

JBoss – Haute disponibilité

David THIBAU – 2015

david.thibau@gmail.com



Agenda

❖ Introduction

- Concepts du clustering
- Fonctionnalités de clustering de Jboss

❖ JGroups

❖ Apache comme répartiteur HTTP

- mod_jk
- mod_proxy
- mod_proxy_ajp
- mod_cluster

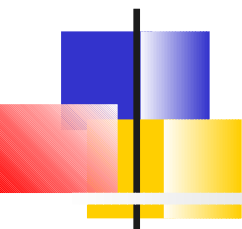
❖ Infinispan

❖ Réplication de session http

❖ Clustering d'EJBs

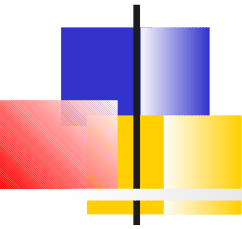
- EJBs Session
- EJBs Entité

❖ Clustering de serveurs HornetQ



Introduction

Concepts du clustering
Fonctionnalités de clustering de
JBoss



Concepts du clustering

Définitions

Typologies de cluster

Répartition de charge

Haute-disponibilité et réplication



Clustering

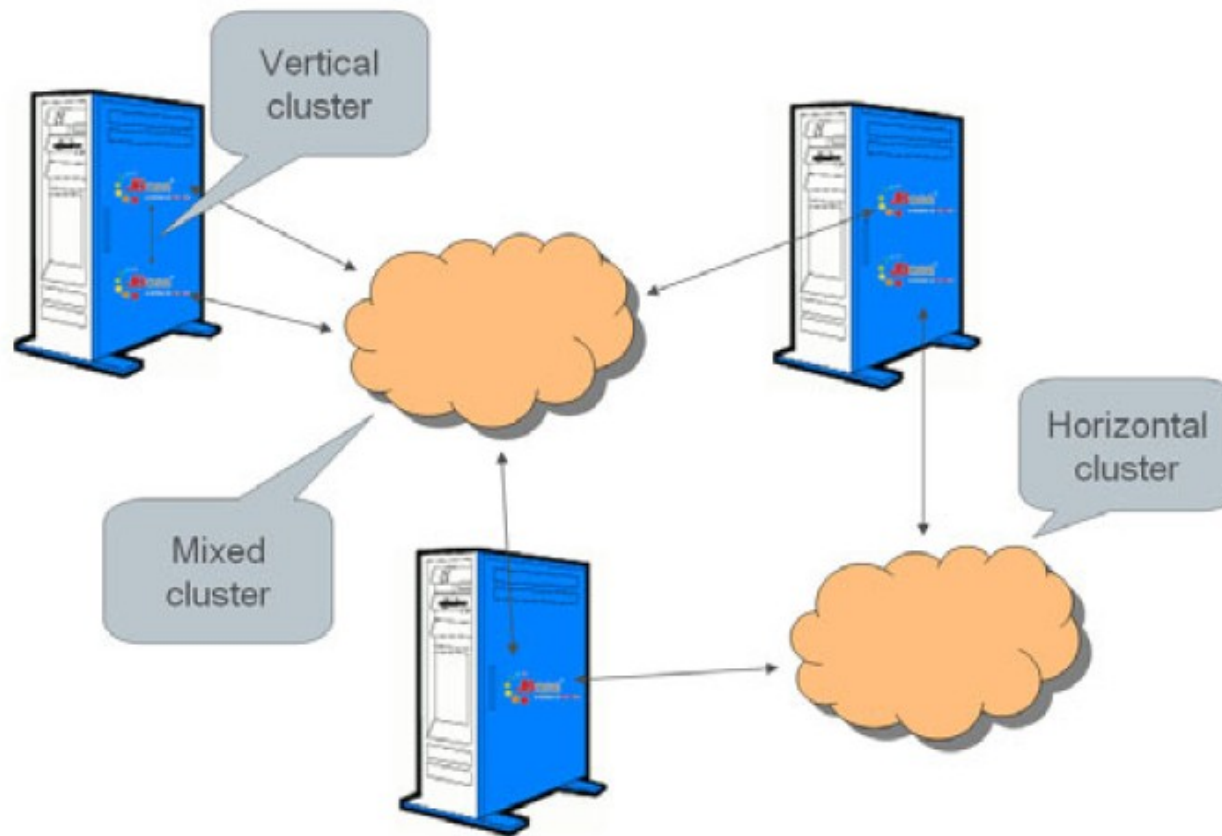
- ❖ Le clustering consiste à exécuter la même application simultanément sur différents serveurs : les nœuds
- ❖ Le clustering a deux objectifs principaux :
 - Augmenter la résistance du système à la charge utilisateur en ajoutant des serveurs
 - Permettre de rendre un service malgré des défaillances matérielles en insérant de la redondance : la tolérance aux pannes
- ❖ Une architecture est dite « **scalable** » si elle permet facilement (sans modification de l'application) d'ajouter des nœuds au cluster

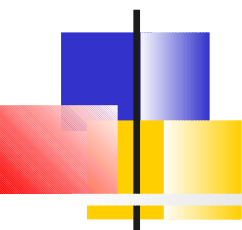


Topologies de cluster

- ❖ On distingue deux topologies de cluster (qui peuvent être mixées):
 - **Vertical** : Dupliquer le nombre de serveurs sur une machine
=> Optimisation de la puissance de la machine
 - **Horizontal** : Dupliquer le nombre de machines
=> architecture plus résistante aux défaillances matérielles

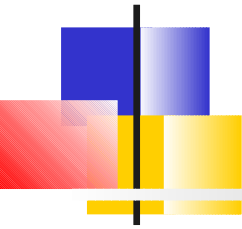
Example





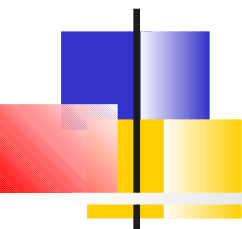
Comparaison des topologies

- ❖ Si l'on dispose d'un serveur avec beaucoup de RAM et plusieurs processeurs; une seule instance d'une JVM ne pourra pas utiliser toutes les ressources.
=> Dans ce cas, un cluster **vertical** sera approprié et aura l'avantage d'éviter des temps latence induits par les échanges réseau.
- ❖ Les clusters **horizontaux** s'imposent lorsque les ressources d'une machine sont pleinement utilisées.
 - Ils sont également utilisés lorsque les nœuds se répartissent sur diverses zones géographiques. (Attention aux temps réseau)
 - La tolérance aux fautes et la scalability sont des arguments en faveur des clusters horizontaux.



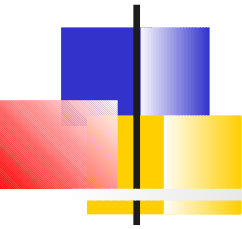
Découverte automatique

- ❖ Il n'est pas toujours nécessaire de spécifier les différents nœuds participant à un cluster
- ❖ La **découverte automatique** est la capacité de découvrir les instances d'un serveur présents sur le réseau susceptibles de former un cluster
- ❖ Cette fonctionnalité est possible grâce au multicast.
Les instances écoutent sur un canal de communication utilisé par chacun pour envoyer des messages



Uniformité des nœuds

- ❖ En général, les applications et services déployés sur les différents nœuds d'un cluster sont identiques. On parle alors de clusters **homogènes**
- ❖ Cependant, les clusters **hétérogènes** sont quelquefois inévitables dans certains environnements de production où les ressources matérielles sont limitées et partagées.
- ❖ Certains services des clusters, comme par exemple JNDI, s'adaptent mieux aux clusters homogènes.



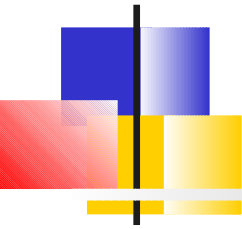
Déploiement d'application

- ❖ Les techniques de déploiement d'application peuvent varier. On distingue trois options :
 - **Indépendant** : chaque serveur a son propre système de fichier avec une copie de l'application
 - **Partagé** : Système de fichier partagé où réside l'application
 - **Géré** : Un serveur d'administration fournit l'application aux autres serveurs
 - **Farming** : Chaque nœud du cluster peut être le serveur d'administration



Répartition de charge

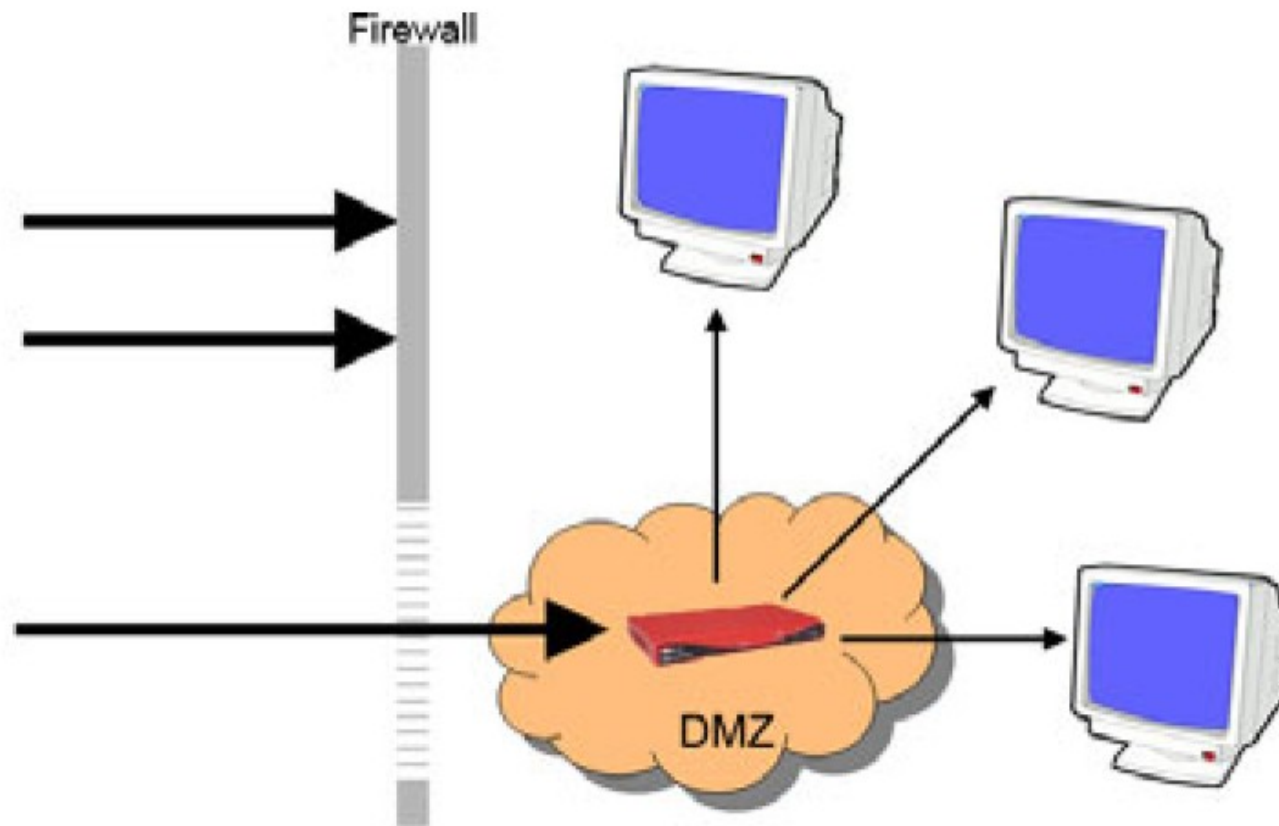
- ❖ La **répartition de charge** (load balancing) consiste à router les requêtes clients vers différents nœuds du serveur
- ❖ Cette tâche est effectuée par un répartiteur qui peut être matériel ou logiciel
- ❖ Le répartiteur est l'unique point d'entrée dans l'environnement de l'application
- ❖ La répartition permet de rendre scalable une application en déclarant de nouveaux nœuds dans le répartiteur.

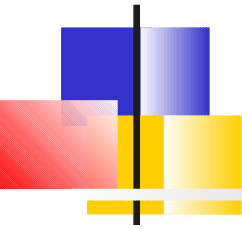


Topologie

- ❖ Lorsque le client accède à un site, le serveur DNS route la requête vers le répartiteur qui la redirige vers un des nœuds du cluster
- ❖ Le répartiteur distribue la charge sur les différents serveurs applicatifs et minimise les risques de sécurité liés à des accès directs aux serveurs applicatifs
- ❖ En général, le répartiteur est donc présent en DMZ

Répartiteur en DMZ





Stratégies de répartition

- ❖ Le répartiteur peut appliquer différentes stratégies de répartition, les plus communes sont :
 - **Au hasard** : Le répartiteur envoie la requête à un serveur tiré au hasard
 - **Round robin** : Le répartiteur parcourt séquentiellement une liste de serveurs
 - **Sticky session** ou **First available** : Pour la première requête d'un client donné, le répartiteur choisit un nœud selon une des 2 autres stratégies. Les requêtes suivantes du même client sont routées vers ce même nœud.
- ❖ Ces différentes stratégies peuvent être agrémentées en donnant des poids sur les différents nœuds.



Répartiteur de charge matériel

- ❖ En général, les répartiteurs matériels exposent une unique adresse IP pour tout le cluster et maintiennent une table d'adresses internes pour chaque nœud du cluster
- ❖ Lorsque le répartiteur reçoit une requête, il réécrit son entête afin de la diriger vers un nœud particulier du cluster
- ❖ Lorsqu'un nœud défaille, le répartiteur est informé afin de le supprimer de sa table de routage



Répartiteur logiciel

- ❖ Les répartiteurs logiciels sont moins chers et plus faciles à mettre en place mais consomment de la mémoire et du CPU et sont à la merci des autres logiciels s'exécutant sur le système
- ❖ Il existe des répartiteurs au niveau du système d'exploitation
 - Microsoft's Network Load Balancing Service (NLBS)
 - Pure Load Balancer (PLB) for Unix
- ❖ En général pour les applications Java EE, un serveur Web est utilisé (JBoss Web Server, Apache, ou IIS).



Haute disponibilité

- ❖ Lorsqu'un serveur défaille, le service peut continuer grâce aux autres serveurs
- ❖ Le but recherché est donc la disponibilité du service (*availability*)
- ❖ Afin qu'une défaillance soit complètement transparente pour l'utilisateur, il est nécessaire de répliquer l'état de sa session utilisateur.
- ❖ Une autre solution peut être de perdre la session utilisateur et de le rediriger vers un autre serveur

=> Ces deux types de redondance n'ont pas du tout les mêmes impacts sur la performance globale du cluster



Quelques chiffres

Service OK	Nombre de 9	Temps down cumulé sur une année
98%		7,3 jours
99%	2-neuf	87,6 heures
99,5%		43,8 heures
99,9%	3-neuf	8,8 heures
99,95%		4,4 heures
99,99%	4-neuf	53 minutes
99,999%	5-neuf	5,3 minutes
99,9999%	6-neuf	31 secondes



Modes de réplication

- ❖ Pour garantir à 100% la tolérance aux pannes, une réplication **synchrone** doit être mise en place :

la réponse n'est rendue au client que lorsque l'état de la session a été répliquée sur tous les nœuds

- ❖ Pour améliorer la réactivité du système, une réplication **asynchrone** est généralement utilisée :

la réponse est rendue avant que la réplication ne soit terminée.

=> Cette stratégie ne peut pas garantir une tolérance aux pannes à 100%.



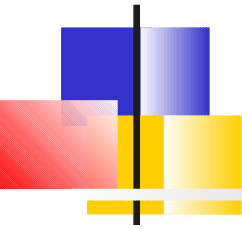
Réplication totale ou de voisinage

- ❖ La réplication **totale** consiste à répliquer un état sur tous les nœuds d'un cluster.
Elle induit un énorme usage mémoire car chaque nœud contient la totalité des sessions utilisateur à un instant t .
Les CPUs sont également fortement sollicités pour la réplication
- ❖ La réplication de **voisinage** (*buddy replication*) consiste à ne répliquer l'état d'un nœud que sur un sous-ensemble du cluster.
Si un nœud défaille, l'état peut être restauré sur un des nœuds en backup



Invalidation de cache

- ❖ Lors de la réplication de données provenant d'un support persistant (EJBs entités), les mises à jour sur ces entités n'ont pas nécessairement besoin d'être répliquées.
- ❖ Une alternative est d'envoyer un message invalidant les caches des autres nœuds qui pourront rafraîchir leurs données grâce au support de persistance (la base de données)
- ❖ Les messages d'invalidation sont beaucoup plus petits que les messages servant à la réplication ainsi les temps de latence dus au réseau sont diminués.



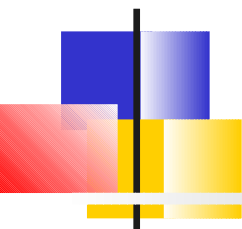
Fonctionnalités de clustering de JBoss



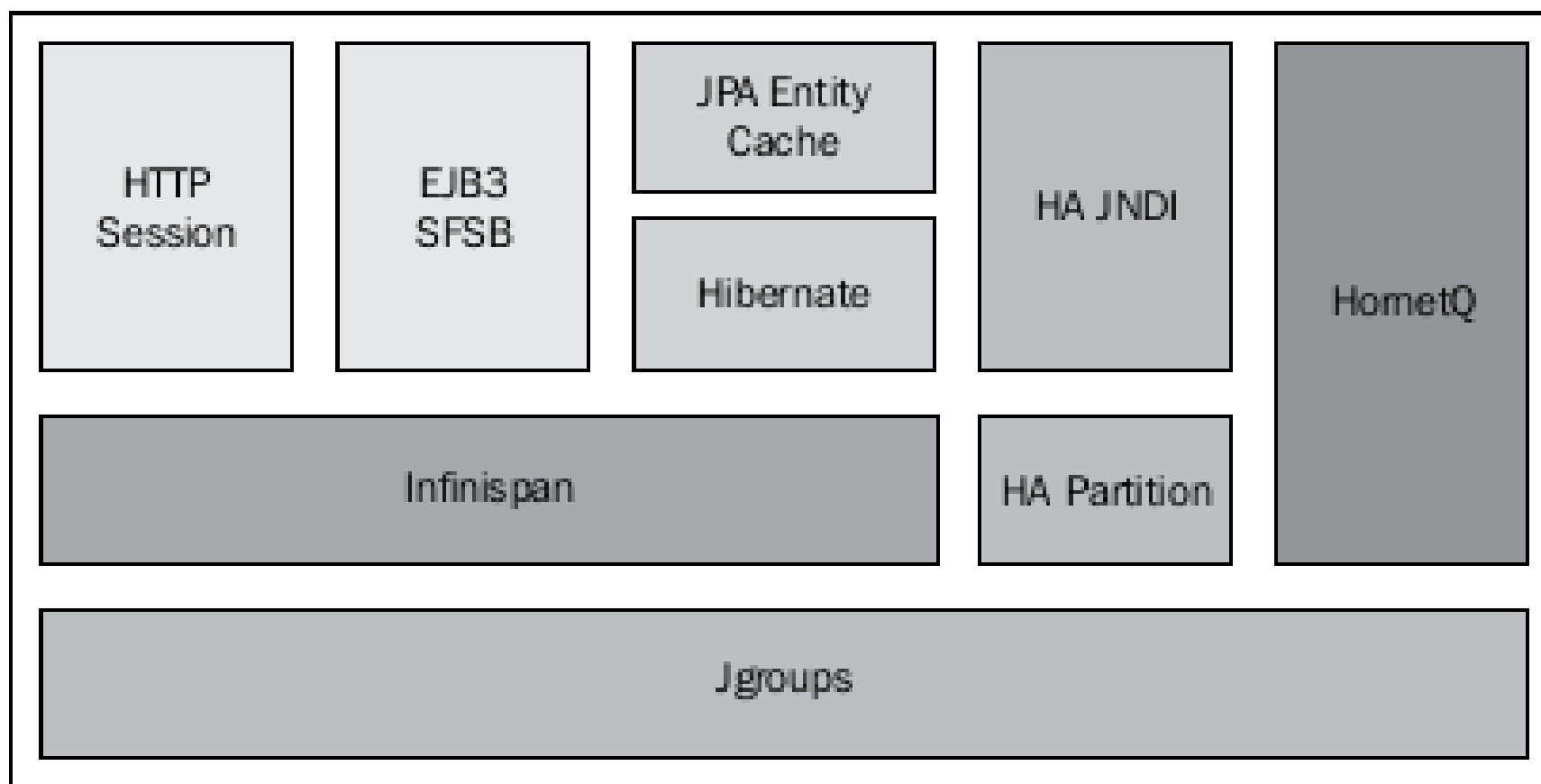
Sous-systèmes

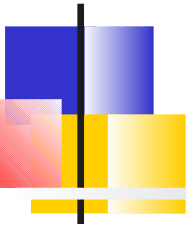
Les différents sous-systèmes prenant part à la mise en place d'un cluster Jboss sont

- **JGroups** utilisé pour la communication entre nœuds
- **Infinispan** qui fait office de cache de réplication et synchronisation entre noeuds
- **HornetQ** avec ses propres capacités de clustering



Architecture





Mode standalone et domaine

AS 7 est distribué avec des fichiers de configuration dédiés au cluster :

- ***standalone-ha.xml*** en mode standalone
- Profils ***ha-profile*** en mode domaine

Un cluster de serveurs *standalone* est un choix possible.

Cependant, il sera alors nécessaire de mettre au point ses propres outils de gestion des nœuds

Une organisation en domaine permet de bénéficier de ce type d'outils



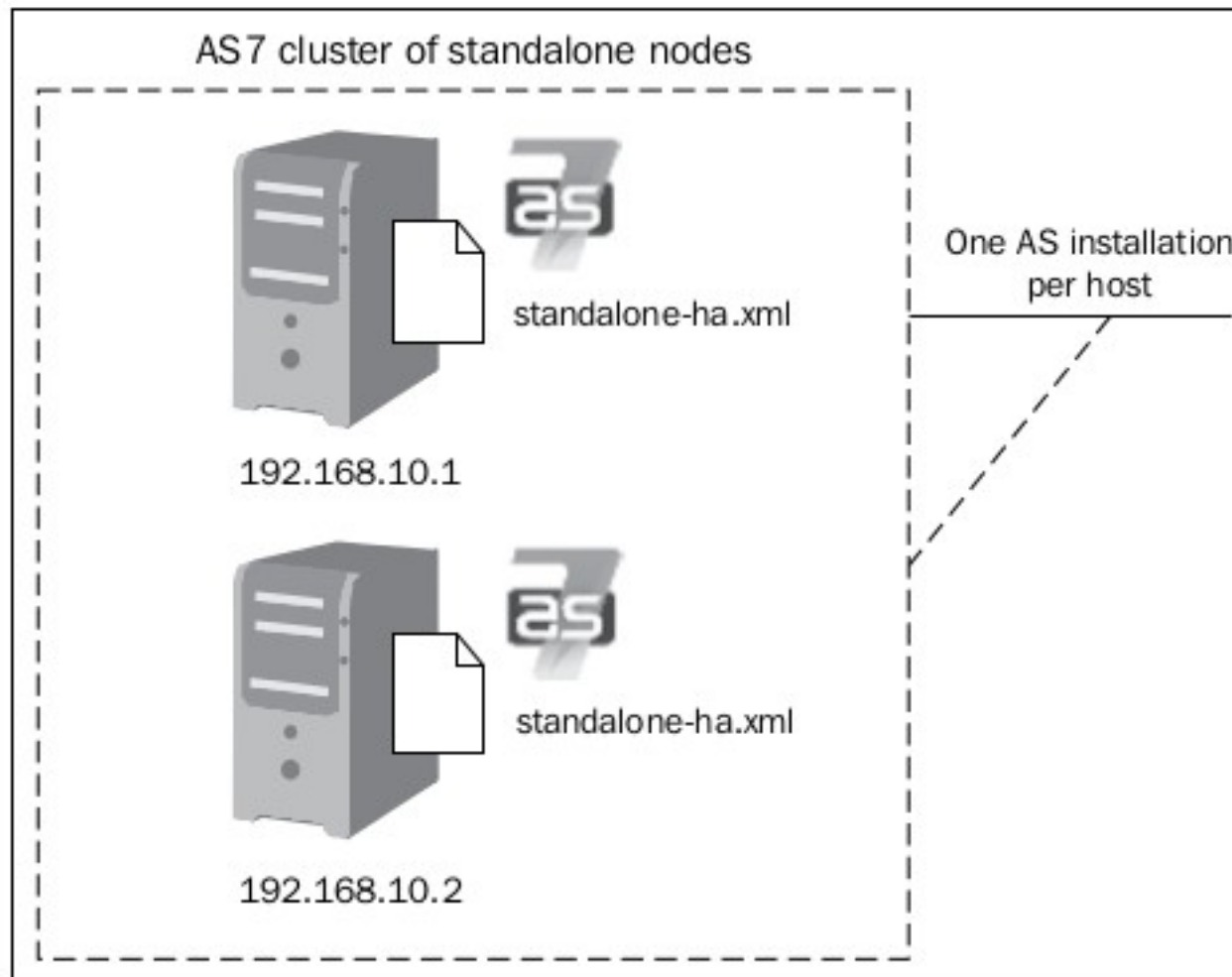
Vertical/Horizontal

Un cluster horizontal nécessite le moins de configuration ; la seule chose à faire est d'associer le serveur à son adresse IP et de démarrer le serveur avec une configuration cluster.

Pour un cluster vertical, les conflits de port doivent être évités. 2 options sont possible :

- Définir plusieurs adresses IP sur la même machine
- Définir un offset de port pour chaque serveur

Horizontal





Exemple horizontal

```
<interfaces>
```

```
<interface name="management"><inet-address value="192.168.10.1"/></interface>
```

```
<interface name="public"><inet-address value="192.168.10.1"/></interface>
```

```
</interfaces>
```

Seconde machine (192.168.10.2)

```
<interfaces>
```

```
<interface name="management"><inet-address value="192.168.10.2"/></interface>
```

```
<interface name="public"><inet-address value="192.168.10.2"/></interface>
```

```
</interfaces>
```

...

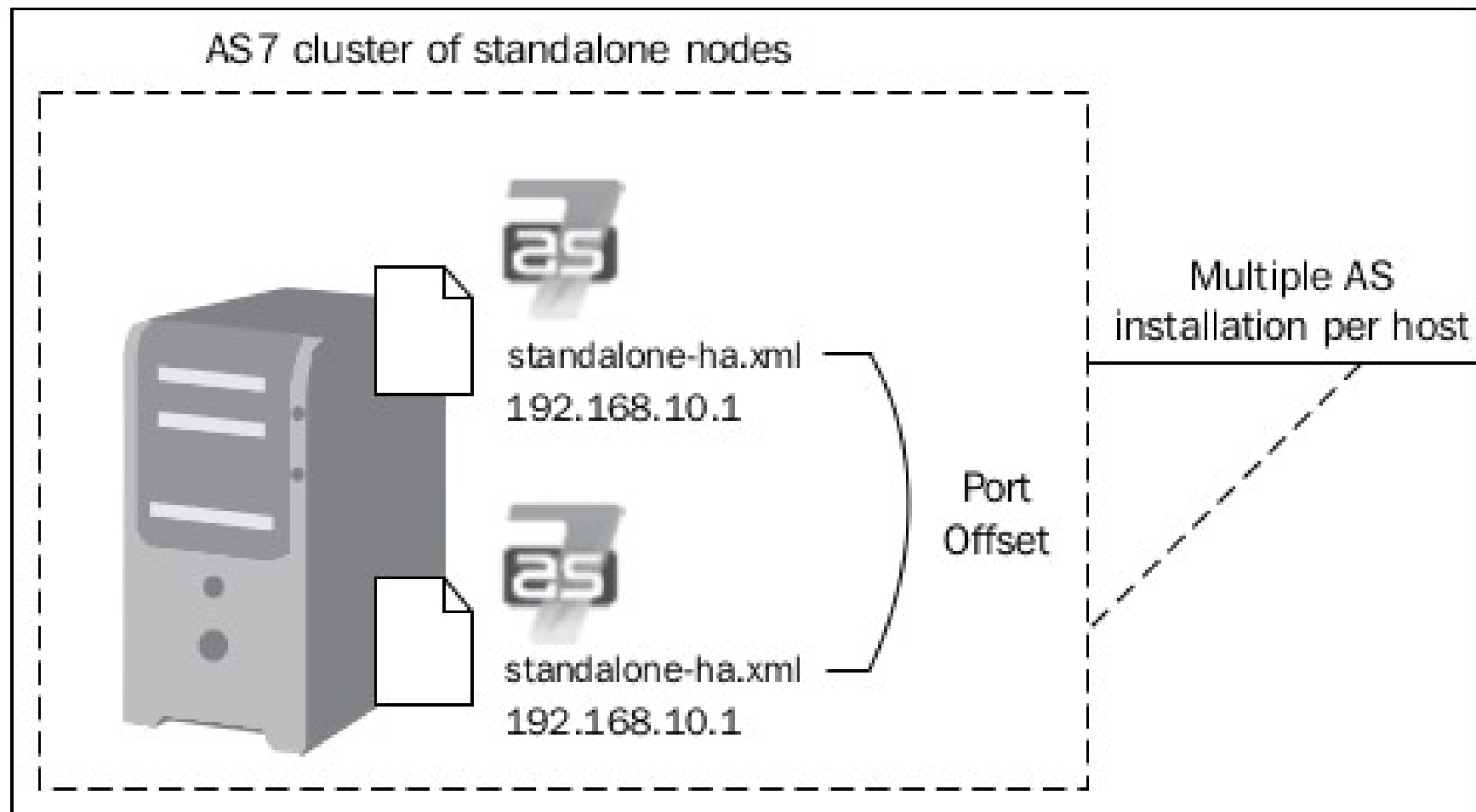
```
standalone.bat --server-config=standalone-ha.xml
```



Utilisation d'adresses IP différentes

- ❖ Une autre alternative pour mettre en place un cluster vertical est de définir plusieurs adresses IP sur la même machine
- ❖ Par exemple sur Linux, binder un nœud vers *localhost* et l'autre sur l'adresse IP
- ❖ Avec Windows, on peut créer un *Microsoft Loopback Adapter* avec 2 adresses IP
 - Dans le panneau de configuration, aller à « Ajouter du matériel »
 - Cocher « Oui, j'ai déjà connecté le matériel »
 - Sélectionner tout en bas « Ajouter un nouveau matériel »
 - Sélectionner l'option d'installation manuelle, puis « Carte réseau »
 - Sélectionner « Microsoft » comme fabricant et « Carte de bouclage Microsoft »
 - Ensuite, aller dans « Connexion réseau » dans le panneau de configuration
 - Sélectionner les propriétés TCP/IP de la nouvelle carte réseau et indiquer une adresse non routable (ex : 192.168.1.140) avec un masque de sous-réseau de 255.255.255.0.
 - Sélectionner « Avancé »... et ajouter une autre adresse IP (exemple 192.168.1.141) avec le même masque de sous-réseau
 - Redémarrer la machine ou le réseau !!

Vertical avec offset de port





Exemple

```
<!--server 1 UNCHANGED -->
```

```
<socket-binding-group name="standard-sockets" default-  
interface="public">
```

```
. . . . .
```

```
</socket-binding-group>
```

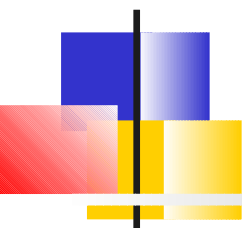
Déclaration de l'offset de port (150):

```
<!--server 2 -->
```

```
<socket-binding-group name="standard-sockets" default-  
interface="public" port-offset="150">
```

```
. . . . .
```

```
</socket-binding-group>
```

Interfaces d'administration

```
<!--server 1 UNCHANGED -->
<socket-binding-group name="standard-sockets" default-interface="public" >
. . . . .
<socket-binding name="management-native" interface="management" port="9999"/>
<socket-binding name="management-http" interface="management" port="9990"/>
. . . . .
</socket-binding-group>
```

Sur le second serveur, un port différent

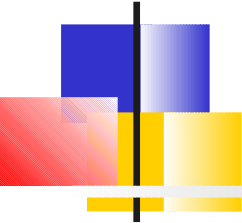
```
<!--server 2 -->
<socket-binding-group name="standard-sockets" default-interface="public" port-
  offset="150">
. . . . .
<socket-binding name="management-native" interface="management" port="9999"/>
<socket-binding name="management-http" interface="management" port="9990"/>
. . . . .
</socket-binding-group>
```



Domaine

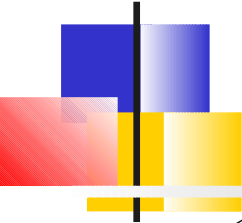
La configuration cluster est déjà présente sous la forme du profil **ha-profile** dans *domain.xml*.

Il suffit alors d'associer un groupe de serveurs à ce profil



Groupe de serveur vers profil ha

```
<server-groups>
  <server-group name="main-server-group" profile="ha">
    <jvm name="default">
      <heap size="64m" max-size="512m"/>
    </jvm>
    <socket-binding-group ref="ha-sockets"/>
  </server-group>
  <server-group name="other-server-group" profile="ha">
    <jvm name="default">
      <heap size="64m" max-size="512m"/>
    </jvm>
    <socket-binding-group ref="ha-sockets"/>
  </server-group>
</server-groups>
```



<socket-binding>

```
<socket-binding-group name="ha-sockets" default-interface="public">
  <socket-binding name="http" port="8080"/>
  <socket-binding name="https" port="8443"/>
  <socket-binding name="jgroups-diagnostics" port="0" multicast-address="224.0.75.75" multicast-
port="7500"/>
  <socket-binding name="jgroups-mping" port="0" multicast-address="230.0.0.4" multicast-
port="45700"/>
  <socket-binding name="jgroups-tcp" port="7600"/>
  <socket-binding name="jgroups-tcp-fd" port="57600"/>
  <socket-binding name="jgroups-udp" port="55200" multicast-address="230.0.0.4" multicast-
port="45688"/>
  <socket-binding name="jgroups-udp-fd" port="54200"/>
  <socket-binding name="jmx-connector-registry" port="1090"/>
  <socket-binding name="jmx-connector-server" port="1091"/>
  <socket-binding name="jndi" port="1099"/>
  <socket-binding name="modcluster" port="0" multicast-address="224.0.1.105" multicast-
port="23364"/>
  <socket-binding name="osgi-http" port="8090"/>
  <socket-binding name="remoting" port="4447"/>
  <socket-binding name="txn-recovery-environment" port="4712"/>
  <socket-binding name="txn-status-manager" port="4713"/>
</socket-binding-group>
```



host.xml

Dans *host.xml*, les serveurs doivent être associés au socket binding group **ha-sockets**

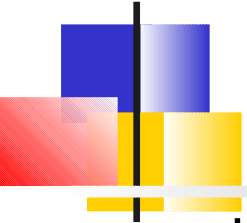
```
<servers>
<server name="server-one" group="main-server-group" >
<socket-binding-group ref="ha-sockets" port-offset="150"/>
<jvm name="default"/>
</server>
<server name="server-two" group="main-server-group" auto-start="true" >
<socket-binding-group ref="ha-sockets" port-offset="250"/>
<jvm name="default"/>
</server>
<server name="server-three" group="other-server-group" auto-start="false">
<socket-binding-group ref="standard-sockets" port-offset="350"/>
<jvm name="default"/>
</server>
</servers>
```



Démarrage en mode lazy

Avec JBoss AS 7 les services du clustering sont démarrés à la demande.

Ainsi, juste démarrer les serveurs en cluster sans application clusterisés ne démarre aucun service ni channels JGroups



Farming – no more !

JBoss AS 7 n'utilise plus le concept de *farming* comme dans les précédentes versions

Pour distribuer une application sur un ensemble de nœud, la meilleure approche est de créer un script CLI qui déploie l'application sur tous les nœuds

Par exemple :

```
connect 192.168.10.1
```

```
deploy Example.war
```

```
connect 192.168.10.2
```

```
deploy Example.war
```

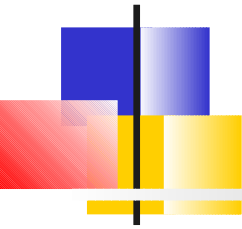
Puis :

```
jboss-cli.sh --user=username --password=password  
--file=deploy.cli
```



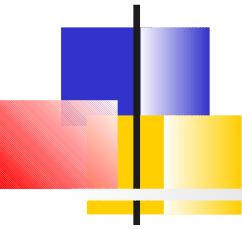
Capacités de clustering

- ❖ Permet la répartition de charge via :
 - Répartiteur HTTP matériel ou logiciel
 - Proxy EJB (Stateless, Stateful, MDB) ou contexte JNDI
- ❖ Permet la tolérance aux pannes via la réplication d'état
 - Sessions HTTP
 - EJB stateful 2.1 ou 3.0
 - EJB entités 3.0
- ❖ Clustering des services J2EE
 - JMS
 - JNDI



Répartiteurs

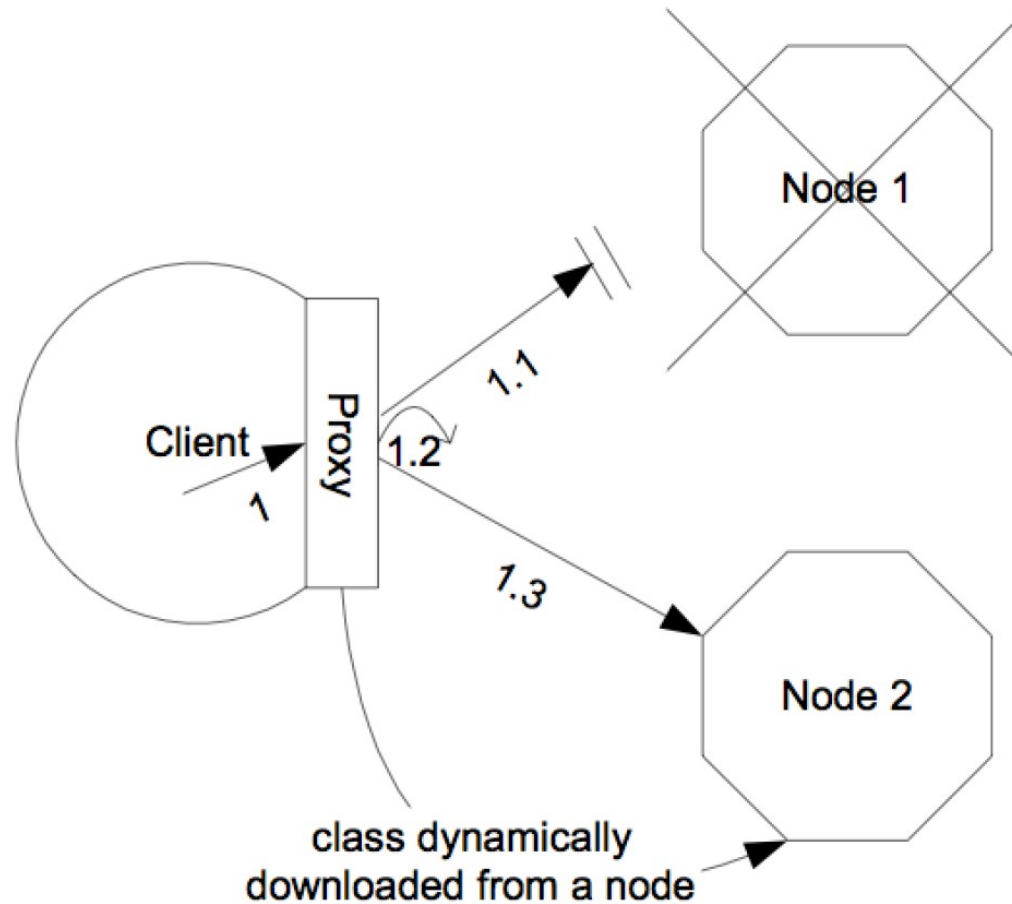
- ❖ Les architectures du cluster du point de vue du client qui y accède peuvent être classifiées en deux types.
 - Insertion d'un **intercepteur** côté client :
Client des EJBs (client lourd, servlet ou Java Beans), Client des services JNDI ou RMI
 - Insertion d'un **répartiteur** de charge :
Client léger accédant en HTTP



Intercepteur côté client

- ❖ Pour les appels distants, le serveur JBoss génère une classe *stub* exposant les services distants et permettant au client de cacher les communications réseau.
- ❖ Dans le cas d'un cluster, l'objet *stub* maintient une connaissance de la constitution du cluster afin de router les appels clients vers les nœuds actifs.
- ❖ Il contient également les algorithmes de répartition de charge et/ou de tolérance aux défaillances

Intercepteur / proxy

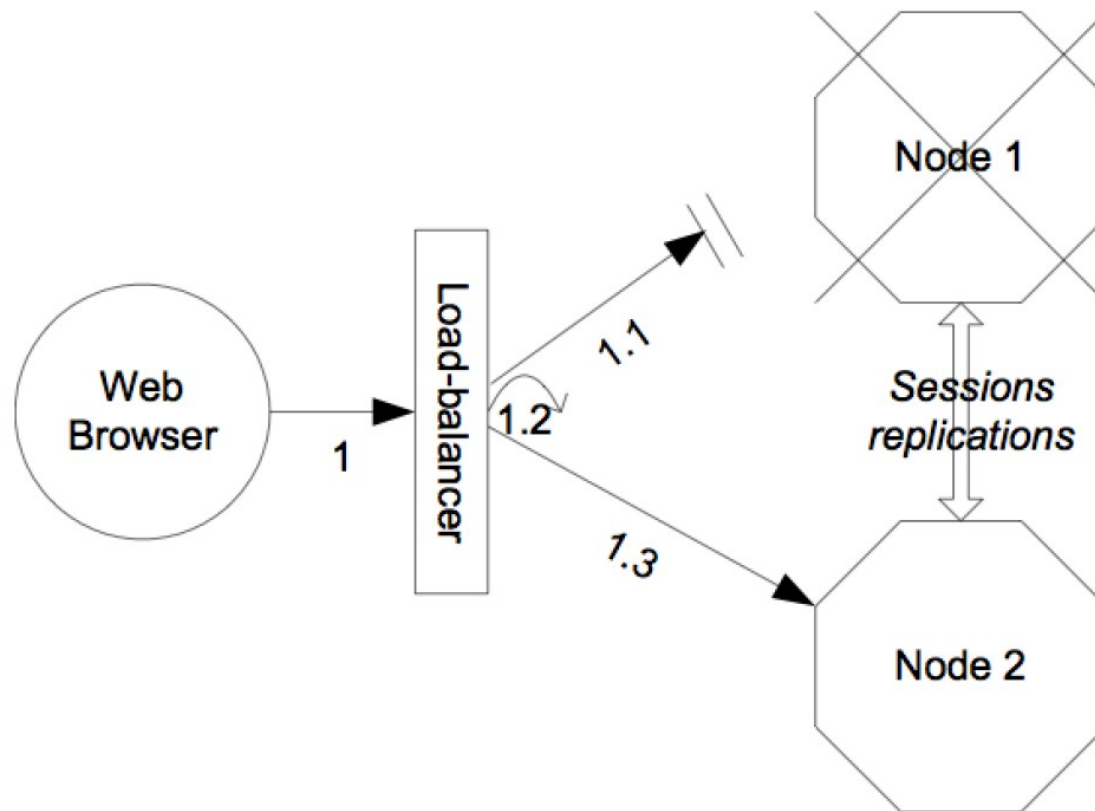




Service HTTP

- ❖ Dans le cas du service HTTP, un répartiteur de charge inclut au cluster dirige les requêtes vers les nœuds actifs en appliquant également les algorithmes de répartition de charge et de tolérance aux fautes.
- ❖ Le client ne s'adresse alors qu'au répartiteur

Répartiteur de charge





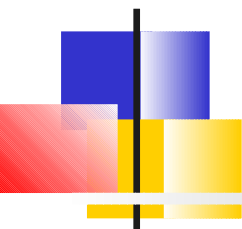
Stratégie de répartition

- ❖ Dans les deux types d'architectures, JBoss propose 3 stratégies de répartition :
 - **Round-robin** : Chaque requête est alternativement dirigée vers un nouveau nœud. Le premier nœud est choisi au hasard
 - **First-available** : Un nœud disponible est choisi pour la première requête et toutes les requêtes suivantes sont dirigées vers ce nœud.
 - **First-AvailableIdenticalAllProxies** : Identique à *First-Available* mais le nœud élu est partagé par tous les intercepteurs de la même famille, i.e. les intercepteurs appelant le même EJB cible



TP

Démarrage d'un cluster vertical

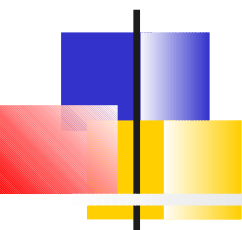


JGroups



Introduction

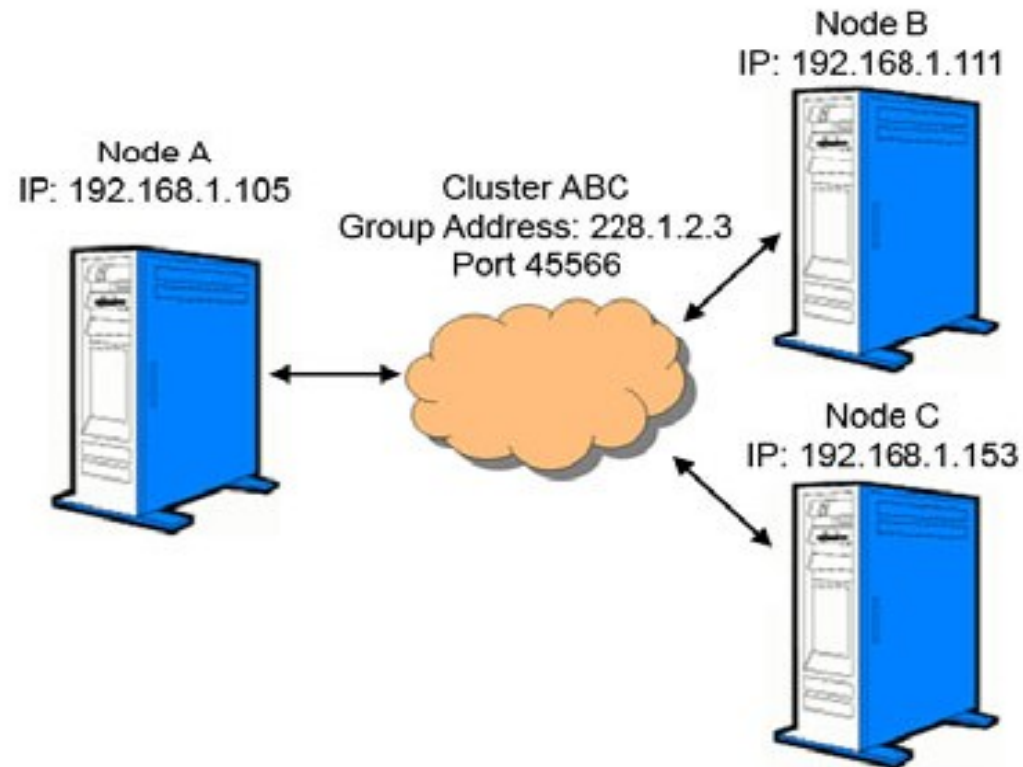
- ❖ La librairie ***JGroups*** permet des communications *peer-to-peer* entre les nœuds d'un cluster.
- ❖ Elle s'appuie sur une pile de protocoles de communication permettant principalement :
 - le transport sûr de messages
 - la découverte de nœud
 - la détection de panne
 - la gestion de groupes.
- ❖ *JGroups* utilise les capacités des protocoles de transport sous-jacent pour envoyer les messages en multicast (UDP) ou de simuler le multicast avec TCP
- ❖ Cette librairie intervient dans tous les services de clustering de *JBoss*



Comparaisons des protocoles

- ❖ Comparaison des deux principaux protocoles de transport
 - **UDP** est le protocole recommandé surtout dans un environnement LAN
 - **TCP** est le plus verbeux (multiple unicast) mais le plus sûr. Il est présent sur toutes les machines connecté au WAN (Internet).

