Cenni di WildFly Clustering

Alma Mater Studiorum - Università di Bologna
CdS Laurea Magistrale in Ingegneria Informatica
I Ciclo - A.A. 2017/2018

Corso di Sistemi Distribuiti M (8 cfu)

11 - Cenni di Clustering di Applicazioni in WildFly Application Server

Docente: Paolo Bellavista
paolo.bellavista@unibo.it

http://lia.disi.unibo.it/Courses/sd1718-info/
http://lia.disi.unibo.it/Staff/PaoloBellavista/

Clustering

Approfittiamo della parentesi su architettura JBoss e JMX per parlare di questo AS in relazione a una funzionalità molto importante in molti servizi distribuiti di livello enterprise: ***clustering***

Partiamo con alcune domande.

Che cosa intendete quando usate i termini:

- Cluster ***fisico***?
- Cluster ***logico***?
- Clustering o clusterizzazione di una ***applicazione***?
- ***Supporto hw/sw*** al clustering?

Esempio di Funzionalità Specifica di JBoss: Clustering Service

Approfittiamo della parentesi su architettura JBoss e JMX per parlare di questo AS in relazione a una funzionalità molto importante in molti servizi distribuiti di livello enterprise: ***clustering***

Clustering, perché?

- ***esecuzione su server multipli in parallelo*** fornendo una visione singola ai clienti applicativi (ad es. motori di ricerca, siti di e-commerce articolati e ad alto carico di utenti, …)
- cruciale per:
  - ***Tolleranza ai guasti e disponibilità***
  - ***Load Balancing***
  - ***Scalabilità*** (miglioramento di performance tramite semplice aggiunta di nuovi nodi al cluster e load balancing)

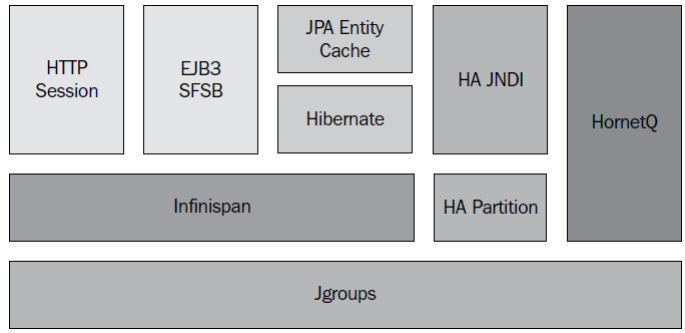
JBoss Clustering è una soluzione con ***buona trasparenza*** (***cluster mantenuto automaticamente, approccio modulare***) e open-source

Se utilizzerete JBoss AS 7 e successivi…

Novità più rilevante:

- ***servizio Infinispan*** per gestione della ***replicazione stato applicativo*** (invece di JBoss Cache, indicata nei lucidi successivi)

Cambiano i nomi dei file di configurazione: `standalone-ha.xml` o `standalone-full-ha.xml`

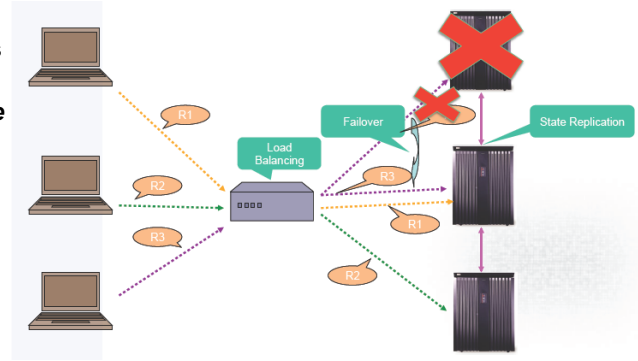


Esempio di Funzionalità Specifica di JBoss: Clustering Service

Invece nelle esercitazioni suggerite, se utilizzate versioni precedenti di JBoss, per utilizzare JBoss clustering è sufficiente:

- Usare la ***configurazione "all"*** di JBoss
  - `run.bat -c all`
  - `./run.sh -c all`

Abilita tutte le librerie necessarie al clustering, come *JGroups.jar, jboss-cache.jar*, ...



Comunicazione all'interno del Cluster

Un ***cluster (o partizione)*** JBoss è semplicemente un ***insieme di nodi***; ogni nodo JBoss è definito come una istanza del server JBoss (anche più nodi "logici" su una stessa macchina, con differenti porte associate ovviamente)

Uso di canali JGroup

Un canale JGroup:

- Tiene traccia ***automaticamente di chi fa parte del cluster*** (sulla base di configurazione e nome del canale JGroup utilizzato)
- Supporta ***scambio affidabile di messaggi*** all'interno del cluster
- Default: comunicazione multicast UDP (anche possibilità di TCP – quando lo sceglieresti?)

