BAS 2009

"Infrastruttura Middleware"

Sommario

Sommario....2
1 Architettura hardware di rete....3
2 Architettura middleware....3
2.1 Server Web....4
2.2 Server Applicativi....6
2.2.1 Deployment Descriptor....7
2.2.2 Apache Tomcat....8
2.2.3 L'architettura a connettori di Tomcat....8
2.2.4 I connettori HTTP....9
2.2.5 Architetture eterogenee con Tomcat e Apache Server....11
2.2.6 mod_proxy VS mod_jk....13
2.3 DBMS....14
2.4 Modelli architetturali....16
2.4.1 Modello scalabile....16
3 Architettura Applicativa....19
3.1 Protocollo AJP13 e tecniche di bilanciamento....19
3.2 Cluster IMS....19
3.3 Cluster Siti Istituzionali....19
3.4 DBMS Replicator e Database Server Backup....19
4 Tuning Apache HTTP Server....19
5 Tuning Apache Tomcat....19
5.1 mod_jk e configurazione dei workers....20
5.2 Clustering, riciclo delle sessioni e load balancing....20
5.3 Sleepers....20
5.4 Configurazione del connettore Https nel cluster IMS....20
5.5 Configurazioni aggiuntive....20
6 Tuning JVM....20
6.1 Java Heap Memory....20
6.2 Garbage Collector e algoritmi di garbage collecting....20

6.3 JVM monitoring....20
6.4 Heap Dump....20
7 DBMS....20
7.1 Tuning MySql....21
7.2 Tungsten Replicator....21
8 Monitoring....21
8.1 Introduzione a Nagios....21
8.2 Host, servizi, comandi e event handlers....21
8.3 pnp4nagios....21
8.4 check_mk e check_mk_agent....21
8.5 Un caso d'uso: monitoring di tomcat....21
8.6 Compatibilità e dipendenze 21
9 Test di carico (Stress Test)....21
9.1 Test prestazionali sui siti istituzionali....22
10 Analisi dei siti....22
10.1 Google Analytics sui siti istituzionali....22
11 Tavole....22
11.1 Elenco delle applicazioni (repository e deploy)....22
11.2 Databases e utenti 22
11.3 Elenco degli script principali 22
11.4 Utenti SO e permessi....22
11.5 Utenti applicativi....22
11.6 Servizi di Nagios....22
11.7 Event Handlers di Nagios....22

1 Architettura hardware di rete

In questo capitolo verranno fornite informazioni dettagliate sulle apparecchiature hardware utilizzate nell'infrastruttura, corredandole di schede tecniche e diagrammi. Il sistema è composto da otto server, sei di frontend applicativi inseriti nella DMZ e due di backend in cui risiedono i database.

[SCHEDE SERVER]

2 Architettura middleware

Lo strato middleware dell'architettura è stato implementato utilizzando noti strumenti del mondo opensource. Come server http è stato adoperato Apache web server 2.2, come server applicativo J2EE, invece, Tomcat 6 per i siti istituzionali e Tomcat 5.5 per l'IMS e come DBMS il noto MySQL 5. Nel capitolo verranno illustrati sommariamente questi strumenti e i vari modelli di configurazione middleware, ponendo l'accento sui limiti di quelli standard e sui motivi che ci hanno spinti a progettare un'architettura altamente scalabile, modulare e personalizzata.