Multicasting

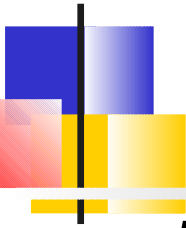




UDP et multicast

Lors de l'utilisation de UDP et du multicast plusieurs problèmes typiques peuvent survenir :

- Les nœuds sont derrière un firewall qui bloque les ports multicast
- Le réseau est derrière un gateway qui ne prend pas en compte le multicast (réseau du domicile par exemple).
=> ajouter une route au gateway afin qu'il redirige le trafic multicast sur le réseau local



Test multicast avec JGroups

JGroups est distribué avec deux programmes de test du multicast: ***McastReceiverTest*** et ***McastSenderTest***

Démarrer *McastReceiverTest*, :

```
java -classpath jgroups-3.0.0.Final.jar org.jgroups.tests.  
McastReceiverTest -mcast_addr 224.10.10.10 -port 5555
```

Puis *McastSenderTest*:

```
java -classpath jgroups-3.0.0.Final.jar  
org.jgroups.tests.McastSenderTest -mcast_addr  
224.10.10.10 -port 5555
```

Les caractères saisis dans la fenêtre de l'émetteur doivent s'afficher dans la fenêtre du récepteur

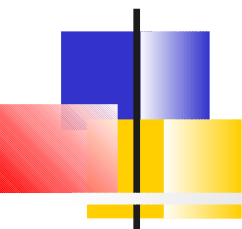


Channels

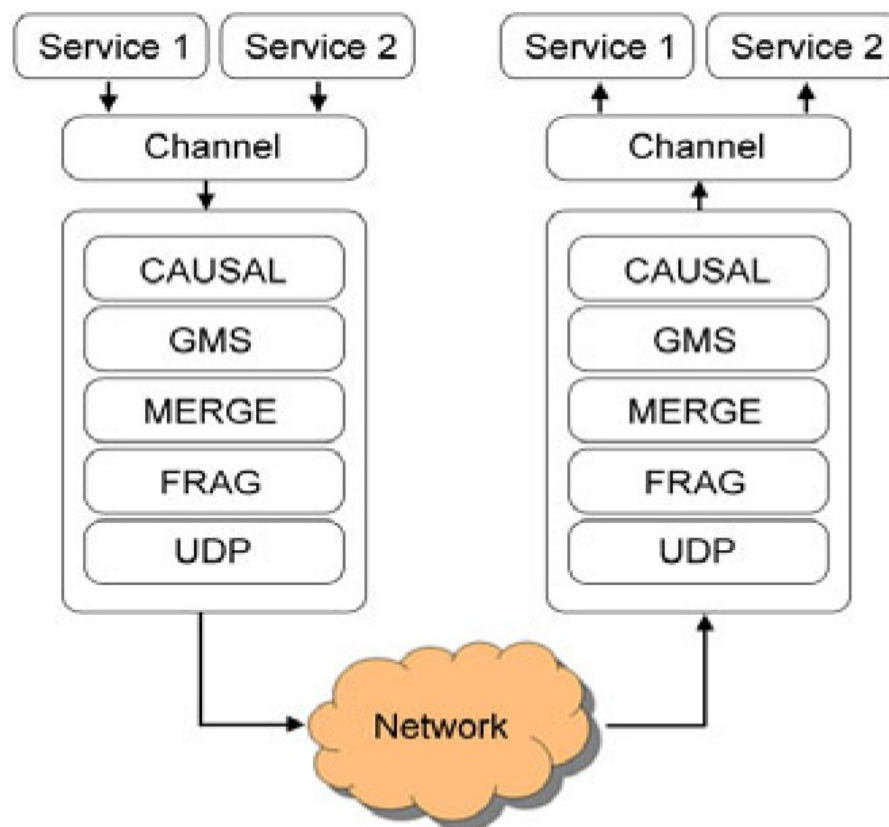
- ❖ Un **channel** fournit l'API des applications pour envoyer des messages aux autres membres du cluster.

Un channel est une **pile de protocoles**

- Lorsqu'un message est envoyé, il traverse la pile de protocoles
 - La pile est parcourue dans le sens inverse lorsque le message est reçu
- ❖ Chaque protocole apporte des services liés au clustering et peut envoyer, recevoir, modifier, réordonner, faire passer ou bloquer des messages.



Exemple de pile JGroups





Types de protocoles

Transport : UDP, TCP

Découverte : PING, TCPPING, TCPGOSSIP, MPING

Fragmentation/Réassemblage : FRAG, FRAG2

Transmission sûre : CAUSAL, NAKACK, pbcast.NAKACK, SMACK, UNICAST, PBCAST, pbcast.STABLE

Gestion d'appartenance : pbcast.GMS, MERGE, MERGE2, VIEW_SYNC

Détection de panne : FD, FD_SIMPLE, FD_PING, FD_ICMP, FD SOCK, VERIFY_SUSPECT

Sécurité : AUTH

Transfert d'état : pbcast.STATE_TRANSFER, pbcast.STREAMING_STATE_TRANSFER

Debug : PERF_TP, SIZE, TRACE

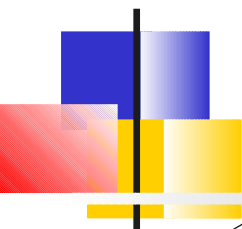
Autres : COMPRESS, pbcast.FLUSH



Configuration JGroups

L'API *JGroups* était déjà utilisé dans les versions précédentes de JBoss AS. Les différents *channels* étaient alors définis dans des fichiers de configuration spécifiques

Avec JBoss AS 7, la configuration JGroups est embarquée dans le fichier de configuration principal dans le sous-système *jgroups*



Sous-système JGroups

```
<subsystem xmlns="urn:jboss:domain:jgroups:1.0" default-stack="udp">
  <stack name="udp">
    <transport type="UDP" socket-binding="jgroups-udp" diagnostics-socket-binding="jgroups-diagnostics"/>
    <protocol type="PING"/>
    <protocol type="MERGE2"/>
    <protocol type="FD SOCK" socket-binding="jgroups-udp-fd"/>
    <protocol type="FD"/>
    <protocol type="VERIFY_SUSPECT"/>
    <protocol type="BARRIER"/>
    <protocol type="pbcast.NAKACK"/>
    <protocol type="UNICAST"/>
    <protocol type="pbcast.STABLE"/>
    <protocol type="VIEW_SYNC"/>
    <protocol type="pbcast.GMS"/>
    <protocol type="UFC"/>
    <protocol type="MFC"/>
    <protocol type="FRAG2"/>
    <protocol type="pbcast.STREAMING_STATE_TRANSFER"/>
    <protocol type="pbcast.FLUSH"/>
  </stack>
  <!-- More stacks -->
</subsystem>
```



Configuration des piles

La configuration définit donc des *stack* que l'on peut attribuer aux services de clustering

L'attribut ***default-stack*** du sous-système définit la pile Jgroups par défaut.

Voir *docs/schema/jboss-jgroups.xsd*



UDP / TCP

Avec une socket UDP multicast , un client peut contacter plusieurs serveurs avec un paquet unique sans connaître les adresses IP des différents serveurs.

Pour utiliser UDP avec JBoss AS 7 il faut modifier l'attribut ***default-stack***

Par exemple :

```
<subsystem xmlns="urn:jboss:domain:jgroups:1.0"  
default-stack="udp">
```



Socket bindings

```
<socket-binding-group name="clustering-sockets" default-  
  interface="loopback" port-offset="0">  
  <socket-binding name="jgroups-udp" port="55200"  
    multicast-address="230.0.0.4" multicast-port="45688"/>  
  <socket-binding name="jgroups-udp-fd" port="54200"/>  
  <socket-binding name="jgroups-diagnostics" port="0"  
    multicast-address="224.0.75.75" multicast-port="7500"/>  
  <socket-binding name="jgroups-tcp" port="7600"/>  
  <socket-binding name="jgroups-tcp-fd" port="57600"/>  
  <socket-binding name="jgroups-mping" port="0"  
    multicast-address="230.0.0.4" multicast-port="45700"/>  
</socket-binding-group>
```



Personnalisation des protocoles

Les protocoles définies dans une stack ont de nombreux paramètres de configuration par défaut.

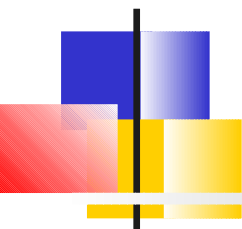
Il est possible de surcharger la configuration par défaut en définissant une nouvelle valeur.

```
<stack name="udp">
  <transport type="UDP" ...>
    <property name="enable_bundling">true</property>
    <property name="ip_ttl">0</property>
  </transport>
  <!-- ... -->
</stack>
```



Configuration des threads

```
<subsystem xmlns="urn:jboss:domain:clustering:jgroups:1.0">
  <stack name="udp">
    <transport type="UDP" ...
default-executor="jgroups-default" oob-executor="jgroups-oob" timer-executor="jgroups-timer"/>
```



Répartiteur HTTP Apache

mod_jk
mod_proxy
mod_proxy_ajp
mod_cluster



Introduction

Le serveur http apache est souvent mis en frontal des applications web déployées sur Jboss

Le serveur Apache apporte en effet quelques avantages :

- Il est généralement plus rapide pour servir les ressources statiques que JBoss Web server.
- Le serveur applicatif qui contient des données sensibles peut alors être placé dans une zone sécurisée. Apache se comportant comme un serveur proxy
- Apache peut également devenir un répartiteur de charge, distribuant les requêtes vers les différentes instances de JBoss Web server. En cas de défaillance, Apache continue de façon transparente à distribuer les requêtes vers les nœuds actifs



Types de connexions

La connexion entre Apache et JBoss AS peut se faire via différents module Apache :

- La librairie Tomcat's ***mod_jk***
- La librairie Apache ***mod_proxy***
- Avec Jboss 7, la nouvelle API ***mod_cluster*** API



Apache

- ❖ Le module mod_jk utilise alors le protocole ***ajp*** pour dispatcher les requêtes HTTP vers Tomcat/JBoss.
- ❖ Côté JBoss, 2 stratégies d'équilibrage de charge sont disponibles :
 - Round-robin
 - First available



Mise en pratique

- ❖ Installer Apache et charger le module *mod_jk*
- ❖ Mapper les URLs destinées à Tomcat
- ❖ Configurer les workers dans le *mod_jk* via le fichier *workers.properties*
- ❖ Configurer JBoss/Tomcat
- ❖ Éventuellement, configurer la réplication de session



httpd.conf

Inclure la configuration de mod_jk

```
# Include mod_jk's specific configuration file  
Include conf/mod-jk.conf
```

Ou

```
ln -s mods-available/jk.conf mods-enabled/jk.conf
```



jk.conf

```
# Defining mounting points for all VirtualHost
JkMountCopy all
# 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]"
# JkRequestLogFormat
JkRequestLogFormat "%w %V %T"
# Mount your applications
JkMount /application/* loadbalancer
```



workers.properties

```
# 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.node1.port=8009
worker.node1.host=node1.mydomain.com
worker.node1.type=ajp13
worker.node1.lbfactor=1
worker.node1.cachesize=10
# Define Node2
# modify the host as your host IP or DNS name.
worker.node2.port=8009
worker.node2.host= node2.mydomain.com
worker.node2.type=ajp13
worker.node2.lbfactor=1
worker.node2.cachesize=10
# Load-balancing behaviour
worker.loadbalancer.type=lb
worker.loadbalancer.balance_workers=node1,node2
worker.loadbalancer.sticky_session=1
#worker.list=loadbalancer
# Status worker for managing load balancer
worker.status.type=status
```



JBoss

Du côté JBoss, il faut déclarer le connecteur AJP

```
<subsystem xmlns="urn:jboss:domain:web:1.1">
  <connector name="http" protocol="HTTP/1.1" socket-
binding="http" scheme="http" />
  <connector name="AJP" protocol="AJP/1.3"
socket-binding="ajp" />
  <virtual-server name="localhost">
    <alias name="example.com" />
  </virtual-server>
</subsystem>
```




Port AJP

```
<socket-binding-group name="standard-sockets"  
default-interface="default">  
  <socket-binding name="http" port="8080"/>  
  <socket-binding name="ajp" port="8009"/>  
  . . . .  
</socket-binding-group>
```



mod_proxy

Depuis Apache 1.3, il est possible de configurer le module ***mod_proxy*** qui configure Apache comme serveur proxy

Cela permet de propager les requêtes vers Jboss sans avoir à configurer un connecteur web comme *mod_jk*.

C'est le moyen le plus facile pour mettre Apache en frontal de Jboss mais c'est également le plus lent



Httpd.conf

```
LoadModule proxy_module modules/mod_proxy.so
```

```
ProxyPass /myapp http://localhost:8080/myapp
```

```
ProxyPassReverse /myapp http://localhost:8080/myapp
```



Proxy ajp

Depuis Apache 2.2, il existe un autre module nommé ***mod_proxy_ajp*** qui peut être utilisé de la même façon.

Cependant, il utilise le protocole AJP

```
LoadModule proxy_ajp_module modules/mod_proxy_ajp.so  
ProxyPass / ajp://localhost:8009/  
ProxyPassReverse / ajp://localhost:8009/
```



JBoss

Côté Jboss, le connecteur AJP doit être activé

```
<subsystem xmlns="urn:jboss:domain:web:1.1">
  <connector name="AJP" protocol="AJP/1.3" socket-binding="ajp" />
</subsystem>

. . . . .

<socket-binding-group name="standard-sockets" default-
interface="default">
  <socket-binding name="ajp" port="8009"/>

. . . .
</socket-binding-group>
```

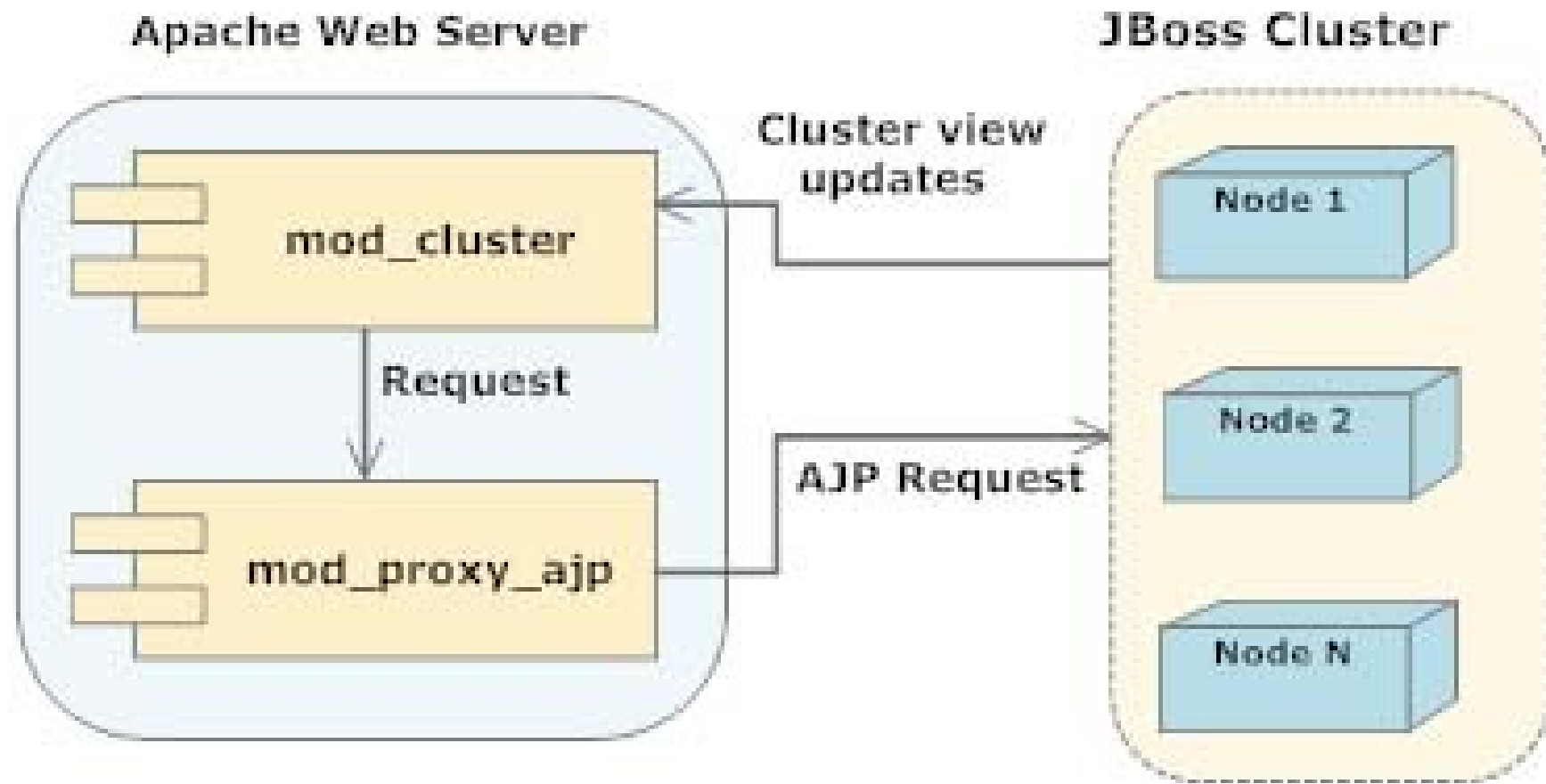


mod_cluster

L'utilisation du module ***mod_cluster*** apporte les avantages suivant:

- Configuration dynamique du cluster
- Possibilité d'obtenir des métriques de charge
- Notification du cycle de vie de l'application

Architecture



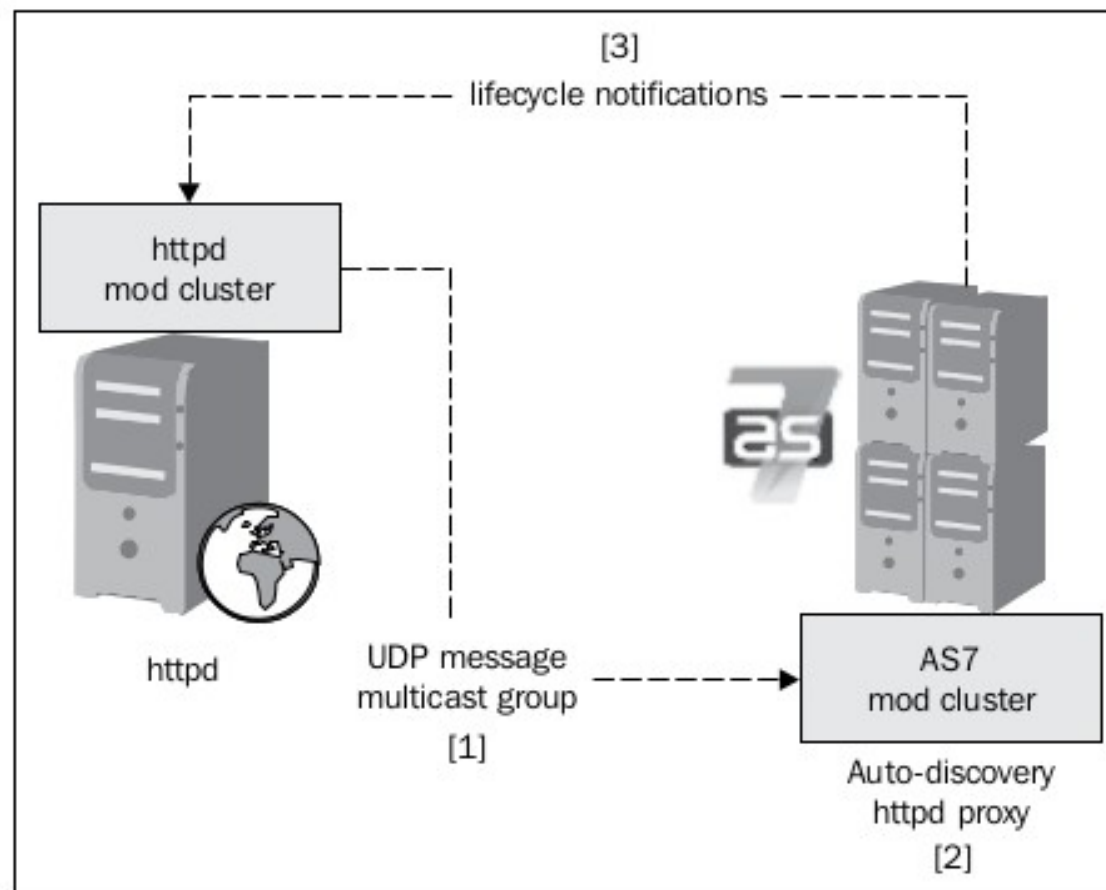
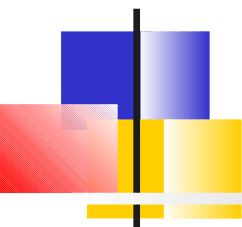


Ajout dynamique de noeud

Avec *mod_cluster*, il est possible d'ajouter ou retirer dynamiquement des nœuds car ils sont découverts automatiquement

La librairie *mod_cluster* du côté *httpd* envoie des messages UDP sur un groupe multicast auquel se sont abonnés les nœuds Jboss

Cela leur permet de découvrir automatiquement les proxy http lors des notifications du cycle de vie applicatif.





Installation de *mod_cluster*

Du côté JBoss 7, le module *mod_cluster* 1.1.3 est déjà présent dans la configuration cluster soit dans *standalone-ha.xml* file ou dans le profil ha de *domain.xml* :

```
<subsystem xmlns="urn:jboss:domain:modcluster:1.0">
  <mod-cluster-config advertise-socket="modcluster"/>
</subsystem>

<socket-binding name="modcluster" port="0" multicast-
address="224.0.1.105" multicast-port="23364"/>
```

Il faut cependant ajouter un attribut dans la configuration du sous-système web :

```
<subsystem xmlns="urn:jboss:domain:web:1.1" default-virtual-
server="default-host" instance-id="${jboss.node.name}"
native="false">
```



Configuration *mod_cluster*

Attributs du sous-système *mod_cluster*

proxy-list : Liste des proxy Apache

proxy-url : Dans le cas d'un seul proxy

advertise : Découverte automatique des proxies, *proxy-list* n'est pas renseigné

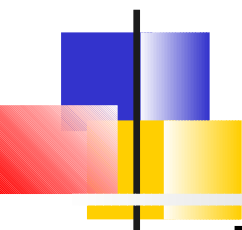
advertise-security-key : Les checksum des messages avec les proxies sont vérifiées avec une clé

excluded-contexts (défaut : *ROOT, admin-console, invoker, jbossws, jmx-console, juddi, web-console*) : Les contextes à exclure

auto-enable-contexts (défaut true) : Contextes automatiquement inclus dans la gestion du proxy

stop-context-timeout (défaut : 10 secondes) : Un contexte est désactivé 10 secondes après son arrêt

socket-timeout (défaut 20 seconds) : Timeout pour déclarer un proxy comme down



Configuration *mod_cluster*

Désactiver les statistiques

```
<subsystem xmlns="urn:jboss:domain:modcluster:1.0">  
<mod-cluster-config proxy-list="192.168.0.1:6666"/>  
</subsystem>
```

Exclure des contextes particuliers

```
<subsystem xmlns="urn:jboss:domain:modcluster:1.0">  
<mod-cluster-config excluded-contexts="ROOT, foo"/>  
</subsystem>
```



Configuration des proxies

Le sous-système `mod_cluster` permet également de configurer le comportement du proxy

stickySession (défaut : `true`) Garder le même nœud pour un client

stickySessionRemove (défaut `false`) : Enlever le mode sticky, si le répartiteur ne peut pas atteindre le nœud

stickySessionForce (défaut `false`) : Retourner une erreur si le répartiteur ne peut pas router vers le nœud sticky

workerTimeout : Nombre de secondes à attendre avant qu'un worker soit disponible



Métriques de charges

Le fournisseur de métriques par défaut donne un poids de 1 à tous les nœuds :

```
<subsystem xmlns="urn:jboss:domain:modcluster:1.0">  
  <mod-cluster-config>  
    <simple-load-provider load="1"/>  
  </mod-cluster-config>  
</subsystem>
```



Métriques dynamiques

Le ***dynamic load provider*** permet de répartir la charge entre les nœuds selon des métriques dynamiques prédéfinis ou personnalisés : *cpu, mem, heap, sessions, receive-traffic, send-traffic, requests, busyness*



Métriques prédéfinis

cpu : Charge CPU

mem : Mémoire système

heap : Heap Java

sessions : Sessions actives

requests : Requêtes actives

send-traffic : Trafic reçu en kb/s

receive-traffic : Trafic envoyé en kb/s

busyness : Pourcentage des threads occupés dans le pool de threads

connection-pool : Connections occupés d'un pool de connexion (JDBC)



Configuration

Les différents métriques peuvent être configurés via 2 attributs qui influe sur le calcul du facteur de charge global d'un nœud (valeur comprise entre 0 et 100) :

- **weight** (défaut 1) : Le poids de la métrique par rapport aux autres
- **capacity** (requis ou optionnel en fonction des métrique): Sert à normaliser la valeur de charge du métrique ainsi qu'à favoriser certains nœuds.

La capacité doit être choisie de telle sorte que :
facteur de charge / capacity < 1



Calcul de charge

La contribution d'un métrique au calcul de charge global d'un nœud est alors calculé comme suit :

$$(charge / capacity) * weight / total weight$$

Il est également possible de lisser les valeurs fournies en prenant en compte dans le calcul l'historique des valeurs fournies

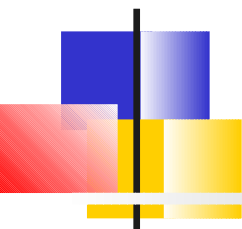


Apache

Du côté Apache, les librairies doivent être installées

Pour Jboss 7.1 *mod_cluster 1.2*, il faut télécharger soit la version d 'Apache pré-pakagée ou seulement les modules et les charger dans *httpd.conf*

http://www.jboss.org/mod_cluster/Downloads/1-1-3.



Modules

Chacun des modules couvre un aspect de la répartition de charge :

- ***mod_proxy*** et ***mod_proxy_ajp*** sont les modules cœur propageant les requêtes via *http/https* ou *ajp*
- ***mod_manager*** est un module qui lit des information de Jboss 7 et met à jour la mémoire partagée en coordination avec ***mod_slotmem***.
- ***mod_proxy_cluster*** est le module qui contient le répartiteur pour *mod_proxy*.
- ***mod_advertise*** est un module additionnel qui permet à Apache de signaler les changements de statuts applicatif via des paquets multicast packets



Httpd.conf

```
LoadModule proxy_module modules/mod_proxy.so
LoadModule proxy_ajp_module modules/mod_proxy_ajp.so
LoadModule slotmem_module modules/mod_slotmem.so
LoadModule manager_module modules/mod_manager.so
LoadModule proxy_cluster_module modules/mod_proxy_cluster.so
LoadModule advertise_module modules/mod_advertise.so
```



Httpd.conf

```
Listen 192.168.10.1:8888
<VirtualHost 192.168.10.1:8888>
<Location />
Order deny,allow
Deny from all
Allow from 192.168.10.
</Location>
KeepAliveTimeout 60
MaxKeepAliveRequests 0
ServerAdvertise On
</VirtualHost>
```



Gestion via CLI

CLI permet d'administrer et de récupérer des information du cluster lors de son exécution

La commande ***list-proxies*** retourne les hôtes (et ports) des proxies connectés

```
[standalone@localhost:9999 subsystem=modcluster] :list-proxies
{
  "outcome" => "success",
  "result" => [
    "CP11-010:8888",
    "CP12-010:8888"
  ]
}
```

La commande ***read-proxies-info*** permet d'avoir des informations détaillées



Example

```
[standalone@localhost:9999 subsystem=modcluster] :read-proxies-info
{
  "outcome" => "success",
  "result" => [
    "CP11-010:8888",
    "Node: [1],Name: de6973fe-b63d-31dc-a806-04ec16870cfa,Balancer:mycluster,LBGroup: ,Host:
      192.168.10.1,Port: 8080,Type:http,Flushpackets: Off,Flushwait:
10,Ping: 10,Smax: 65,Ttl: 60,Elected: 0,Read: 0,Transferred: 0,Connected:0,Load: 1
Vhost: [1:1:2], Alias: localhost
Vhost: [1:1:3], Alias: example.com
Context: [1:1:1], Context: /, Status: ENABLED",
    "CP12-010:8888",
    "Node: [1],Name: re5673ge-c83d-25dv-y104-02rt16456cfa,Balancer:mycluster,LBGroup: ,Host:
      192.168.10.2,Port: 8080,Type:http,Flushpackets: Off,Flushwait:
10,Ping: 10,Smax: 65,Ttl: 60,Elected: 0,Read: 0,Transferred: 0,Connected:0,Load: 1
Vhost: [1:1:2], Alias: localhost
Vhost: [1:1:3], Alias: example.com
Context: [1:1:1], Context: /, Status: ENABLED"
  ]
}
```




Autres commandes CLI

add-proxy permet d'ajouter un proxy non capturé par la configuration httpd.

```
[standalone@localhost:9999 subsystem=modcluster]
: add-proxy(host= CP15-022, port=9999)

{"outcome" => "success"}
```

remove-proxy permet d'enlever un proxy :

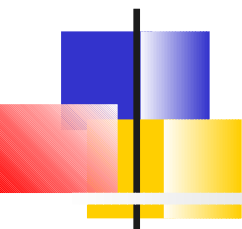
```
[standalone@localhost:9999 subsystem=modcluster]
: remove-proxy(host=CP15-022, port=9999)

{"outcome" => "success"}
```



TP : Load balancing

- ❖ Mettre en place une architecture de load balancing avec Apache avec le sticky session



Infinispan



Introduction

Depuis JBoss AS 6, **Infinispan** est utilisé comme cache distribué dans les configuration cluster

L'API d'Infinispan remplace l'API *JBoss Cache* et peut être utilisé en standalone ou embarqué dans le serveur JBoss

Infinispan permet la réplication de cache pour :

- Les SFSBs
- Le cache Hibernate (entités)
- Les sessions HTTP



Stratégies de cache

Dans Jboss AS 7, le sous-système Infinispan peut configurer plusieurs éléments ***cache-container***

Un *cache-container* contient une ou plusieurs stratégies de cache qui détermine comment les données sont synchronisées.

Les stratégies disponibles sont :

- **Local**: Les données sont stockées seulement sur le nœud local (cache local)
- **Réplication**: Les données sont répliquées sur tous les nœuds
- **Distribution**: Les données sont distribuées sur un sous-ensemble de nœuds
- **Invalidation**: Les données sont stockées seulement dans un *cache store* (Hibernate). Lorsqu'un nœud nécessite une entrée, il la charge à partir du cache. Le cache peut alors être invalidée sur tous les nœuds

Les cache containers sont automatiquement inscrit dans l'annuaire JNDI sous le nom : `java:jboss/infinispan/container-name`



Example

```
<subsystem xmlns="urn:jboss:domain:infinispan:1.1" default-cache-  
container="cluster">  
  <cache-container name="web" aliases="standard-session-cache"  
default-cache="repl">  
    <replicated-cache name="repl" mode="ASYNC" batching="true">  
      <file-store/>  
    </replicated-cache>  
    <replicated-cache name="sso" mode="SYNC" batching="true"/>  
    <distributed-cache name="dist" l1-lifespan="0" mode="ASYNC"  
    batching="true">  
      <file-store/>  
    </distributed-cache>  
  </cache-container>  
  ...  
</subsystem>
```



Mode de synchronisation

La synchronisation de données entre les nœuds peut être effectuée par des messages **synchrones** (SYNC) ou **asynchrones** (ASYNC).

- Le mode synchrone est le moins performant puisque chaque nœud doit recevoir un acquittement de chaque membre du cluster. Cependant, le mode synchrone est nécessaire lorsque tous les nœuds doivent accéder aux données cachées de façon cohérente.
- Le mode asynchrone privilégie la rapidité à la cohérence, ce mode est particulièrement avantageux pour la réplication de session HTTP en mode sticky

Le mode de synchronisation n'est effectif que pour les caches distribués sur plusieurs nœuds (modes :réplication, distribution et invalidation)



Concurrence et isolation

- ❖ Le cache est également complètement thread-safe.
- ❖ Lors de la configuration par défaut, il est possible de fixer le degré de concurrence de la même façon que l'on fixe les niveaux d'isolation d'une base de données :

REPEATABLE_READ est le niveau d'isolation par défaut d'Infinispan. Avec ce niveau d'isolation, la transaction positionne des verrous en lecture sur toutes les données récupérées. (des lectures fantômes peuvent potentiellement arriveres) .

L'autre niveau supporté est **READ_COMMITTED** qui fournit un gain de performance mais les données récupérées ne sont pas protégées des modifications concurrentes



file-store

L'élément ***file-store*** configure l'emplacement de stockage des données cachées.

Par défaut :

```
jboss.server.data.dir/<cache-container>
```

```
<file-store relative-to="..." path="..." />
```

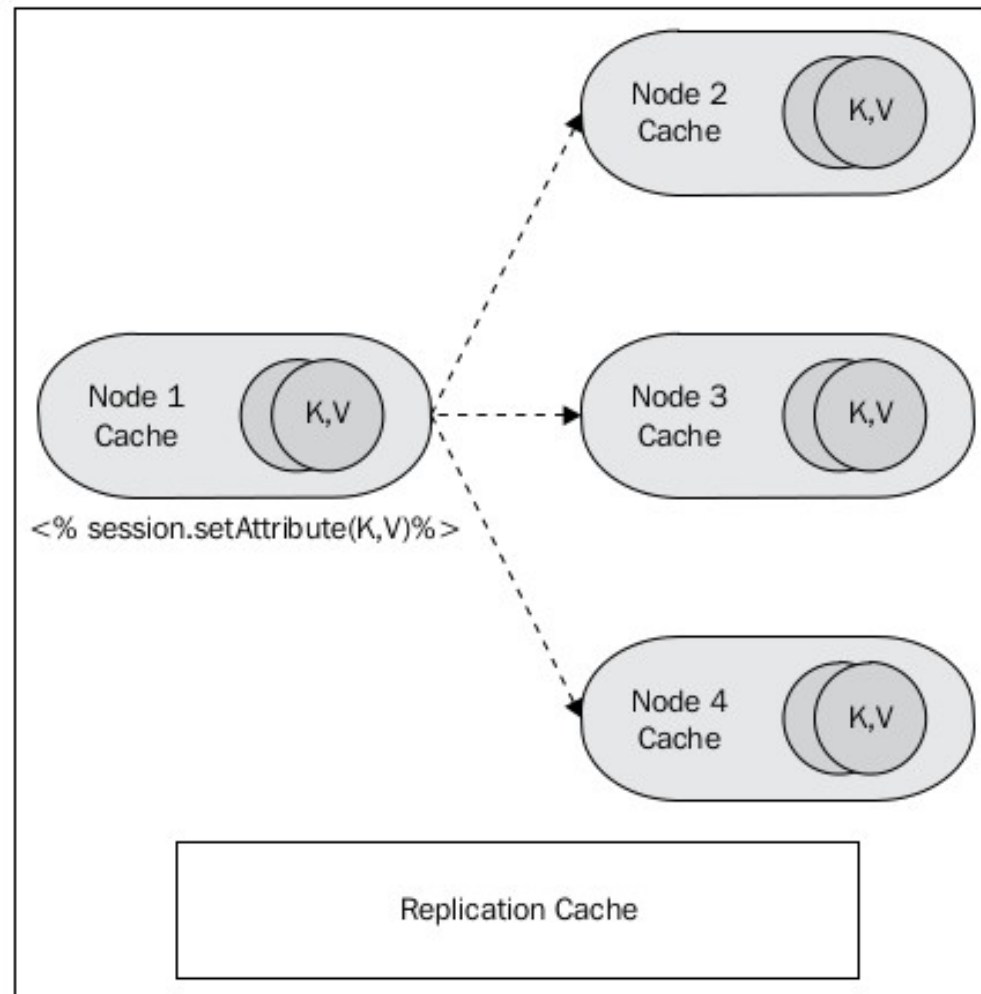


Réplication

Lors de l'utilisation de la réplication, *Infinispan* stocke chaque entrée du cache sur chaque nœud

- La scalabilité de la réplication est alors dépendante de la taille du cluster
- Si $\text{DATA_SIZE} * \text{NUMBER_OF_HOSTS}$ est inférieure à la taille de la mémoire disponible sur chaque hôte, la réplication peut être un choix valable.

Réplication





Distribution

Dans le cas de la distribution, Infinispan stocke chaque entrée du cache sur un sous-ensemble de nœuds permettant une meilleure scalabilité

La distribution utilise un algorithme de hashage pour déterminer où les données doivent être distribuées

L'algorithme est configuré avec le nombre de copies devant être gérées sur le cluster

Le nombre de copies représente le compromis entre performance et la durabilité des données

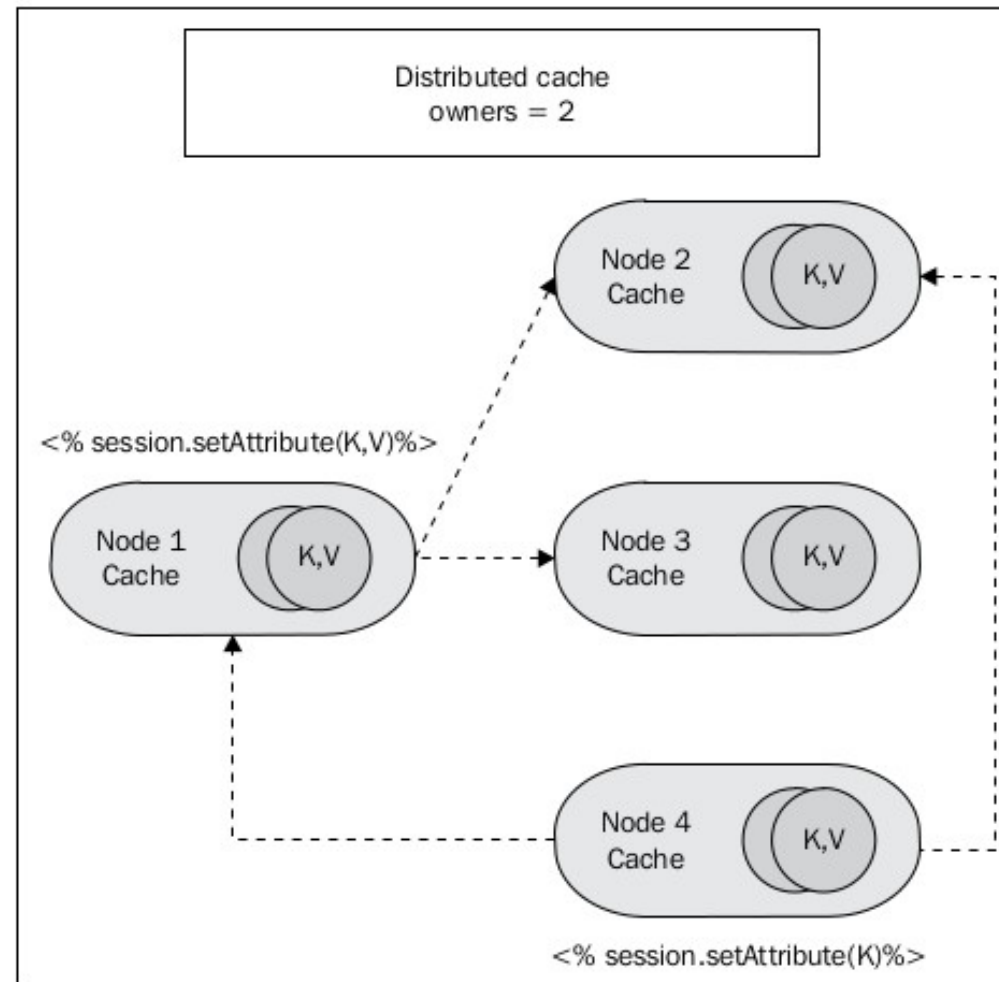
Le paramètre **owners** (par défaut 2) définit le nombre de copies

```
<distributed-cache owners="3" mode="ASYNC" name="dist"
batching="true">
```

```
. . . . .
```

```
</distributed-cache>
```

Distribution





Threads

Il est possible d'externaliser la configuration des threads d'Infinispan dans un pool de threads

Des pools différents peuvent être configurés pour les différents cache-container

Les pools disponibles sont :

- **transport** : Taille du pool de thread limité dédié au transport de données sur le réseau
- **listener-executor** : Taille du pool de threads utilisé pour enregistré et être notifié des événements concernant le cache
- **replication-queue-executor** : Taille du pool de threads planifiées pour la réplication des données du cache
- **eviction-executor** : Taille du pool de threads planifiées pour périodiquement nettoyer le cache



Gestion des threads

```
<subsystem xmlns="urn:jboss:domain:clustering:infinispan:1.0">
  <cache-container name="cluster" listener-executor="infinispan-listener"
    eviction-executor="infinispan-eviction" replication-queue-executor="infinispan-repl-queue">
    <transport executor="infinispan-transport"/>
    <!-- Caches -->
  </cache-container>
</subsystem>

<subsystem xmlns="urn:jboss:domain:threads:1.0">
  <bounded-queue-thread-pool name="infinispan-listener" blocking="true">
    <max-threads count="1" per-cpu="2"/>
    <queue-length count="100000" per-cpu="200000"/>
  </bounded-queue-thread-pool>
  <bounded-queue-thread-pool name="infinispan-transport" blocking="true">
  <!-- ... --></bounded-queue-thread-pool>
  <scheduled-thread-pool name="infinispan-eviction">
  <!-- ... --></scheduled-thread-pool>
  <scheduled-thread-pool name="infinispan-repl-queue">
  <!-- ... --></scheduled-thread-pool>
</subsystem>
```



Transport

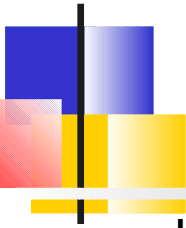
Par défaut, les caches d'Infinispan utilise la pile par défaut de JGroups : *default-stack*

Il est cependant possible de choisir une couche transport différente pour chaque *cache-container* :

```
<cache-container name="web" default-cache="repl">  
<transport stack="tcp"/>  
</cache-container>
```

La configuration par défaut UDP est en général adaptée au grands clusters ou lorsque l'on utilise la réplication ou l'invalidation comme il ouvre beaucoup moins de sockets

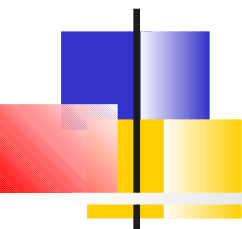
La pile TCP est plus adapté aux petits clusters ou à la distribution



Utilisation directe d'Infinispan

Il est possible de se faire injecter le cache Infinispan dans ses classes Java

```
@ManagedBean
public class MyBean<K, V> {
    @Resource(lookup = "java:jboss/infinispan/mycontainer")
    private org.infinispan.manager.CacheContainer container;
    private org.infinispan.Cache<K, V> cache;
    @PostConstruct
    public void start() {
        this.cache = this.container.getCache();
    }
    // Use cache
}
```



Réplication de session HTTP



Introduction

Par défaut, la réplication de session utilise le cache nommé **web**

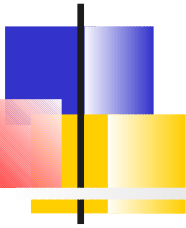
Il est possible de dédier un cache à une application via *jboss-web.xml*

```
<jboss-web>
  <replication-config>
    <cache-name>web.dist</cache-name>
  </replication-config>
</jboss-web>
```



Cache *web*

```
<cache-container name="web" default-cache="repl">
  <alias>standard-session-cache</alias>
  <replicated-cache mode="ASYNC" name="repl" batching="true">
    <locking isolation="REPEATABLE_READ"/>
    <file-store/>
  </replicated-cache>
  <distributed-cache mode="ASYNC" name="dist" batching="true">
    <locking isolation="REPEATABLE_READ"/>
    <file-store/>
  </distributed-cache>
</cache-container>
```



Réplication de sessions HTTP

- ❖ Les descripteurs de déploiement de l'application web doivent être configurés :
 - *web.xml* doit comporter l'élément **`<distributed/>`** (après l'élément `<description>`)
 - *jboss-web.xml*

```
<jboss-web>
```

```
  <replication-config>
```

```
    <cache-name>web.dist</cache-name>
```

```
  </replication-config>
```

```
</jboss-web>
```



Interprétation du nom

Détermination du cache à partir du nom spécifié dans le descripteur de déploiement

1. Interprété comme étant le nom du cache

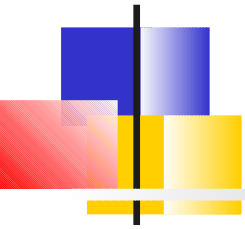
exemple : `jboss.infinispan.web.dist`

2. Interprété comme nom du container, le cache par défaut est alors utilisé

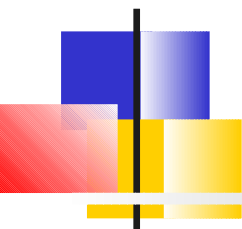
exemple : `jboss.infinispan.web`

3. Assume que le nom de base est *jboss.infinispan*

exemple : `web`, `web.dist`

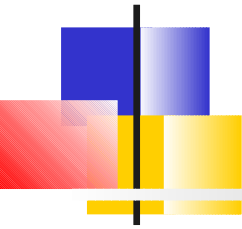


TP : Réplication de session



Clustering d'EJBs

EJBs Session
EJBs Entité



EJBs Session

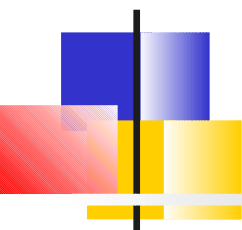


EJB Session Stateless

- ❖ Pas de synchronisation d'états entre les différents EJB.
- ❖ EJB 2.1 - Descripteur de déploiement spécifique *jboss.xml* :

```
<jboss>
<enterprise-beans>
<session><ejb-name>nextgen.StatelessSession</ejb-name>
  <jndi-name>nextgen.StatelessSession</jndi-name>
  <clustered>true</clustered>
  <cluster-config>
    <partition-name>${jboss.partition.name}</partition-name>
    <home-load-balance-policy>org.jboss.ha.framework.interfaces.RoundRobin</home-load-balance-policy>
    <bean-load-balance-policy>
      org.jboss.ha.framework.interfaces.RoundRobin
    </bean-load-balance-policy>
  </cluster-config>
</session>
</enterprise-beans>
</jboss>
```

- ❖ EJB 3.0 : Le descripteur de déploiement est remplacé par l'annotation **@Clustered** avec comme attributs la politique de load balancing et le nom de la partition



Algorithme de répartition

Plusieurs choix sont possibles pour l'algorithme de répartition :

- **Round robin** : par défaut
- **RandomRobin** : Au hasard
- **FirstAvailable** : Au hasard, puis sticky pour un proxy donné
- **FirstAvailableIdenticalAllProxies** : Au hasard, puis sticky pour tous les proxies du même EJB



Exemple EJB 3.0

```
@Stateless
@Clustered(loadBalancePolicy=RoundRobin.class)
public class CounterBean implements Counter {

    /**
     * Default constructor.
     */
    public CounterBean() {
    }

    public void printCount(int messageNumber) {
        System.out.println(messageNumber) ;
    }
}
```