A default JBoss utilizza 4 canali JGroup separati:

- ***Un canale*** usato dal servizio general-purpose di ***HAPartition***
- ***Tre canali*** creati da JBoss Cache, anche per supportare ***replicazione dello stato***

Comunicazione all'interno del Cluster

Configurazione JGroup

- File `cluster-service.xml` nella directory `/deploy`
  descrive la ***configurazione per la partizione di default*** del cluster
  configurazioni JGroups come attributi innestati di servizi MBean del cluster
- Attributo `PartitionConfig` di MBean `ClusterPartition`
  descrive e configura lo ***stack di protocolli JGroups***
  configurazione di default usa UDP con IP multicast

```
<mbean code="org.jboss.ha.framework.server.ClusterPartition"
name="jboss:service={jboss.partition.name:DefaultPartition}">
…
<attribute name="PartitionConfig">
<Config>
<UDP mcast_addr="${jboss.partition.udpGroup:228.1.2.3}"
mcast_port="${jboss.hapartition.mcast_port:45566}"
tos="8" …
```


Comunicazione all'interno del Cluster

Ad esempio, PING è il protocollo per la scoperta iniziale dei membri del cluster

```
…
<PING timeout="2000"
down_thread="false" up_thread="false"
num_initial_members="3"/>

<MERGE2 max_interval="100000"
down_thread="false" up_thread="false"
min_interval="20000"/>
…
<FD timeout="10000" max_tries="5"
down_thread="false" up_thread="false" shun="true"/>
<VERIFY_SUSPECT timeout="1500" down_thread="false"
up_thread="false"/>
…
<pbcast.STATE_TRANSFER down_thread="false" up_thread="false"/>
…
```

Ad esempio, MERGE2 è per la fusione di nuovi sottogruppi già discovered

FD è invece il protocollo per failure detection

Comunicazione all'interno del Cluster

HA Partition

Servizio general-purpose di ***alta disponibilità*** (*High Availability*) utilizzato per diversi compiti in JBoss AS clustering

- Astrazione costruita sulla ***base dei canali JGroups***
- Supporto per effettuare e ricevere ***invocazioni RPC/RMI*** da e verso nodi cluster
- Supporto a ***registry distribuito*** (servizi del cluster e su quali nodi eseguono)
- ***Notifiche a listener*** per modifiche nell'appartenenza al cluster o per cambiamenti di servizi nel registry
- Nucleo di molti altri servizi di clustering:
  - ***smart proxy*** lato cliente
  - ***farming***
  - ***HA-JNDI***
  - …

Comunicazione all'interno del Cluster

Esempio di configurazione HA Partition

```
<mbean code="org.jboss.ha.framework.server.ClusterPartition"
name="jboss:service=DefaultPartition">
<attribute name="PartitionName">
   ${jboss.partition.name:DefaultPartition}</attribute>
<! – Indirizzo usato per determinare il nome del nodo -->
<attribute name="NodeAddress">${jboss.bind.address}</attribute>
<! -- deadlock detection abilitata o no -->
<attribute name="DeadlockDetection">False</attribute>
<! -- Max time (in ms) di attesa per il completamento del trasferimento di
   stato -->
<attribute name="StateTransferTimeout">30000</attribute>
<! – configurazione protocolli JGroups -->
<attribute name="PartitionConfig”>... ... </attribute>
</mbean>
```

Per fare parte dello stesso cluster, ***i nodi devono semplicem. avere lo stesso*** `PartitionName` e gli stessi elementi `PartitionConfig`

Comunicazione all'interno del Cluster

Possibilità di ***deployment omogeneo*** (replica su tutti i nodi del cluster – ***automatico attraverso farming***) o ***disomogeneo*** (sconsigliato dagli sviluppatori JBoss, ad es. per mancanza di supporto a transazioni distribuite sul cluster)

Clustering JNDI

Il cliente usa un ***proxy HA-JNDI per invocare JNDI lookup***

Lato server, ***servizio HA-JNDI mantiene albero JNDI*** (*context tree*) sul cluster:

- Albero sempre disponibile fino a che almeno un nodo è presente nel cluster
- Ogni nodo ha sua copia locale di parte dell'albero JNDI (spazio di nomi parzialm. replicato e partizionato)
- Il servizio HA-JNDI su di un nodo può effettuare binding di oggetti presenti nella sua copia locale JNDI

