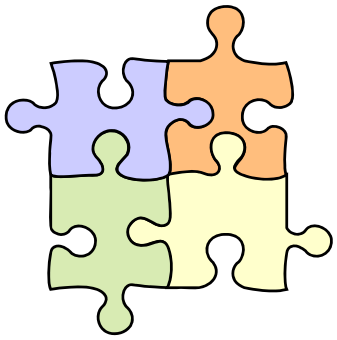
Luca Cabibbo



Architetture Software

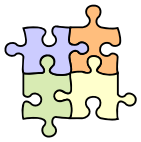
Cluster per architetture a componenti

Dispensa ASW 442

ottobre 2014

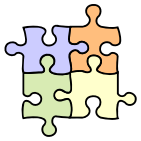
Un buon progetto produce benefici in più aree.

Trudy Benjamin



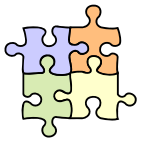
- Fonti

- [IBM] Clustering Solutions Overview, IBM RedPaper 4072
  - http://www.redbooks.ibm.com/redpapers/pdfs/redp4072.pdf
- [Microsoft] MSDN Library
  - Server Clustering, http://msdn.microsoft.com/en-us/library/ff649250.aspx
  - Load Balanced Cluster, http://msdn.microsoft.com/en-us/library/ff648960.aspx
  - Failover Cluster, http://msdn.microsoft.com/en-us/library/ff650328.aspx
- [Glassfish] Clustering in Glassfish Version 3.1
  - http://glassfish.java.net/public/clustering31.html (con imprecisioni)
  - http://dsc.sun.com/appserver/reference/techart/glassfishcluster/ (relativo alla versione 2, ma corretto)

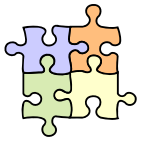


- Obiettivi e argomenti

- Obiettivi
  - conoscere alcuni aspetti relativi al deployment di architetture basate su componenti, per sostenere scalabilità e disponibilità
- Argomenti
  - introduzione
  - cluster
  - cluster per application server

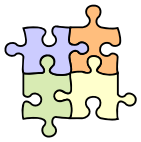


- Wordle



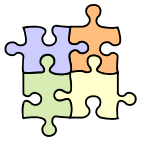
* Introduzione

- Le architetture a componenti fanno uso di un Application Server
  - con l'obiettivo di delegare all'Application Server le responsabilità relative alle proprietà di qualità delle applicazioni – in particolare, prestazioni, scalabilità e disponibilità
    - questo consente di focalizzare lo sviluppo dei componenti sugli aspetti funzionali
  - per raggiungere questi obiettivi, l'implementazione dell'Application Server è basata sullo stile architetturale Container e altri pattern correlati
- In pratica, alcuni obiettivi di qualità possono essere raggiunti solo insieme a un'opportuna implementazione *distribuita* del Container
  - questa dispensa discute alcuni aspetti relativi all'architettura fisica (di deployment) delle architetture basate su componenti – comunemente basata su un cluster di computer



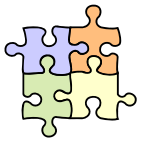
* Cluster

- Un **cluster** è un gruppo di due o più computer (server) interconnessi – chiamati *nodi* o membri del cluster – che lavorano insieme per offrire un servizio come un sistema singolo
  - per "sistema singolo" si intende che il gruppo dei nodi interconnessi che forma il cluster viene visto dall'utente/client del servizio come una singola entità
  - un cluster, oltre ai nodi (computer), comprende anche altri elementi, hardware e software
    - ad esempio, certamente la rete di interconnessione
    - di solito, inoltre, su ciascun nodo è in esecuzione un servizio per gestire l'appartenenza al cluster
    - inoltre, è di solito presente anche un elemento per il bilanciamento del carico