EJB Session stateful

- ❖ Dans le cas des stateful, JBoss réplique l'état du bean entre les différents nœuds.
- ❖ Le cache Infinispan utilisé est nommé **sfsb** qui réplique les sessions sur tous les nœuds
- ❖ Si un défaillance survient, le proxy EJB le détecte et choisit un autre nœud ou les données session ont été répliquées
- ❖ En version 3.0 les annotations sont utilisées



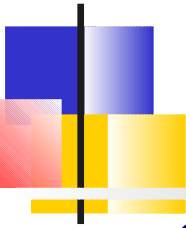
sfsb

```
<cache-container name="sfsb" default-cache="repl">
  <alias>sfsb-cache</alias>
  <alias>jboss.cache:service=EJB3SFSBClusteredCache</alias>
  <replicated-cache mode="ASYNC" name="repl" batching="true">
    <locking isolation="REPEATABLE_READ"/>
    <eviction strategy="LRU" max-entries="10000"/>
    <file-store/>
  </replicated-cache>
</cache-container>
```



EJB Session stateful 3.0

- ❖ Le bean est annotée par **@Clustered** et peut également configurer le cache par **@Cache**
 - **name** : Si on ne veut pas utiliser sfsb
 - **maxSize** : Taille du cache avant passivation
 - **idleTimeoutSeconds** : Tps idle avant passivation
 - **removalTimeoutSeconds** : Tps avant suppression
 - **replicationIsPassivation** : Pour appeler les méthodes de callback **@PrePassivate** du bean lors d'une réplication



Exemple EJB3 session stateful

```
@Stateful
```

```
@Clustered
```

```
@CacheConfig(maxSize=5000, removalTimeoutSeconds=18000)
```

```
public class MyBean implements MySessionInt {
```

```
    private int state = 0;
```

```
    public void increment() {
```

```
        System.out.println("counter: " + (state++));
```

```
    }
```

```
}
```



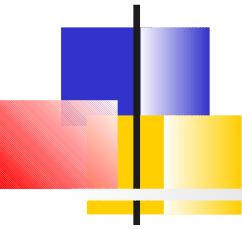

TP

❖ Phase 1

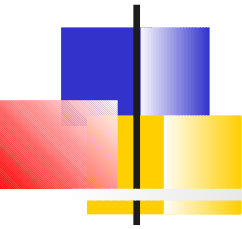
- Déploiement d'un EJB stateless « clusterisé »
- Construction du code client
- Observer la répartition de charge

❖ Phase 2

- Déploiement d'un EJB stateful « clusterisé »
- Construction du code client
- Observer le mode sticky et la tolérance aux pannes

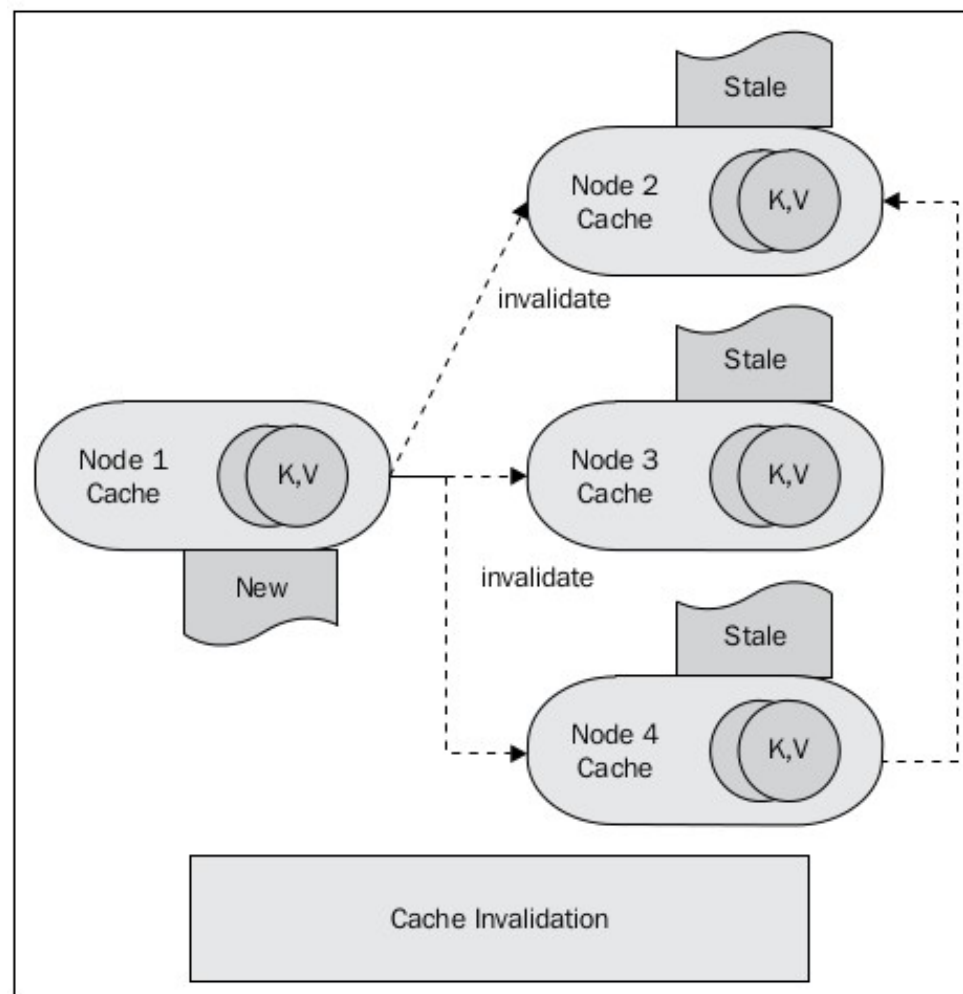
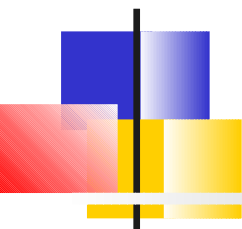


EJBs Entités



Introduction

- ❖ Les entités n'étant pas accessibles à distance ne sont pas concernées par la répartition de charge
- ❖ Le cache Hibernate de second-niveau (SessionFactory) est alors utilisé pour minimiser les accès à la base de données
- ❖ Le cache stocke des entités ou des collections d'entités provenant de requêtes Hibernate
- ❖ Infinispan est alors utilisé pour invalider les caches de second niveau sur les nœuds du cluster





Configuration

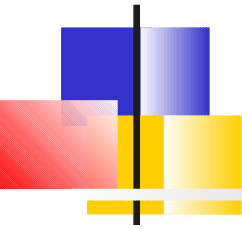
Le profil ha définit donc le cache container hibernate qui définit 3 caches :

- Le cache **local** : le cache de second niveau hibernate utilisé par chaque noeud
- Le cache **d'invalidation** permettant l'invalidation du cache local
- Le cache **timestamp** qui stocke un timestamp de la dernière modification pour chaque table.
Ce



Configuration

```
<cache-container name="hibernate" default-cache="local-query">
  <local-cache name="local-query">
    <eviction strategy="LRU" max-entries="10000"/>
    <expiration max-idle="100000"/>
  </local-cache>
  <invalidation-cache mode="SYNC" name="entity">
    <eviction strategy="LRU" max-entries="10000"/>
    <expiration max-idle="100000"/>
  </invalidation-cache>
  <replicated-cache name="timestamps" mode="ASYNC">
    <transaction mode="NONE"/>
    <eviction strategy="NONE"/>
  </replicated-cache>
</cache-container>
```



Configuration des EJBs

- ❖ Les EJBs doivent également être configurés pour pouvoir être cachés
- ❖ La configuration consiste
 - A indiquer à Hibernate l'utilisation du cache via le fichier *persistence.xml*
 - Annoter les EJBs que l'on veut cacher
 - Éventuellement, préciser les stratégies d'éviction pour certains EJBs
 - A indiquer des dépendances sur les modules *org.hibernate* et *org.infinispan*



persistence.xml

```
<!-- Autoriser le cache -->
```

```
<property name="hibernate.cache.use_second_level_cache" value="true"/>
```

```
<property name="hibernate.cache.use_query_cache" value="true"/>
```

```
<!-- Surcharger le container par défaut -->
```

```
<property name="hibernate.cache.infinispan.cachemanager"  
value="java:jboss/infinispan/mycontainer"/>
```

```
<!-- Surcharger les régions de cache -->
```

```
<property name="hibernate.cache.infinispan.entity.cfg" value="entity"/>
```

```
<property name="hibernate.cache.infinispan.collection.cfg" value="entity"/>
```

```
<property name="hibernate.cache.infinispan.query.cfg" value="local-query"/>
```

```
<property name="hibernate.cache.infinispan.timestamp.cfg" value="timestamp"/>
```




persistence.xml / JPA2.0

- ❖ L'élément **<shared-cache-mode>** (JPA 2.0) doit être renseigné avec un des valeurs suivantes :
 - *ALL* : Toutes les entités et les données associées sont cachées
 - *NONE* : Le cache est désactivé pour cette unité de persistance.
 - *ENABLE_SELECTIVE* : Le cache est autorisé si l'annotation *@Cacheable* est spécifiée sur la classe entité.
 - *DISABLE_SELECTIVE* : Le cache est activé sauf pour les entités annotées avec *@Cacheable(false)*
- ❖ La propriété ***hibernate.cache.use_minimal_puts*** permet d'optimiser le nombre d'écriture dans le cache
- ❖ La propriété ***hibernate.cache.use_query_cache*** permet d'activer le cache de requêtes hibernate



persistence.xml

```
<shared-cache-mode>ENABLE_SELECTIVE</shared-cache-mode>

<properties>
  <property name=
    "hibernate.cache.use_second_level_cache" value="true"/>
  <property name="hibernate.cache.use_minimal_puts"
    value="true"/>
  <property name="hibernate.cache.use_query_cache"
    value="true"/>
</properties>
```



persistence.xml

```
<shared-cache-mode>ENABLE_SELECTIVE</shared-cache-mode>

<properties>
  <property name=
    "hibernate.cache.use_second_level_cache" value="true"/>
  <property name="hibernate.cache.use_minimal_puts"
    value="true"/>
  <property name="hibernate.cache.use_query_cache"
    value="true"/>
</properties>
```



Configuration pour Infinispan en mode cluster

Ces attributs permettent de donner l'implémentation du gestionnaire de cache via JNDI :

```
<property name="hibernate.cache.region.factory_class"
  value="org.jboss.as.jpa.hibernate4.infinispan.InfinispanRegionFactory"/>
<property name="hibernate.cache.infinispan.cachemanager"
  value="java:jboss/infinispan/container/hibernate"/>
<property name="hibernate.transaction.manager_lookup_class"
  value="org.hibernate.transaction.JBossTransactionManagerLookup"/>
<property name="hibernate.cache.use_second_level_cache" value="true"/>
```



Annotations

Une fois l'unité de persistance configurée, il peut être nécessaire d'indiquer par des annotations les entités qui seront effectivement cachées.

- Soit directement sur les **entités**
- Soit sur des ***NamedQuery***



Annotations entités

- ❖ Règle classiquement mise en œuvre :
cacher les EJBs qui changent peu et
sont principalement accédés en lecture

```
@Entity
```

```
@Cacheable
```

```
@Cache (usage=CacheConcurrencyStrategy.TRANSACTIONAL,  
        region = "Customer"
```

```
)
```

```
public class Customer implements Serializable {
```

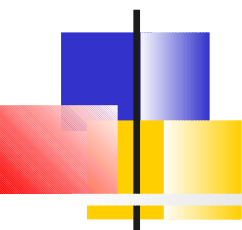
```
// ... ..
```

```
}
```

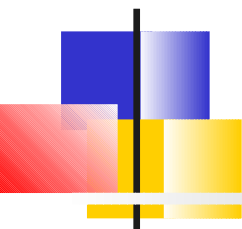


Annotations @NamedQuery

```
@Entity
@Cache(usage = CacheConcurrencyStrategy.TRANSACTIONAL,
      region = "Account")
@NamedQueries(
{
    @NamedQuery(
name = "account.bybranch",
query = "select acct from Account as acct where acct.branch = ?1",
hints = { @QueryHint(name = "org.hibernate.cacheable", value =
      "true") }
    )
})
public class Account implements Serializable
{
    // ... ..
}
```



TP



Clustering HornetQ



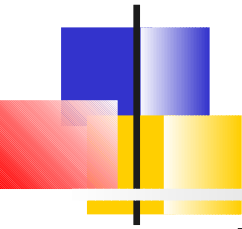
Introduction

Le clustering avec **HornetQ** permet de définir des groupes de serveurs HornetQ afin de partager la charge de traitement des messages.

Chaque nœud actif du cluster est un serveur HornetQ gérant ses propres messages et ses propres connexions

Les serveurs doivent être configurés en cluster en positionnant l'élément **clustered** dans la configuration et en définissant des connexions « clusterisées »

Ce type de connexion permet de répartir la charge entre les nœuds



Capacités de clustering

L'architecture de clustering peut suivre différentes topologies qui offre différentes capacités :

- Répartition du traitement des messages
- Redistribution des messages
- Répartition des connexions clientes
- Découverte automatique des nœuds
- Haute disponibilité et fail-over

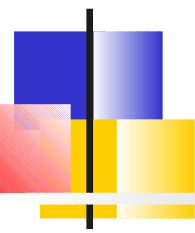


Découverte des nœuds

La découverte automatique est un mécanisme basé sur UDP qui permet de propager les détails de connexions au

- **Clients** : Un client peut alors se connecter aux serveurs du cluster sans connaître précisément les serveurs actifs
- **Aux autres serveurs** : Les serveurs peuvent créer leurs connexions cluster sans connaître à priori sa constitution

La topologie du cluster est envoyée au client par des connexions normale et aux serveurs via les connexions clusterisées. La première connexion est alors établie soit en utilisant UDP soit en fournissant la liste des connecteurs initiaux



Répartition de charge serveur

Si des connexions clusterisées sont définies entre les nœuds, *HornetQ* répartit la charge des messages arrivant sur un nœud spécifique.

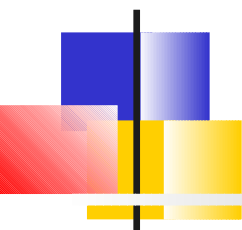
- Par exemple, si le cluster est formé des nœuds A et B et qu'un client envoie des messages sur le nœud A. Les messages seront réparties (Round-robin par défaut) sur A et B



Répartition des connexions clientes

Avec cette fonctionnalité, les différentes session créées avec la même usine à session peuvent être connectées à différents nœuds du cluster.

La stratégie de répartition est configurable (Round-robin, Random ou custom)



Redistribution de messages

La redistribution de messages permet de redistribuer les messages d'une file pour lesquels il n'y a pas de consommateur disponible sur le nœud vers des nœuds qui ont de tels consommateurs.

La redistribution peut être configurée pour se déclencher dès que le dernier consommateur d'une file est fermé ou après un certain délai

Par défaut, la redistribution n'est pas activée



Topologies de cluster

Il y a 2 topologies principales de cluster :

- Cluster **symétrique** : Chaque nœud est connecté à tous les autres (JBoss). Chaque nœud connaît toutes les files existantes sur les autres nœuds ainsi que leurs consommateurs. Avec cette connaissance, la charge peut être répartie
- Cluster en **chaîne** : Les nœuds forment une chaîne et chaque nœud connaît le nœud précédent et le nœud suivant



HA et *failover*

HornetQ permet :

- la haute-disponibilité : Le système continue à fonctionner même si un ou plusieurs serveurs tombent en panne
- Le failover : en cas de défaillance d'un serveur, le client peut continuer à fonctionner en migrant sur un autre serveur



Serveur de backup

HornetQ permet de lier des serveurs actifs à des serveurs de backup

- Chaque serveur actif peut avoir un ou plusieurs backup
- Un serveur de backup est détenu par un seul serveur actif

Les serveurs de backup ne sont pas opérationnels tant que le serveur actif fonctionne. (Redondance passive)

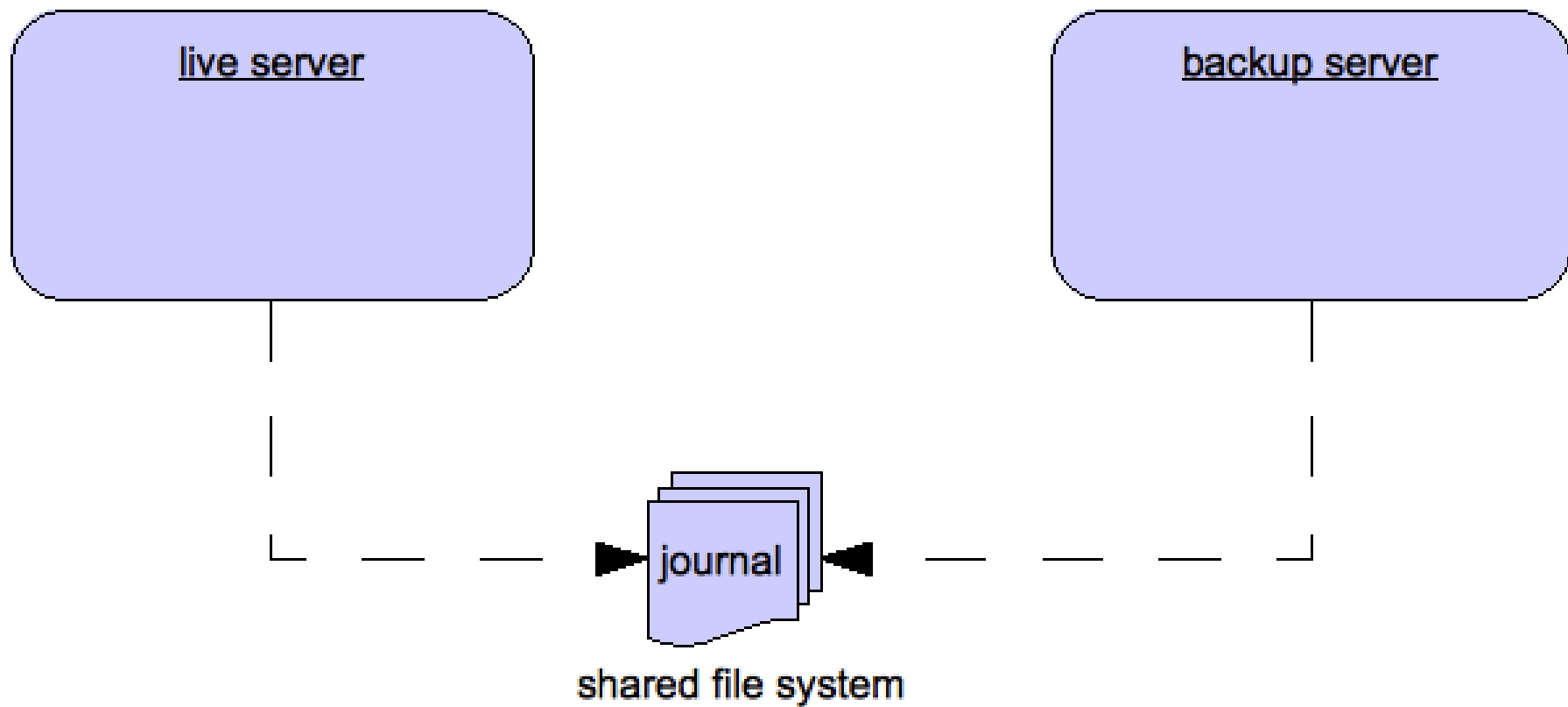
Lorsque le serveur actif défaille, le serveur passif devient actif et un nouveau serveur de backup devient passif

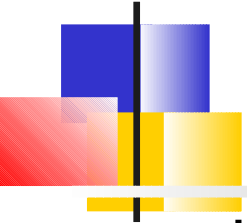
Si le serveur actif redémarre il redevient le serveur primaire

Ce mode de backup est implémenté via un support de persistance partagé



Actif/Passif





Fail-over

HornetQ propose deux modes de failover :

- Failover **automatique** : Les clients sont configurés afin d'avoir connaissance des serveurs actifs et des serveurs de backup. Il n'est alors pas nécessaire de coder la logique de reconnexion lors d'une défaillance
- Failover **applicatif** : La logique de reconnexion est codée dans les clients

HornetQ fournit également le ré-attachement automatique et transparent au même serveur en cas de problèmes temporaire réseau



Notification d'erreur via JMS

JMS fournit un mécanisme standard afin d'être notifié de façon asynchrone d'une perte de connexion :

java.jms.ExceptionListener

Un *ExceptionListener* est appelé par HornetQ lors d'une perte de connexion même si le mécanisme de *failover* a été effectif. Il faut alors inspecter le code d'erreur de l'exception JMS qui peut prendre 2 valeurs : FAILOVER ou DISCONNECT

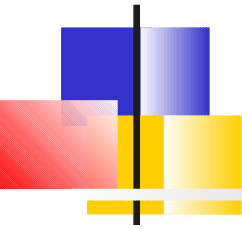


Configuration JBoss

La configuration JBoss s'effectue dans le sous-système ***messaging***.

L'élément ***clustered*** doit être positionné à *true*

```
<subsystem xmlns="urn:jboss:domain:messaging:1.1">
  <hornetq-server>
    <clustered>true</clustered>
    . . . . .
  </hornetq-server>
</subsystem>
```



Configuration multi-cast

Chaque nœud utilise UDP :

- pour diffuser ses informations sur ses connecteurs : *broadcast-group*
- pour découvrir les informations des connecteurs des autres serveurs : *discovery-group*



Connexions cluster

Les connexions cluster doivent également être configurées à l'intérieur de l'élément `<hornetq-server>` :

```
<cluster-connections>
<cluster-connection name="mycluster">
  <address>jms</address>
  <connector-ref>netty</connector-ref>
  <retry-interval>500</retry-interval>
  <use-duplicate-detection>true</use-duplicate-detection>
  <forward-when-no-consumers>false</forward-when-no-consumers>
  <max-hops>1</max-hops>
</cluster-connection>
</cluster-connections>
```




Configuration

L'attribut ***name*** définit la connexion cluster, ils peut avoir plusieurs connexions cluster dans le même sous-système de *messaging*

L'élément ***address*** est obligatoire et détermine comment les messages sont distribués sur le cluster. Dans l'exemple précédent, tous les messages envoyés à une adresse commençant par *jms* seront répartis

L'élément ***connector-ref*** référence le connecteur définit dans la section *connectors* du sous-système de *messaging*

L'élément ***retry-interval*** détermine l'intervalle en ms entre les tentative de relivraison au même nœud

L'élément ***use-duplicate-detection*** lorsqu'il est activé détecte les messages dupliqués qui seront alors ignorés

L'élément ***forward-when-no-consumers*** permet la redistribution de message

L'option ***max-hops*** (Par défaut 1) permet de déterminer les nœuds accessible pour la répartition. 1 signifie donc que seuls les nœuds en connexions directes peuvent être choisis



Algorithme de répartition

L'algorithme de répartition peut être spécifié sous l'élément *connection-factory*.

Les algorithmes fournis par HornetQ sont :

- Round-Robin (*RoundRobinConnectionLoadBalancingPolicy*)
- Random (*RandomConnectionLoadBalancingPolicy*).

Un algorithme customisé peut être mis en place en implémentant l'interface

org.hornetq.api.core.client.loadbalance.ConnectionLoadBalancingPolicy

```
<connection-factory name="InVmConnectionFactory">
. . . .
<connection-load-balancing-policy-class-name>
org.hornetq.api.core.client.loadbalance.RandomConnectionLoadBalancingPolicy
</connection-load-balancing-policy-class-name>
</connection-factory>
```



Sécurité

Lorsque les connexions sont créées entre les nœuds, *HornetQ* utilise un login/mot de passe.

Il est impératif de changer les valeurs par défaut sinon des clients distants seront capable de faire des connexions au serveur

```
<cluster-user>user</cluster-user>
```

```
<cluster-password>password</cluster-password>
```

Un message de trace rappelle à l'administrateur que les mots de passe doivent être changés

```
09:29:07,573 WARNING [org.hornetq.core.server.impl.HornetQServerImpl]
(MSC service thread 1-1) Security risk! It has been detected that the cluster admin
user and password have not been changed from the installation default. Please see
the HornetQ user guide, cluster chapter, for instructions on how to do this.
```



Configuration pour HA

La mise en place d'un serveur de backup nécessite la mise en place d'un journal partagé

La configuration est identique du côté du serveur actif et du backup



Exemple

```
<hornetq-server>
  <!-- true for backup, false for live -->
  <backup>true</backup>
  <!-- Persistence activée sur le disque -->
  <persistence-enabled>true</persistence-enabled>
  <!-- Le journal est partagé -->
  <shared-store>true</shared-store>
  <!-- Répertoire accessible par les 2 serveurs (live et backup) -->
  <journal-directory path="path/to/journal" relative-to="user.home"/>
  <bindings-directory path="path/to/bindings" relative-to="user.home"/>
  <large-messages-directory path="path/to/large-message" relative-to="user.home"/>
  <paging-directory path="path/to/paging" relative-to="user.home"/>
  <journal-file-size>102400</journal-file-size><!-- you may tune this -->
  <journal-min-files>2</journal-min-files><!-- you may tune this -->
  <!-- Failover lors d'un shutdown normal -->
  <failover-on-shutdown>true</failover-on-shutdown>
  <!-- ... -->
</hornetq-server>
```



Failover du client

Afin que le client puisse basculer sur le serveur actif en cas de défaillance, il doit obtenir une référence d'une usine à connexion *ha*.

Cette usine à connexion HA est configurée de telle sorte qu'elle référence le groupe de découverte

Le client doit également être accessible via le multicast UDP correspondant



Configuration Usine HA

```
<jms-connection-factories>
  <connection-factory name="RemoteConnectionFactory">
    <discovery-group-ref discovery-group-name="dg-group1"/>
    <entries>
      <entry
name="java:jboss/exported/jms/RemoteConnectionFactory"/>
    </entries>
    <ha>true</ha>
    <reconnect-attempts>-1</reconnect-attempts>
  </connection-factory>
</jms-connection-factories>
```



MDBs et HornetQ

- ❖ Les MDBs beans consommant les messages JMS peuvent être déployés en mode cluster. Ils sont alors présents sur l'ensemble des nœuds
- ❖ Lors de l'envoi d'un message par un client JMS, le message peut être traité par un des pools de MDB du cluster.



TP

❖ Jboss Messaging

- Queue distribuée
- Failover
- MDBs



Ressources

- ❖ <http://www.jboss.com> : Marketing et services
- ❖ <http://labs.jboss.com> : Téléchargement
- ❖ <http://wiki.jboss.org> : Contributions utilisateurs JBoss

- ❖ Documents de référence disponibles sur le site de JBoss
 - Clustering
- ❖ Livres :
 - “JBoss in action” – Manning publications



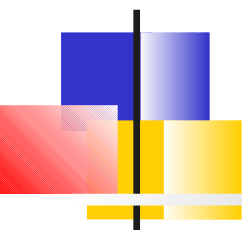
Merci!!!

❖ MERCI DE VOTRE ATTENTION



Annexes

Protocoles JGroups



Protocoles de découverte

- ❖ Utilisés pour découvrir les nœuds actifs
- ❖ Il dépend en général du protocole de transport :
 - PING et UDP Conf JBoss AS
 - TCPPING et TCP Conf JBoss AS : (la liste des nœuds possibles est alors statique et doit être fournie) :
 - MPING et TCP



Configuration UDP

- ❖ C'est la configuration par défaut. Plusieurs attributs peuvent être définis :
 - **mcast_addr** et **mcast_port** sont les adresses et le port de multicast. Ils peuvent être mappés sur des variables JBoss fournit au démarrage (`jgroups.udp.mcast_addr`)
 - **ip_ttl** indique le *time-to-live*, c'est le nombre maximal de saut réseau qu'un message peut faire. Chaque router décrémente cette valeur. Cependant, de nombreux routeurs ne propage pas les messages multicast. Mais lors d'un environnement WAN, cet attribut peut être augmenté.
 - **ip_mcast** est utile lorsque l'on désire faire de l'unicast sur UDP
- ❖ IANA (Internet Assigned Numbers Authority) contrôle les adresses IP de multicast et autorise des adresses comprises entre 224.0.0.0 et 239.255.255.255. Pour un LAN, les adresses à utiliser sont comprises entre 224.0.0.0 jusqu'à 224.0.0.255.
- ❖ L'adresse configurée par défaut est en dehors de cette plage (228.11.11.11). Elle doit donc être modifiée pour être sûr que les routeurs ne propagent pas les messages à l'extérieur du LAN.



Configuration TCP

- ❖ Le protocole TCP ne supportant pas le multicast, la découverte automatique n'est pas possible directement.
 - Si possible, utiliser MPING (configuration par défaut)
 - Sinon fournir les hôtes initiaux dans la configuration de TCPPING.

- ❖ Exemple :

```
<TCPPING timeout="3000" down_thread="false" up_thread="false"  
initial_hosts="localhost[7600],localhost[7601]"  
port_range="1"  
num_initial_members="3"/>
```

- ❖ La liste des hôtes initiaux peut également être fournie par la variable JBoss *jgroups.tcpping.initial_hosts*.



Protocoles de détection de panne

❖ 6 protocoles de détection de panne

- **FD** Conf JBoss AS : Le plus verbeux, le plus réactif mais également le plus suspicieux
- **FD SOCK** Conf JBoss AS : Détection passive => le moins réactif
- **FD_SIMPLE** : Niveau de tolérance adaptable
- **FD_PING** : Exécution d'un script ping
- **FD_ICMP** : *InetAddress.isReachable()*, Java5
- **FD_ALL** : Multicast régulier



Protocoles de détection

❖ **FD**

- Chaque nœud envoie périodiquement un message de type « Es-tu vivant ? » à son voisin
- Si pas de réponse, il envoie un message de type SUSPECT à l'ensemble du cluster
- Le coordinateur courant du groupe vérifie que le nœud est effectivement en panne avant de mettre à jour la vue du cluster.

❖ **FD SOCK**

- Pas de message supplémentaires,
- La panne n'est détectée que lorsqu'une connexion normale ne peut pas aboutir.



Protocoles sûrs

- ❖ Les protocoles sûrs permettent de garantir que l'ordre de réception des paquets de données est identique à l'ordre d'émission.
- ❖ 3 protocoles sont proposés :
 - **UNICAST** Conf JBoss AS basé sur des acquittements positifs (ACK), i.e l'émetteur renvoie le message tant qu'il n'a pas reçu d'acquittement
`<UNICAST timeout="100,200,400,800"/>`
 - **NAKACK** Conf JBoss AS basé sur des acquittements négatifs (NAK), le récepteur demande une ré-émission d'un message lorsqu'il s'aperçoit d'un gap entre 2 messages
`<pbcast.NAKACK
max_xmit_size="8192"
use_mcast_xmit="true"
retransmit_timeout="600,1200,2400,4800"/>`
 - **SMACK** : Non documenté



Gestion de groupe

- ❖ Le protocole de gestion de groupes **pbcast.GMS** Conf JBoss AS gère l'ajout de nouveaux membres, les demandes de retrait du groupes et les messages de suspicion envoyés par le protocole de détection de panne.
- ❖ Il est basé sur la notion de coordinateur
- ❖ L'algorithme pour l'acceptation d'un nouveau membre est le suivant :
 - Découverte des membres initiaux
 - Si pas de réponse => singleton et exit
 - Sinon détermine le coordinateur à partir des réponses (membre le + ancien)
 - Envoie d'une requête JOIN vers le coordinateur
 - Si réponse au JOIN, récupère la vue du cluster et exit
 - Sinon sleep de 5s et retour à 1



Configuration

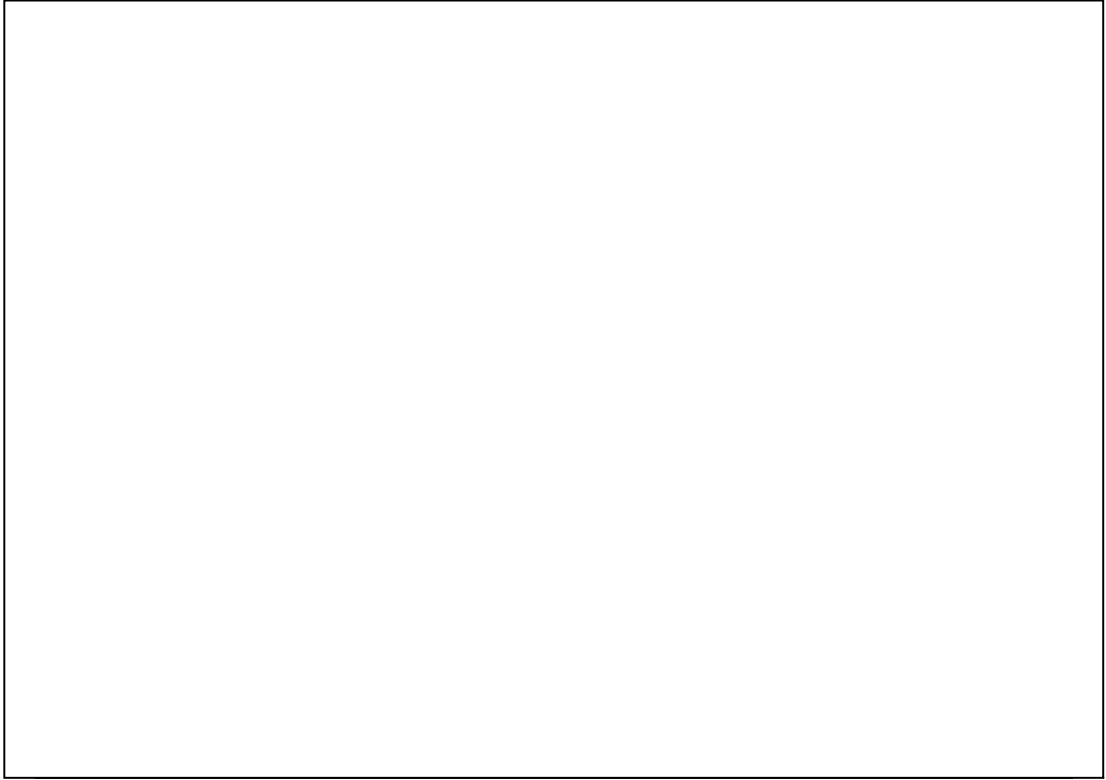
❖ Quelques attributs de *pbcast.GMS*

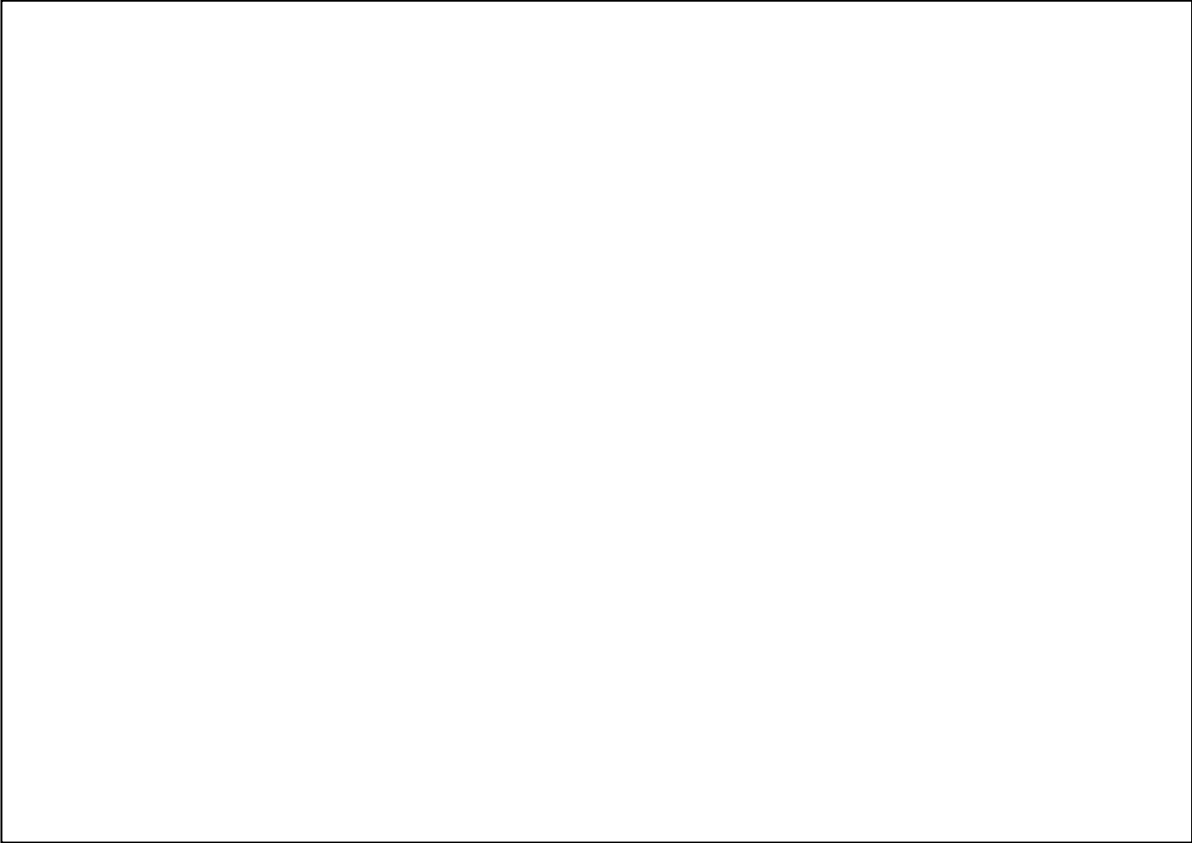
- ***join_timeout*** : Délai d'attente de la requête *JOIN* (valeur par défaut 5s)
- ***join_retry_timeout*** : Délai d'attente avant de réessayer
- ***leave_timeout*** : Délai d'attente de la requête *LEAVE*
- ***print_local_addr*** : Affichage sur la console de l'adresse du nouveau membre. Valeur par défaut true

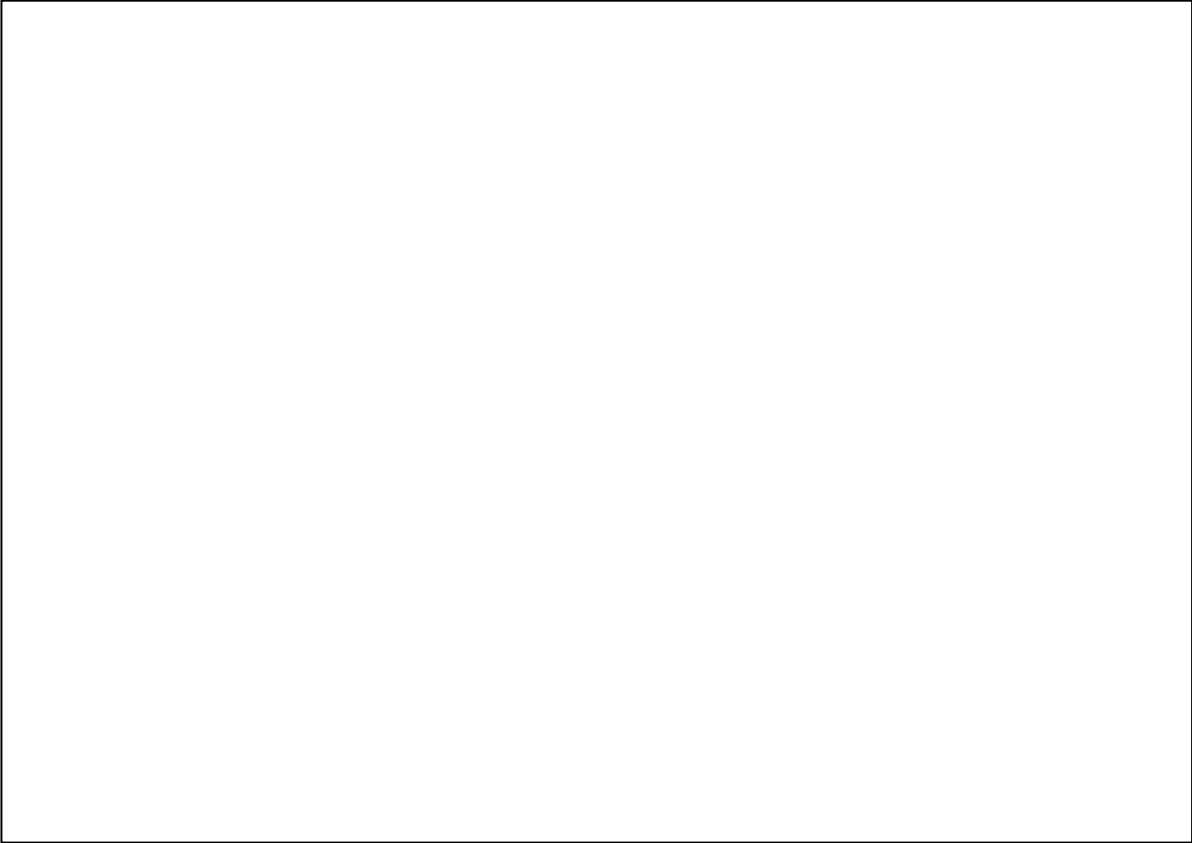


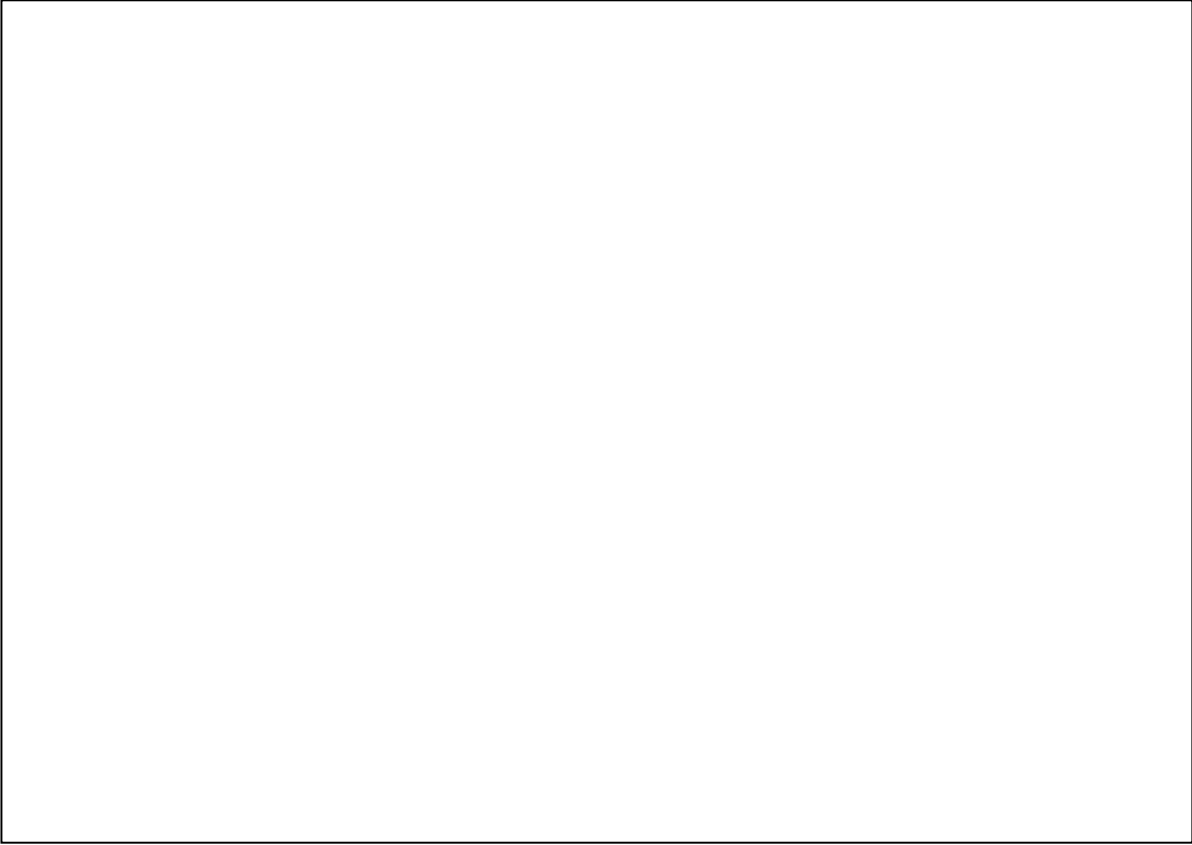
Transfert d'état

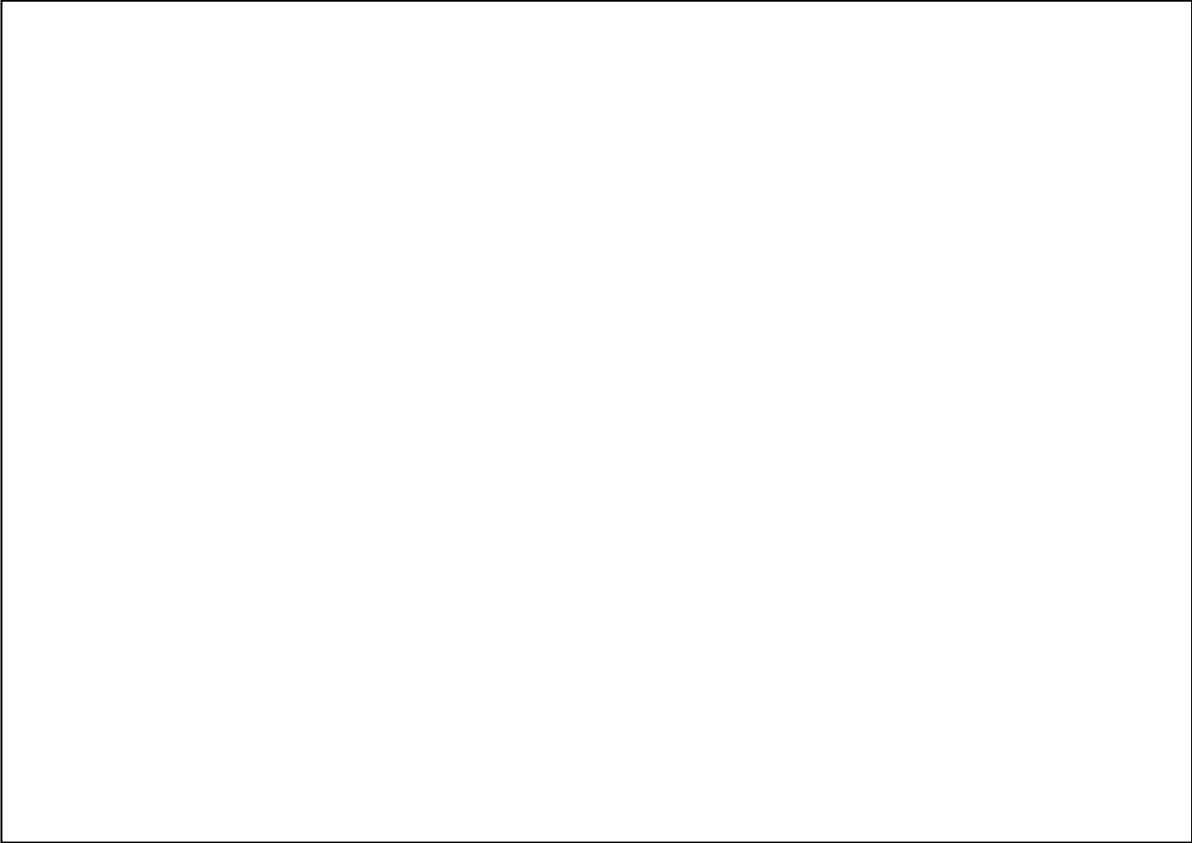
- ❖ L'état de l'application est transféré au nouveau membre d'un cluster.
- ❖ Ce transfert s'effectue soit
 - Globalement : *pbcast.STATE_TRANSFER* Conf
JBoss AS
 - Soit par morceau :
pbcast.STREAMING_STATE_TRANSFER