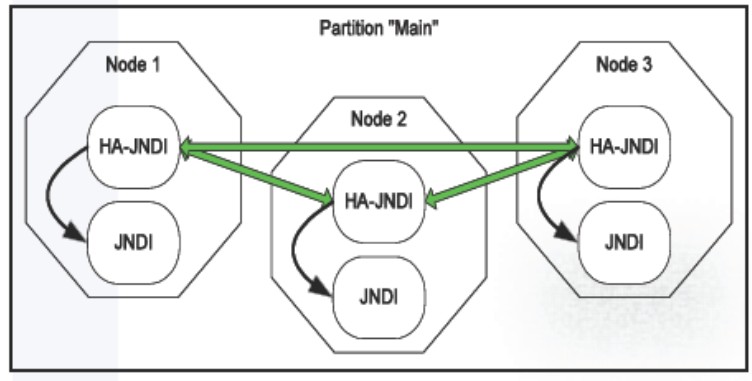
All'atto del lookup:

- Se binding non è nell'albero del nodo interrogato, ***delega a JNDI locale***
- Se non trovato, ***coordinamento fra componenti HA-JNDI nel cluster***
- `NameNotFoundException` se non presente in nessun JNDI del cluster

Comunicazione all'interno del Cluster

Clustering JNDI:

- Componenti HA-JNDI mantengono ***conoscenza reciproca*** e ***copia parziale*** dell'albero JNDI locale
- Componenti JNDI locali possono avere alberi più estesi rispetto ai loro HA-JNDI



Comunicazione Client-to-Cluster

Due tipi di configurazione:

1) JBoss fat client

- I clienti usano ***HA smart proxy*** che contengono ***logica di load balancing*** (ad es. Round Robin) ***e capacità di resilienza*** (*failover*)
- La logica di HA smart proxy può essere ***modificata e plugged-in*** da parte del programmatore (differenti politiche di failover e load balancing basate su pesi differenziati e carico)

Clienti EJB usano necessariamente HA Smart Proxy, che realizza uno stub RMI significativamente accresciuto

2) JBoss thin client

- Tutta la logica di load-balancing + failover è ***lasciata allo sviluppatore*** che può adottare soluzioni hw o sw (ad es. Apache mod_jk/mod_cluster + Tomcat)

Clienti Web accedono trasparentem. attraverso Apache mod_jk/mod_cluster

Comunicazione Client-to-Cluster

Nel caso semplice e specifico di ***invocazioni RMI***:

- Logica di clustering messa in atto dagli ***stub cliente*** (*HA-RMI*)
- Stub contiene la ***lista dei nodi disponibili*** nel cluster + ***politica di load balancing***
  - Random/Round Robin/First Available
- Se avviene una ***modifica nella composizione del cluster***, alla invocazione seguente il server fa ***piggybacking*** della lista aggiornata
- ***Lista*** dei nodi ***mantenuta automaticamente*** lato server tramite ***JGroups***
- Solita soluzione tramite ***meccanismi trasparenti di intercettazione client-side*** che si occupano di load balancing e failover all'invocazione di metodi nell'interfaccia dello stub

Replicazione dello Stato

Possibile attività progettuale?
Magari su approfondimento Infinispan

JBoss Cache

JBoss Cache è un ***framework per il supporto a cache distribuita*** (che può essere utilizzato anche in altri ambienti AS)

Realizza il ***caching di oggetti Java acceduti frequentemente*** per motivi di performance

In JBoss, fornisce servizi di caching per

- Sessioni HTTP
- Stato di session bean EJB 3.0
- Entity EJB 3.0

Ogni servizio di cache è definito in un Mbean separato (con il suo canale JGroups)

JBoss Cache è cluster-aware: lo stato è mantenuto ***consistente fra i vari nodi*** nel cluster

- Tolleranza ai guasti in caso di server crash
- Necessità di invalidazione e/o aggiornamento dello stato nella cache

Replicazione dello Stato, tipicam per Sessioni HTTP

Failover e Load Balancing di sessioni HTTP

Replicazione dello stato della sessione gestita da JBoss

- Configurazione ***"all"*** di JBoss include ***session state replication***
- Occorre ***configurare session state replication*** da gestire ***tramite JBoss Cache***