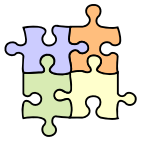
Perché il clustering

- Due (anzi, tre) motivazioni principali per l'uso di un cluster
  - scalabilità
    - se il carico del servizio eccede quello di un singolo nodo (o dei nodi attualmente presenti), è possibile risolvere il problema aggiungendo ulteriori nodi (scalabilità orizzontale)
    - il carico del servizio è distribuito tra i diversi nodi mediante l'uso di tecniche di load balancing (bilanciamento del carico)
  - high availability (alta disponibilità)
    - se uno o più nodi fallisce, il servizio può essere ancora erogato tramite i nodi sopravvissuti – con un'interruzione di servizio ridotta
    - mediante l'uso di tecniche di fault tolerance (tolleranza ai guasti) – ad esempio, il failover (in caso di fallimento di un nodo A, muovere servizi da A ad un altro nodo B)
  - queste benefici possono essere anche ottenuti congiuntamente



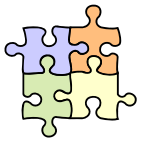
Tipi di cluster

- Esistono diversi tipi di cluster, con finalità diverse – ad esempio
  - high-performance cluster
    - usati di solito in contesti scientifici – ad esempio, per le previsioni del tempo
    - è importante soprattutto la possibilità di distribuire il carico *di singoli job* tra più nodi – ad esempio, map-reduce
    - non prendiamo in considerazione cluster di questo tipo
  - high availability cluster
    - usati di solito per ospitare applicazioni commerciali, per poter gestire un numero elevato di *transazioni concorrenti*
    - è importante sia la scalabilità che l'alta disponibilità
  - disaster recovery
    - requisiti stringenti per la disponibilità (ad es., business continuity) possono essere gestiti mediante cluster i cui nodi sono distribuiti tra più siti, geograficamente distanti tra loro



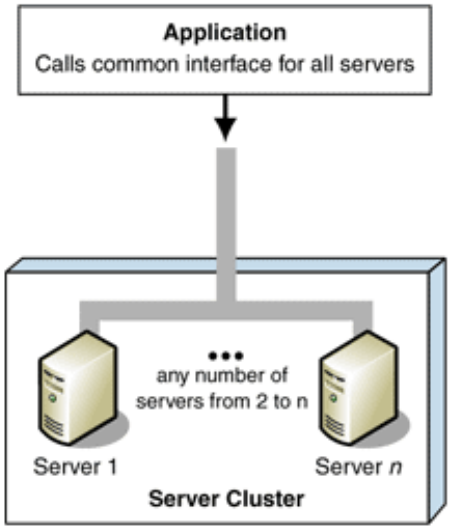
Cluster e application server

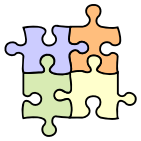
- Come detto, un cluster ha lo scopo di offrire un servizio come un sistema singolo
  - il servizio offerto da un cluster potrebbe essere un server web, un database server, un application server, ...
  - nel seguito di questa dispensa, ci concentriamo soprattutto su cluster di supporto agli application server – anche se molte considerazioni hanno validità più generale
  - in questo caso, il cluster consente di eseguire un'applicazione (a componenti) su più nodi, in modo concorrente, fornendo un punto di accesso singolo ai client dell'applicazione



Configurazioni

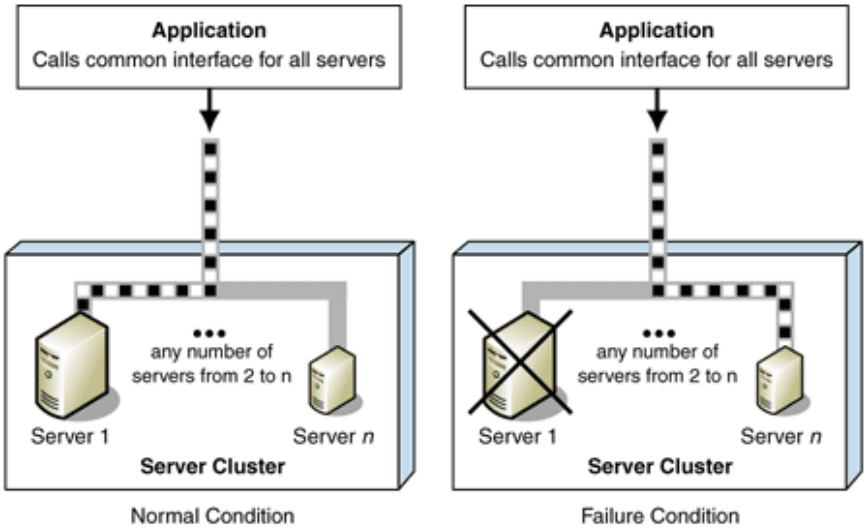
- Esistono diverse configurazioni comuni per i cluster
  - la configurazione di base per un cluster è un gruppo di due o più nodi che lavorano insieme per offrire un servizio come un sistema singolo