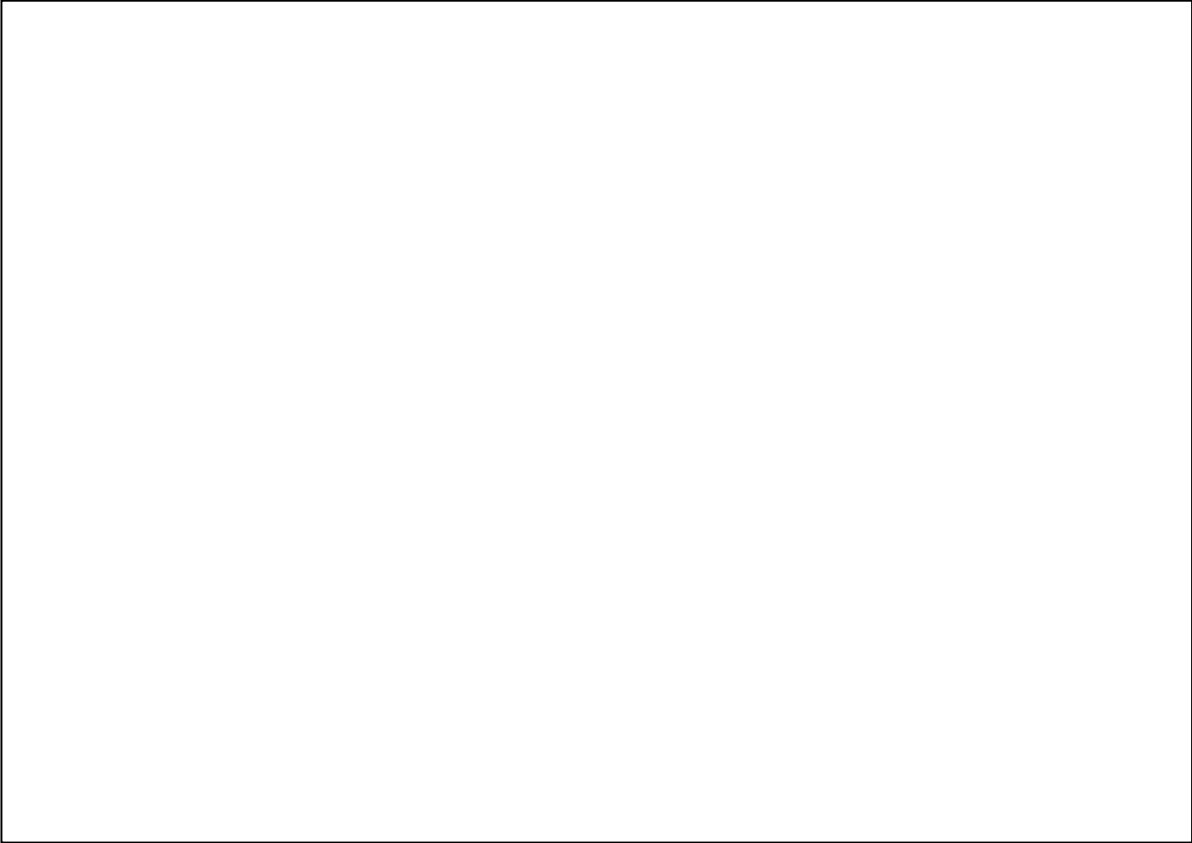


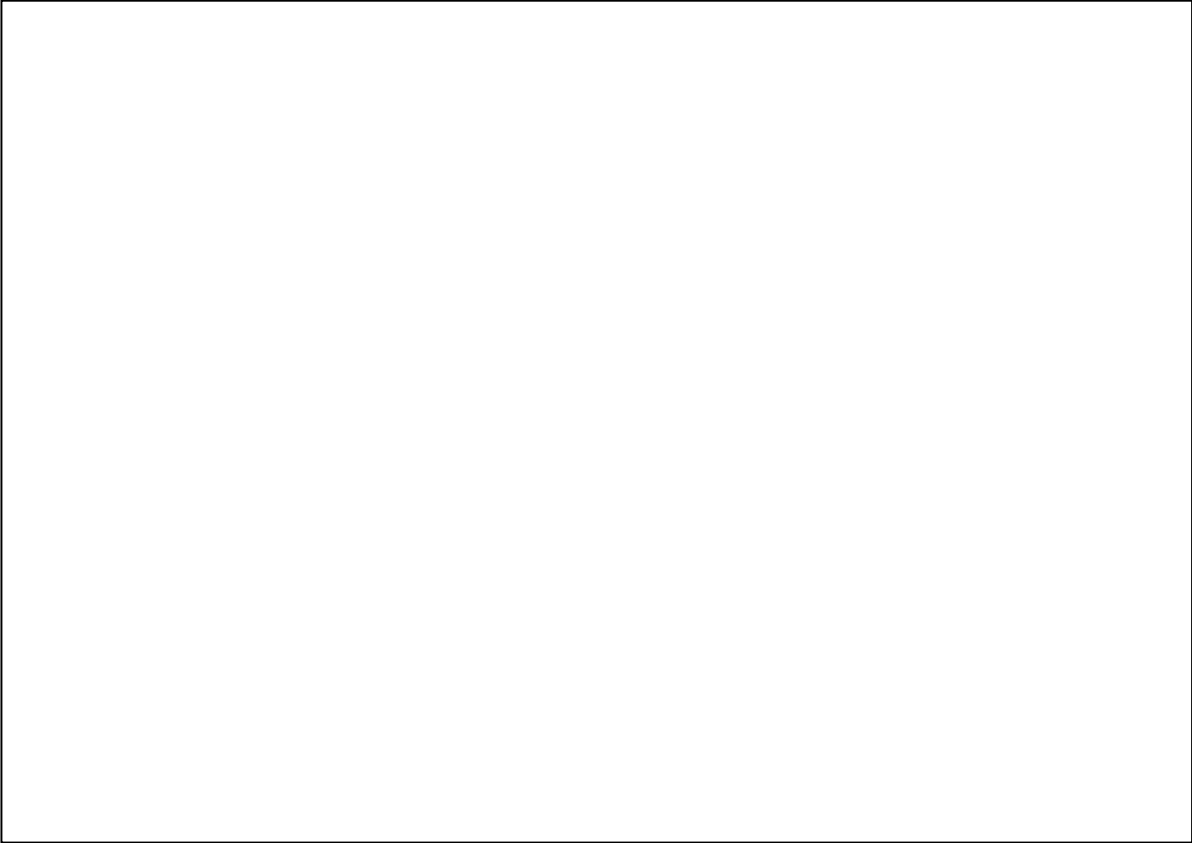


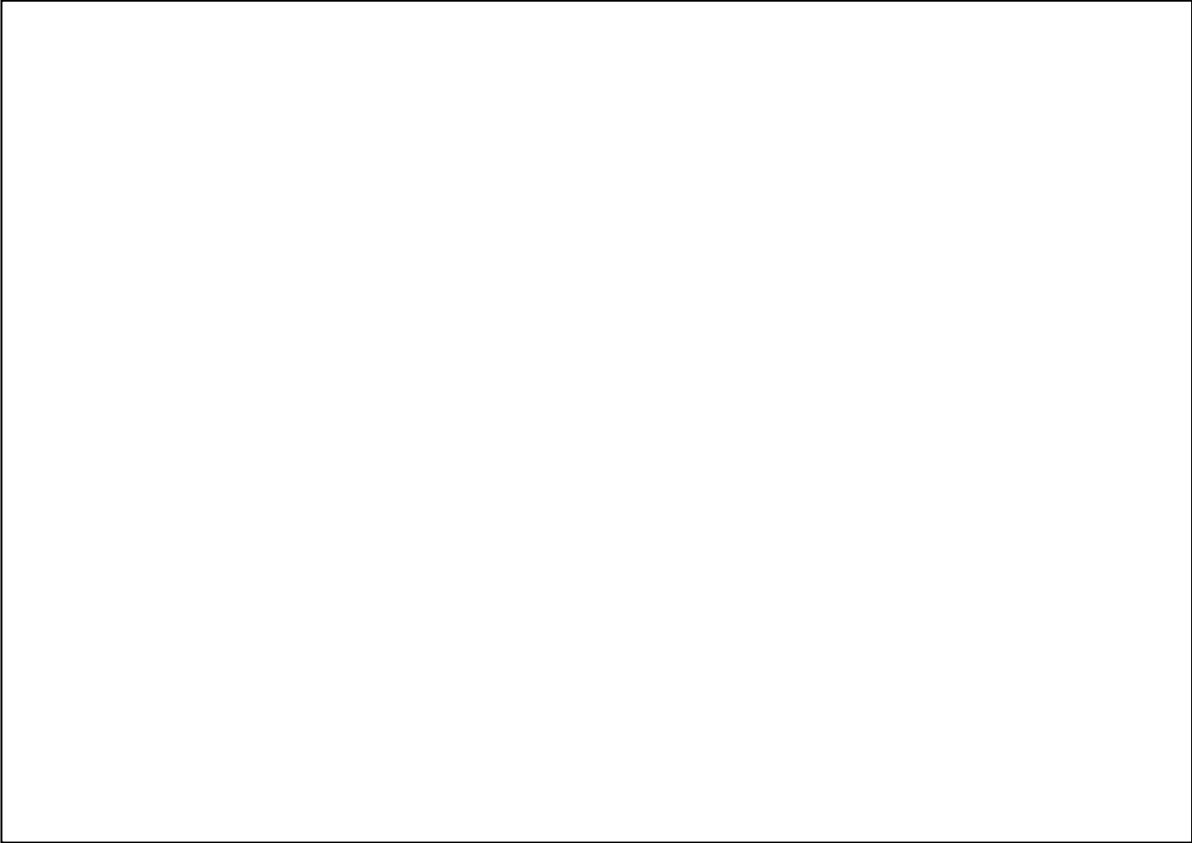


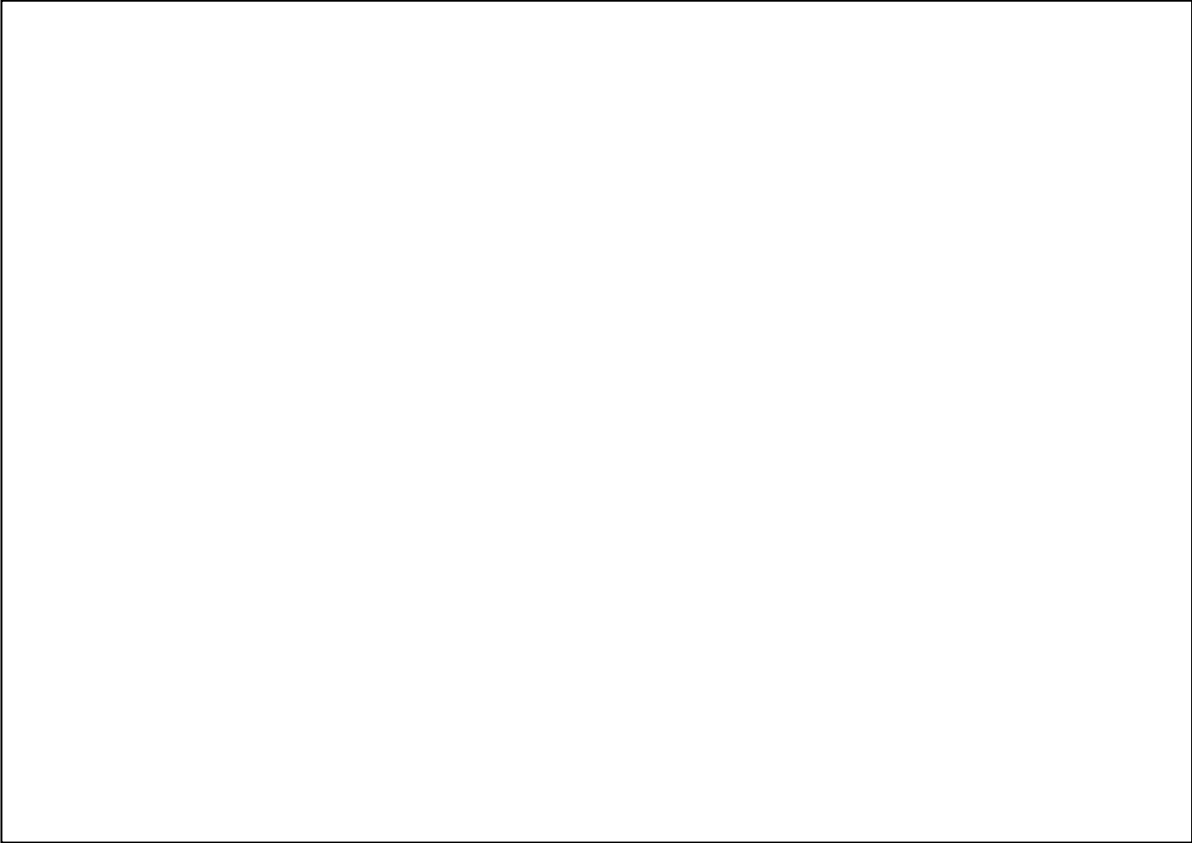


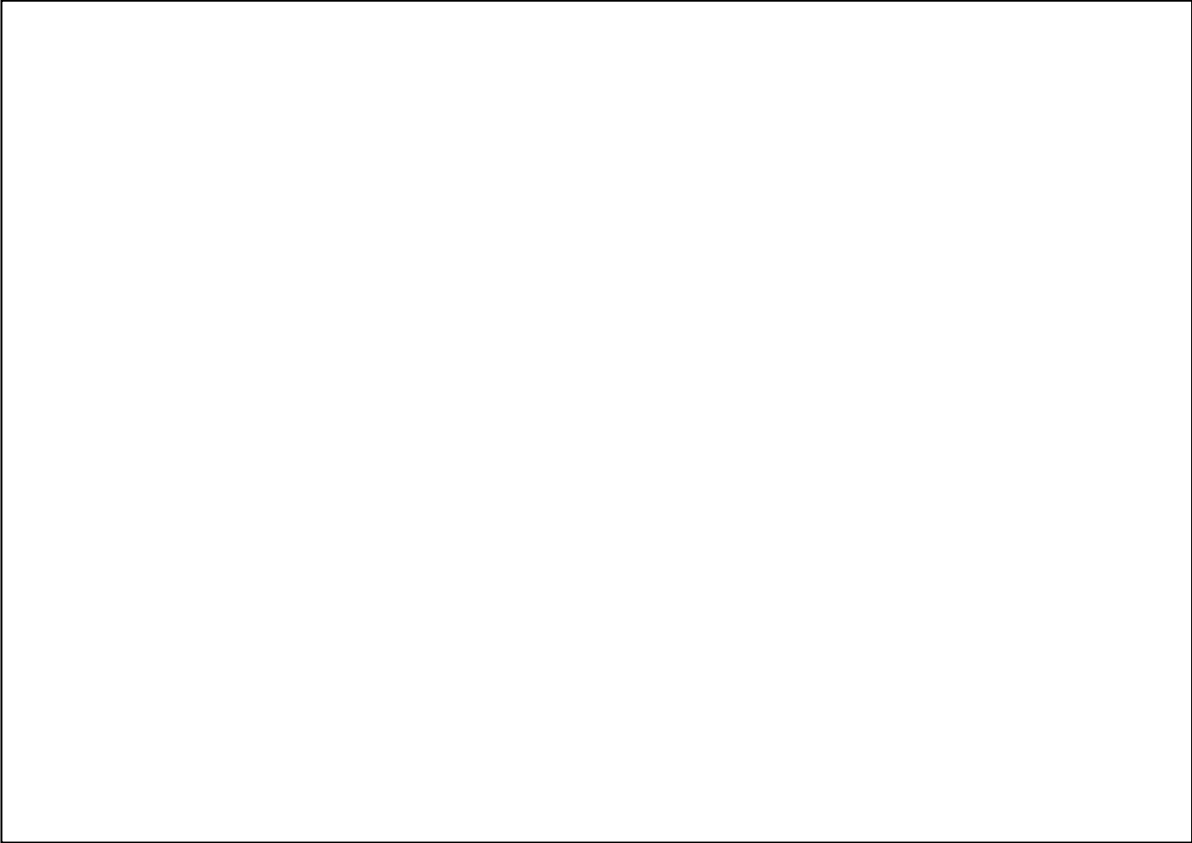


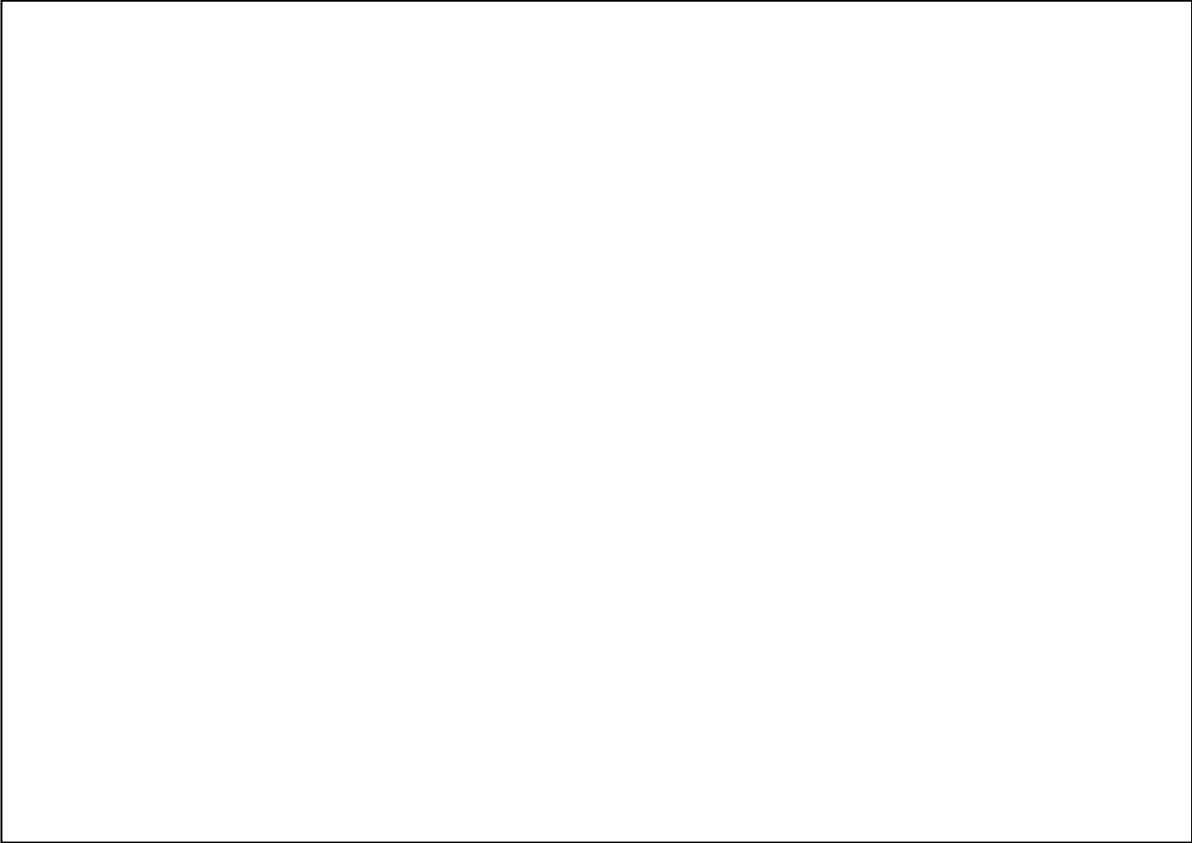


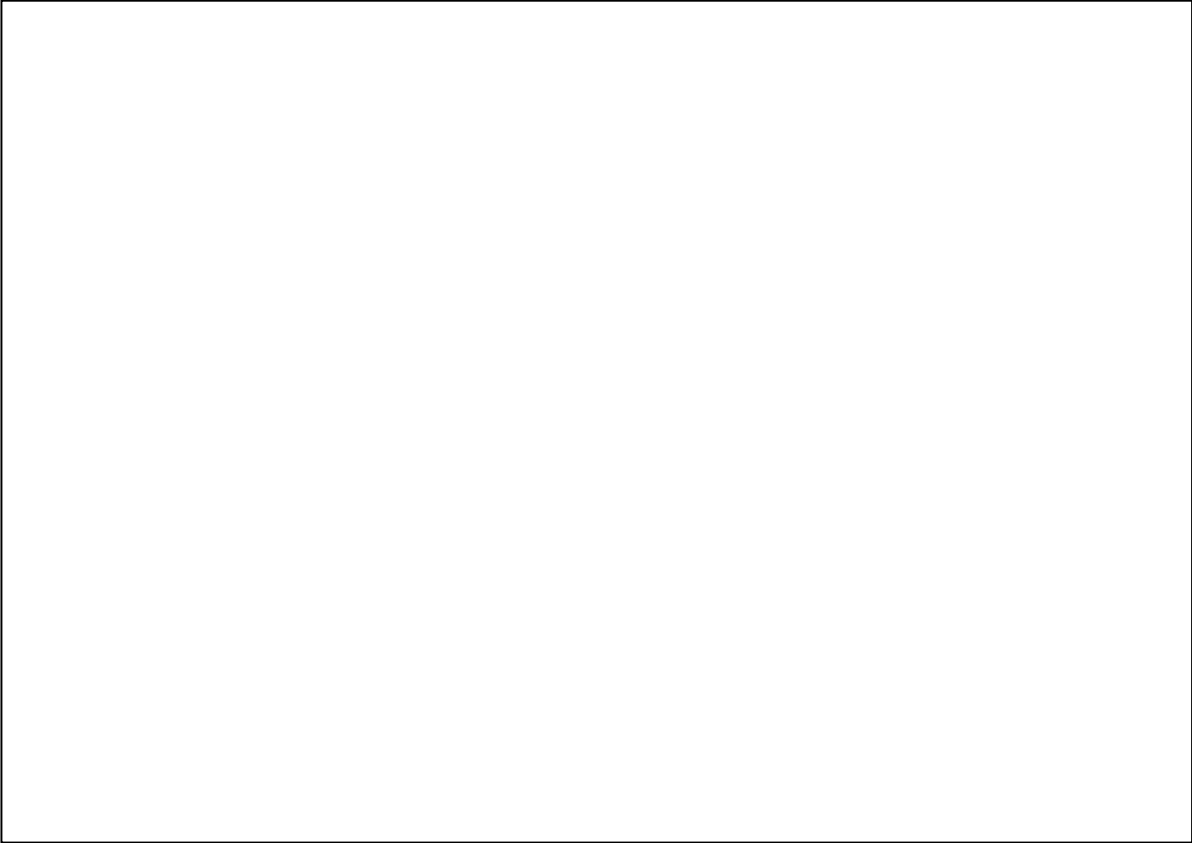


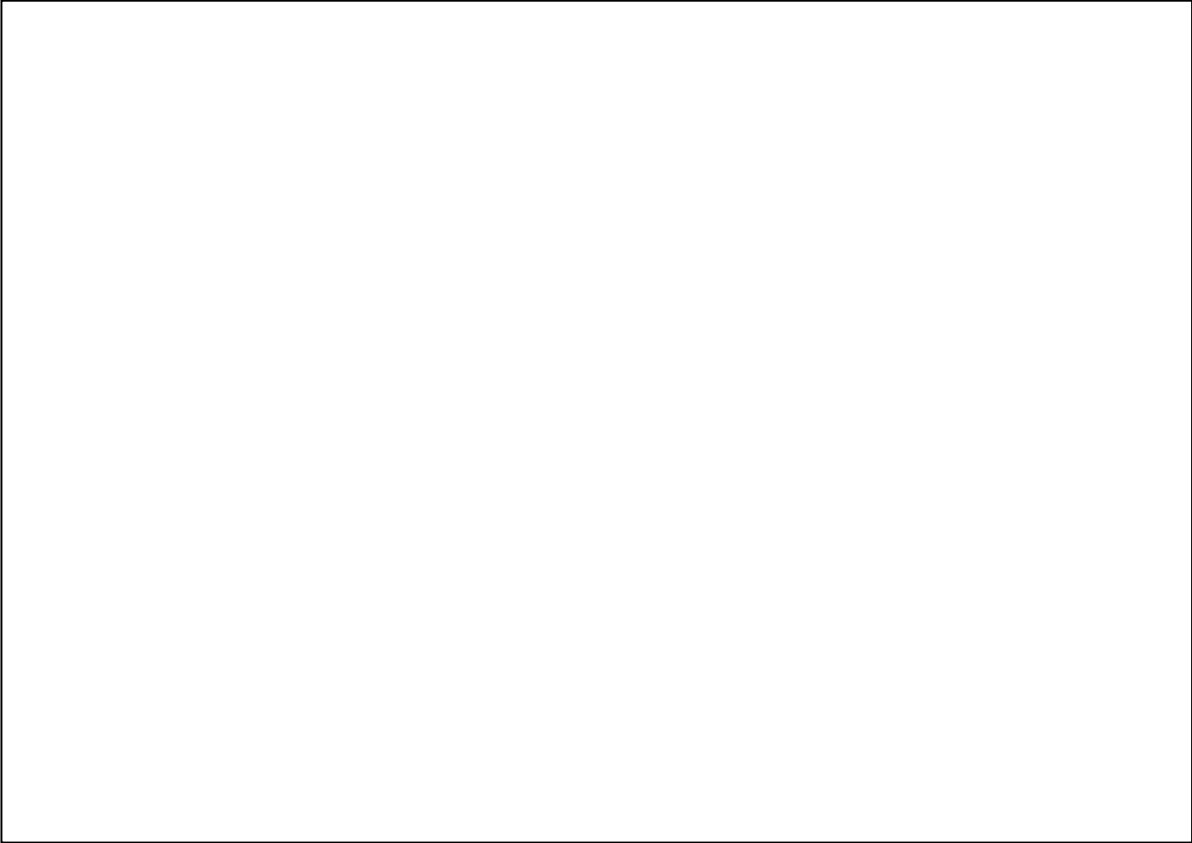


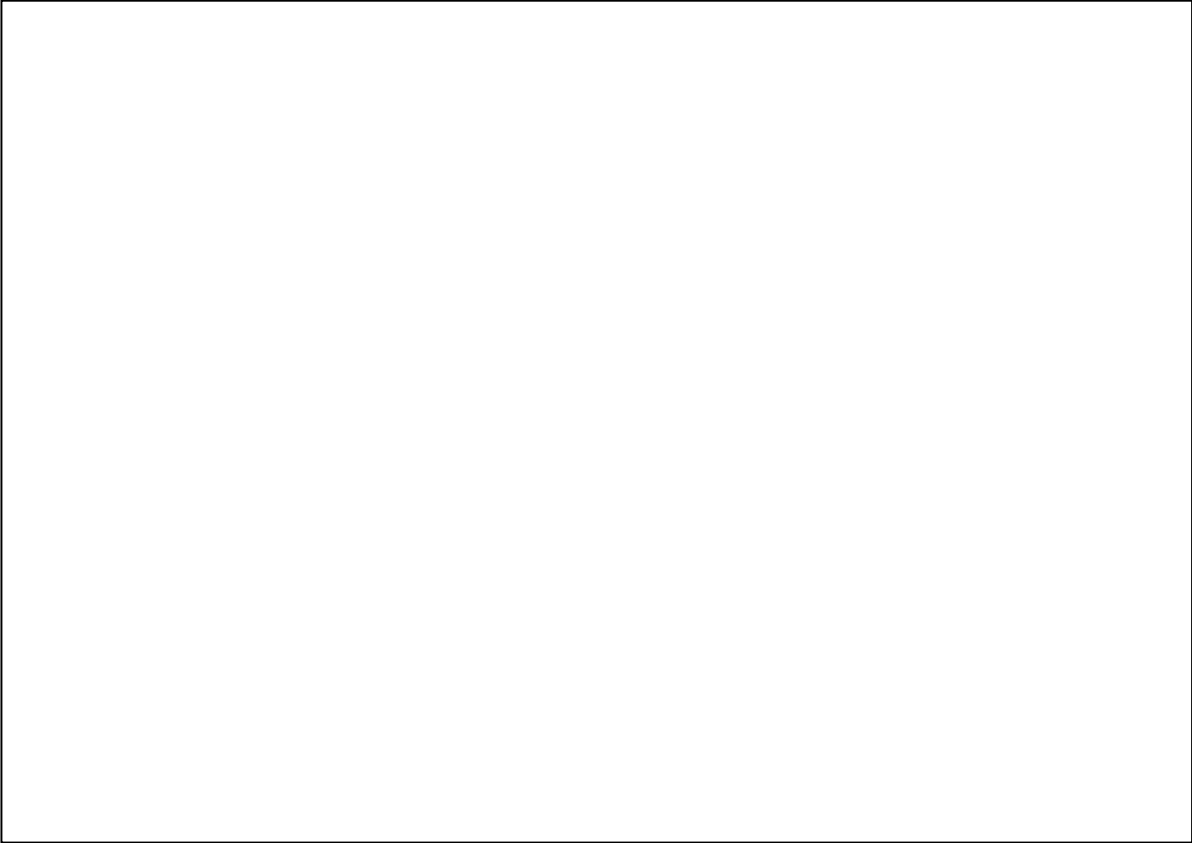


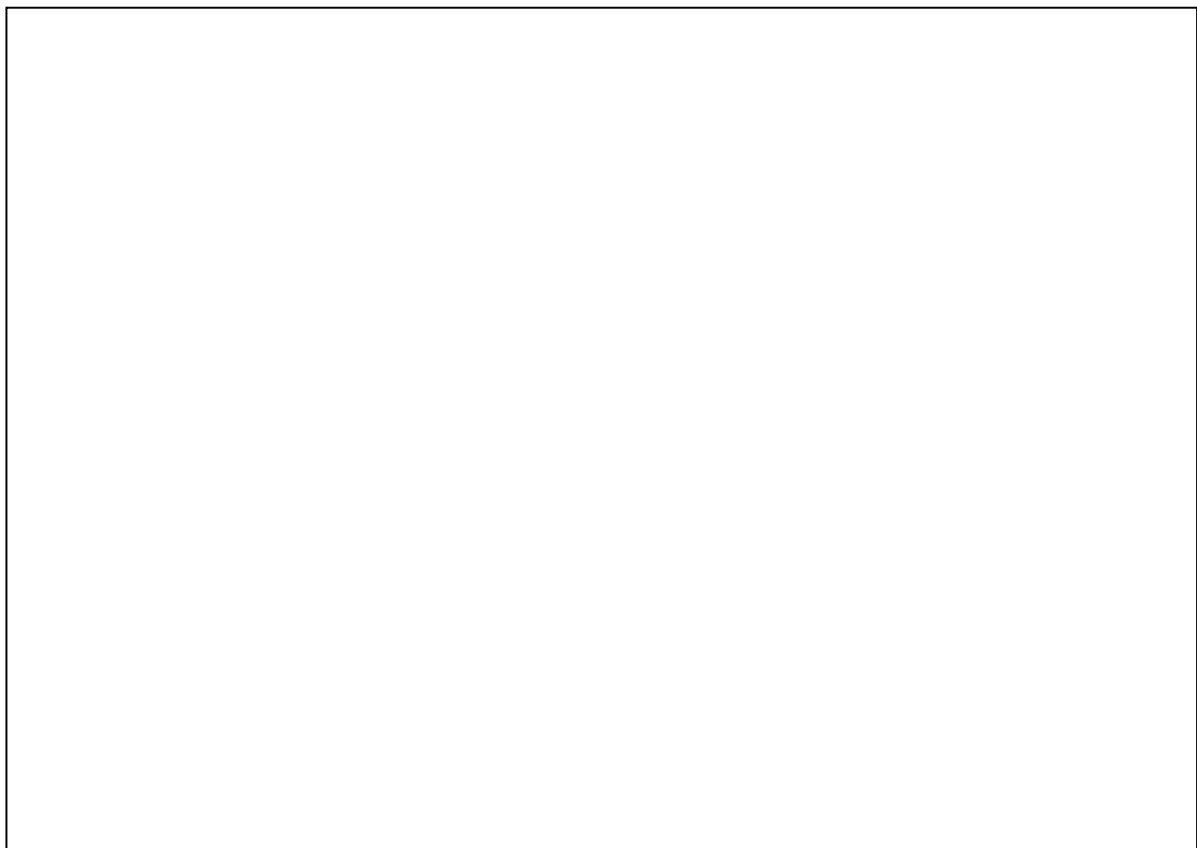








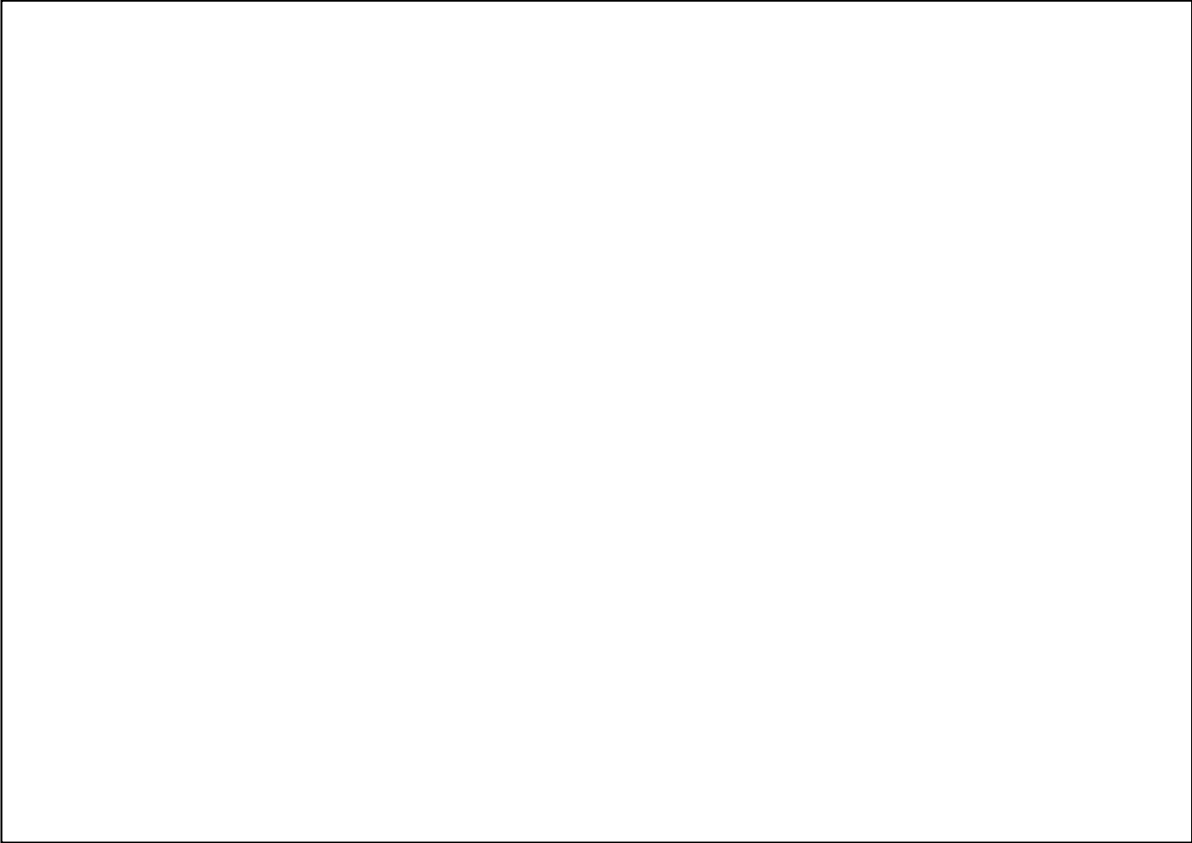


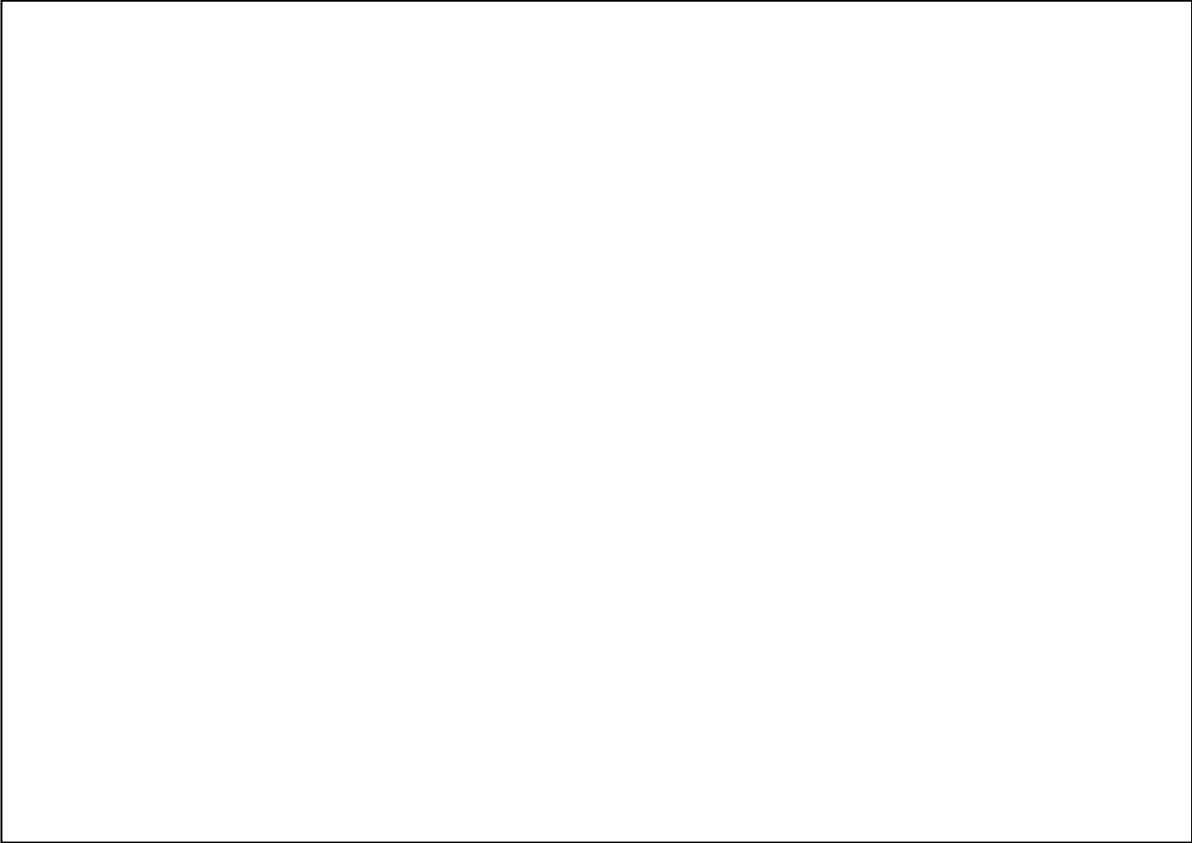


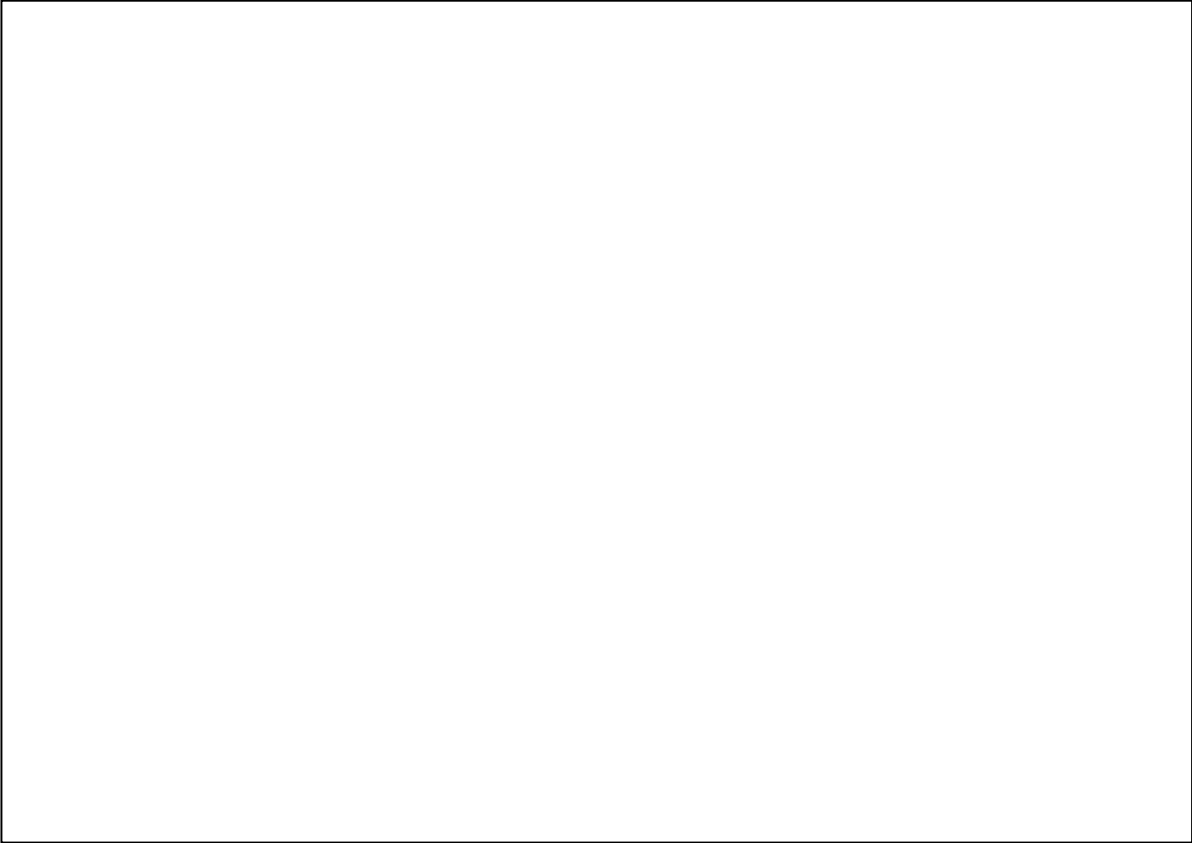


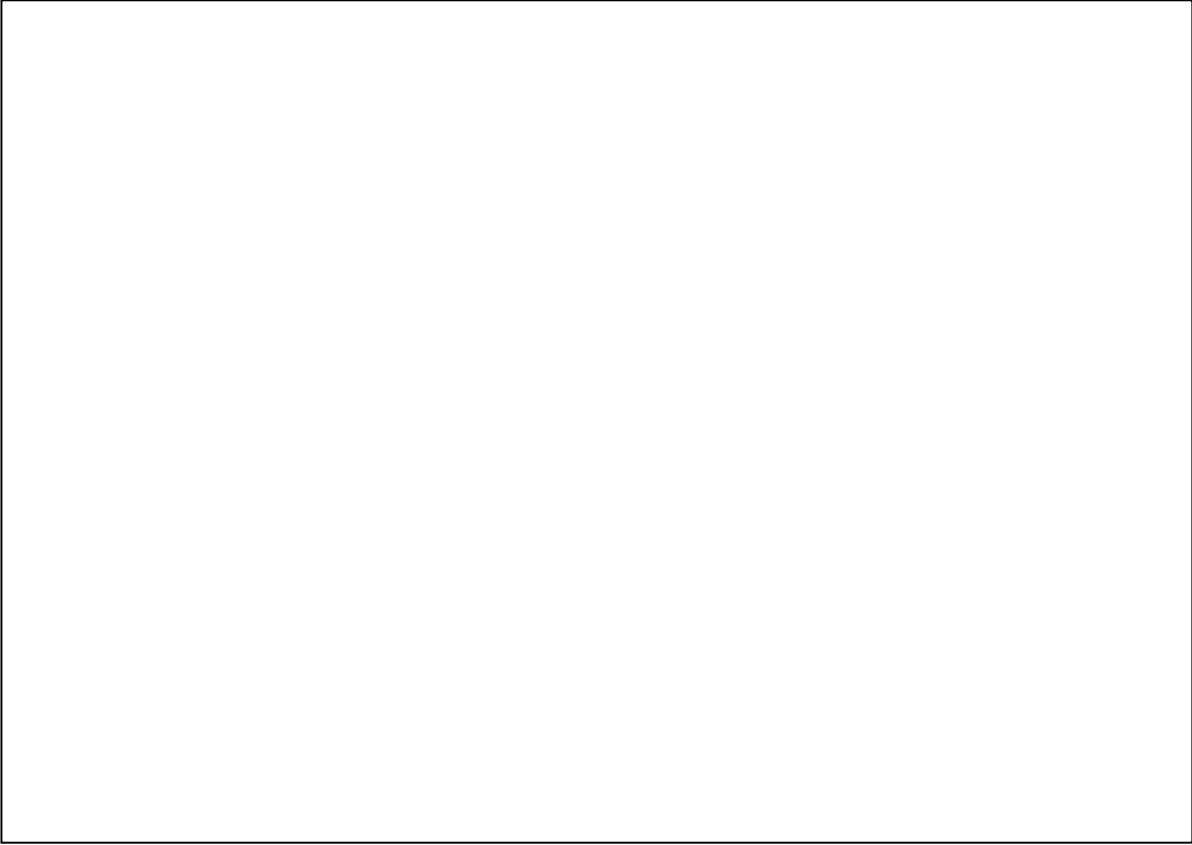
Quelques chiffres

Service OK	Nombre de 9	Temps down cumulé sur une année
98%		7,3 jours
99%	2-neuf	87,6 heures
99,5%		43,8 heures
99,9%	3-neuf	8,8 heures
99,95%		4,4 heures
99,99%	4-neuf	53 minutes
99,999%	5-neuf	5,3 minutes
99,9999%	6-neuf	31 secondes











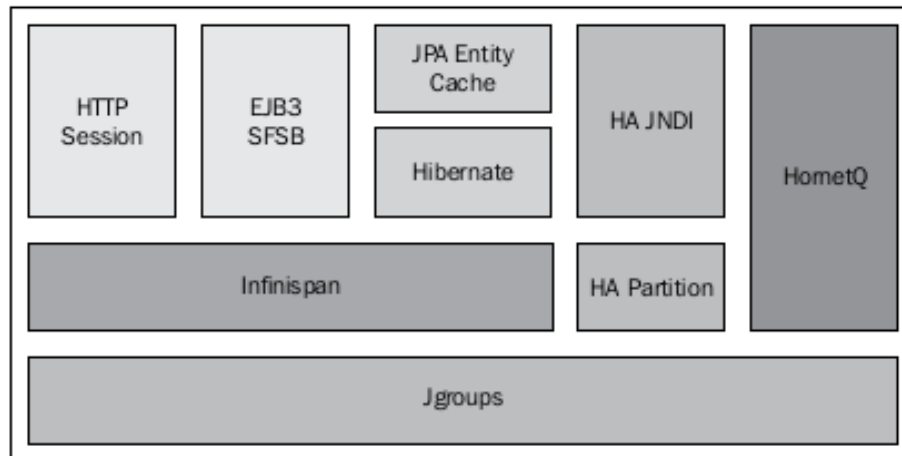
Sous-systèmes

Les différents sous-systèmes prenant part à la mise en place d'un cluster Jboss sont

- **JGroups** utilisé pour la communication entre nœuds
- **Infinispan** qui fait office de cache de réplication et synchronisation entre noeuds
- **HornetQ** avec ses propres capacités de clustering



Architecture





Mode standalone et domaine

AS 7 est distribué avec des fichiers de configuration dédiés au cluster :

- ***standalone-ha.xml*** en mode standalone
- Profils ***ha-profile*** en mode domaine

Un cluster de serveurs *standalone* est un choix possible.