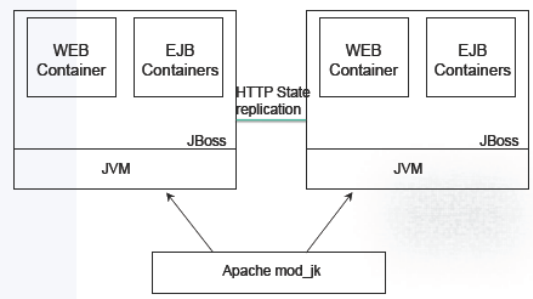
Load balancing è invece gestito usualmente da software esterno o a livello hardware:

- Ad es., suggeriti da JBoss

Apache ***mod_jk***

o più recentemente

Apache ***mod_cluster***



Apache mod_cluster

Rispetto al più semplice e «tradizionale» Apache mod_jk:

- ***Configurazione dinamica cluster***
  - integrato con meccanismi di advertising su JGroups basati su multicast UDP
  - nodi cluster fanno discovery automatico di ***uno o più nodi load balancer disponibili***
- ***Diverse metriche*** per misurazione carico
  - cpu, memoria, n° connessioni, ...
- Notifiche stato applicativo
  - possibilità di intercettare eventi di re-deploy/un-deploy/... delle applicazioni presenti su AS e prendere azioni di management conseguenti

A partire da JBoss AS 7,
Infinispan

Sostituisce JBoss Cache (testimonia «hot topic» replicazione stato)

Infinispan: framework open-source basato su JGroups, anche utilizzabile aldifuori di JBoss

- Svolge il ruolo di infrastruttura di caching e replicazione per sessione HTTP, stato di SFSB, nomi JNDI e secondo livello di Hibernate

Quattro strategie di caching:

- ***Locale*** (utilizzato tipicamente per secondo livello di Hibernate)
- ***Replicazione***
- ***Distribuzione***
- ***Invalidazione***

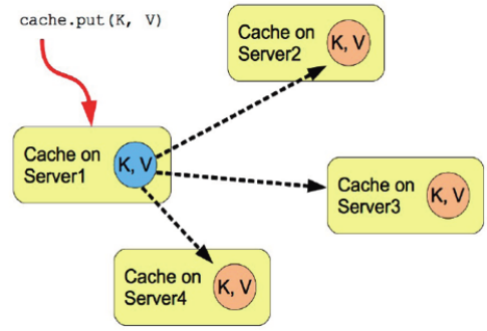
Infinispan

- ***Replicazione***

Tutti oggetti in cache replicati su tutti i nodi del cluster

Tipicamente adatta per piccoli cluster

Scarsamente scalabile



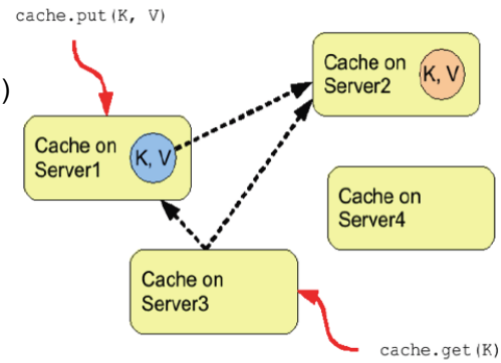
Infinispan

- ***Distribuzione***

Tutti oggetti in cache replicati solo su sottoinsieme fisso (configurabile) di nodi del cluster

Minori performance quando oggetti non disponibili localmente

Migliore scalabilità



Infinispan

- ***Invalidazione***

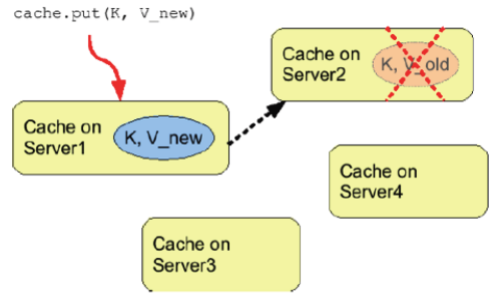
Nessuna replicazione ma rimozione di entry non valide

Tipicamente utilizzata per dati comunque disponibili su datastore persistente

Replicazione e invalidazione possono avvenire in modo ***bloccante/non-bloccante***

In modo non-bloccante:

- anche coda con ***modifiche/invalidazioni multiple***
- Migliori performance ma errori solo riportati in log ed eventualmente utilizzati in rollback se transazioni



Infinispan Cache

- ***Eviction*** (rimozione di oggetti in cache, ***solo locale***)

Strategie unordered, FIFO, LRU

- ***Cache Loader***

Connessione fra Infinispan e versione su memoria persistente (file system, DB, cloud store come AmazonS3) della cache estesa