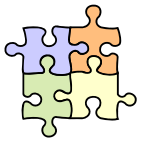




Cluster asimmetrico

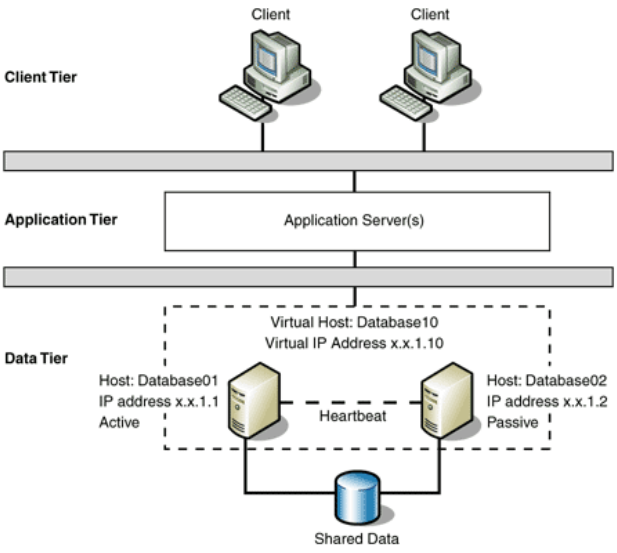
- *Cluster asimmetrico*
  - un servizio viene erogato solo da un nodo (*server primario*)
  - c'è un *server secondario* (*standby server*) che può sostituire il server primario nell'erogazione del servizio – la sostituzione avviene solo in caso di fallimento dei server primario

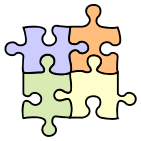




Cluster asimmetrico

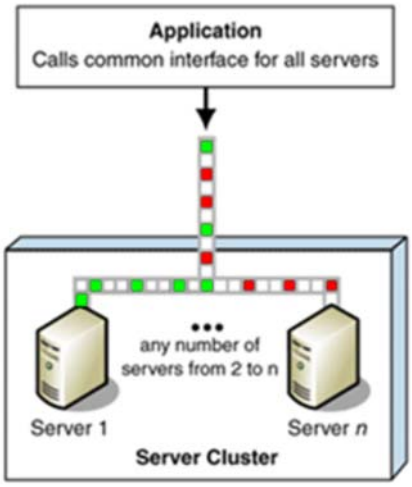
- *Failover*
  - quando viene riconosciuto (nell'esempio, mediante heartbeat) un fallimento del server primario (nell'esempio, per il database)
  - il *failover* consiste nel far prendere al server secondario il ruolo del primario

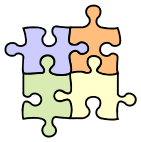




Cluster simmetrico

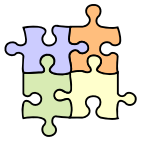
- *Cluster simmetrico*
  - ciascun nodo svolge del lavoro utile, erogando uno o più servizi
  - di solito, ciascun nodo è il server primario per un certo insieme di servizi – ma può essere anche server secondario (standby server) per altri servizi
  - in caso di fallimento di un nodo, ciascuno dei suoi "servizi primari" viene riassegnato a un server secondario (per quel servizio) tra quelli sopravvissuti