Cependant, il sera alors nécessaire de mettre au point ses propres outils de gestion des nœuds

Une organisation en domaine permet de bénéficier de ce type d'outils



Vertical/Horizontal

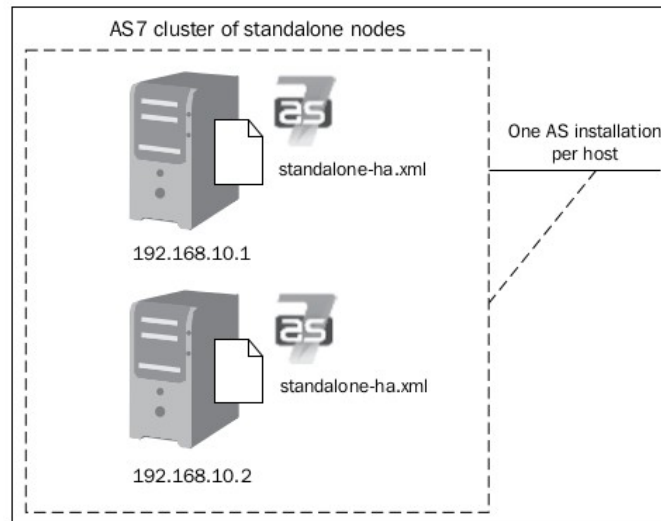
Un cluster horizontal nécessite le moins de configuration ; la seule chose à faire est d'associer le serveur à son adresse IP et de démarrer le serveur avec une configuration cluster.

Pour un cluster vertical, les conflits de port doivent être évités. 2 options sont possible :

- Définir plusieurs adresses IP sur la même machine
- Définir un offset de port pour chaque serveur



Horizontal





Exemple horizontal

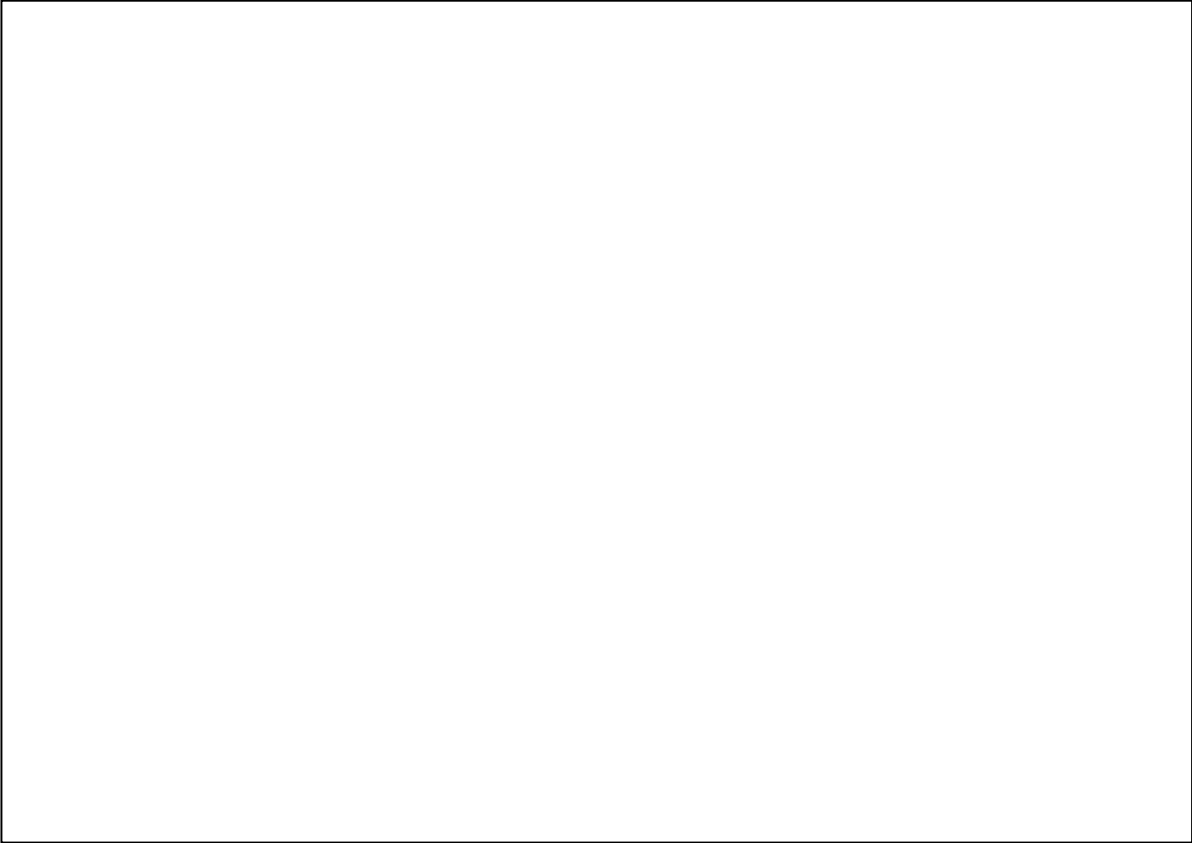
```
<interfaces>
  <interface name="management"><inet-address value="192.168.10.1"/></interface>
  <interface name="public"><inet-address value="192.168.10.1"/></interface>
</interfaces>

Seconde machine (192.168.10.2)

<interfaces>
  <interface name="management"><inet-address value="192.168.10.2"/></interface>
  <interface name="public"><inet-address value="192.168.10.2"/></interface>
</interfaces>

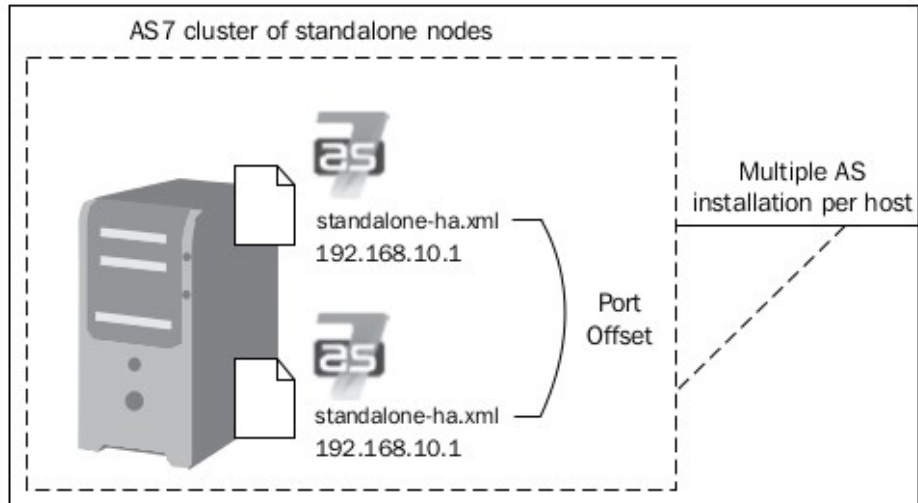
...

standalone.bat --server-config=standalone-ha.xml
```





Vertical avec offset de port





Exemple

```
<!--server 1 UNCHANGED -->  
<socket-binding-group name="standard-sockets" default-  
interface="public">  
.  
.  
.  
</socket-binding-group>
```

Déclaration de l'offset de port (150):

```
<!--server 2 -->  
<socket-binding-group name="standard-sockets" default-  
interface="public" port-offset="150">  
.  
.  
.  
</socket-binding-group>
```



Interfaces d'administration

```
<!--server 1 UNCHANGED -->
<socket-binding-group name="standard-sockets" default-interface="public" >
. . . . .
<socket-binding name="management-native" interface="management" port="9999"/>
<socket-binding name="management-http" interface="management" port="9990"/>
. . . . .
</socket-binding-group>
```

Sur le second serveur, un port différent

```
<!--server 2 -->
<socket-binding-group name="standard-sockets" default-interface="public" port-
offset="150">
. . . . .
<socket-binding name="management-native" interface="management" port="9999"/>
<socket-binding name="management-http" interface="management" port="9990"/>
. . . . .
</socket-binding-group>
```



Domaine

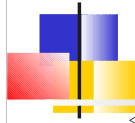
La configuration cluster est déjà présente sous la forme du profil **ha-profile** dans *domain.xml*.

Il suffit alors d'associer un groupe de serveurs à ce profil



Groupe de serveur vers profil ha

```
<server-groups>
  <server-group name="main-server-group" profile="ha">
    <jvm name="default">
      <heap size="64m" max-size="512m"/>
    </jvm>
    <socket-binding-group ref="ha-sockets"/>
  </server-group>
  <server-group name="other-server-group" profile="ha">
    <jvm name="default">
      <heap size="64m" max-size="512m"/>
    </jvm>
    <socket-binding-group ref="ha-sockets"/>
  </server-group>
</server-groups>
```

<socket-binding>

```
<socket-binding-group name="ha-sockets" default-interface="public">
  <socket-binding name="http" port="8080"/>
  <socket-binding name="https" port="8443"/>
  <socket-binding name="jgroups-diagnostics" port="0" multicast-address="224.0.75.75" multicast-
port="7500"/>
  <socket-binding name="jgroups-mping" port="0" multicast-address="230.0.0.4" multicast-
port="45700"/>
  <socket-binding name="jgroups-tcp" port="7600"/>
  <socket-binding name="jgroups-tcp-fd" port="57600"/>
  <socket-binding name="jgroups-udp" port="55200" multicast-address="230.0.0.4" multicast-
port="45688"/>
  <socket-binding name="jgroups-udp-fd" port="54200"/>
  <socket-binding name="jmx-connector-registry" port="1090"/>
  <socket-binding name="jmx-connector-server" port="1091"/>
  <socket-binding name="jndi" port="1099"/>
  <socket-binding name="modcluster" port="0" multicast-address="224.0.1.105" multicast-
port="23364"/>
  <socket-binding name="osgi-http" port="8090"/>
  <socket-binding name="remoting" port="4447"/>
  <socket-binding name="txn-recovery-environment" port="4712"/>
  <socket-binding name="txn-status-manager" port="4713"/>
</socket-binding-group>
```



host.xml

Dans *host.xml*, les serveurs doivent être associés au socket binding group **ha-sockets**

```
<servers>
  <server name="server-one" group="main-server-group" >
    <socket-binding-group ref="ha-sockets" port-offset="150"/>
    <jvm name="default"/>
  </server>
  <server name="server-two" group="main-server-group" auto-start="true" >
    <socket-binding-group ref="ha-sockets" port-offset="250"/>
    <jvm name="default"/>
  </server>
  <server name="server-three" group="other-server-group" auto-start="false">
    <socket-binding-group ref="standard-sockets" port-offset="350"/>
    <jvm name="default"/>
  </server>
</servers>
```



Démarrage en mode lazy

Avec JBoss AS 7 les services du clustering sont démarrés à la demande.

Ainsi, juste démarrer les serveurs en cluster sans application clusterisés ne démarre aucun service ni channels JGroups



Farming – no more !

JBoss AS 7 n'utilise plus le concept de *farming* comme dans les précédentes versions

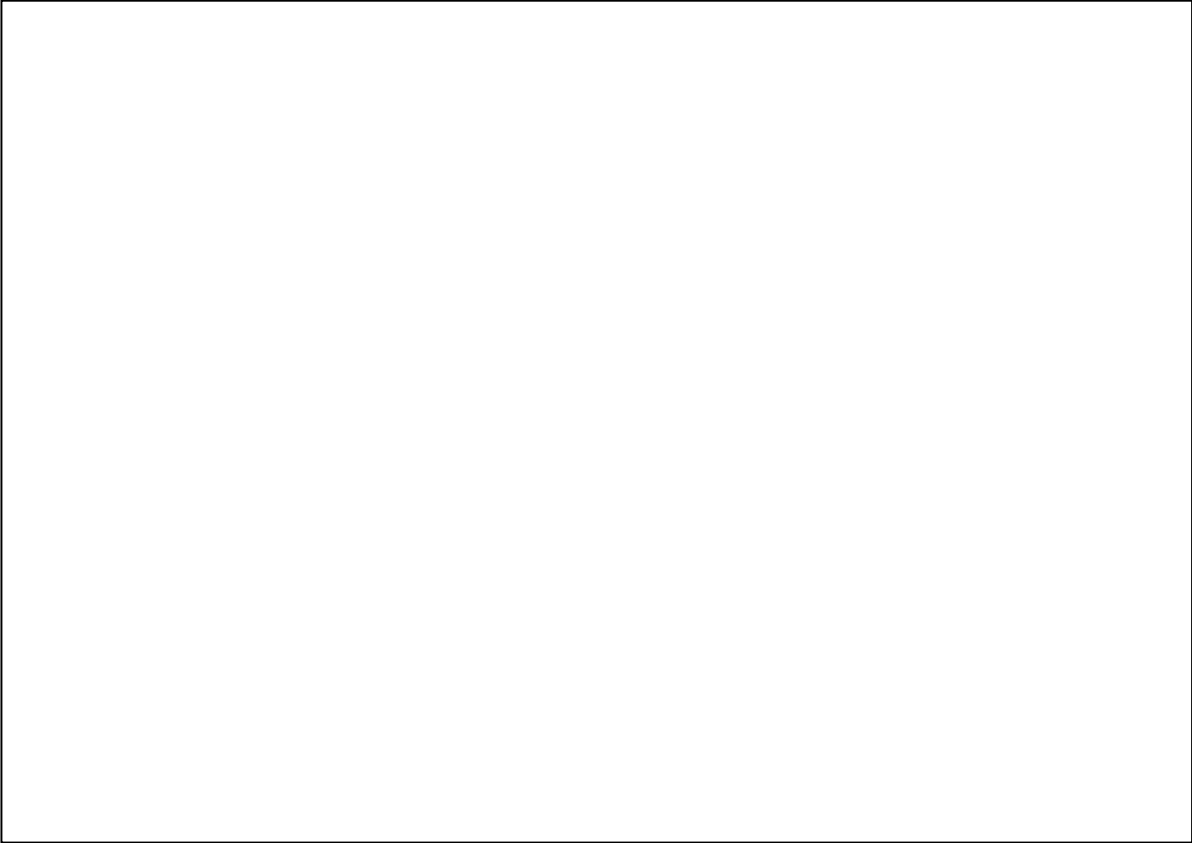
Pour distribuer une application sur un ensemble de nœud, la meilleure approche est de créer un script CLI qui déploie l'application sur tous les nœuds

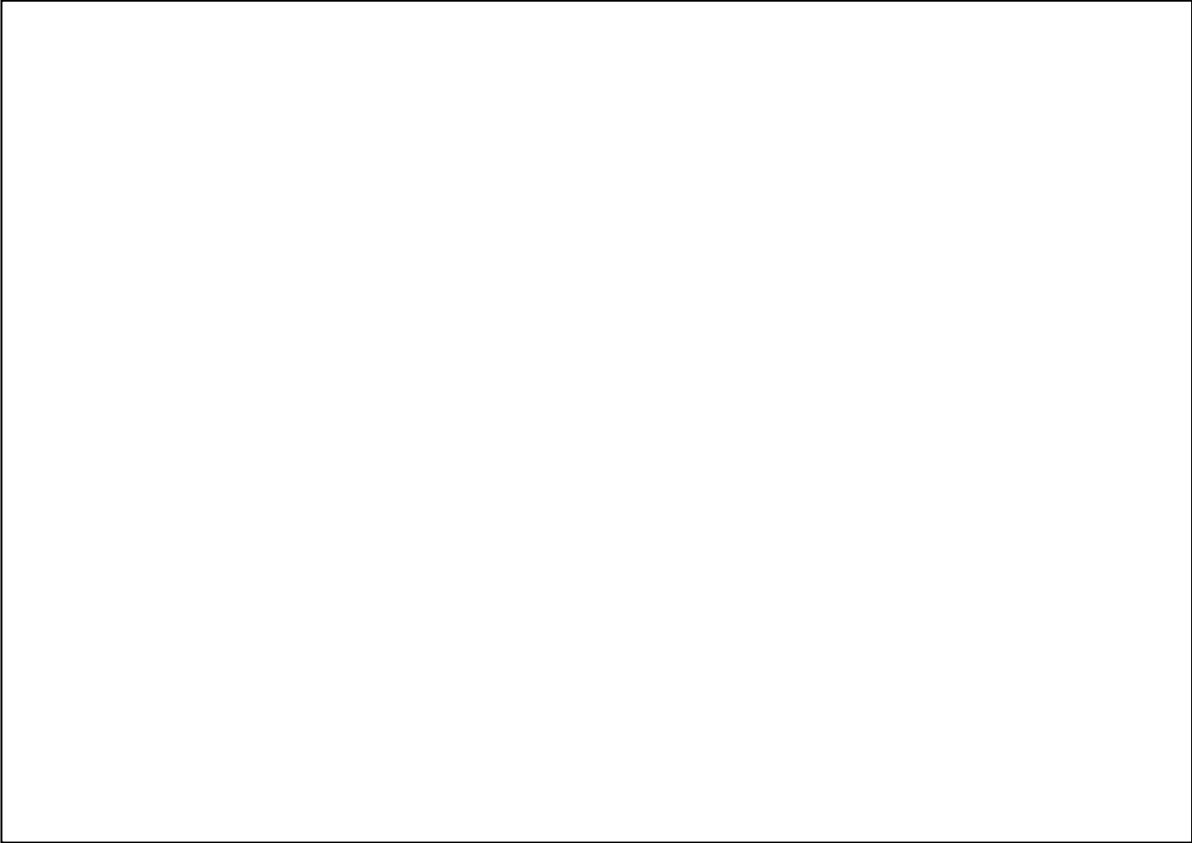
Par exemple :

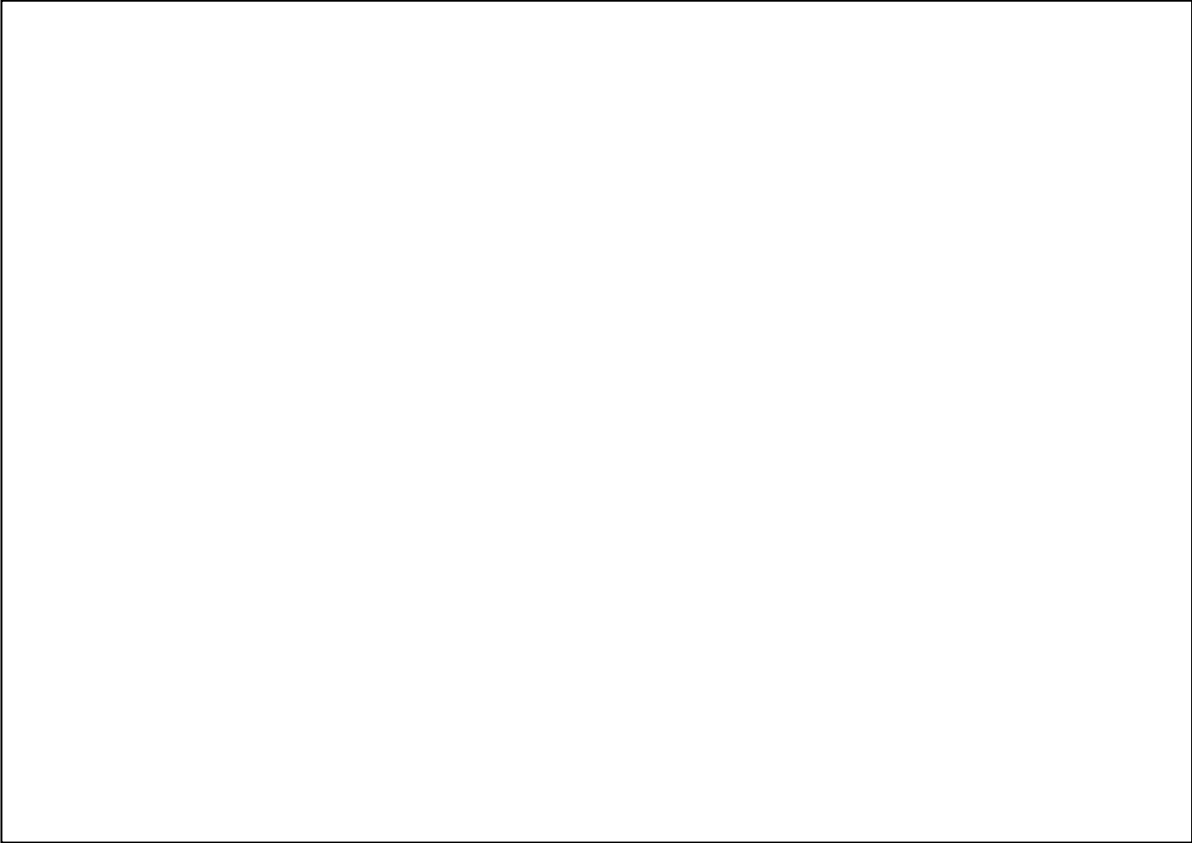
```
connect 192.168.10.1
deploy Example.war
connect 192.168.10.2
deploy Example.war
```

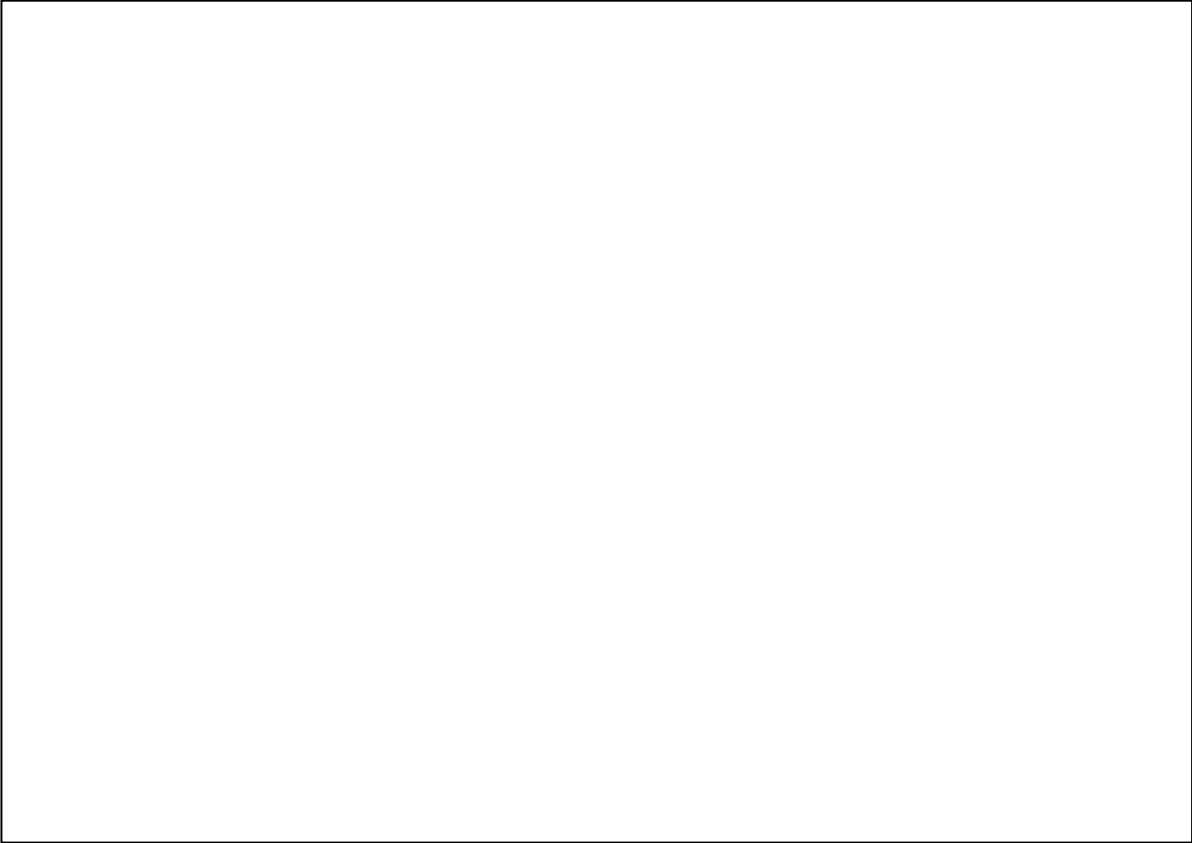
Puis :

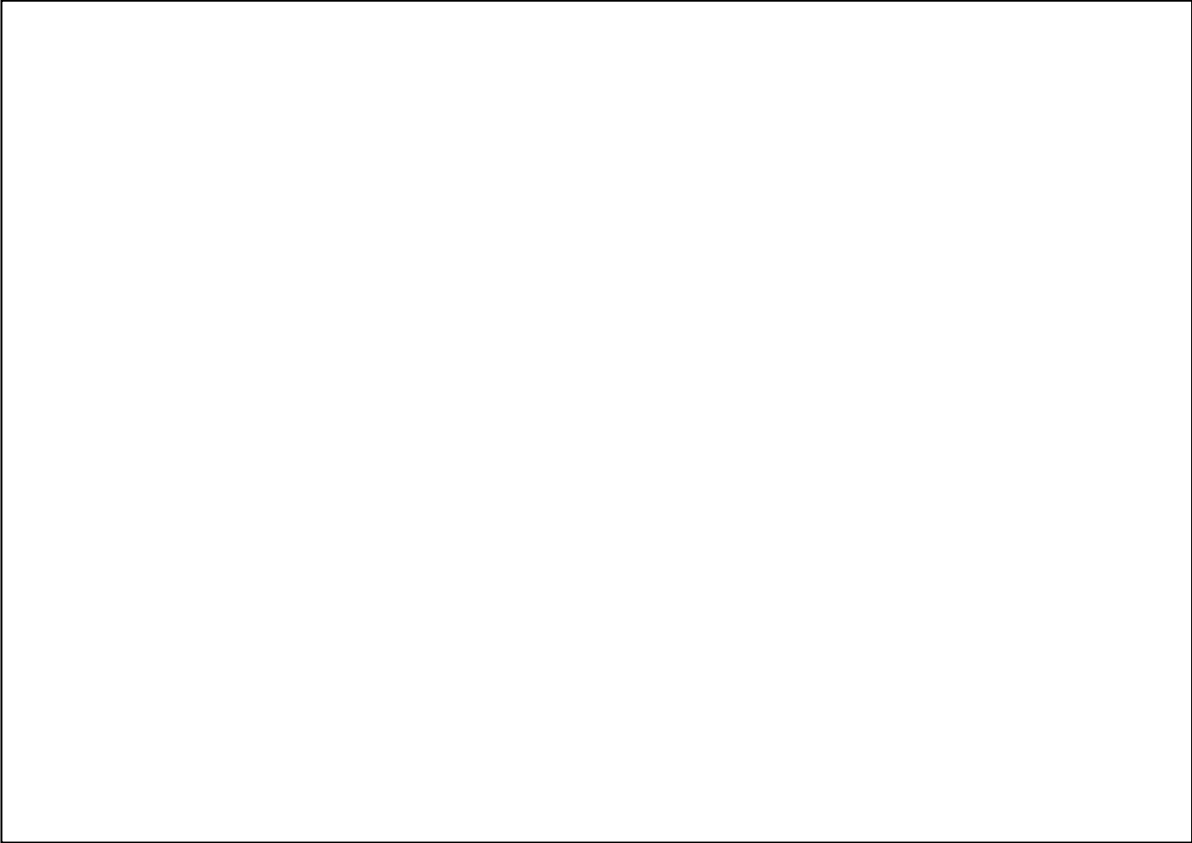
```
jboss-cli.sh --user=username --password=password
--file=deploy.cli
```

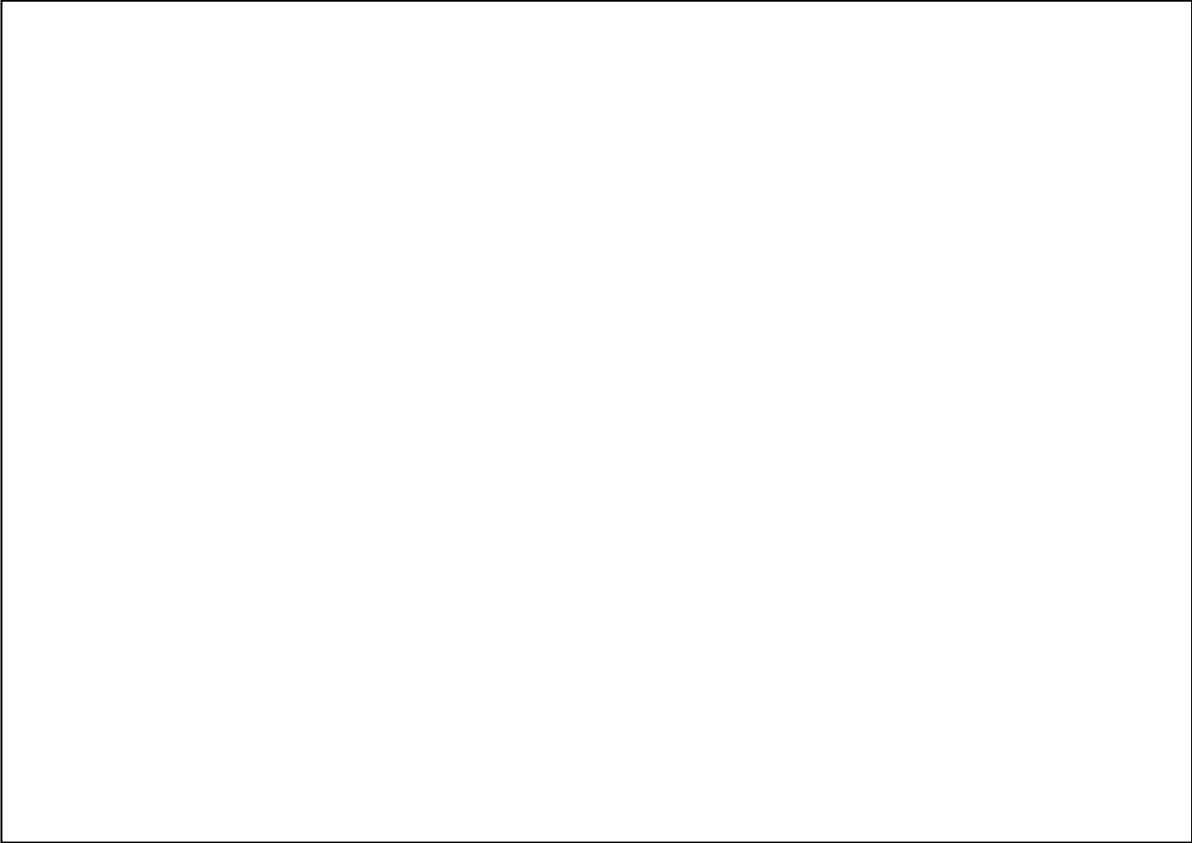


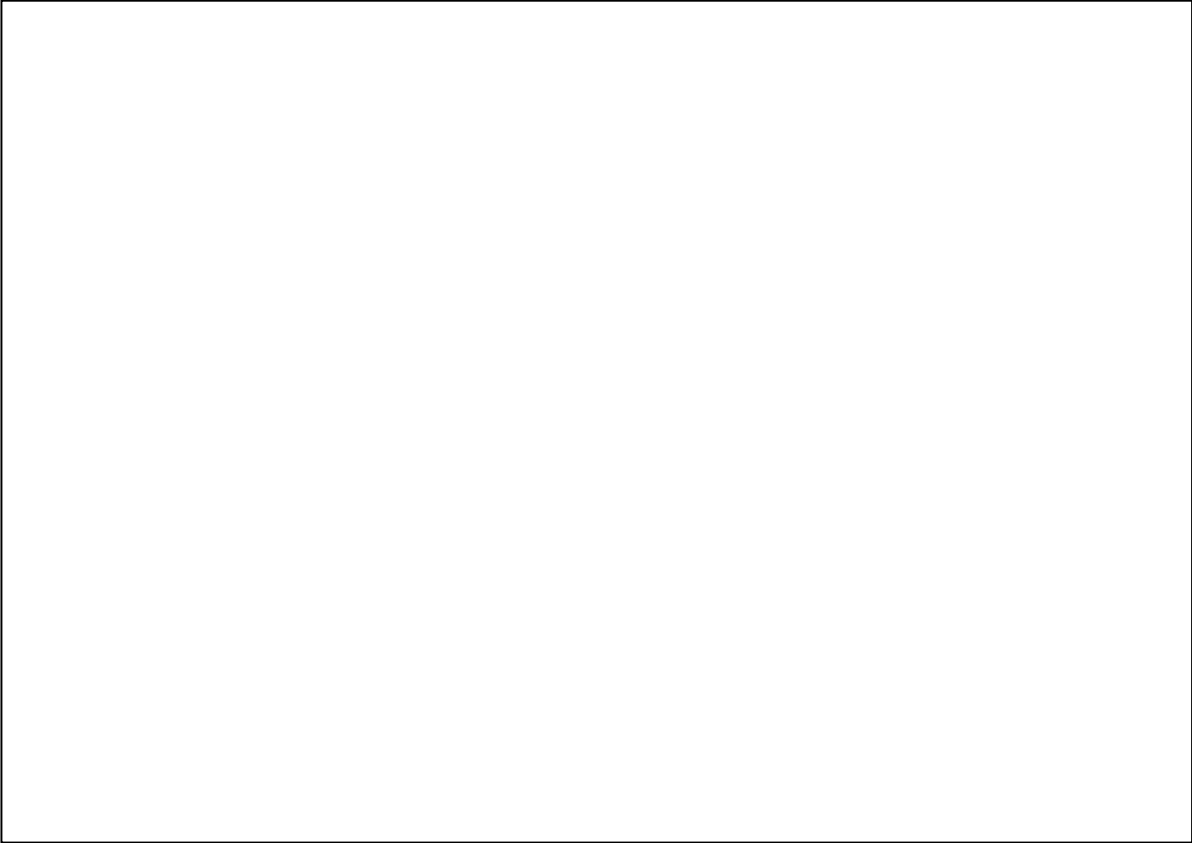


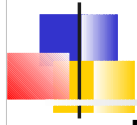






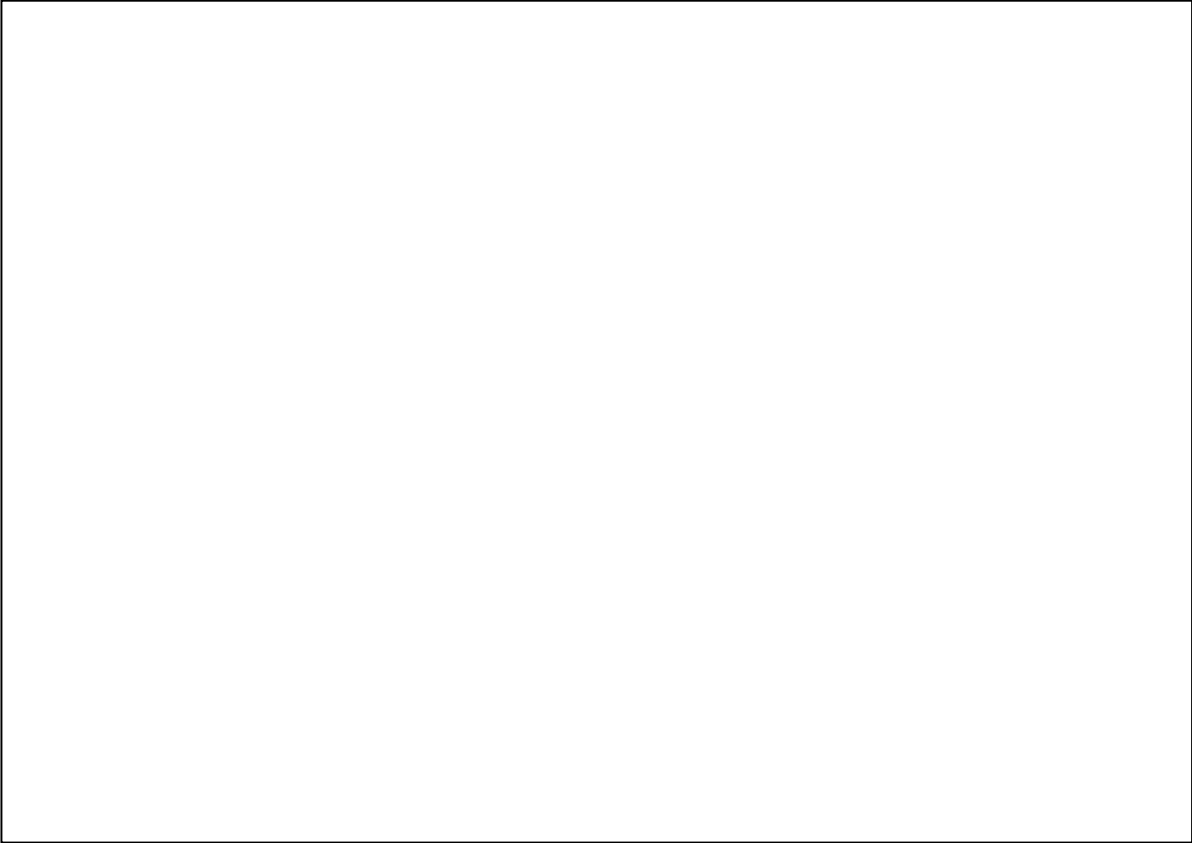


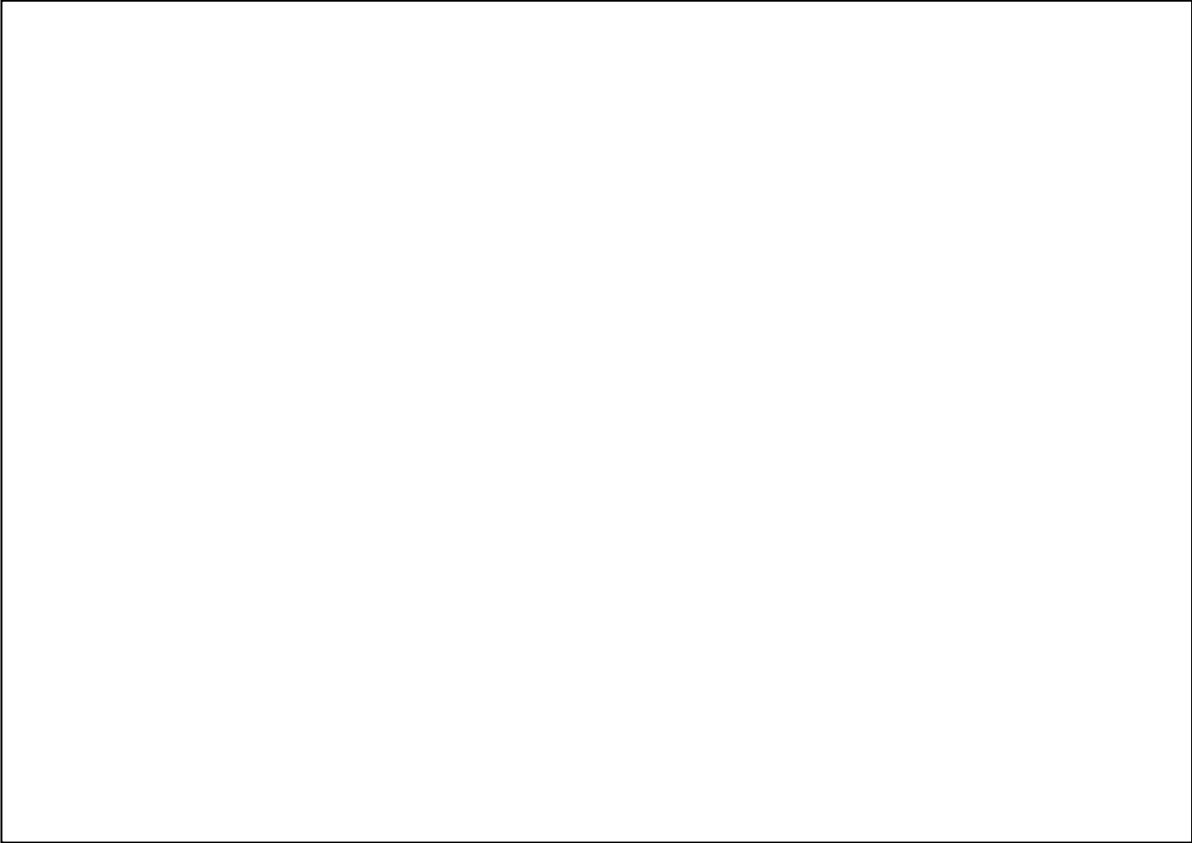


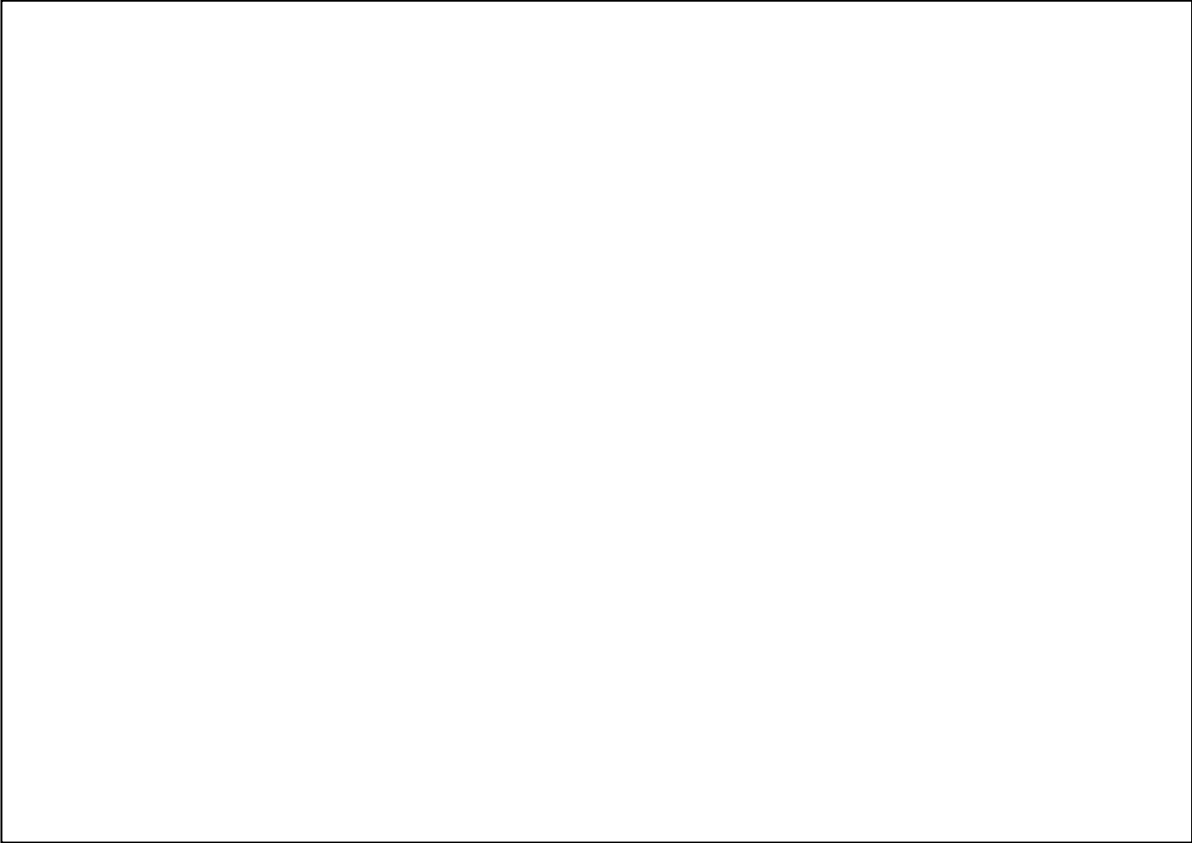


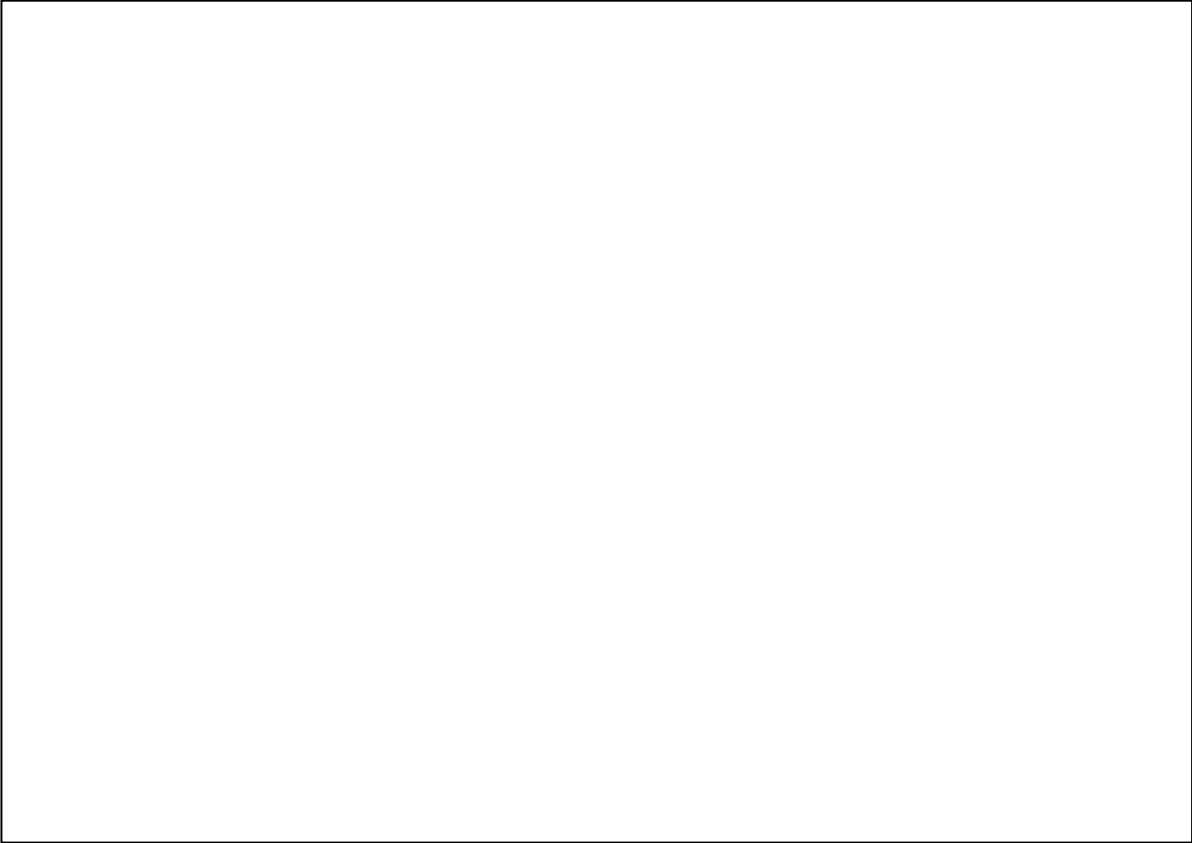
TP

Démarrage d'un cluster vertical











UDP et multicast

Lors de l'utilisation de UDP et du multicast plusieurs problèmes typiques peuvent survenir :

- Les nœuds sont derrière un firewall qui bloque les ports multicast
- Le réseau est derrière un gateway qui ne prend pas en compte le multicast (réseau du domicile par exemple).
=> ajouter une route au gateway afin qu'il redirige le trafic multicast sur le réseau local



Test multicast avec JGroups

JGroups est distribué avec deux programmes de test du multicast: ***McastReceiverTest*** et ***McastSenderTest***