Se ***passivazione abilitata***, scrittura in datastore solo in caso di eviction

- ***Livelli di isolation e locking*** (vecchie conoscenze ☺)
  - REPEATABLE_READ (sono possibili phantom read)
  - READ_COMMITTED (possibili non repeatable read ma maggiori performance)
  - Optimistic locking e verifiche a posteriori in fase di commit

Replicazione dello Stato

In alternativa, ***sticky session***

Si può chiedere un ***clustering con sticky session*** (***nessuna replicazione di sessioni*** HTTP)

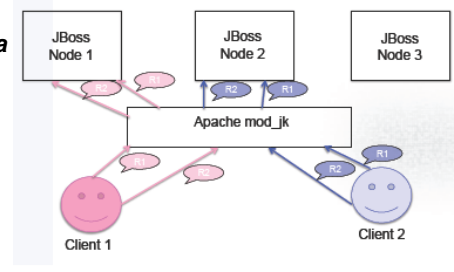
- Tutte le richieste dello ***stesso cliente*** vengono inviate allo ***stesso nodo del cluster***

Pro

- Disabilitazione di HTTP session replication => notevoli ***vantaggi in termini di performance***

Contro

- Caduta del server produce ***perdita dello stato di sessione***



Replicazione dello Stato

Soluzioni ibride: Sticky session + HTTP state replication

Spesso, in casi reali, si utilizzano ***soluzioni ibride più articolate***. JBoss supporta:

- HTTP replication ***sincrona e asincrona*** (come già visto per Infinispan)
  - HTTP replication asincrona ha migliore scalabilità
- Si può inoltre specificare la ***granularità desiderata per la replicazione*** (per motivi di performance)
  - `session` - ***intera sessione***
  - `attribute` – solo gli ***attributi modificati*** (*dirty attributes*) + alcuni dati specifici di sessione come `lastAccessTime`. Particolarmente adatto per ***sessioni con stato di ampia dimensione***

  con decrescenti livelli di overhead

Replicazione dello Stato

In modo molto pratico, ***configurazione di HTTP session replication*** sui vari framework di supporto:

- Session replication viene abilitata a livello di singola applicazione Web in **`WEB-INF/web.xml`**, tag **`<distributable>`**
- Configurazione di JBoss (replicazione sincrona o asincrona) viene fatta in **`WEB-INF/jboss-web.xml`**
- Replicazione della sessione viene configurata nel container per la sessione Web, ad es. ***Tomcat***, in **`deploy/tc5-cluster-service.xml`**

```
...
<!- Modalità consentite: LOCAL, REPL_ASYNC e REPL_SYNC ! -->
<attribute name="CacheMode">REPL_SYNC</attribute>
...
```


Clustering di Componenti EJB

- Stateless Session Bean
  - Facile ☺, data la mancanza di stato, le chiamate possono essere bilanciate su ogni nodo
- Stateful Session Bean
  - ***Stato replicato e sincronizzato*** sul cluster
  - Ad ogni modifica sullo stato di uno di questi bean, entra ***in azione session state replication di JBoss Cache o di Infinispan***
- Entity Bean
  - Non interrogabili da remoto e quindi non prevista abilitazione clustering
  - solo per secondo livello Hibernate possibilità di utilizzare Infinispan
- MessageDriven Bean
  - Utilizzo di HornetQ in JBoss AS 7

Clustering di Componenti EJB

Come deve essere fatto in questo caso HA smart proxy (pensiamo al caso di partenza di semplice fat client RMI)?

Riflettiamo sui diversi casi di:

- Stateless Session Bean
- Stateful Session Bean
- MessageDriven Bean
- Entity Bean

Replicazione dello Stato

Nel migliore dei casi ☺, è sufficiente uso di una ***annotazione*** (specifica e proprietaria di JBoss) per ***informare il container EJB*** di JBoss che il componente considerato deve essere ***clustered***

Annotazione `@Clustered`. Applicabile a:

- ***SLSB*** con smart proxy. Politiche di load balancing
  - RoundRobin (default), FirstAvailable, FirstAvailableIdenticalAllProxies (tutti smart proxy per un bean verso stesso nodo), RandomRobin
- ***SFSB*** con smart proxy. Politiche di load balancing
  - FirstAvailable (con abilitazione replicazione stato)
- ***MDB***. Politiche di load balancing
  - RoundRobin, Random, Custom (tramite implementazione interfaccia `ConnectionLoadBalancingPolicy`)