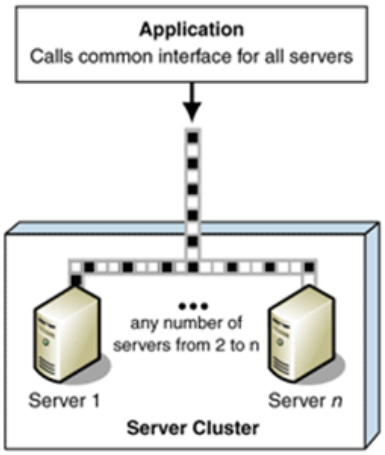
Cluster simmetrico

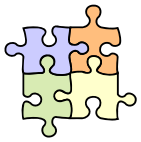
- Considerazioni sull'uso di un cluster simmetrico
  - in un cluster simmetrico, rispetto a un cluster asimmetrico, le risorse di calcolo sono sfruttate meglio – poiché la maggior parte dei nodi viene utilizzato (in modo "utile") la maggior parte del tempo
  - tuttavia, lo spostamento di un servizio da un nodo A verso un altro nodo B potrebbe portare a un carico eccessivo per quel nodo B – con un degrado dei servizi che offre, che addirittura potrebbe poi portare a un fallimento anche del nodo B
  - come vedremo tra breve, un cluster simmetrico può essere utilizzato anche diversamente – distribuendo il carico di un servizio su più nodi



Cluster e load balancing

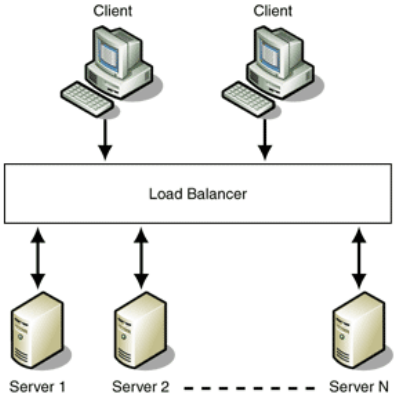
- *Cluster per load balancing*
  - è un cluster di tipo simmetrico, in cui un servizio viene erogato attivamente da più nodi – dunque, non c'è un singolo server primario per il servizio – al limite, tutti i nodi del cluster partecipano attivamente all'erogazione del servizio
  - è necessario utilizzare un meccanismo aggiuntivo di load balancing
  - può essere necessario anche un meccanismo di sincronizzazione tra i nodi del cluster

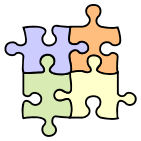




Cluster e load balancing

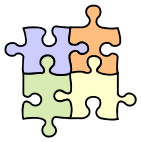
- *Load balancing*
  - un servizio è erogato da più nodi del cluster
  - le richieste per il servizio vengono ricevute da un load balancer – che le gira ai nodi pertinenti del cluster sulla base di un'opportuna politica
    - ad esempio, round robin, carico attuale dei nodi, ...
  - un aumento del carico del servizio può essere gestito mediante l'adozione di più nodi





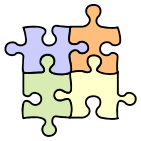
* Cluster per application server

- Come studio di caso, viene discusso l'uso dei cluster a supporto degli application server (AS) – nel seguito, faremo riferimento a Glassfish, un AS per Java EE
  - per *applicazione* (*applicazione Java EE*) intendiamo, ad esempio, un insieme di componenti Java EE
  - un'*istanza di server* (*server instance*) è un processo Java EE che ospita una o più applicazioni Java EE
  - un *nodo* è un computer che ospita una o più istanze di server
  - un *cluster* è formato da un insieme di nodi, di cui è di interesse la configurazione delle istanze di server
- L'obiettivo del cluster è
  - sostenere scalabilità
  - tollerare crash delle istanze di server
  - tollerare guasti di nodi



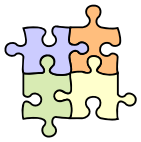
Cluster per application server

- Una configurazione comune (ma non l'unica possibile) per il cluster, a cui faremo riferimento, è la seguente
  - nel cluster, ovviamente, ci sono due o più nodi – ciascuno dei quali può fallire (indipendentemente dagli altri)
  - su ciascun nodo ci sono due istanze di server – se un'istanza di server fallisce, l'altra può sopravvivere
  - ogni istanza di server ospita tutte le applicazioni di interesse per il cluster
- Inoltre, è comune anche la presenza di un database (DB) server (distinto dall'AS)
  - il DB gestisce certamente lo stato persistente delle applicazioni in esecuzione sull'application server
  - ipotizziamo che il DB adotti le sue soluzioni per la disponibilità – e lo consideriamo dunque altamente disponibile