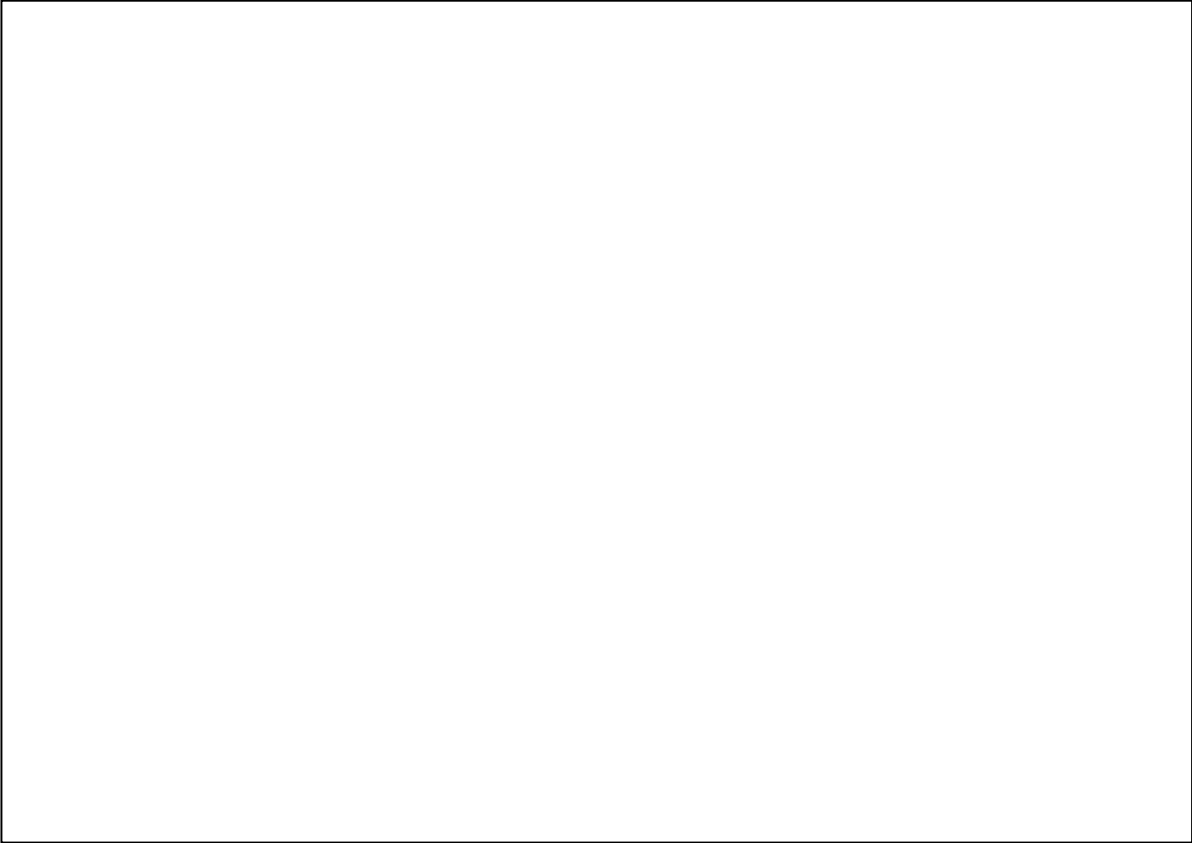
Démarrer *McastReceiverTest*, :

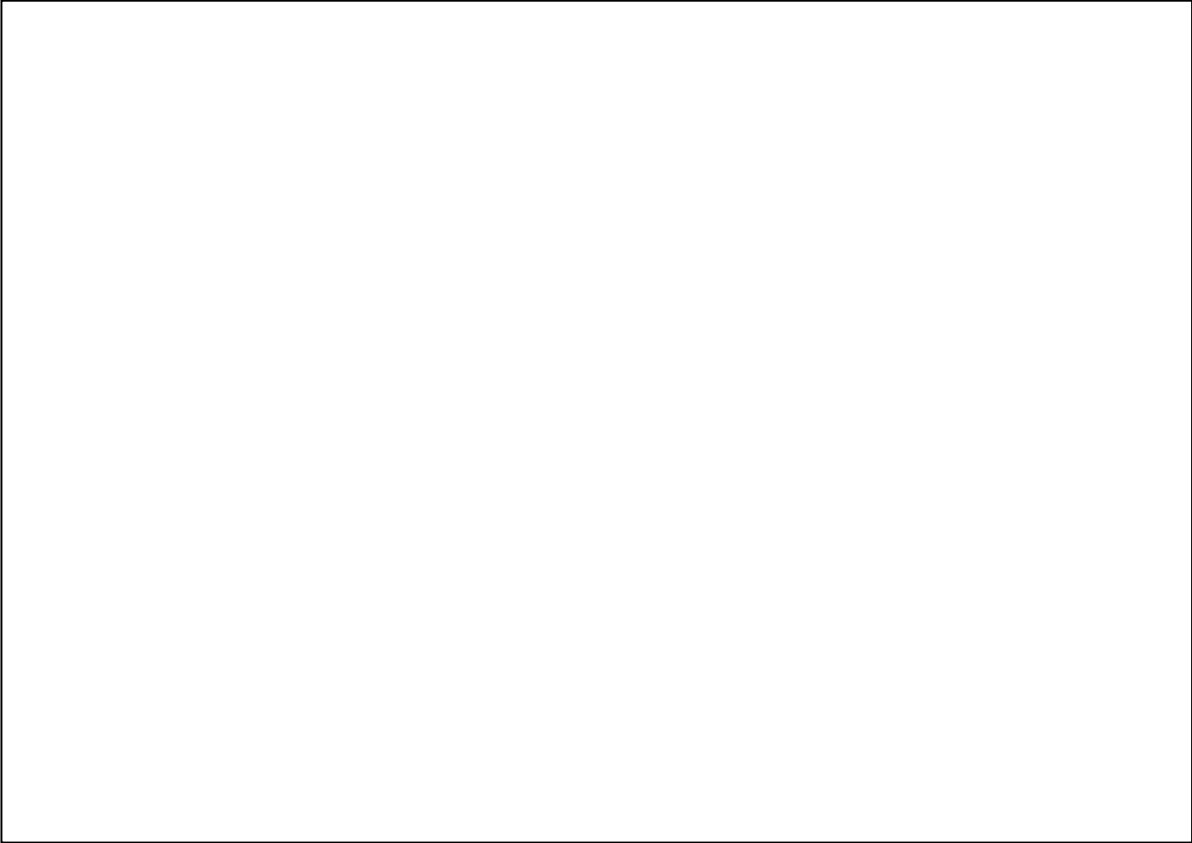
```
java -classpath jgroups-3.0.0.Final.jar org.jgroups.tests.  
McastReceiverTest -mcast_addr 224.10.10.10 -port 5555
```

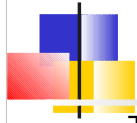
Puis *McastSenderTest*:

```
java -classpath jgroups-3.0.0.Final.jar  
org.jgroups.tests.McastSenderTest -mcast_addr  
224.10.10.10 -port 5555
```

Les caractères saisis dans la fenêtre de l'émetteur doivent s'afficher dans la fenêtre du récepteur







Types de protocoles

Transport : UDP, TCP

Découverte : PING, TCPPING, TCPGOSSIP, MPING

Fragmentation/Réassemblage: FRAG, FRAG2

Transmission sûre : CAUSAL, NAKACK, pbcast.NAKACK, SMACK, UNICAST, PBCAST, pbcast.STABLE

Gestion d'appartenance : pbcast.GMS, MERGE, MERGE2, VIEW_SYNC

Détection de panne : FD, FD_SIMPLE, FD_PING, FD_ICMP, FD SOCK, VERIFY_SUSPECT

Sécurité : AUTH

Transfert d'état : pbcast.STATE_TRANSFER, pbcast.STREAMING_STATE_TRANSFER

Debug : PERF_TP, SIZE, TRACE

Autres : COMPRESS, pbcast.FLUSH



Configuration JGroups

L'API *JGroups* était déjà utilisé dans les versions précédentes de JBoss AS. Les différents *channels* étaient alors définis dans des fichiers de configuration spécifiques

Avec JBoss AS 7, la configuration JGroups est embarquée dans le fichier de configuration principal dans le sous-système *jgroups*



Sous-système JGroups

```
<subsystem xmlns="urn:jboss:domain:jgroups:1.0" default-stack="udp">
  <stack name="udp">
    <transport type="UDP" socket-binding="jgroups-udp" diagnostics-socket-binding="jgroups-diagnostics"/>
    <protocol type="PING"/>
    <protocol type="MERGE2"/>
    <protocol type="FD_SOCK" socket-binding="jgroups-udp-fd"/>
    <protocol type="FD"/>
    <protocol type="VERIFY_SUSPECT"/>
    <protocol type="BARRIER"/>
    <protocol type="pbcast.NAKACK"/>
    <protocol type="UNICAST"/>
    <protocol type="pbcast.STABLE"/>
    <protocol type="VIEW_SYNC"/>
    <protocol type="pbcast.GMS"/>
    <protocol type="UFC"/>
    <protocol type="MFC"/>
    <protocol type="FRAG2"/>
    <protocol type="pbcast.STREAMING_STATE_TRANSFER"/>
    <protocol type="pbcast.FLUSH"/>
  </stack>
  <!-- More stacks -->
</subsystem>
```



Configuration des piles

La configuration définit donc des *stack* que l'on peut attribuer aux services de clustering

L'attribut ***default-stack*** du sous-système définit la pile Jgroups par défaut.

Voir *docs/schema/jboss-jgroups.xsd*



UDP / TCP

Avec une socket UDP multicast , un client peut contacter plusieurs serveurs avec un paquet unique sans connaître les adresses IP des différents serveurs.

Pour utiliser UDP avec JBoss AS 7 il faut modifier l'attribut ***default-stack***

Par exemple :

```
<subsystem xmlns="urn:jboss:domain:jgroups:1.0"  
default-stack="udp">
```



Socket bindings

```
<socket-binding-group name="clustering-sockets" default-  
  interface="loopback" port-offset="0">  
  <socket-binding name="jgroups-udp" port="55200"  
    multicast-address="230.0.0.4" multicast-port="45688"/>  
  <socket-binding name="jgroups-udp-fd" port="54200"/>  
  <socket-binding name="jgroups-diagnostics" port="0"  
    multicast-address="224.0.75.75" multicast-port="7500"/>  
  <socket-binding name="jgroups-tcp" port="7600"/>  
  <socket-binding name="jgroups-tcp-fd" port="57600"/>  
  <socket-binding name="jgroups-mping" port="0"  
    multicast-address="230.0.0.4" multicast-port="45700"/>  
</socket-binding-group>
```



Personnalisation des protocoles

Les protocoles définies dans une stack ont de nombreux paramètres de configuration par défaut.

Il est possible de surcharger la configuration par défaut en définissant une nouvelle valeur.

```
<stack name="udp">
  <transport type="UDP" ...>
    <property name="enable_bundling">true</property>
    <property name="ip_ttl">0</property>
  </transport>
  <!-- ... -->
</stack>
```



Configuration des threads

```
<subsystem xmlns="urn:jboss:domain:clustering:jgroups:1.0">
  <stack name="udp">
    <transport type="UDP" ...
default-executor="jgroups-default" oob-executor="jgroups-oob" timer-executor="jgroups-timer"/>
    <!-- Remaining protocols -->
  </stack>
</subsystem>

<subsystem xmlns="urn:jboss:domain:threads:1.0">
  <queueless-thread-pool name="jgroups-default" blocking="false">
    <core-threads count="20" per-cpu="40"/>
    <max-threads count="200" per-cpu="400"/>
    <keepalive-time time="5" unit="seconds"/>
  </queueless-thread-pool>
  <queueless-thread-pool name="jgroups-oob" blocking="false">
    <!-- ... -->
  </queueless-thread-pool>
  <scheduled-thread-pool name="jgroups-timer">
    <!-- ... -->
  </scheduled-thread-pool>
</subsystem>
```





Introduction

Le serveur http apache est souvent mis en frontal des applications web déployées sur Jboss

Le serveur Apache apporte en effet quelques avantages :

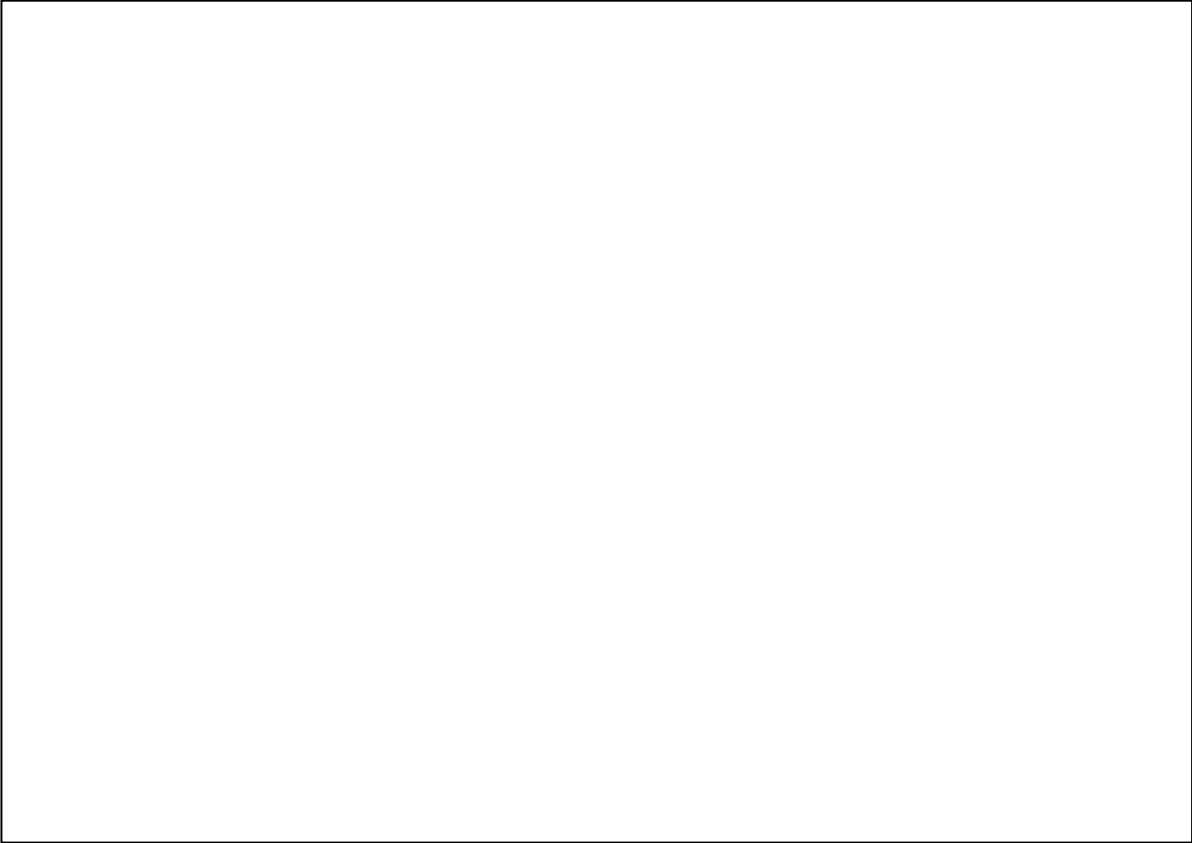
- Il est généralement plus rapide pour servir les ressources statiques que JBoss Web server.
- Le serveur applicatif qui contient des données sensibles peut alors être placé dans une zone sécurisée. Apache se comportant comme un serveur proxy
- Apache peut également devenir un répartiteur de charge, distribuant les requêtes vers les différentes instances de JBoss Web server. En cas de défaillance, Apache continue de façon transparente à distribuer les requêtes vers les nœuds actifs

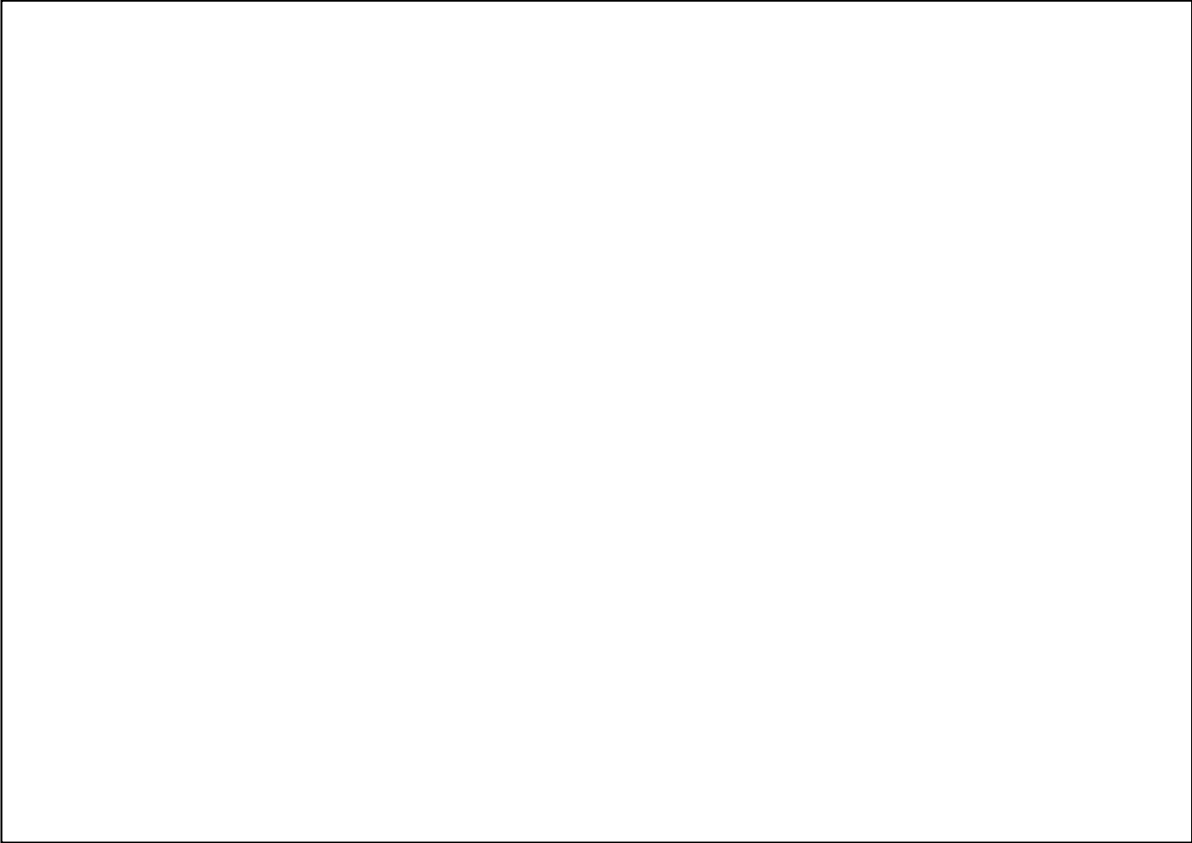


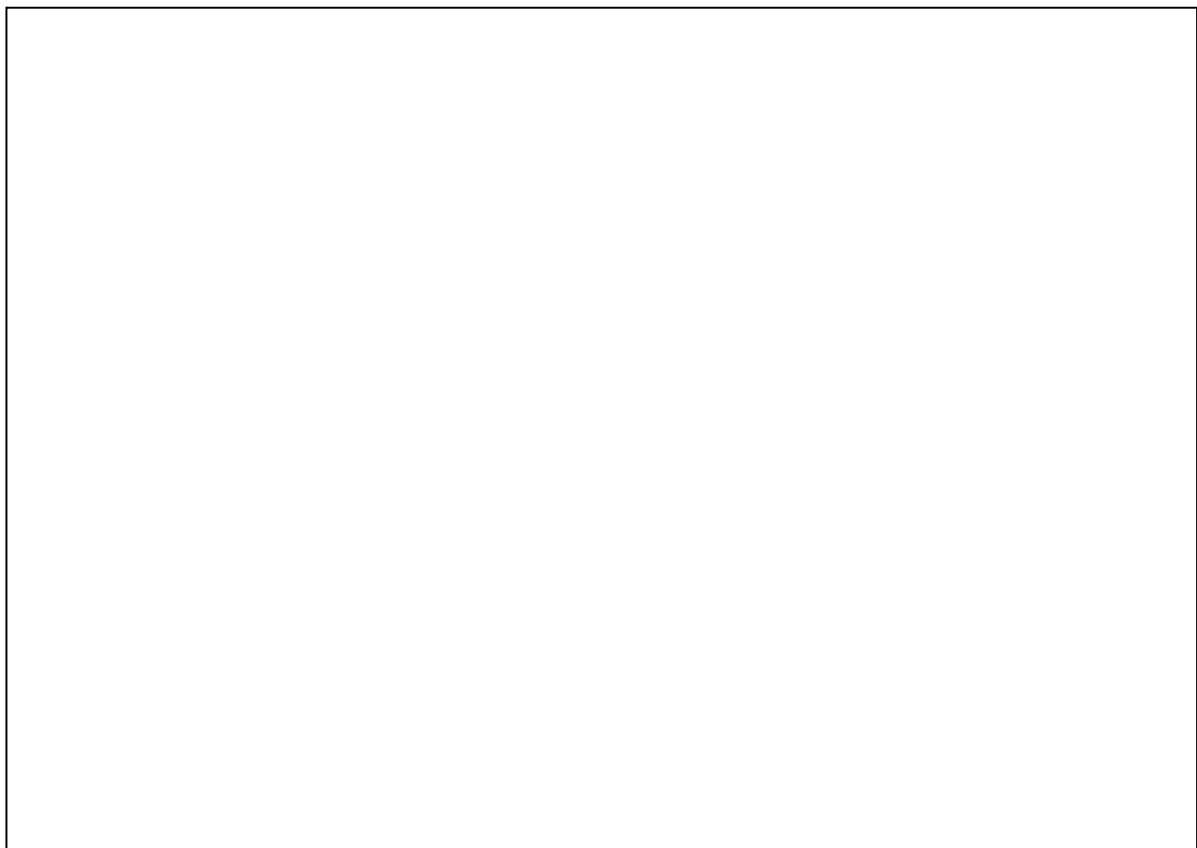
Types de connexions

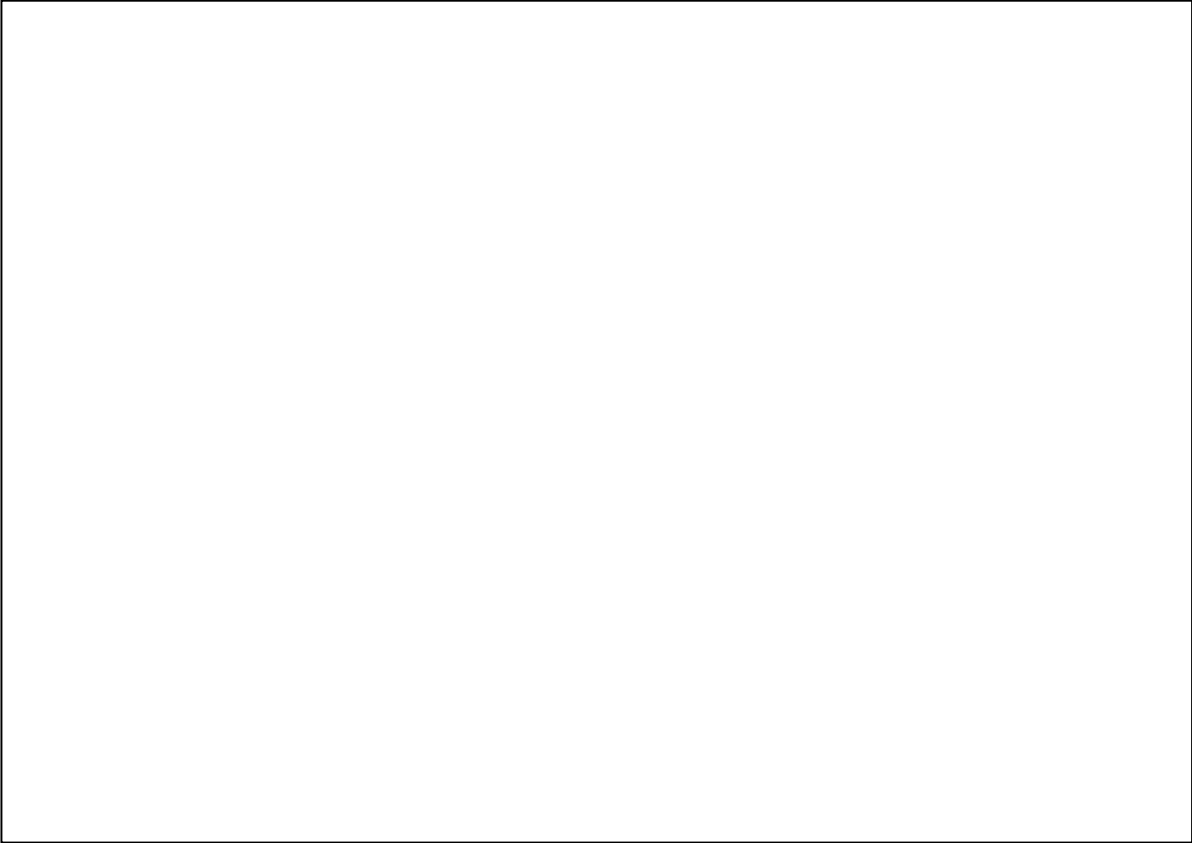
La connexion entre Apache et JBoss AS peut se faire via différents module Apache :

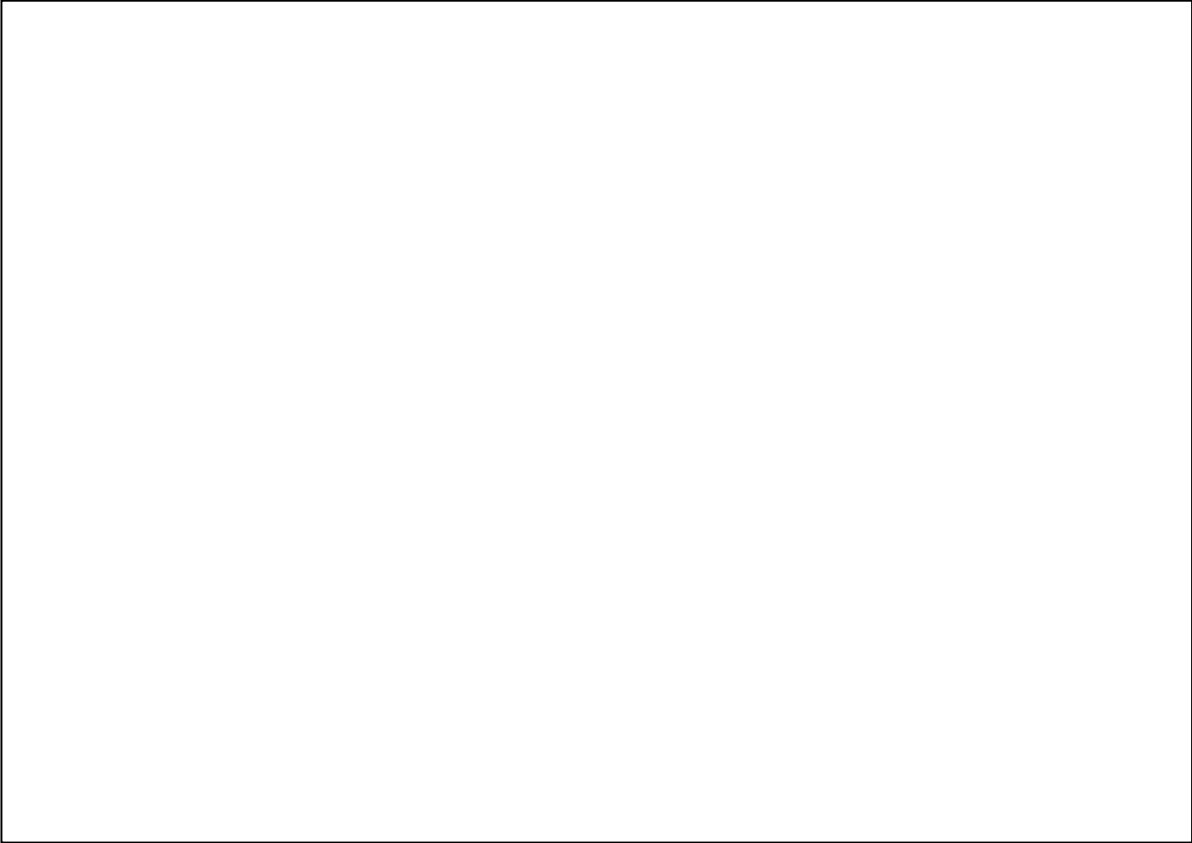
- La librairie Tomcat's ***mod_jk***
- La librairie Apache ***mod_proxy***
- Avec Jboss 7, la nouvelle API ***mod_cluster*** API













JBoss

Du côté JBoss, il faut déclarer le connecteur AJP

```
<subsystem xmlns="urn:jboss:domain:web:1.1">
  <connector name="http" protocol="HTTP/1.1" socket-
binding="http" scheme="http"/>
  <connector name="AJP" protocol="AJP/1.3"
socket-binding="ajp" />
  <virtual-server name="localhost">
    <alias name="example.com"/>
  </virtual-server>
</subsystem>
```



Port AJP

```
<socket-binding-group name="standard-sockets"
  default-interface="default">
  <socket-binding name="http" port="8080"/>
  <socket-binding name="ajp" port="8009"/>
  . . . .
</socket-binding-group>
```



mod_proxy

Depuis Apache 1.3, il est possible de configurer le module ***mod_proxy*** qui configure Apache comme serveur proxy

Cela permet de propager les requêtes vers Jboss sans avoir à configurer un connecteur web comme *mod_jk*.

C'est le moyen le plus facile pour mettre Apache en frontal de Jboss mais c'est également le plus lent



Httpd.conf

```
LoadModule proxy_module modules/mod_proxy.so
```

```
ProxyPass /myapp http://localhost:8080/myapp
```

```
ProxyPassReverse /myapp http://localhost:8080/myapp
```




Proxy ajp

Depuis Apache 2.2, il existe un autre module nommé ***mod_proxy_ajp*** qui peut être utilisé de la même façon.

Cependant, il utilise le protocole AJP

```
LoadModule proxy_ajp_module modules/mod_proxy_ajp.so
ProxyPass / ajp://localhost:8009/
ProxyPassReverse / ajp://localhost:8009/
```



JBoss

Côté Jboss, le connecteur AJP doit être activé

```
<subsystem xmlns="urn:jboss:domain:web:1.1">
  <connector name="AJP" protocol="AJP/1.3" socket-binding="ajp" />
</subsystem>

. . . . .

<socket-binding-group name="standard-sockets" default-
interface="default">
  <socket-binding name="ajp" port="8009"/>
  . . . . .
</socket-binding-group>
```



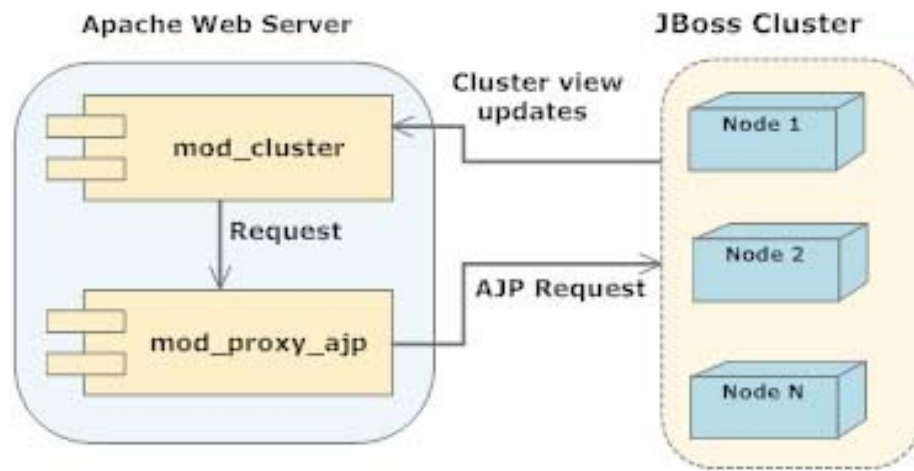
mod_cluster

L'utilisation du module ***mod_cluster*** apporte les avantages suivant:

- Configuration dynamique du cluster
- Possibilité d'obtenir des métriques de charge
- Notification du cycle de vie de l'application



Architecture



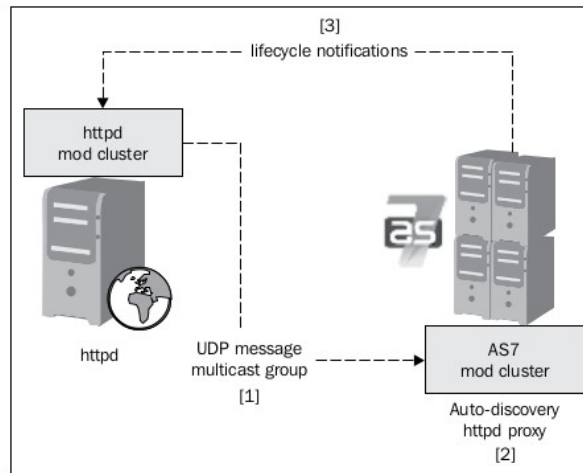


Ajout dynamique de noeud

Avec *mod_cluster*, il est possible d'ajouter ou retirer dynamiquement des nœuds car ils sont découverts automatiquement

La librairie *mod_cluster* du côté *httpd* envoie des messages UDP sur un groupe multicast auquel se sont abonnés les nœuds Jboss

Cela leur permet de découvrir automatiquement les proxy http lors des notifications du cycle de vie applicatif.





Installation de *mod_cluster*

Du côté JBoss 7, le module *mod_cluster* 1.1.3 est déjà présent dans la configuration cluster soit dans *standalone-ha.xml* file ou dans le profil ha de *domain.xml* :

```
<subsystem xmlns="urn:jboss:domain:modcluster:1.0">
  <mod-cluster-config advertise-socket="modcluster"/>
</subsystem>

<socket-binding name="modcluster" port="0" multicast-
address="224.0.1.105" multicast-port="23364"/>
```

Il faut cependant ajouter un attribut dans la configuration du sous-système web :

```
<subsystem xmlns="urn:jboss:domain:web:1.1" default-virtual-
server="default-host" instance-id="${jboss.node.name}"
native="false">
```



Configuration *mod_cluster*

Attributs du sous-système *mod_cluster*

proxy-list : Liste des proxy Apache

proxy-url : Dans le cas d'un seul proxy

advertise : Découverte automatique des proxies, *proxy-list* n'est pas renseigné

advertise-security-key : Les checksum des messages avec les proxies sont vérifiées avec une clé

excluded-contexts (défaut : *ROOT, admin-console, invoker, jbossws, jmx-console, juddi, web-console*) : Les contextes à exclure

auto-enable-contexts (défaut true) : Contextes automatiquement inclus dans la gestion du proxy

stop-context-timeout (défaut : 10 secondes) : Un contexte est désactivé 10 secondes après son arrêt

socket-timeout (défaut 20 seconds) : Timeout pour déclarer un proxy comme down



Configuration *mod_cluster*

Désactiver les statistiques

```
<subsystem xmlns="urn:jboss:domain:modcluster:1.0">  
<mod-cluster-config proxy-list="192.168.0.1:6666"/>  
</subsystem>
```

Exclure des contextes particuliers

```
<subsystem xmlns="urn:jboss:domain:modcluster:1.0">  
<mod-cluster-config excluded-contexts="ROOT, foo"/>  
</subsystem>
```



Configuration des proxies

Le sous-système `mod_cluster` permet également de configurer le comportement du proxy

stickySession (défaut : `true`) Garder le même nœud pour un client

stickySessionRemove (défaut `false`) : Enlever le mode sticky, si le répartiteur ne peut pas atteindre le nœud

stickySessionForce (défaut `false`) : Retourner une erreur si le répartiteur ne peut pas router vers le nœud sticky

workerTimeout : Nombre de secondes à attendre avant qu'un worker soit disponible



Métriques de charges

Le fournisseur de métriques par défaut donne un poids de 1 à tous les nœuds :

```
<subsystem xmlns="urn:jboss:domain:modcluster:1.0">  
  <mod-cluster-config>  
    <simple-load-provider load="1"/>  
  </mod-cluster-config>  
</subsystem>
```



Métriques dynamiques

Le ***dynamic load provider*** permet de répartir la charge entre les nœuds selon des métriques dynamiques prédéfinis ou personnalisés : *cpu*, *mem*, *heap*, *sessions*, *receive-traffic*, *send-traffic*, *requests*, *busyness*



Métriques prédéfinis

cpu : Charge CPU

mem : Mémoire système

heap : Heap Java

sessions : Sessions actives

requests : Requêtes actives

send-traffic : Trafic reçu en kb/s

receive-traffic : Trafic envoyé en kb/s

busyness : Pourcentage des threads occupés dans le pool de threads

connection-pool : Connections occupés d'un pool de connexion (JDBC)



Configuration

Les différents métriques peuvent être configurés via 2 attributs qui influe sur le calcul du facteur de charge global d'un nœud (valeur comprise entre 0 et 100) :

- **weight** (défaut 1) : Le poids de la métrique par rapport aux autres
- **capacity** (requis ou optionnel en fonction des métrique): Sert à normaliser la valeur de charge du métrique ainsi qu'à favoriser certains nœuds.

La capacité doit être choisie de telle sorte que :
facteur de charge / capacity < 1



Calcul de charge

La contribution d'un métrique au calcul de charge global d'un nœud est alors calculé comme suit :

$$(charge / capacity) * weight / total weight$$

Il est également possible de lisser les valeurs fournies en prenant en compte dans le calcul l'historique des valeurs fournies



Apache

Du côté Apache, les librairies doivent être installées

Pour Jboss 7.1 *mod_cluster 1.2*, il faut télécharger soit la version d 'Apache pré-pakagée ou seulement les modules et les charger dans *httpd.conf*

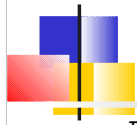
http://www.jboss.org/mod_cluster/Downloads/1-1-3.



Modules

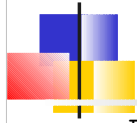
Chacun des modules couvre un aspect de la répartition de charge :

- ***mod_proxy*** et ***mod_proxy_ajp*** sont les modules cœur propageant les requêtes via *http/https* ou *ajp*
- ***mod_manager*** est un module qui lit des information de Jboss 7 et met à jour la mémoire partagée en coordination avec ***mod_slotmem***.
- ***mod_proxy_cluster*** est le module qui contient le répartiteur pour *mod_proxy*.
- ***mod_advertise*** est un module additionnel qui permet à Apache de signaler les changements de statuts applicatif via des paquets multicast packets



Httpd.conf

```
LoadModule proxy_module modules/mod_proxy.so
LoadModule proxy_ajp_module modules/mod_proxy_ajp.so
LoadModule slotmem_module modules/mod_slotmem.so
LoadModule manager_module modules/mod_manager.so
LoadModule proxy_cluster_module modules/mod_proxy_cluster.so
LoadModule advertise_module modules/mod_advertise.so
```



Httpd.conf

```
Listen 192.168.10.1:8888
<VirtualHost 192.168.10.1:8888>
  <Location />
    Order deny,allow
    Deny from all
    Allow from 192.168.10.
  </Location>
  KeepAliveTimeout 60
  MaxKeepAliveRequests 0
  ServerAdvertise On
</VirtualHost>
```



Gestion via CLI

CLI permet d'administrer et de récupérer des information du cluster lors de son exécution

La commande ***list-proxies*** retourne les hôtes (et ports) des proxies connectés

```
[standalone@localhost:9999 subsystem=modcluster] :list-proxies
{
  "outcome" => "success",
  "result" => [
    "CP11-010:8888",
    "CP12-010:8888"
  ]
}
```

La commande ***read-proxies-info*** permet d'avoir des informations détaillées



Exemple

```
[standalone@localhost:9999 subsystem=modcluster] :read-proxies-info
{
  "outcome" => "success",
  "result" => [
    "CP11-010:8888",
    "Node: [1],Name: de6973fe-b63d-31dc-a806-04ec16870cfa,Balancer:mycluster,LBGroup: ,Host:
      192.168.10.1,Port: 8080,Type:http,Flushpackets: Off,Flushwait:
      10,Ping: 10,Smax: 65,Ttl: 60,Elected: 0,Read: 0,Transferred: 0,Connected:0,Load: 1
      Vhost: [1:1:2], Alias: localhost
      Vhost: [1:1:3], Alias: example.com
      Context: [1:1:1], Context: /, Status: ENABLED",
    "CP12-010:8888",
    "Node: [1],Name: re5673ge-c83d-25dv-y104-02rt16456cfa,Balancer:mycluster,LBGroup: ,Host:
      192.168.10.2,Port: 8080,Type:http,Flushpackets: Off,Flushwait:
      10,Ping: 10,Smax: 65,Ttl: 60,Elected: 0,Read: 0,Transferred: 0,Connected:0,Load: 1
      Vhost: [1:1:2], Alias: localhost
      Vhost: [1:1:3], Alias: example.com
      Context: [1:1:1], Context: /, Status: ENABLED"
  ]
}
```



Autres commandes CLI

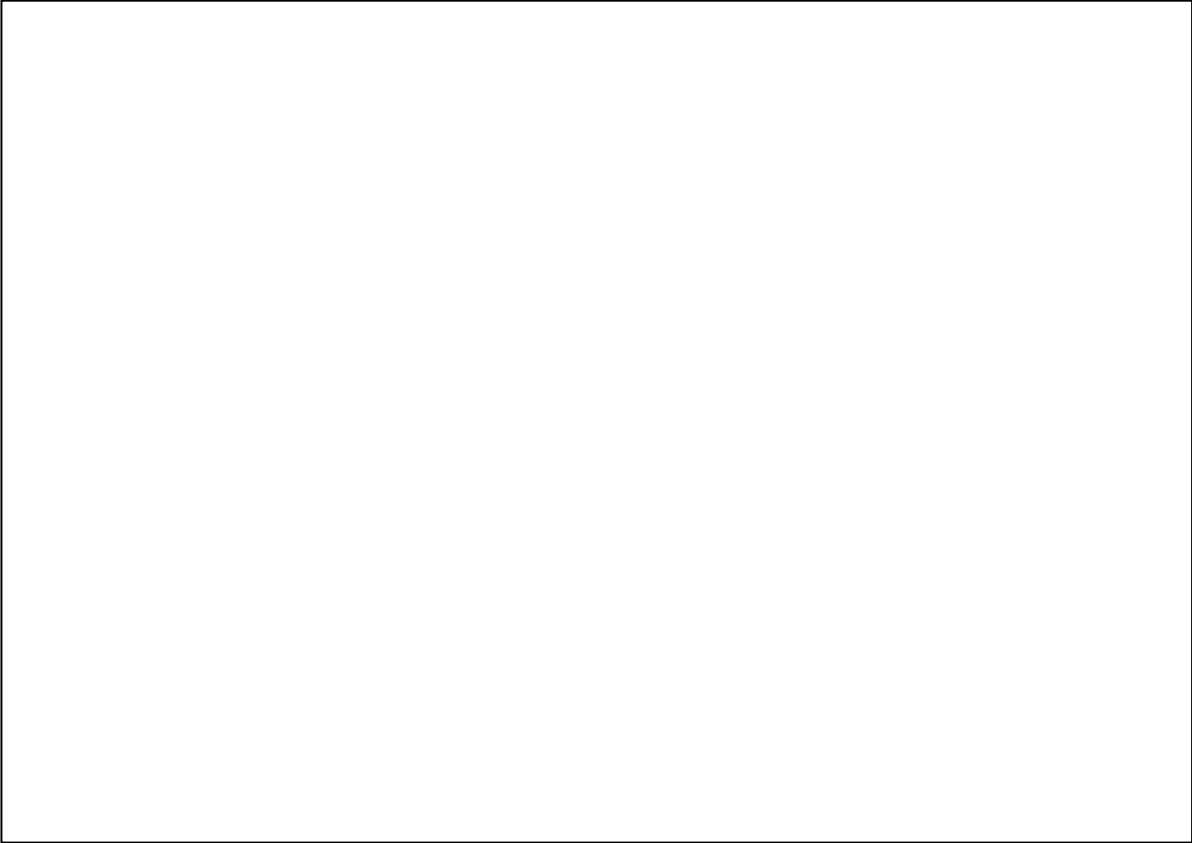
add-proxy permet d'ajouter un proxy non capturé par la configuration httpd.

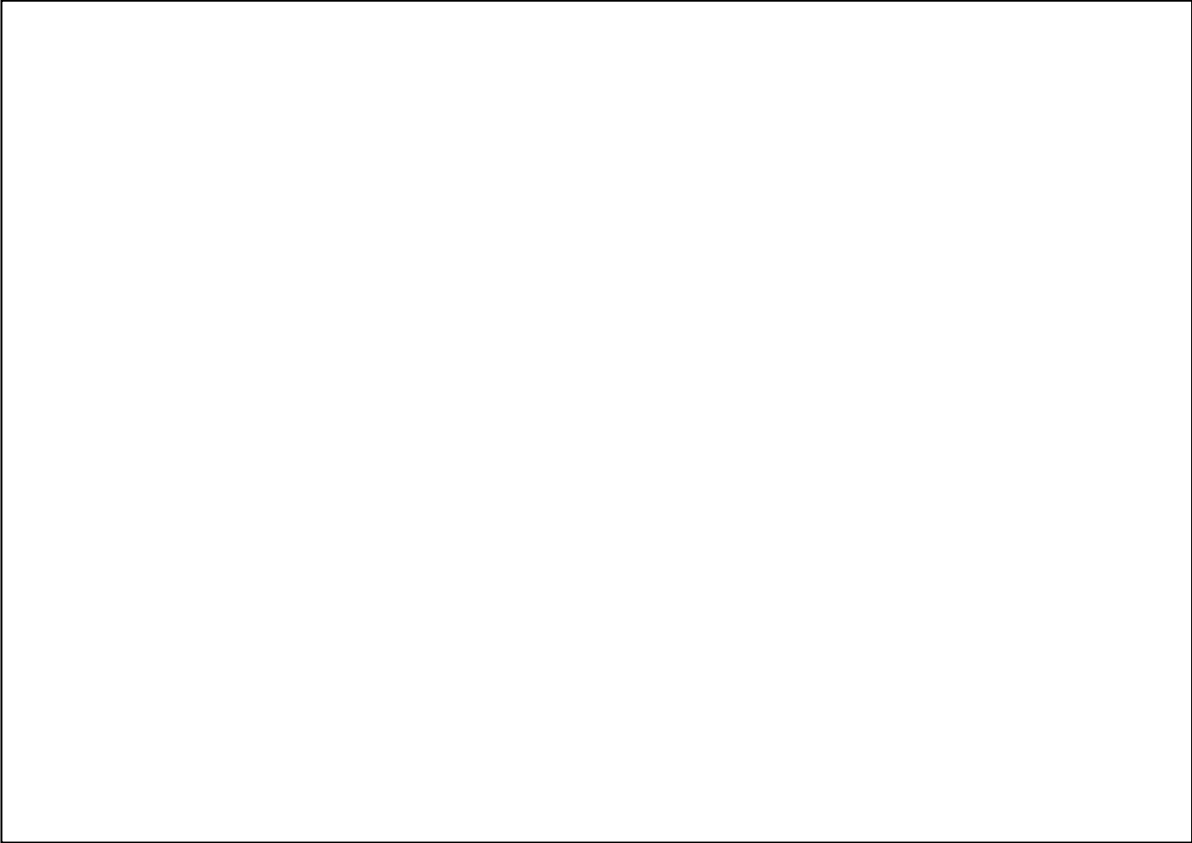
```
[standalone@localhost:9999 subsystem=modcluster]
: add-proxy(host= CP15-022, port=9999)
{"outcome" => "success"}
```

remove-proxy permet d'enlever un proxy :

```
[standalone@localhost:9999 subsystem=modcluster]
: remove-proxy(host=CP15-022, port=9999)
{"outcome" => "success"}
```









Stratégies de cache

Dans Jboss AS 7, le sous-système Infinispan peut configurer plusieurs éléments ***cache-container***

Un *cache-container* contient une ou plusieurs stratégies de cache qui détermine comment les données sont synchronisées.