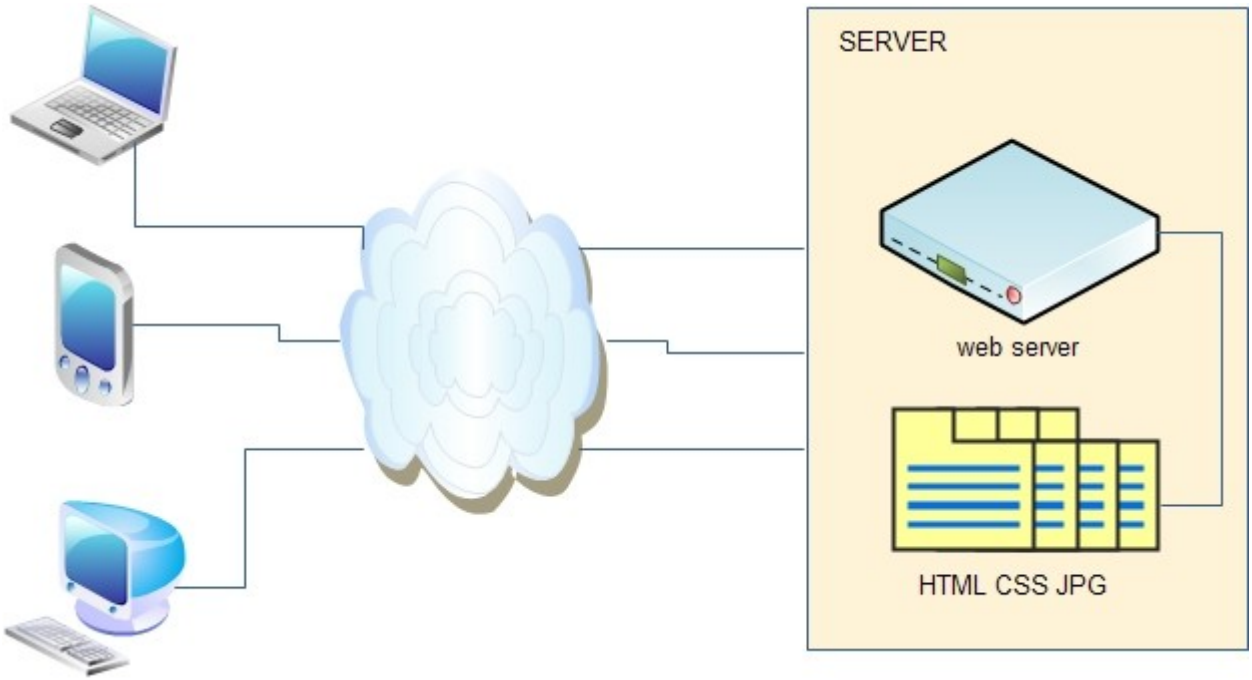
2.1 Server Web

Un Web server non è altro che un software installato in un server con la funzione di elaborare pagine web e generare dinamicamente contenuti.

Le semplici pagine .htm non necessitano di particolari interventi da parte del Web server, il loro codice viene interpretato dal browser del computer client e per questo l'html è definito come un linguaggio client side. Le pagine .php, .asp o .jsp, contengono dei codici destinati a produrre dei comportamenti e a generare dinamicamente html; perchè ciò sia possibile è necessaria la mediazione di un Web server. PHP, ASP e JSP vengono quindi definiti linguaggi server side.

Esemplificando, quando si invia ad un Web server la richiesta di una pagina .htm, statica, esso:

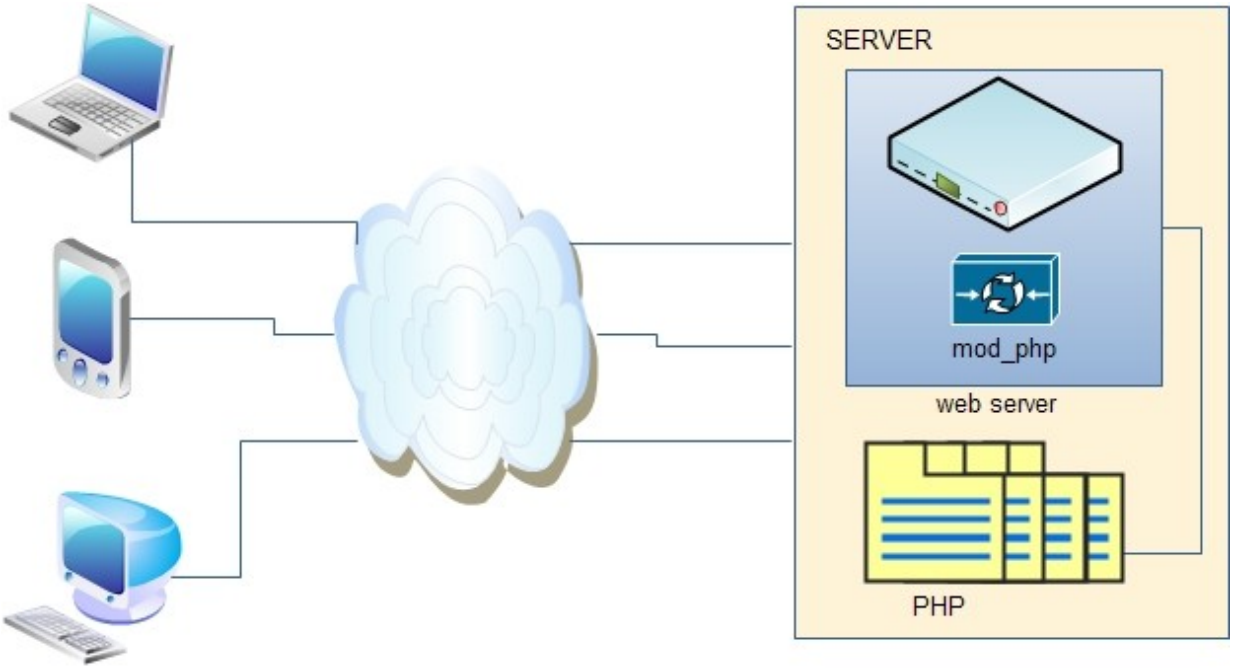
1. riconosce la richiesta
2. cerca e, se presente, trova la pagina nel computer server
3. invia la pagina al browser.