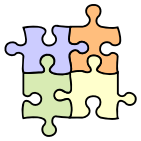
- Gestione dello stato delle sessioni

- Nell'uso di un cluster a supporto di un application server (AS), una risorsa critica – che va dunque gestita in modo opportuno – è costituita dai dati per lo stato delle applicazioni
  - ad es., stato delle sessioni/conversazioni con i client dell'application server (sessioni HTTP e stato degli enterprise bean stateful) – ma anche messaggi JMS
  - preservare questo stato è importante, ai fini della disponibilità del sistema e per gli utenti finali
  - qui ci concentriamo soprattutto sulla gestione dello stato delle sessioni



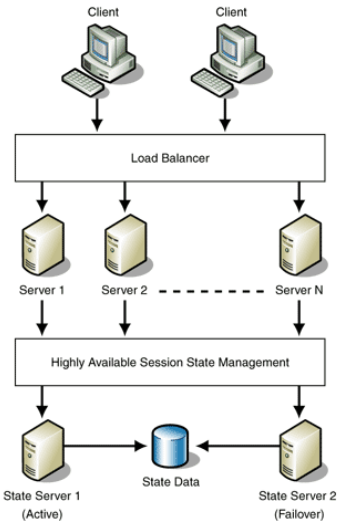
Gestione dello stato delle sessioni

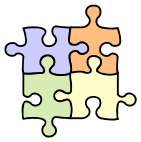
- Gestione dello stato delle sessioni in un application server
  - se, a seguito di un guasto, lo stato di una sessione viene perso, allora quella conversazione deve riprendere dall'inizio
  - se invece lo stato delle sessioni viene replicato in modo opportuno, allora la perdita di "una copia" dello stato di una sessione potrebbe non pregiudicare la prosecuzione della conversazione
- Osservazione
  - un AS può farsi aiutare (o meno) da un database server (affidabile) nella gestione dello stato delle sessioni/conversazioni con i suoi client



. Gestione centralizzata

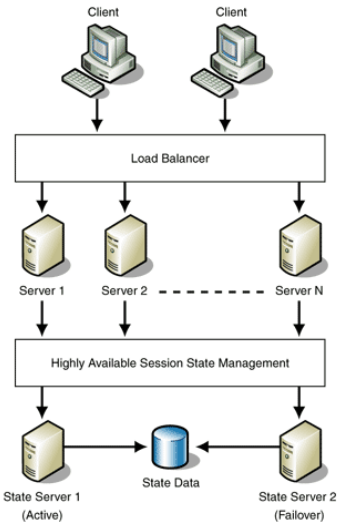
- *Gestione centralizzata dello stato delle sessioni*
  - lo stato delle sessioni è gestito nel DB (o comunque in un livello diverso da quello dell'AS) in modo affidabile (vedi ipotesi)
  - lo stato delle sessioni è condiviso dai diversi nodi – pertanto, ogni richiesta (da parte di client qualunque) può essere gestita da un nodo qualsiasi del cluster
  - quando il load balancer riceve una richiesta, la può girare a un'istanza di server qualunque
  - ogni volta che un'istanza di server riceve una richiesta da parte di un client, allora, prima di gestire la richiesta, carica lo stato della sessione dal DB – durante l'esecuzione dell'operazione richiesta, lo stato della sessione viene aggiornato anche nel DB

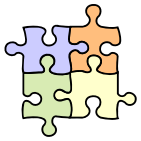




Gestione centralizzata

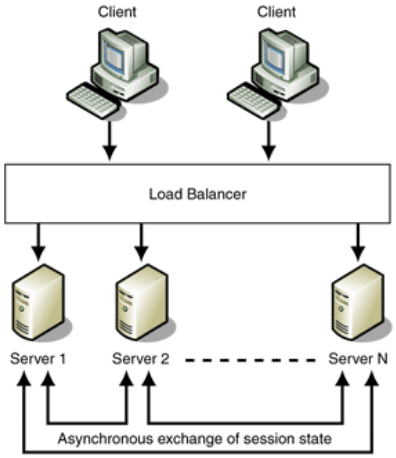
- *Gestione centralizzata dello stato delle sessioni*
  - che succede quando un'istanza di server o un nodo fallisce?
  - per le richieste attualmente in carico su quell'istanza di server o su quel nodo fallito, è possibile fare rollback nel DB e rischedulare la richiesta su un'istanza di server o nodo diverso – in modo completamente trasparente al client
  - poiché i client non sono associati alle specifiche istanze di server, nessuna conversazione è pregiudicata