Les stratégies disponibles sont :

- **Local**: Les données sont stockées seulement sur le nœud local (cache local)
- **Réplication**: Les données sont répliquées sur tous les nœuds
- **Distribution**: Les données sont distribuées sur un sous-ensemble de nœuds
- **Invalidation**: Les données sont stockées seulement dans un *cache store* (Hibernate). Lorsqu'un nœud nécessite une entrée, il la charge à partir du cache. Le cache peut alors être invalidée sur tous les nœuds

Les cache containers sont automatiquement inscrit dans l'annuaire JNDI sous le nom : `java:jboss/infinispan/container-name`



Exemple

```
<subsystem xmlns="urn:jboss:domain:infinispan:1.1" default-cache-  
container="cluster">  
  <cache-container name="web" aliases="standard-session-cache"  
default-cache="repl">  
    <replicated-cache name="repl" mode="ASYNC" batching="true">  
      <file-store/>  
    </replicated-cache>  
    <replicated-cache name="sso" mode="SYNC" batching="true"/>  
    <distributed-cache name="dist" l1-lifespan="0" mode="ASYNC"  
batching="true">  
      <file-store/>  
    </distributed-cache>  
  </cache-container>  
  ...  
</subsystem>
```

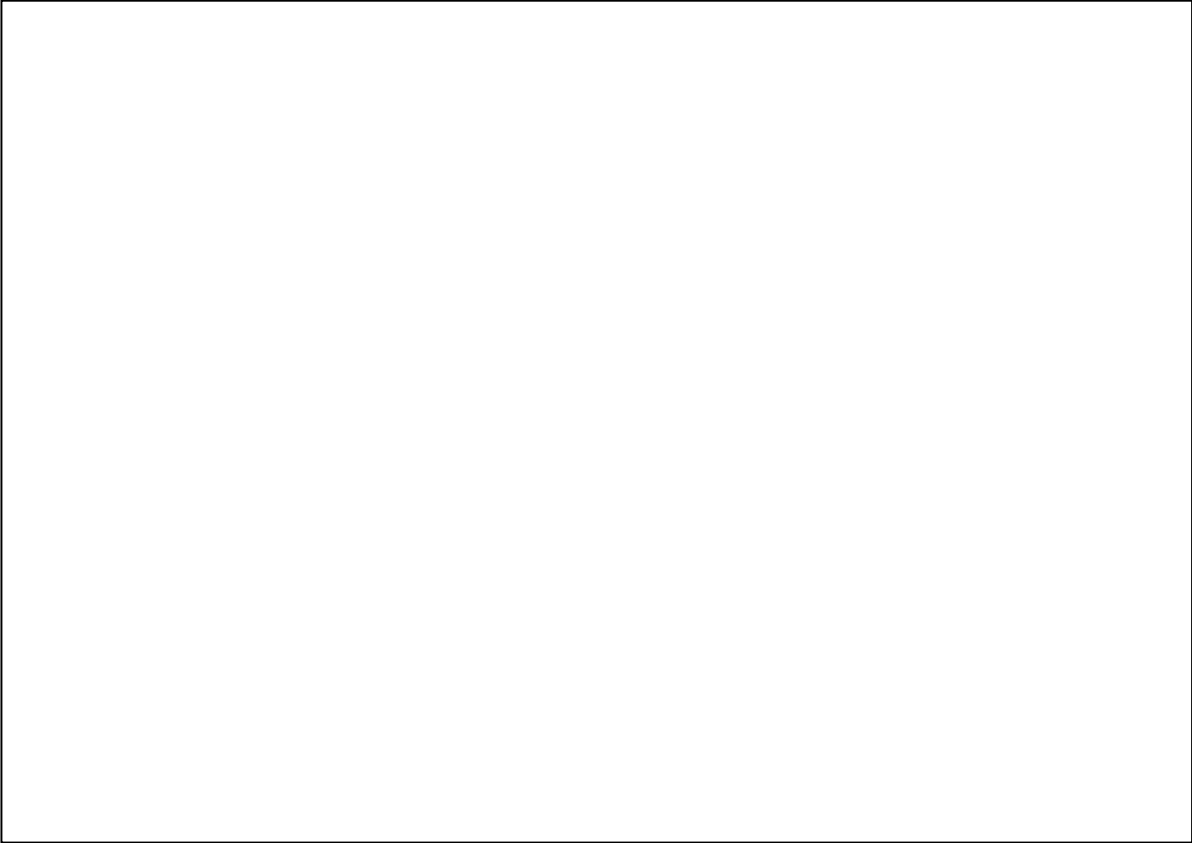


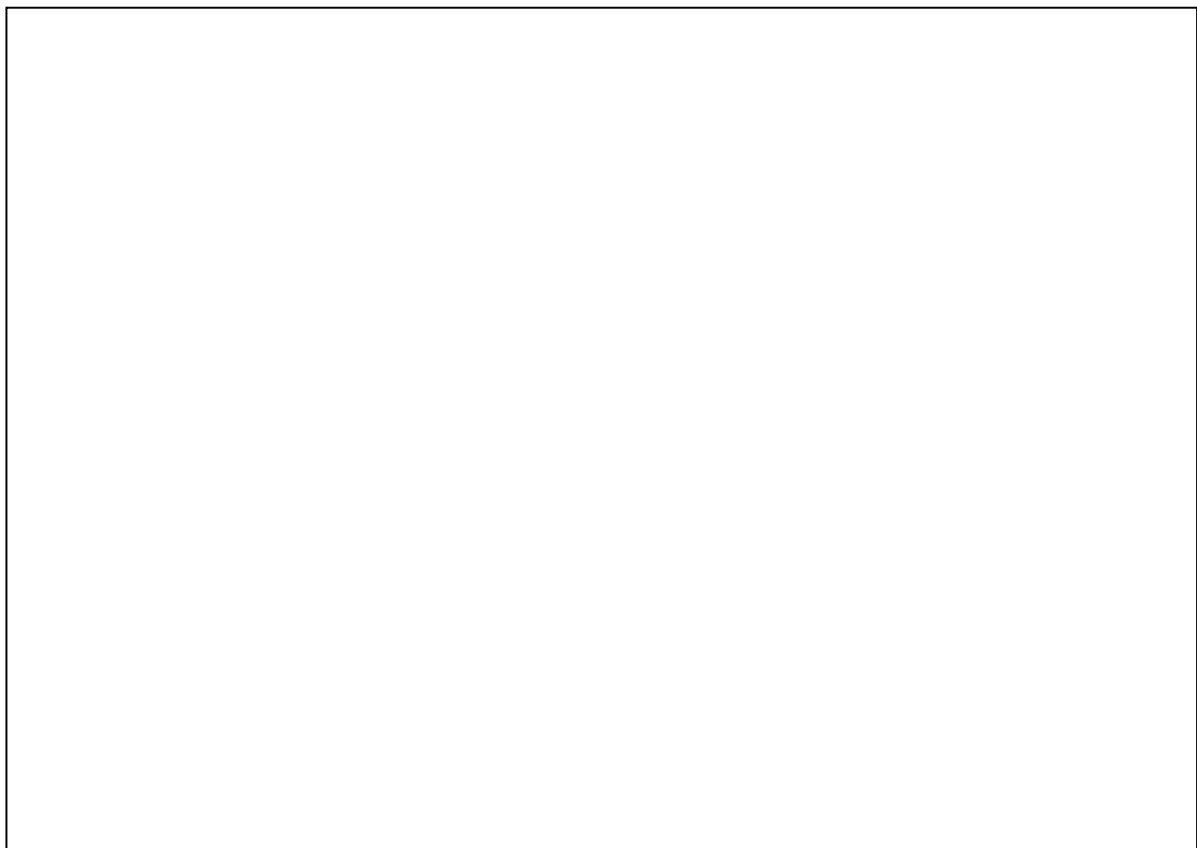
Mode de synchronisation

La synchronisation de données entre les nœuds peut être effectuée par des messages **synchrones** (SYNC) ou **asynchrones** (ASYNC).

- Le mode synchrone est le moins performant puisque chaque nœud doit recevoir un acquittement de chaque membre du cluster. Cependant, le mode synchrone est nécessaire lorsque tous les nœuds doivent accéder aux données cachées de façon cohérente.
- Le mode asynchrone privilégie la rapidité à la cohérence, ce mode est particulièrement avantageux pour la réplication de session HTTP en mode sticky

Le mode de synchronisation n'est effectif que pour les caches distribués sur plusieurs nœuds (modes :réplication, distribution et invalidation)







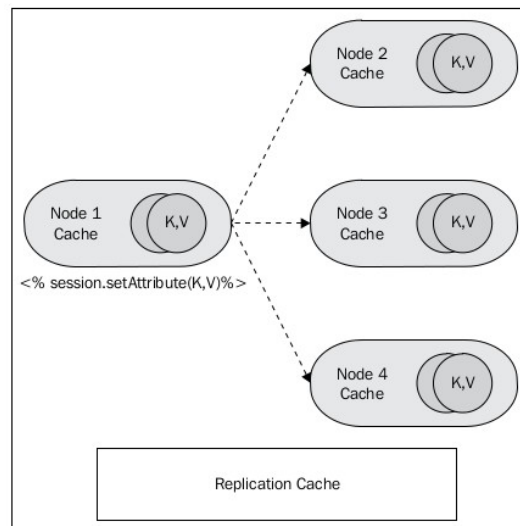
Réplication

Lors de l'utilisation de la réplication, *Infinispan* stocke chaque entrée du cache sur chaque nœud

- La scalabilité de la réplication est alors dépendante de la taille du cluster
- Si $\text{DATA_SIZE} * \text{NUMBER_OF_HOSTS}$ est inférieure à la taille de la mémoire disponible sur chaque hôte, la réplication peut être un choix valable.



Réplication





Distribution

Dans le cas de la distribution, Infinispan stocke chaque entrée du cache sur un sous-ensemble de nœuds permettant une meilleure scalabilité

La distribution utilise un algorithme de hashage pour déterminer où les données doivent être distribuées

L'algorithme est configuré avec le nombre de copies devant être gérées sur le cluster

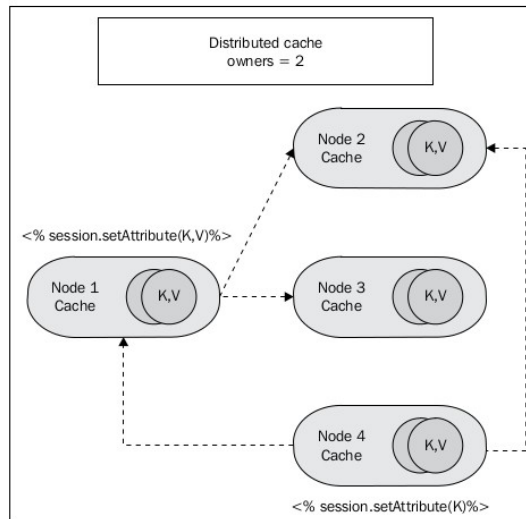
Le nombre de copies représente le compromis entre performance et la durabilité des données

Le paramètre **owners** (par défaut 2) définit le nombre de copies

```
<distributed-cache owners="3" mode="ASYNC" name="dist"
batching="true">
. . . . .
</distributed-cache>
```



Distribution





Threads

Il est possible d'externaliser la configuration des threads d'Infinispan dans un pool de threads

Des pools différents peuvent être configurés pour les différents cache-container

Les pools disponibles sont :

- **transport** : Taille du pool de thread limité dédié au transport de données sur le réseau
- **listener-executor** : Taille du pool de threads utilisé pour enregistré et être notifié des événements concernant le cache
- **replication-queue-executor** : Taille du pool de threads planifiées pour la réplication des données du cache
- **eviction-executor** : Taille du pool de threads planifiées pour périodiquement nettoyer le cache



Gestion des threads

```
<subsystem xmlns="urn:jboss:domain:clustering:infinispan:1.0">
  <cache-container name="cluster" listener-executor="infinispan-listener"
    eviction-executor="infinispan-eviction" replication-queue-executor="infinispan-repl-queue">
    <transport executor="infinispan-transport"/>
    <!-- Caches -->
  </cache-container>
</subsystem>

<subsystem xmlns="urn:jboss:domain:threads:1.0">
  <bounded-queue-thread-pool name="infinispan-listener" blocking="true">
    <max-threads count="1" per-cpu="2"/>
    <queue-length count="100000" per-cpu="200000"/>
  </bounded-queue-thread-pool>
  <bounded-queue-thread-pool name="infinispan-transport" blocking="true">
    <!-- ... --></bounded-queue-thread-pool>
  <scheduled-thread-pool name="infinispan-eviction">
    <!-- ... --></scheduled-thread-pool>
  <scheduled-thread-pool name="infinispan-repl-queue">
    <!-- ... --></scheduled-thread-pool>
</subsystem>
```



Transport

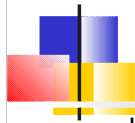
Par défaut, les caches d'Infinispan utilise la pile par défaut de JGroups : *default-stack*

Il est cependant possible de choisir une couche transport différente pour chaque *cache-container* :

```
<cache-container name="web" default-cache="repl">  
  <transport stack="tcp"/>  
</cache-container>
```

La configuration par défaut UDP est en général adaptée au grands clusters ou lorsque l'on utilise la réplication ou l'invalidation comme il ouvre beaucoup moins de sockets

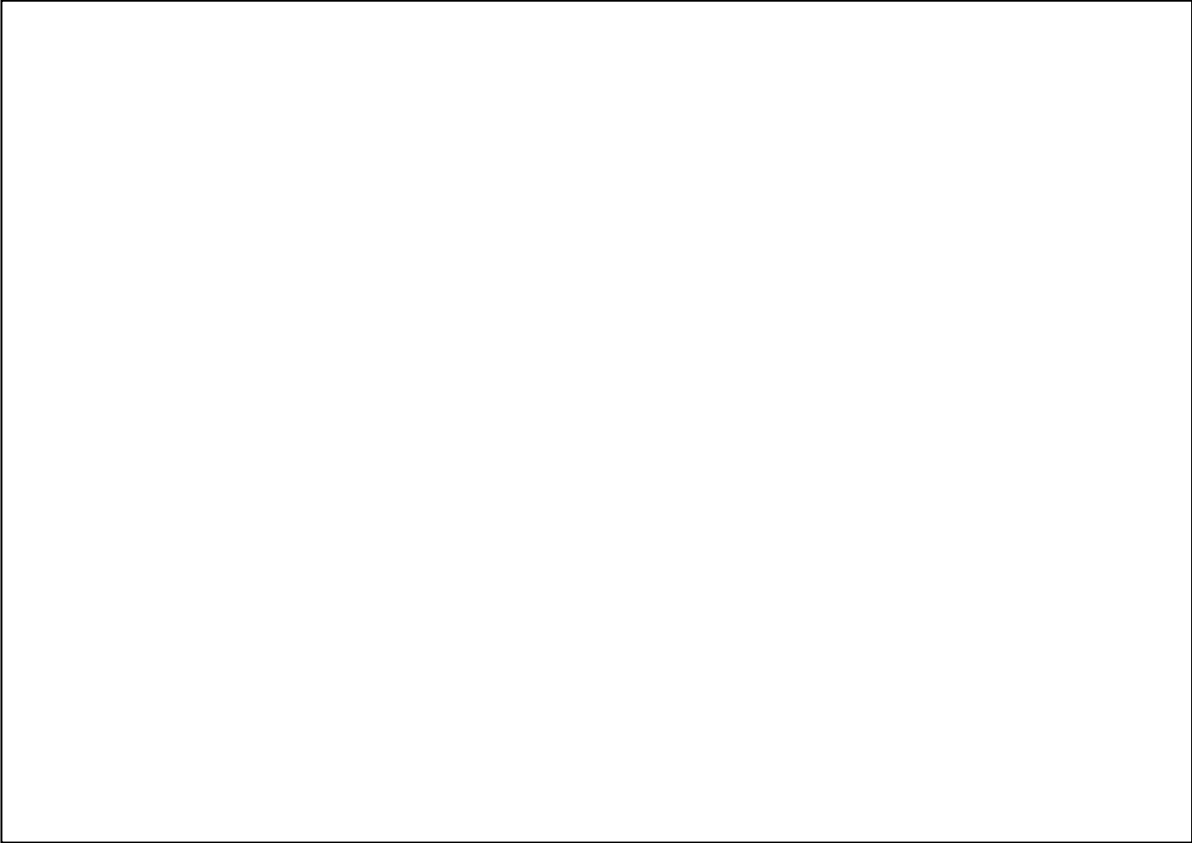
La pile TCP est plus adapté aux petits clusters ou à la distribution



Utilisation directe d'Infinispan

Il est possible de se faire injecter le cache Infinispan dans ses classes Java

```
@ManagedBean
public class MyBean<K, V> {
    @Resource(lookup = "java:jboss/infinispan/mycontainer")
    private org.infinispan.manager.CacheContainer container;
    private org.infinispan.Cache<K, V> cache;
    @PostConstruct
    public void start() {
        this.cache = this.container.getCache();
    }
    // Use cache
}
```





Introduction

Par défaut, le réplication de session utilise le cache nommé **web**

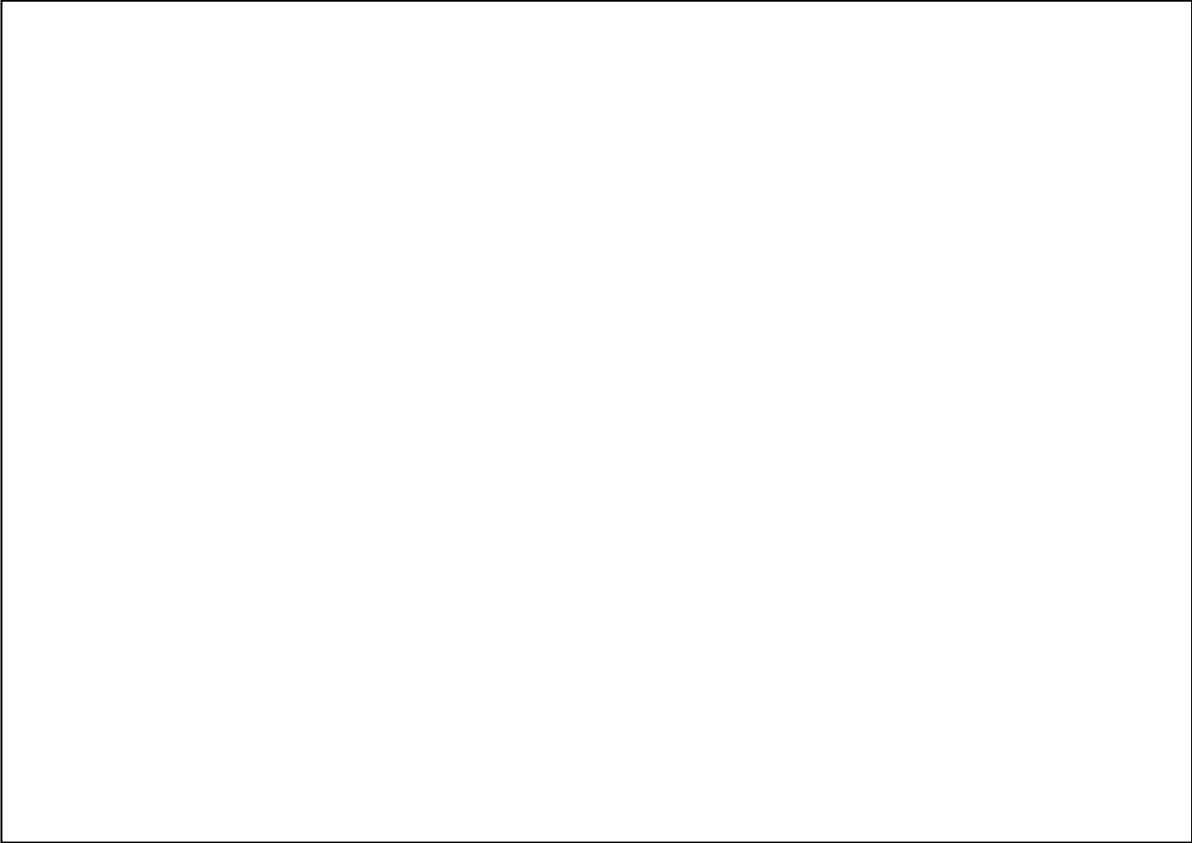
Il est possible de dédier un cache à une application via *jboss-web.xml*

```
<jboss-web>
  <replication-config>
    <cache-name>web.dist</cache-name>
  </replication-config>
</jboss-web>
```




Cache web

```
<cache-container name="web" default-cache="repl">
  <alias>standard-session-cache</alias>
  <replicated-cache mode="ASYNC" name="repl" batching="true">
    <locking isolation="REPEATABLE_READ"/>
    <file-store/>
  </replicated-cache>
  <distributed-cache mode="ASYNC" name="dist" batching="true">
    <locking isolation="REPEATABLE_READ"/>
    <file-store/>
  </distributed-cache>
</cache-container>
```





Interprétation du nom

Détermination du cache à partir du nom spécifié dans le descripteur de déploiement

1. Interprété comme étant le nom du cache

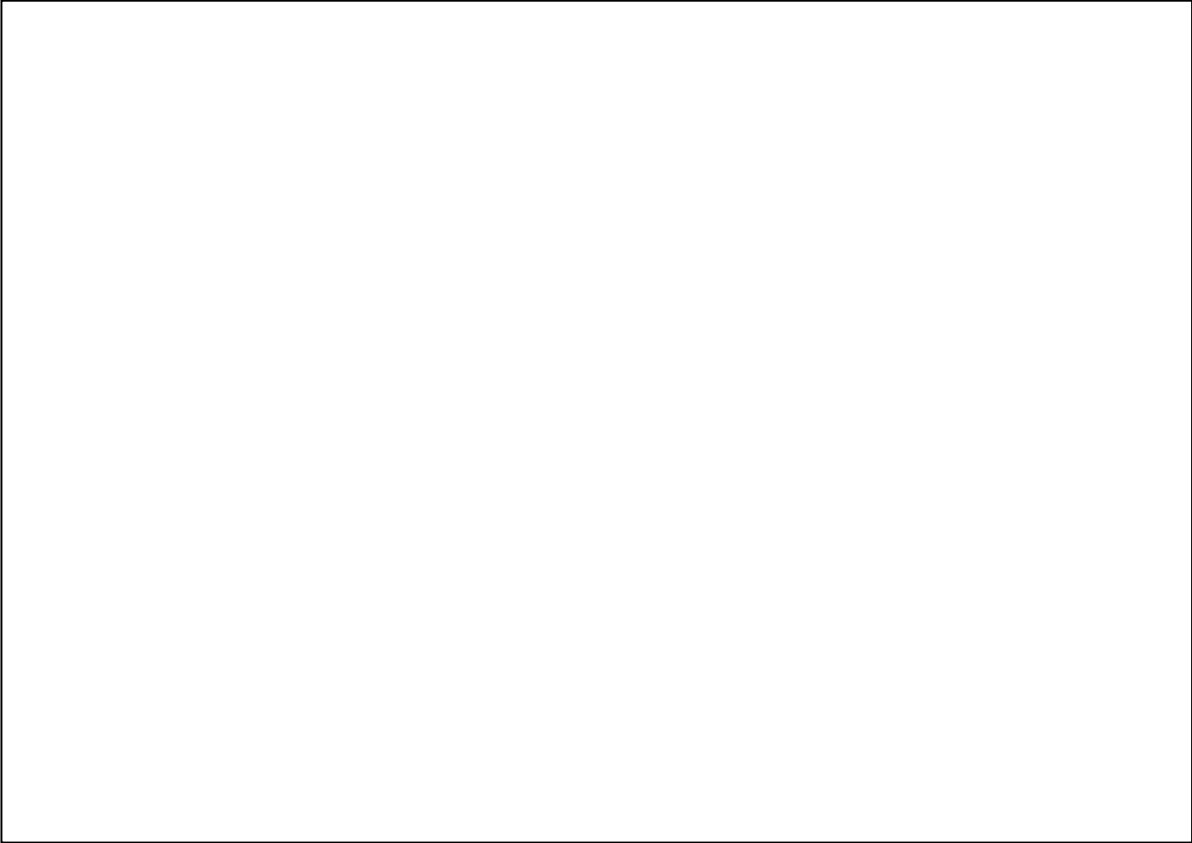
exemple : `jboss.infinispan.web.dist`

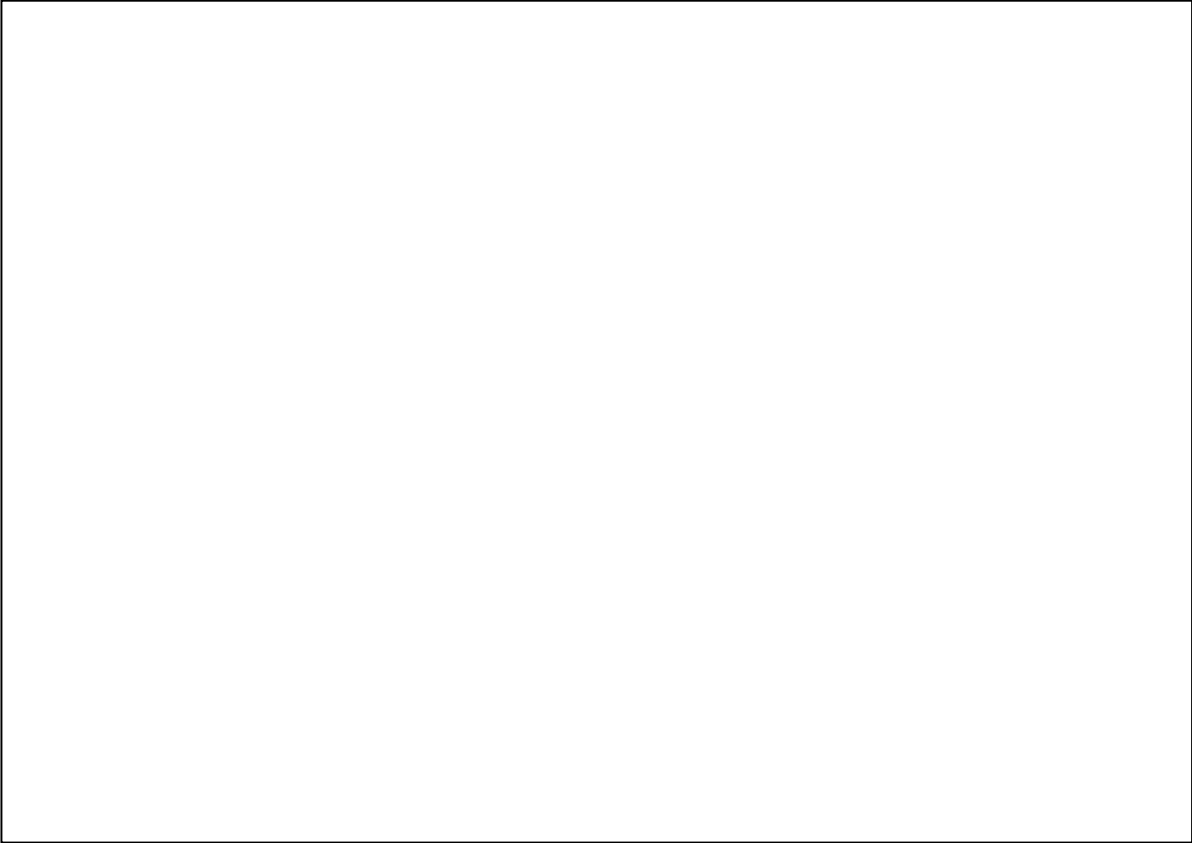
2. Interprété comme nom du container, le cache par défaut est alors utilisé

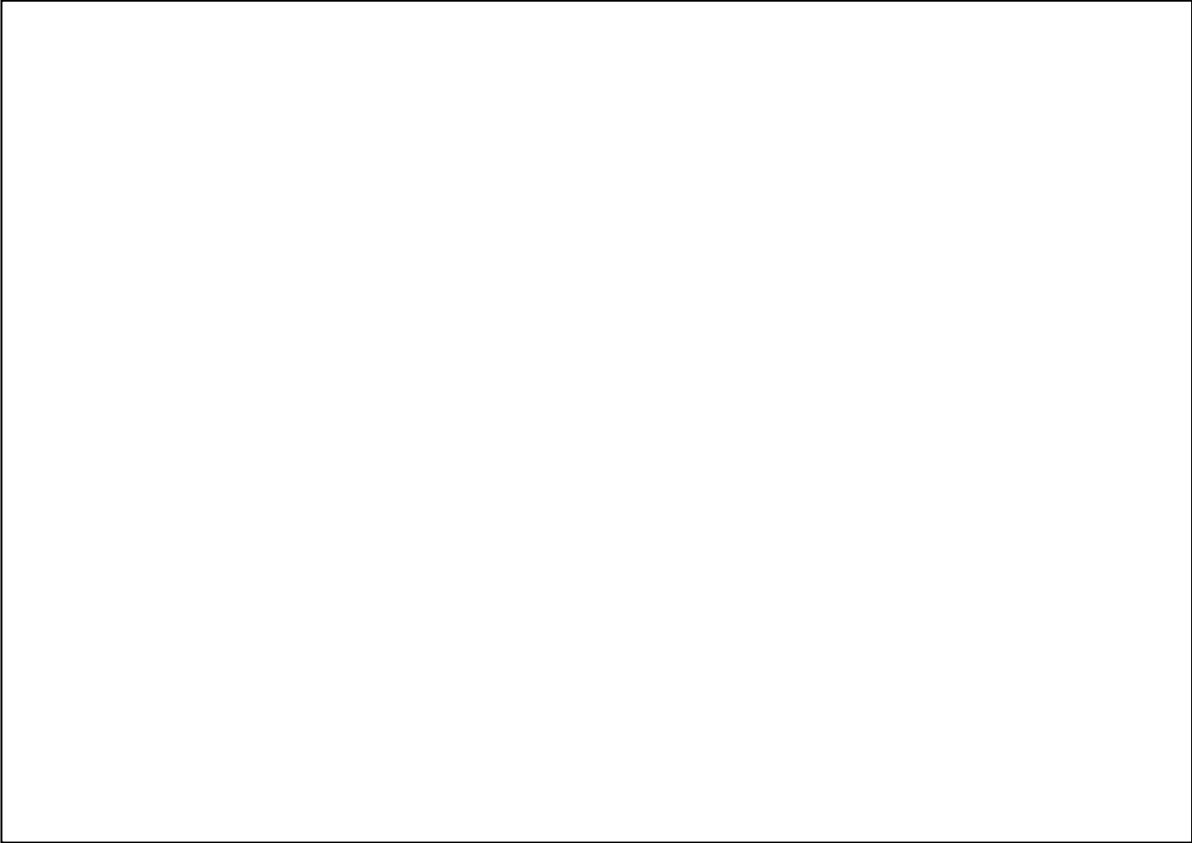
exemple : `jboss.infinispan.web`

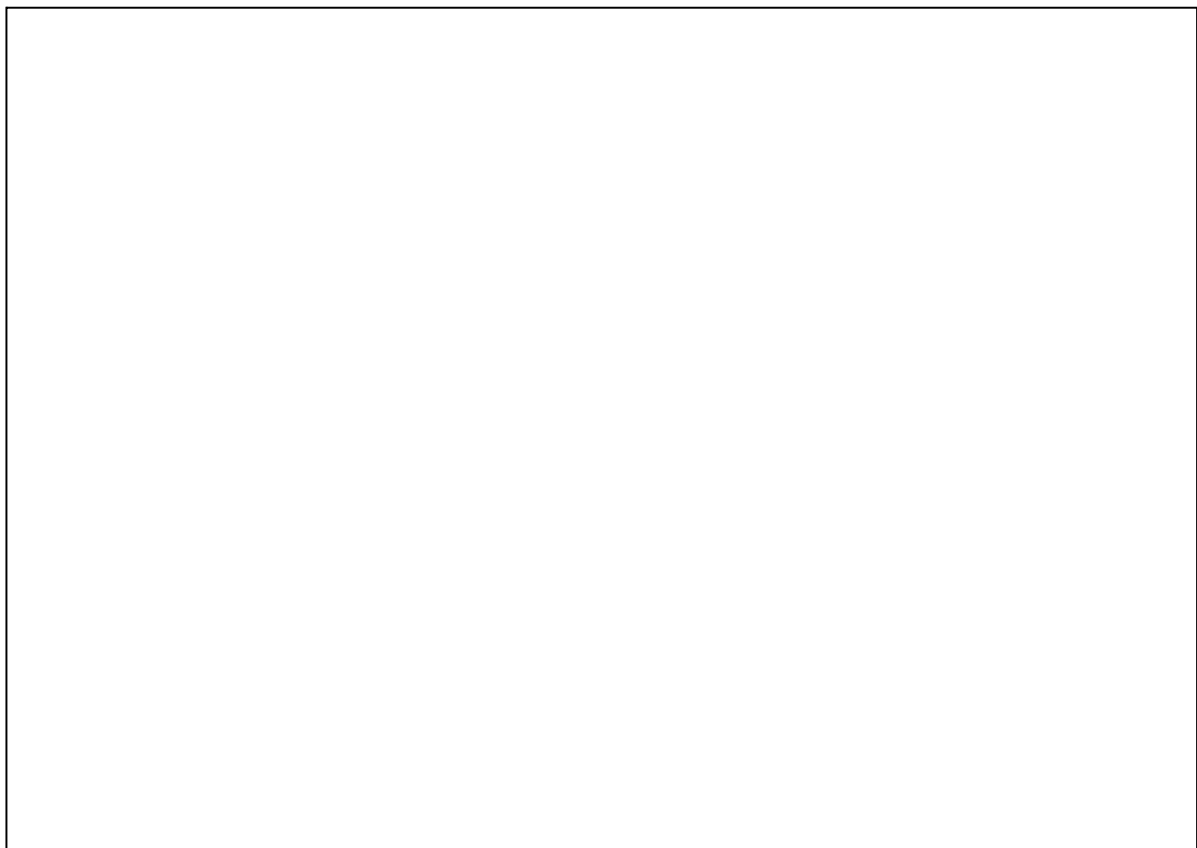
3. Assume que le nom de base est *jboss.infinispan*

exemple : `web, web.dist`







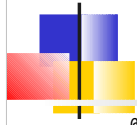




Algorithme de répartition

Plusieurs choix sont possibles pour l'algorithme de répartition :

- **Round robin** : par défaut
- **RandomRobin** : Au hasard
- **FirstAvailable** : Au hasard, puis sticky pour un proxy donné
- **FirstAvailableIdenticalAllProxies** : Au hasard, puis sticky pour tous les proxies du même EJB

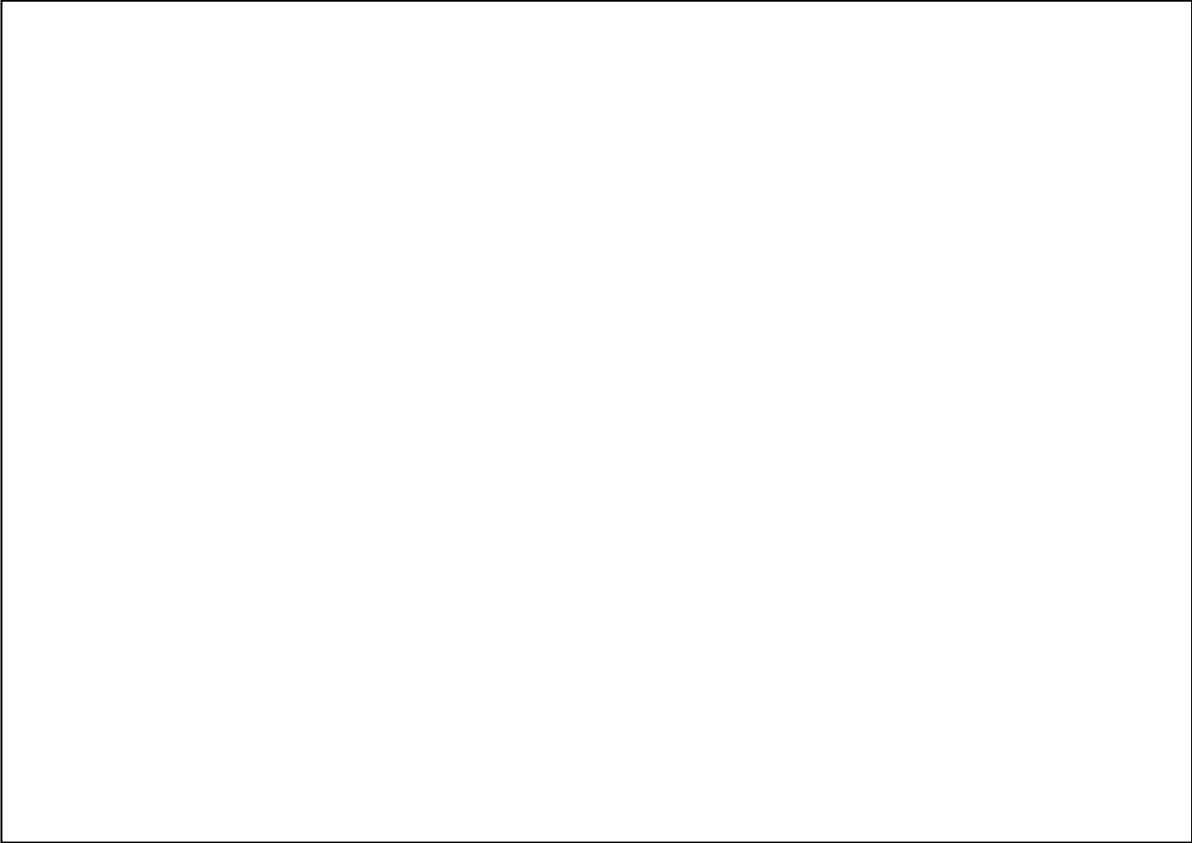


Exemple EJB 3.0

```
@Stateless
@Clustered(loadBalancePolicy=RoundRobin.class)
public class CounterBean implements Counter {

    /**
     * Default constructor.
     */
    public CounterBean() {
    }

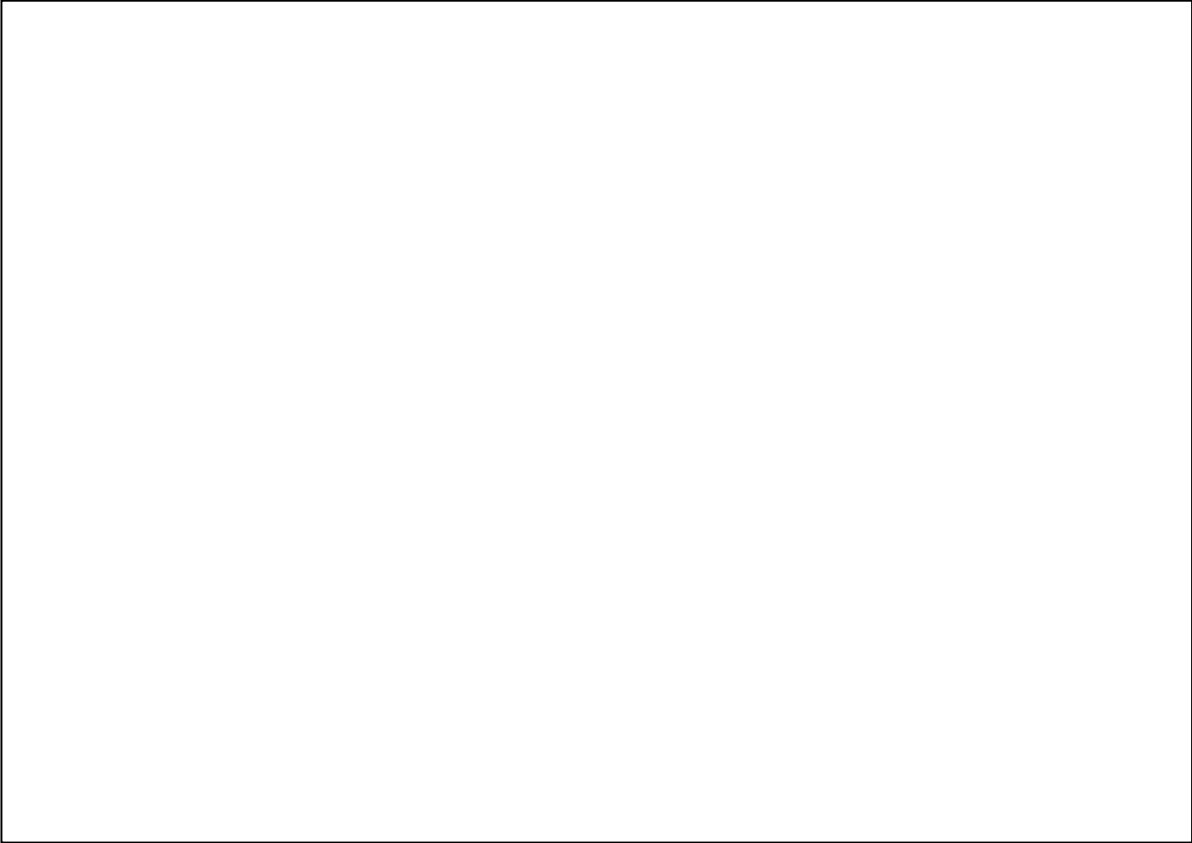
    public void printCount(int messageNumber) {
        System.out.println(messageNumber) ;
    }
}
```





sfsb

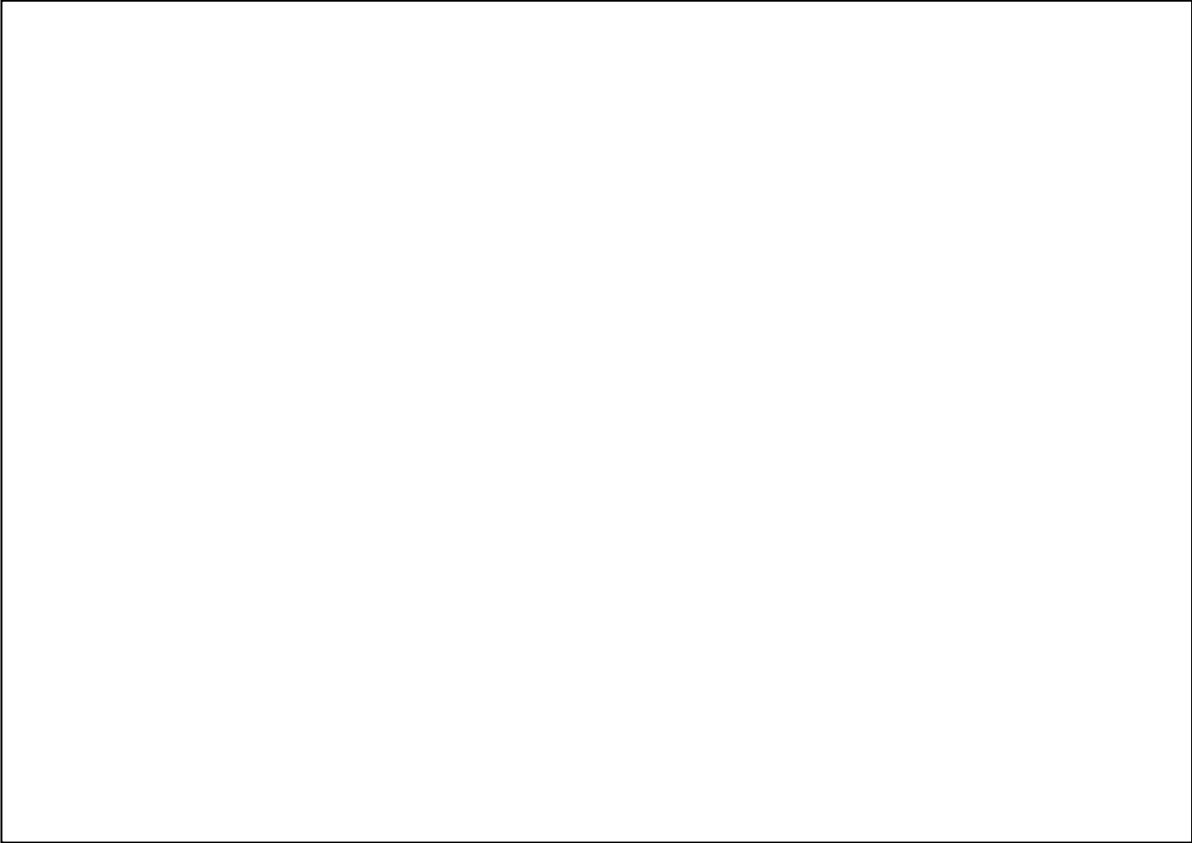
```
<cache-container name="sfsb" default-cache="repl">
  <alias>sfsb-cache</alias>
  <alias>jboss.cache:service=EJB3SFSBClusteredCache</alias>
  <replicated-cache mode="ASYNC" name="repl" batching="true">
    <locking isolation="REPEATABLE_READ"/>
    <eviction strategy="LRU" max-entries="10000"/>
    <file-store/>
  </replicated-cache>
</cache-container>
```