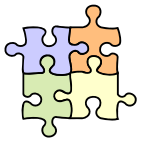




. Gestione asincrona

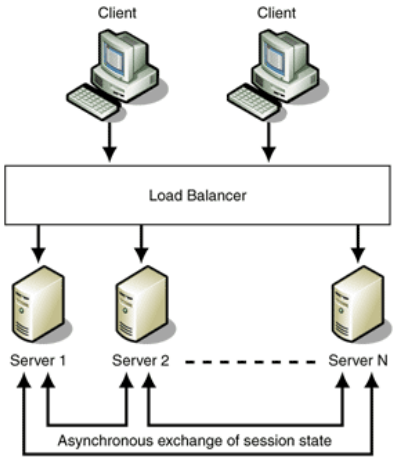
- *Gestione asincrona dello stato delle sessioni*
  - lo stato delle sessioni è gestito in memoria – ogni istanza di server memorizza lo stato di alcune sessioni, in modo tale che lo stato di ciascuna sessione sia memorizzato in almeno due istanze di server (in esecuzione su nodi diversi)
  - ogni volta che un client fa una richiesta, il load balancer la gira a un'istanza di server che ha in carico la gestione della sessione per quel client – quando la richiesta è stata servita, l'istanza che l'ha servita comunica (in modo asincrono) lo stato aggiornato di quella sessione a tutte le altre istanze coinvolte dalla gestione di quella sessione (si veda la tattica passive redundancy/ warm spare)

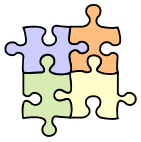




Gestione asincrona

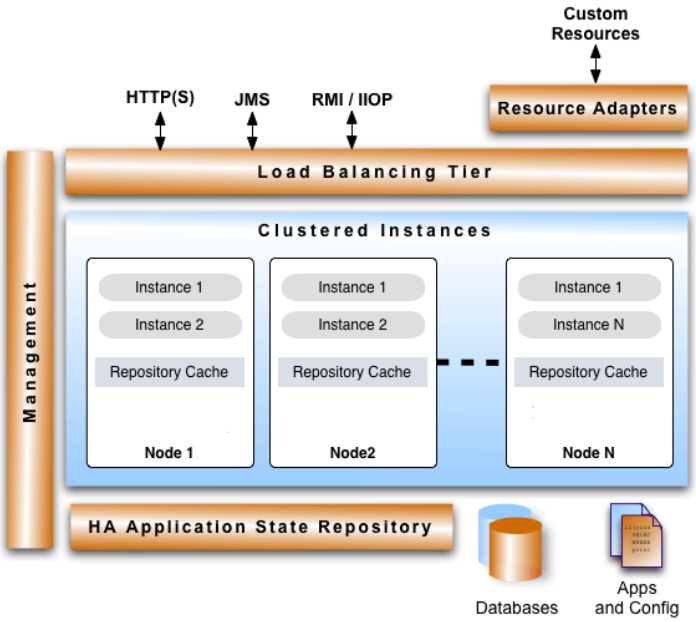
- *Gestione asincrona dello stato delle sessioni*
  - che succede quando un'istanza di server o un nodo fallisce?
  - le richieste in carico su quell'istanza o su quel nodo fallito vengono rigirate ad altre istanze in grado di gestire quelle richieste – perché conoscono lo stato aggiornato delle sessioni con i rispettivi client
  - nel frattempo, lo stato delle sessioni gestite dall'istanza di server o nodo fallito vengono comunicate (sempre in modo asincrono) ad altre istanze, per garantire nel cluster la presenza di almeno due copie dello stato di ciascuna sessione