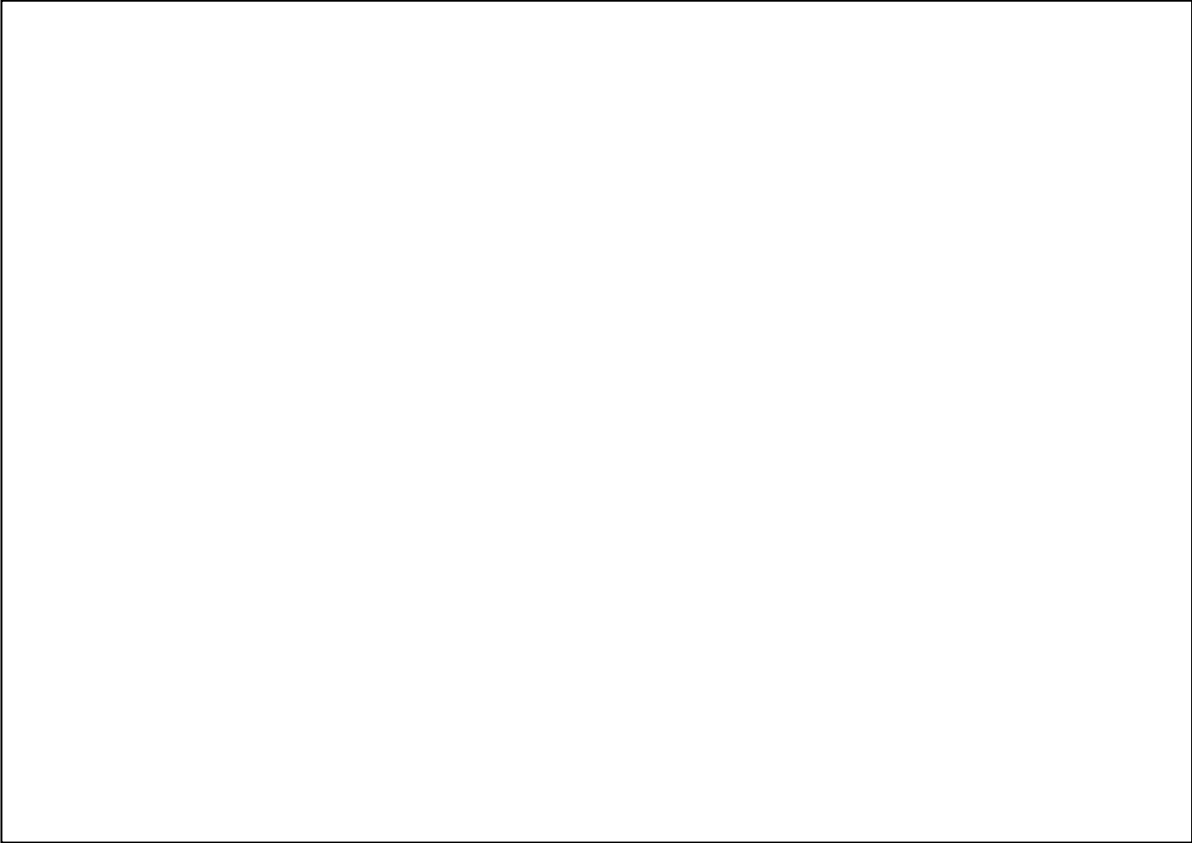


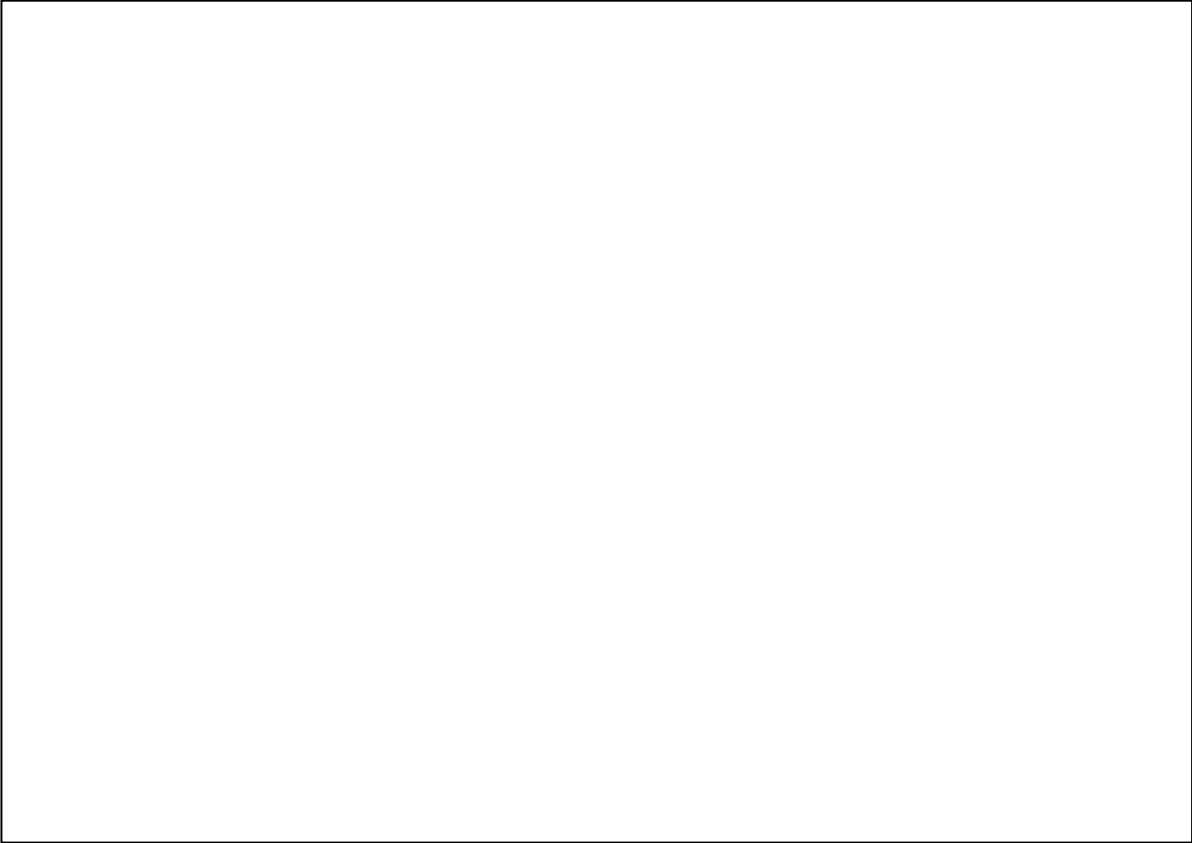


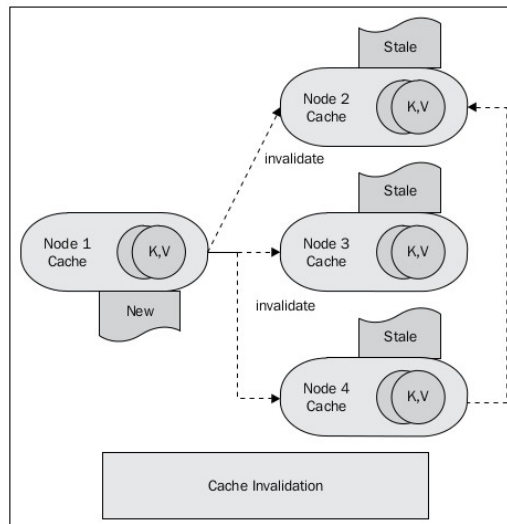
Exemple EJB3 session stateful

```
@Stateful  
@Clustered  
@CacheConfig(maxSize=5000, removalTimeoutSeconds=18000)  
public class MyBean implements MySessionInt {  
    private int state = 0;  
  
    public void increment() {  
        System.out.println("counter: " + (state++));  
    }  
}
```











Configuration

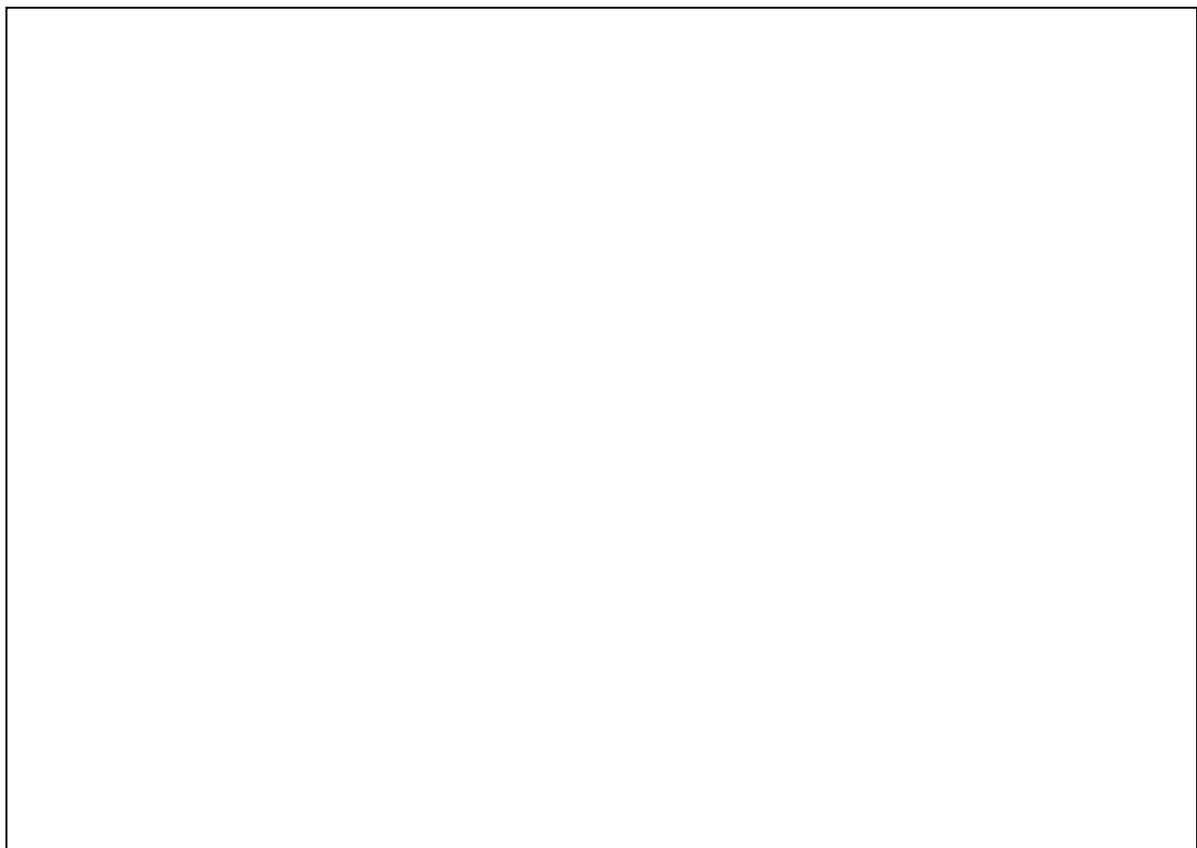
Le profil ha définit donc le cache container hibernate qui définit 3 caches :

- Le cache **local** : le cache de second niveau hibernate utilisé par chaque noeud
- Le cache **d'invalidation** permettant l'invalidation du cache local
- Le cache **timestamp** qui stocke un timestamp de la dernière modification pour chaque table. Ce



Configuration

```
<cache-container name="hibernate" default-cache="local-query">
  <local-cache name="local-query">
    <eviction strategy="LRU" max-entries="10000"/>
    <expiration max-idle="100000"/>
  </local-cache>
  <invalidation-cache mode="SYNC" name="entity">
    <eviction strategy="LRU" max-entries="10000"/>
    <expiration max-idle="100000"/>
  </invalidation-cache>
  <replicated-cache name="timestamps" mode="ASYNC">
    <transaction mode="NONE"/>
    <eviction strategy="NONE"/>
  </replicated-cache>
</cache-container>
```



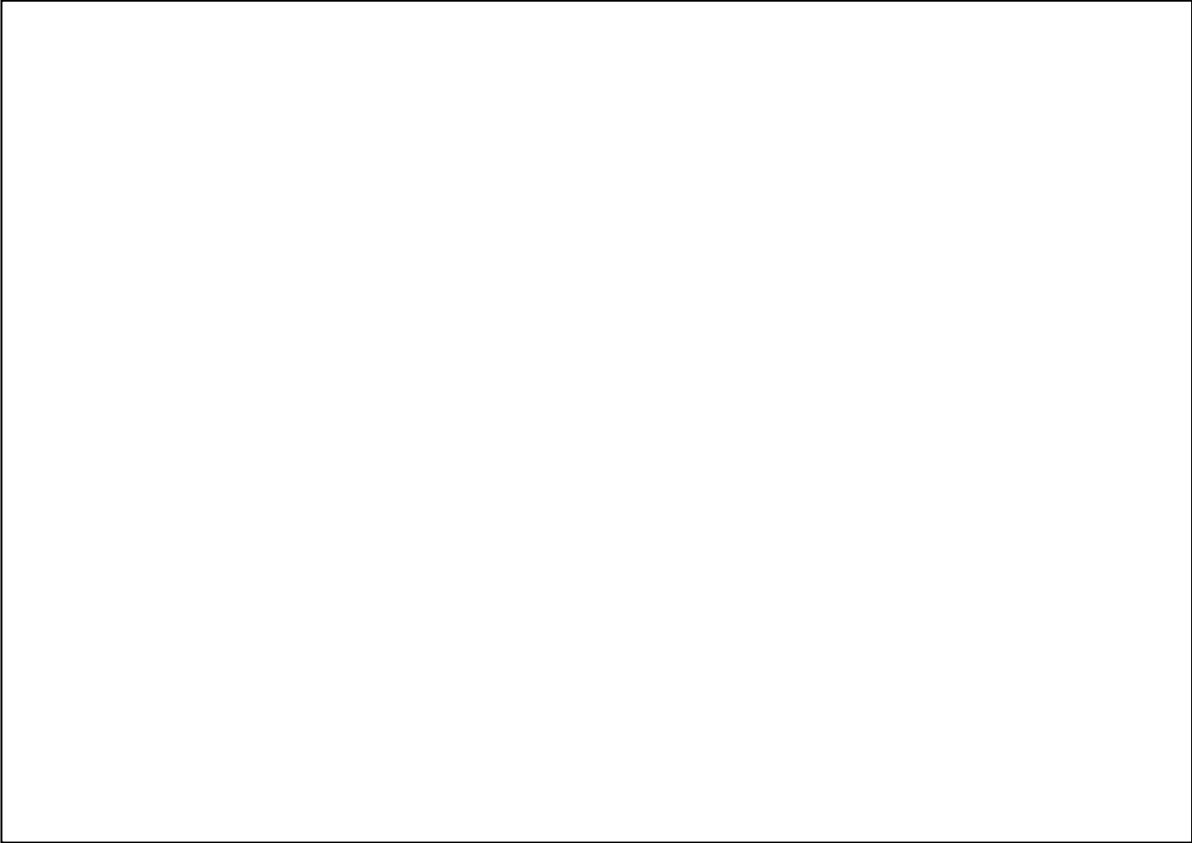


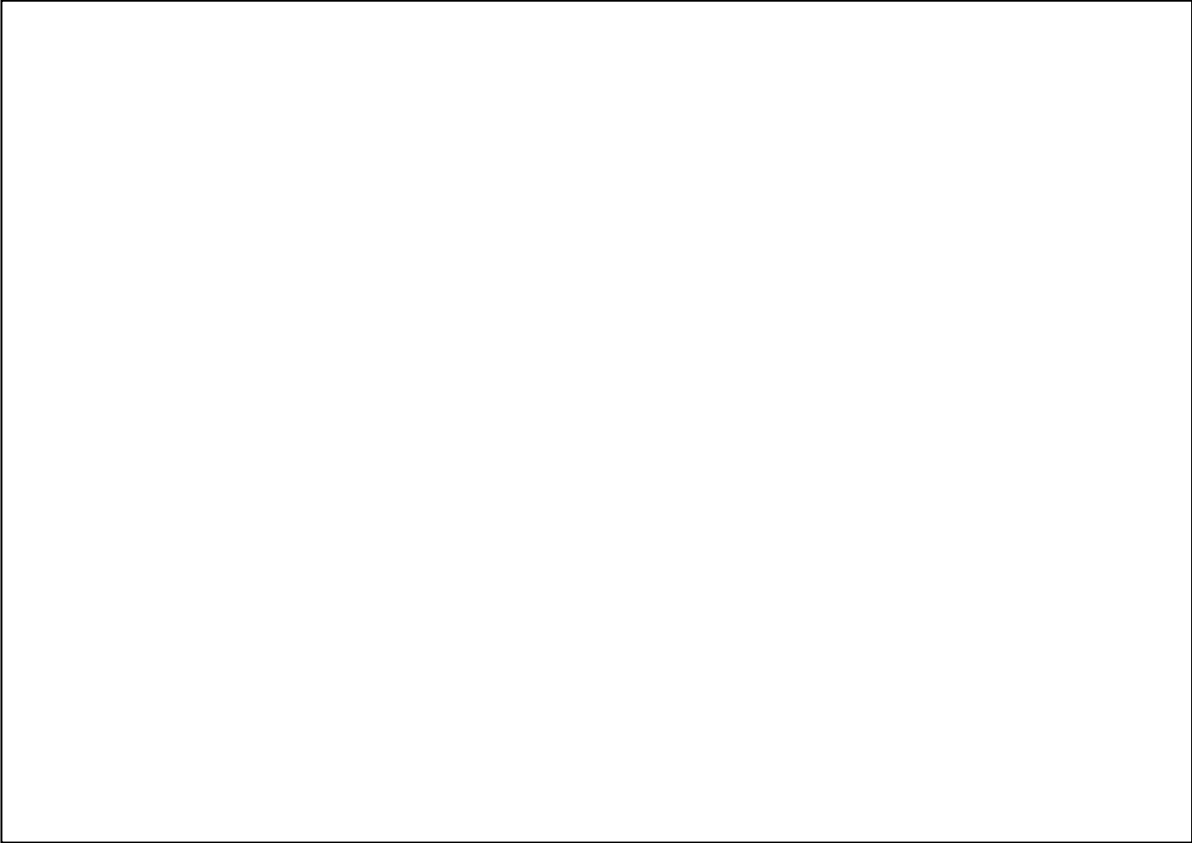
persistence.xml

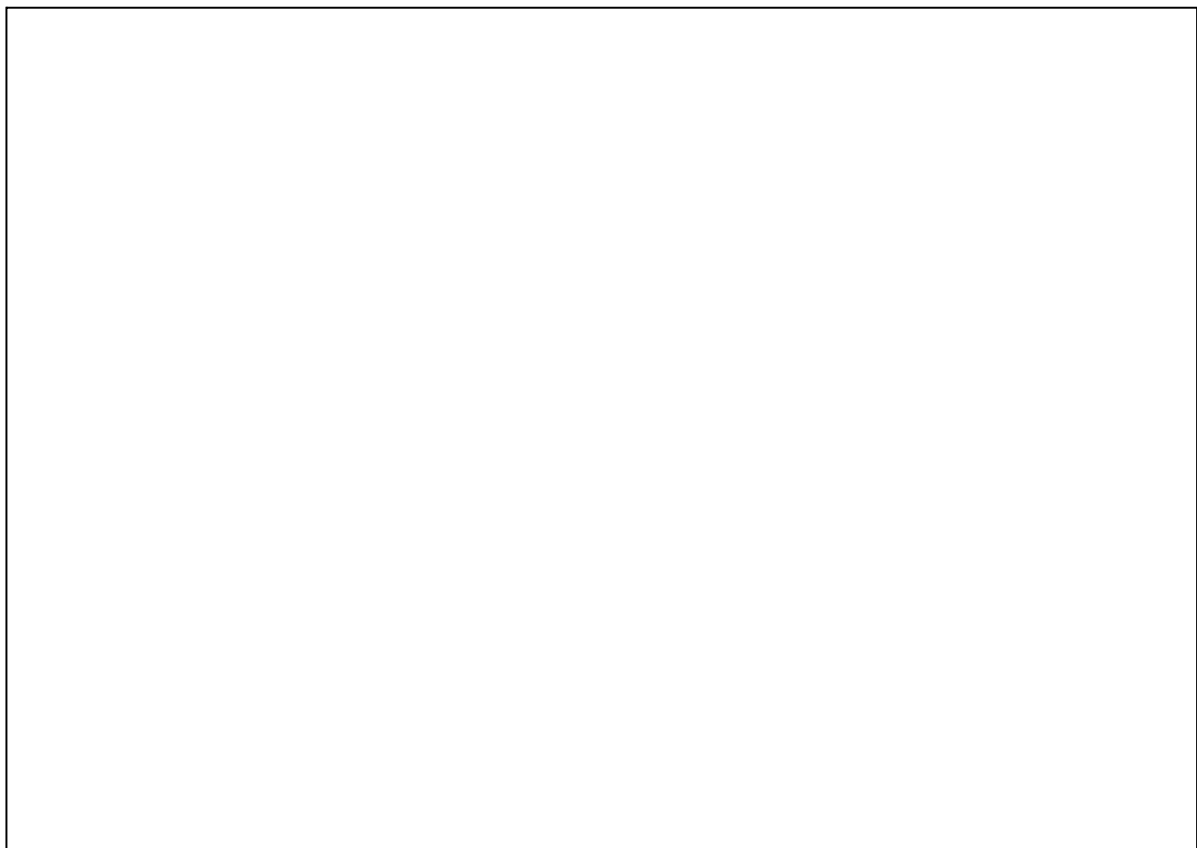
```
<!-- Autoriser le cache -->
<property name="hibernate.cache.use_second_level_cache" value="true"/>
<property name="hibernate.cache.use_query_cache" value="true"/>

<!-- Surcharger le container par défaut -->
<property name="hibernate.cache.infinispan.cachemanager"
value="java:jboss/infinispan/mycontainer"/>

<!-- Surcharger les régions de cache -->
<property name="hibernate.cache.infinispan.entity.cfg" value="entity"/>
<property name="hibernate.cache.infinispan.collection.cfg" value="entity"/>
<property name="hibernate.cache.infinispan.query.cfg" value="local-query"/>
<property name="hibernate.cache.infinispan.timestamp.cfg" value="timestamp"/>
```





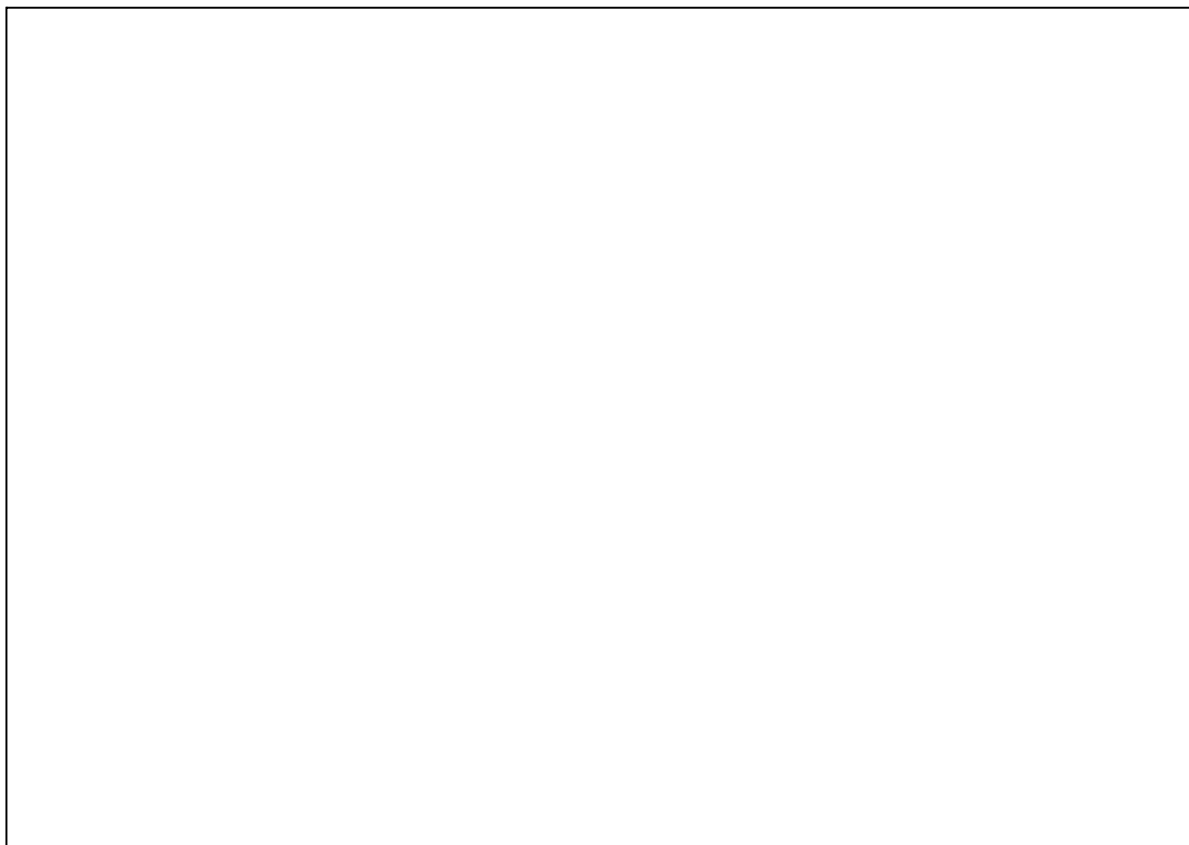


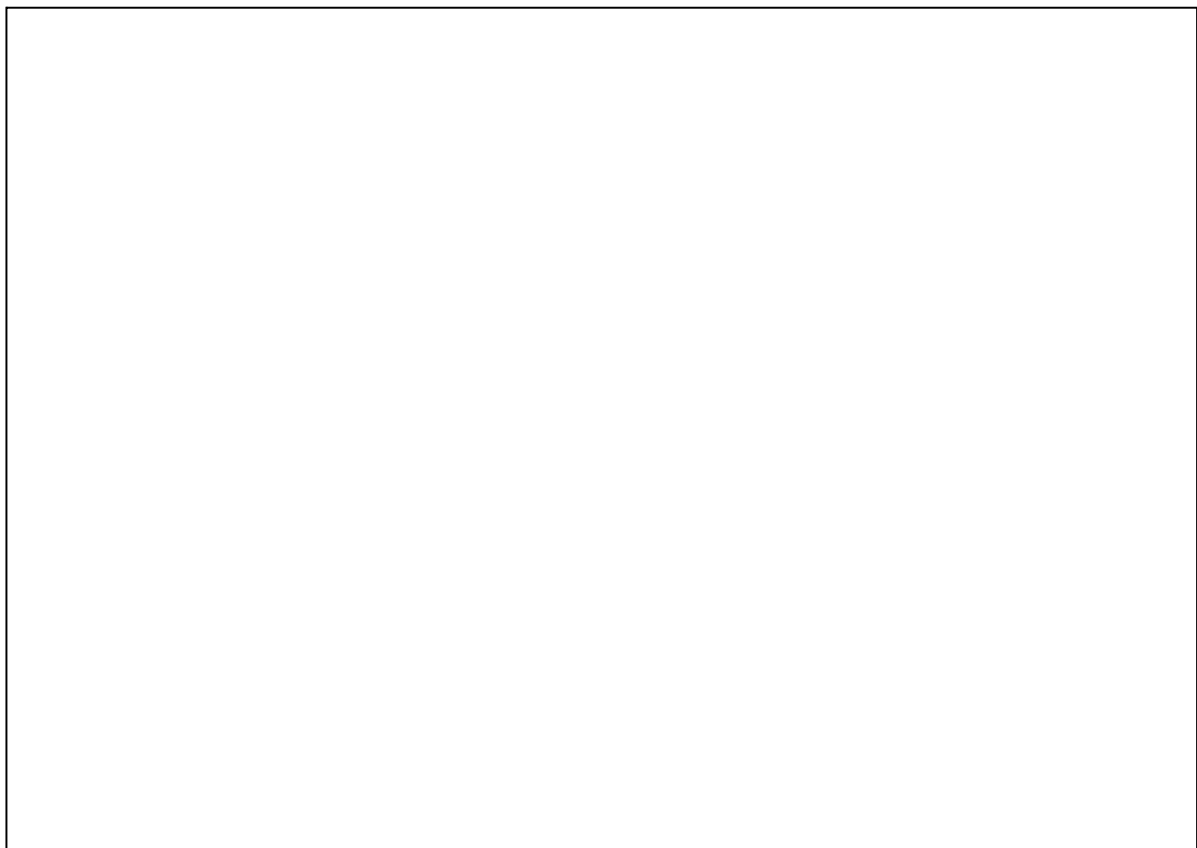


Configuration pour Infinispan en mode cluster

Ces attributs permettent de donner l'implémentation du gestionnaire de cache via JNDI :

```
<property name="hibernate.cache.region.factory_class"
  value="org.jboss.as.jpa.hibernate4.infinispan.InfinispanRegionFactory"/>
<property name="hibernate.cache.infinispan.cachemanager"
  value="java:jboss/infinispan/container/hibernate"/>
<property name="hibernate.transaction.manager_lookup_class"
  value="org.hibernate.transaction.JBossTransactionManagerLookup"/>
<property name="hibernate.cache.use_second_level_cache" value="true"/>
```





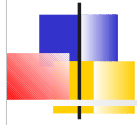


Annotations @NamedQuery

```
@Entity
@Cache(usage = CacheConcurrencyStrategy.TRANSACTIONAL,
      region = "Account")
@NamedQueries(
{
  @NamedQuery(
    name = "account.bybranch",
    query = "select acct from Account as acct where acct.branch = ?1",
    hints = { @QueryHint(name = "org.hibernate.cacheable", value =
      "true") }
  )
})
public class Account implements Serializable
{
  // ... ...
}
```



TP



Clustering HornetQ



Introduction

Le clustering avec **HornetQ** permet de définir des groupes de serveurs HornetQ afin de partager la charge de traitement des messages.

Chaque nœud actif du cluster est un serveur HornetQ gérant ses propres messages et ses propres connexions

Les serveurs doivent être configurés en cluster en positionnant l'élément **clustered** dans la configuration et en définissant des connexions « clusterisées »

Ce type de connexion permet de répartir la charge entre les nœuds



Capacités de clustering

L'architecture de clustering peut suivre différentes topologies qui offre différentes capacités :

- Répartition du traitement des messages
- Redistribution des messages
- Répartition des connexions clientes
- Découverte automatique des nœuds
- Haute disponibilité et fail-over



Découverte des nœuds

La découverte automatique est un mécanisme basé sur UDP qui permet de propager les détails de connexions au

- **Clients** : Un client peut alors se connecter aux serveurs du cluster sans connaître précisément les serveurs actifs
- **Aux autres serveurs** : Les serveurs peuvent créer leurs connexions cluster sans connaître à priori sa constitution

La topologie du cluster est envoyée au client par des connexions normale et aux serveurs via les connexions clusterisées. La première connexion est alors établie soit en utilisant UDP soit en fournissant la liste des connecteurs initiaux



Répartition de charge serveur

Si des connexions clusterisées sont définies entre les nœuds, *HornetQ* répartit la charge des messages arrivant sur un nœud spécifique.

- Par exemple, si le cluster est formé des nœuds A et B et qu'un client envoie des messages sur le nœud A. Les messages seront réparties (Round-robin par défaut) sur A et B



Répartition des connexions clientes

Avec cette fonctionnalité, les différentes session créées avec la même usine à session peuvent être connectées à différents nœuds du cluster.

La stratégie de répartition est configurable (Round-robin, Random ou custom)



Redistribution de messages

La redistribution de messages permet de redistribuer les messages d'une file pour lesquels il n'y a pas de consommateur disponible sur le nœud vers des nœuds qui ont de tels consommateurs.

La redistribution peut être configurée pour se déclencher dès que le dernier consommateur d'une file est fermé ou après un certain délai

Par défaut, la redistribution n'est pas activée



Topologies de cluster

Il y a 2 topologies principales de cluster :

- Cluster **symétrique** : Chaque nœud est connecté à tous les autres (JBoss). Chaque nœud connaît toutes les files existantes sur les autres nœuds ainsi que leurs consommateurs. Avec cette connaissance, la charge peut être répartie
- Cluster en **chaîne** : Les nœuds forment une chaîne et chaque nœud connaît le nœud précédent et le nœud suivant



HA et *failover*

HornetQ permet :

- la haute-disponibilité : Le système continue à fonctionner même si un ou plusieurs serveurs tombent en panne
- Le failover : en cas de défaillance d'un serveur, le client peut continuer à fonctionner en migrant sur un autre serveur



Serveur de backup

HornetQ permet de lier des serveurs actifs à des serveurs de backup

- Chaque serveur actif peut avoir un ou plusieurs backup
- Un serveur de backup est détenu par un seul serveur actif

Les serveurs de backup ne sont pas opérationnels tant que le serveur actif fonctionne. (Redondance passive)

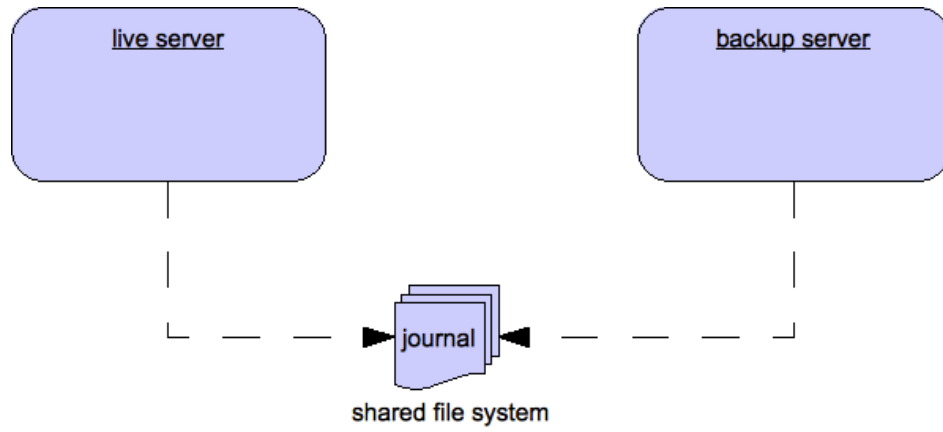
Lorsque le serveur actif défaille, le serveur passif devient actif et un nouveau serveur de backup devient passif

Si le serveur actif redémarre il redevient le serveur primaire

Ce mode de backup est implémenté via un support de persistance partagé



Actif/Passif





Fail-over

HornetQ propose deux modes de failover :

- Failover **automatique** : Les clients sont configurés afin d'avoir connaissance des serveurs actifs et des serveurs de backup. Il n'est alors pas nécessaire de coder la logique de reconnexion lors d'une défaillance
- Failover **applicatif** : La logique de reconnexion est codée dans les clients

HornetQ fournit également le ré-attachement automatique et transparent au même serveur en cas de problèmes temporaire réseau



Notification d'erreur via JMS

JMS fournit un mécanisme standard afin d'être notifié de façon asynchrone d'une perte de connexion :

java.jms.ExceptionListener

Un *ExceptionListener* est appelé par HornetQ lors d'une perte de connexion même si le mécanisme de *failover* a été effectif. Il faut alors inspecter le code d'erreur de l'exception JMS qui peut prendre 2 valeurs : FAILOVER ou DISCONNECT



Configuration JBoss

La configuration JBoss s'effectue dans le sous-système ***messaging***.

L'élément ***clustered*** doit être positionné à *true*

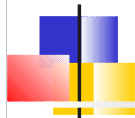
```
<subsystem xmlns="urn:jboss:domain:messaging:1.1">
  <hornetq-server>
    <clustered>true</clustered>
    . . . . .
  </hornetq-server>
</subsystem
```



Configuration multi-cast

Chaque nœud utilise UDP :

- pour diffuser ses informations sur ses connecteurs : *broadcast-group*
- pour découvrir les informations des connecteurs des autres serveurs : *discovery-group*



Connexions cluster

Les connexions cluster doivent également être configurées à l'intérieur de l'élément `<hornetq-server>` :

```
<cluster-connections>
<cluster-connection name="mycluster">
  <address>jms</address>
  <connector-ref>netty</connector-ref>
  <retry-interval>500</retry-interval>
  <use-duplicate-detection>true</use-duplicate-detection>
  <forward-when-no-consumers>>false</forward-when-no-consumers>
  <max-hops>1</max-hops>
</cluster-connection>
</cluster-connections>
```



Configuration

- L'attribut **name** définit la connexion cluster, ils peut avoir plusieurs connexions cluster dans le même sous-système de *messaging*
- L'élément **address** est obligatoire et détermine comment les messages sont distribués sur le cluster. Dans l'exemple précédent, tous les messages envoyés à une adresse commençant par *jms* seront répartis
- L'élément **connector-ref** référence le connecteur définit dans la section *connectors* du sous-système de *messaging*
- L'élément **retry-interval** détermine l'intervalle en ms entre les tentative de relivraison au même nœud
- L'élément **use-duplicate-detection** lorsqu'il est activé détecte les messages dupliqués qui seront alors ignorés
- L'élément **forward-when-no-consumers** permet la redistribution de message
- L'option **max-hops** (Par défaut 1) permet de déterminer les nœuds accessible pour la répartition. 1 signifie donc que seuls les nœuds en connexions directes peuvent être choisis



Algorithme de répartition

L'algorithme de répartition peut être spécifié sous l'élément *connection-factory*.

Les algorithmes fournis par HornetQ sont :

- Round-Robin (*RoundRobinConnectionLoadBalancingPolicy*)
- Random (*RandomConnectionLoadBalancingPolicy*).

Un algorithme customisé peut être mis en place en implémentant l'interface

org.hornetq.api.core.client.loadbalance.ConnectionLoadBalancingPolicy

```
<connection-factory name="InVmConnectionFactory">
. . . .
<connection-load-balancing-policy-class-name>
org.hornetq.api.core.client.loadbalance.RandomConnectionLoadBalancingPolicy
</connection-load-balancing-policy-class-name>
</connection-factory>
```



Sécurité

Lorsque les connexions sont créées entre les nœuds, *HornetQ* utilise un login/mot de passe.

Il est impératif de changer les valeurs par défaut sinon des clients distants seront capable de faire des connexions au serveur

```
<cluster-user>user</cluster-user>
```

```
<cluster-password>password</cluster-password>
```

Un message de trace rappelle à l'administrateur que les mots de passe doivent être changés

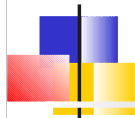
```
09:29:07,573 WARNING [org.hornetq.core.server.impl.HornetQServerImpl]
(MSC service thread 1-1) Security risk! It has been detected that the cluster admin
user and password have not been changed from the installation default. Please see
the HornetQ user guide, cluster chapter, for instructions on how to do this.
```




Configuration pour HA

La mise en place d'un serveur de backup nécessite la mise en place d'un journal partagé

La configuration est identique du côté du serveur actif et du backup



Exemple

```
<hornetq-server>
  <!-- true for backup, false for live -->
  <backup>true</backup>
  <!-- Persistence activée sur le disque -->
  <persistence-enabled>true</persistence-enabled>
  <!-- Le journal est partagé -->
  <shared-store>true</shared-store>
  <!-- Répertoire accessible par les 2 serveurs (live et backup) -->
  <journal-directory path="path/to/journal" relative-to="user.home"/>
  <bindings-directory path="path/to/bindings" relative-to="user.home"/>
  <large-messages-directory path="path/to/large-message" relative-to="user.home"/>
  <paging-directory path="path/to/paging" relative-to="user.home"/>
  <journal-file-size>102400</journal-file-size><!-- you may tune this -->
  <journal-min-files>2</journal-min-files><!-- you may tune this -->
  <!-- Failover lors d'un shutdown normal -->
  <failover-on-shutdown>true</failover-on-shutdown>
  <!-- ... -->
</hornetq-server>
```



Failover du client

Afin que le client puisse basculer sur le serveur actif en cas de défaillance, il doit obtenir une référence d'une usine à connexion *ha*.

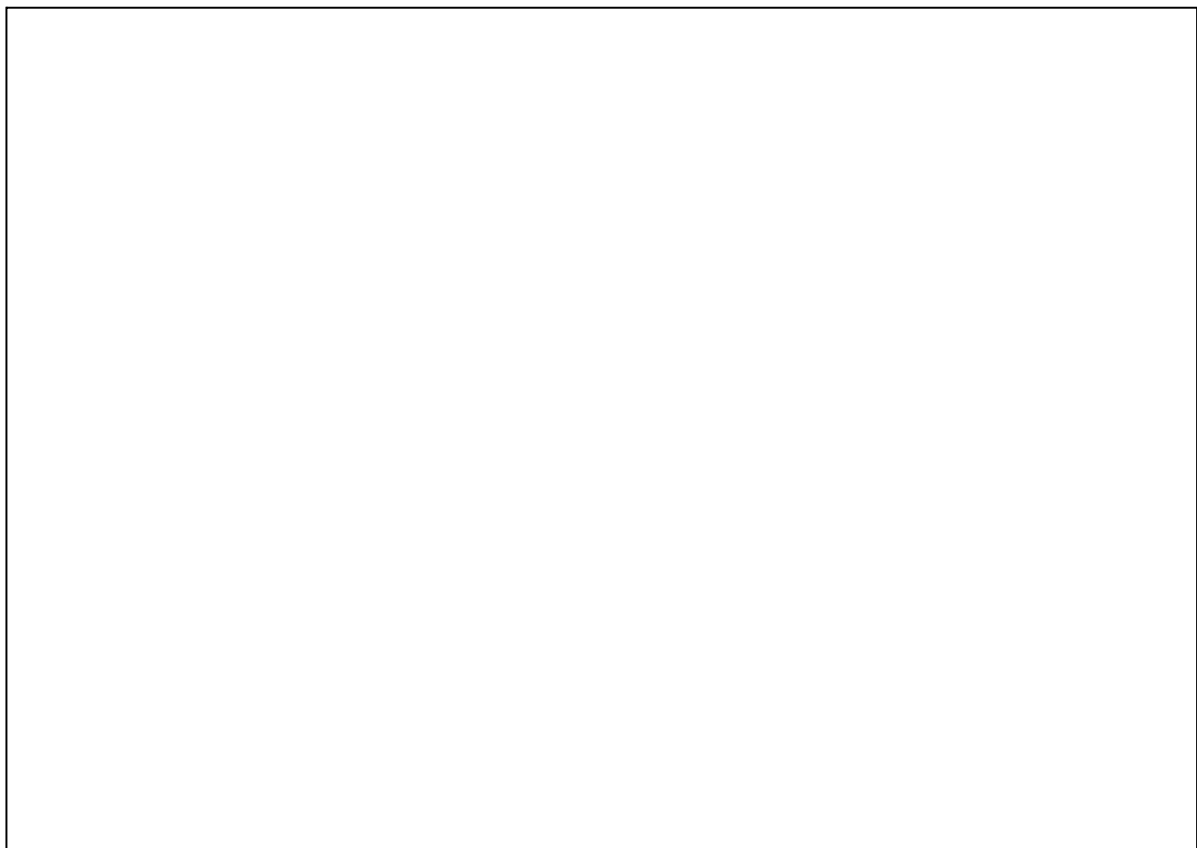
Cette usine à connexion HA est configurée de telle sorte qu'elle référence le groupe de découverte

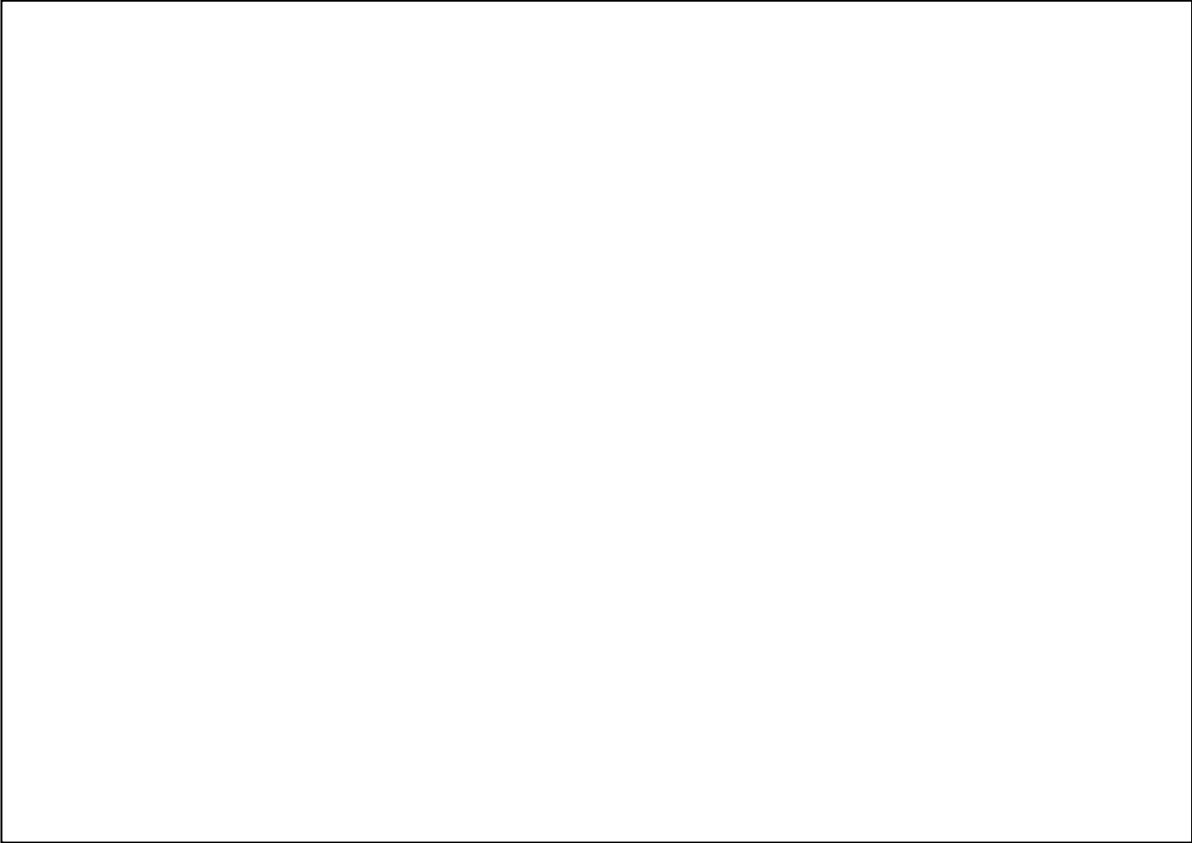
Le client doit également être accessible via le multicast UDP correspondant



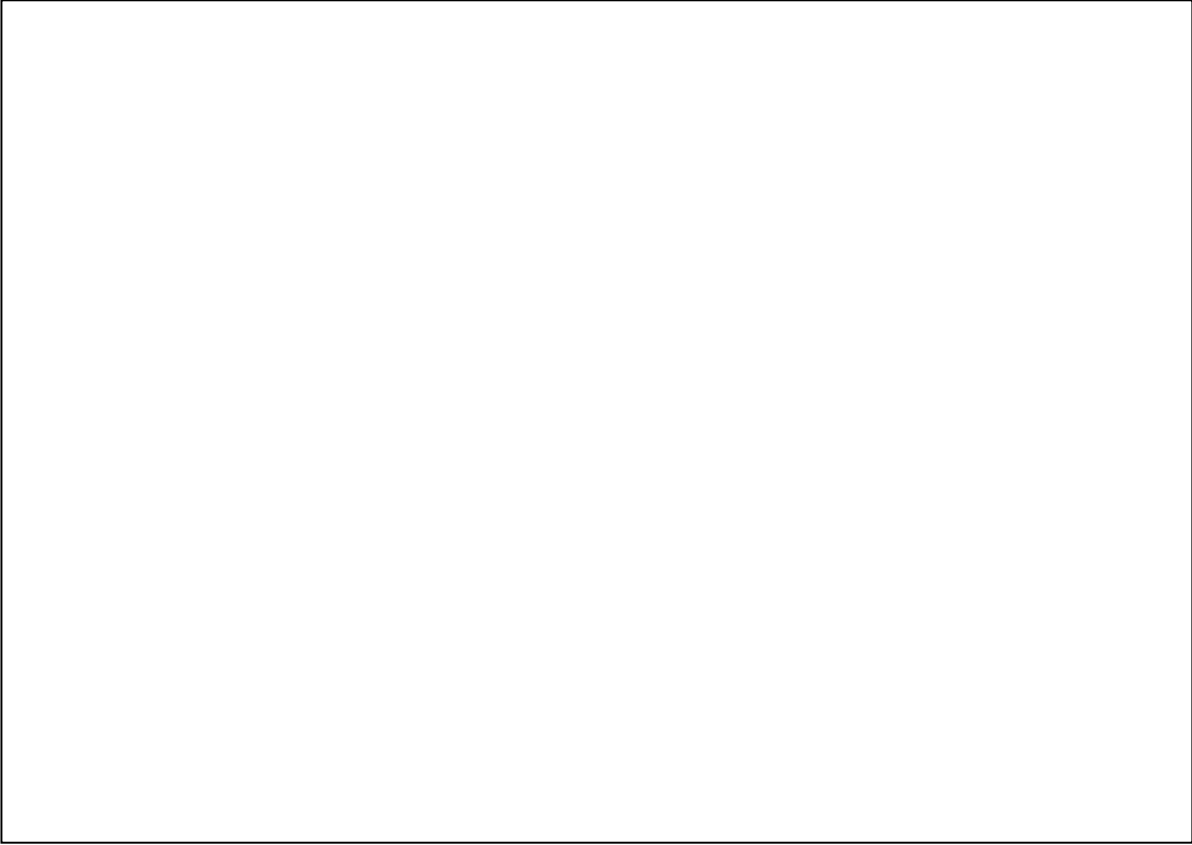
Configuration Usine HA

```
<jms-connection-factories>
  <connection-factory name="RemoteConnectionFactory">
    <discovery-group-ref discovery-group-name="dg-group1"/>
    <entries>
      <entry
name="java:jboss/exported/jms/RemoteConnectionFactory"/>
    </entries>
    <ha>true</ha>
    <reconnect-attempts>-1</reconnect-attempts>
  </connection-factory>
</jms-connection-factories>
```











Annexes

Protocoles JGroups