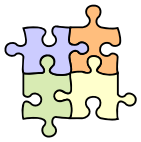




- Il caso di Glassfish

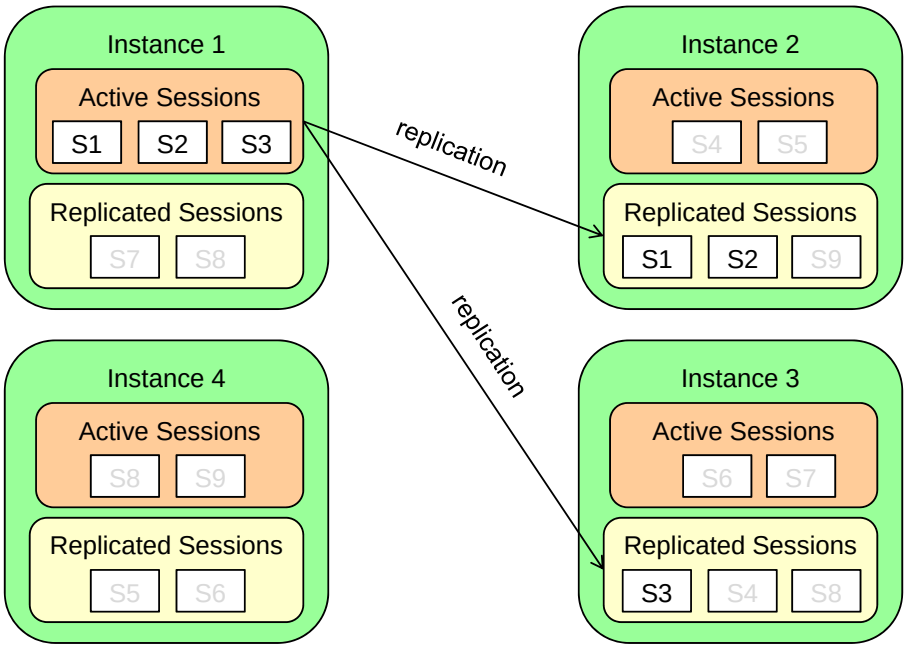
- Glassfish utilizza una gestione asincrona dello stato delle sessioni – in-memory session replication
  - http://glassfish.java.net/public/clustering31.html

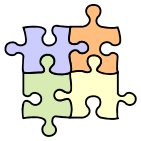




Il caso di Glassfish

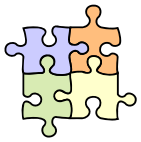
- Lo stato di ciascuna sessione S è replicato su due diverse istanze
  - un'istanza A per cui la sessione S è attiva (A gestisce attivamente la sessione S), l'altra istanza invece ne memorizza una replica





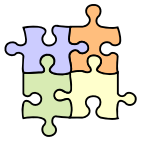
Il caso di Glassfish

- Quando nel cluster arriva una richiesta nell'ambito di una sessione S, il load balancer assegna la gestione della richiesta all'unica istanza A per cui la sessione S è attiva
  - la selezione avviene sulla base di una funzione di hash – applicata all'id della sessione
  - in caso di fallimento di questa istanza A, il failover è gestito dal load balancer, che gira la richiesta a un'altra istanza



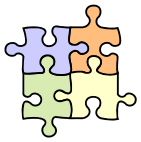
Il caso di Glassfish

- Due possibili scenari di failover
  - caso 1: la richiesta di failover è assegnata a un'istanza B che possiede già la replica dello stato della sessione S
    - l'istanza B diventa proprietaria della sessione S (ovvero, S diventa attiva per B), e si occupa dell'elaborazione della richiesta
    - inoltre, l'istanza B seleziona (usando la funzione di hash) un'altra istanza R che si dovrà occupare di memorizzare la replica dello stato della sessione S, e gliela comunica (in modo serializzato e asincrono)



Il caso di Glassfish

- Due possibili scenari di failover
  - caso 2: la richiesta di failover è assegnata a un'istanza C che non possiede una replica dello stato della sessione S
    - l'istanza C chiede (usando la funzione di hash) lo stato della sessione S all'istanza B che ne possiede la replica
    - l'istanza C diventa proprietaria della sessione S (ovvero, S diventa attiva per C), e si occupa dell'elaborazione della richiesta
    - inoltre, l'istanza C seleziona (usando la funzione di hash) un'altra istanza R che si dovrà occupare di memorizzare la replica dello stato della sessione S, e gliela comunica (in modo serializzato e asincrono)
    - l'istanza B cancella la propria replica della sessione S



- Discussione

- Un Application Server realizza l'infrastruttura di comunicazione tra componenti
  - per raggiungere obiettivi di qualità come prestazioni, scalabilità e alta disponibilità, l'implementazione di un Application Server
    - è basata sullo stile architetturale Container e altri pattern correlativi
    - è concepita con una realizzazione distribuita, comunemente basata su un cluster di computer