Disegno 1: Richiesta di contenuto statico a un server web

Nel caso di una pagina dinamica, invece, il Web server:

1. riconosce la richiesta
2. cerca e trova la pagina all'interno del server Web
3. esegue le istruzioni contenute all'interno del codice producendo dinamicamente contenuti
4. invia la pagina browser



Disegno 2: Richiesta di contenuto dinamico a un server web

Il server web più diffuso al mondo, e adottato nella nostra architettura, è Apache HTTP Server. Apache è un prodotto open source della Apache Software Foundation, nato da uno sforzo collaborativo volto a sviluppare un server web robusto, modulare e ricco di funzionalità. La versione da noi adottata è la 2.2 che presenta numerosi cambiamenti rispetto sia alla 1.3 sia alla 2.0. In particolare sono stati migliorati e resi più semplici da utilizzare i moduli legati alla configurazione e all'autenticazione, nella gestione del caching, invece, sono stati rivisti completamenti i moduli mod_cache, mod_disk_cache e mod_mem_cache, adattandoli alle più moderne esigenze di browser e applicazioni. Per quanto riguarda le funzionalità di proxying, al noto mod_proxy sono stati aggiunti il mod_proxy_balancer per l'implementazione del load balancing e il mod_proxy_ajp che aggiunge il supporto al protocollo Apache JServ utilizzato da Tomcat. Altra importante miglioria è il supporto a file di dimensioni superiori ai 2GB.

Nella nostra architettura sono state inserite due istanze di Apache 2.2, una per le applicazioni web istituzionali e l'altra per le richieste verso l'IMS. Quest'ultima implementa il modulo ssl per le richieste

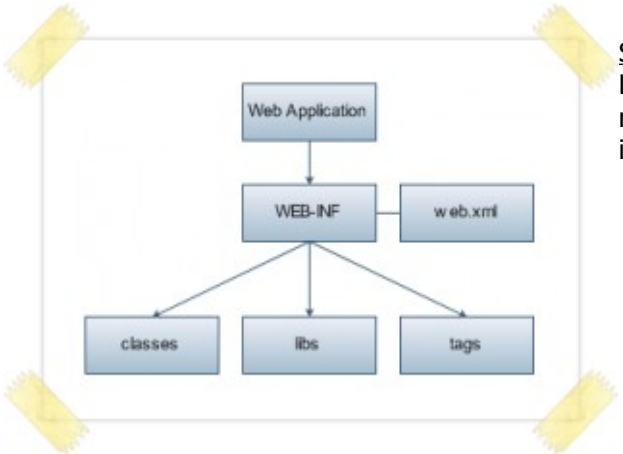
2.2 Server Applicativi

Apache http riesce a gestire molti contenuti dinamici grazie a una serie ben nutrita di moduli aggiuntivi. Per fare un esempio, attraverso il modulo php, Apache è in grado di interpretare gli script PHP e inviare al client il codice html risultante.

Tuttavia ciò non è possibile per ogni tecnologia. In questi casi Apache necessita dell'ausilio di componenti autonomi a cui delegare le operazioni di compilazione o di interpretazione. Questi componenti prendono il nome di server applicativi o web container, particolari strumenti in grado di manipolare applicazioni web basate su web component.

Nella piattaforma Java un web component è un oggetto che estende le funzionalità di un web server aggiungendone di nuove: Servlet e Java Server Pages sono per esempio dei web component.
In particolare le Servlet sono dei programmi Java in grado di processare dinamicamente delle richieste e costruire delle risposte mentre le JSP sono dei documenti di testo che vengono eseguiti come servlet permettendo un approccio più naturale alla costruzione di contenuti statici.
I web component sono in grado di accedere ai servizi offerti da un web container come lo smistamento delle richieste, la sicurezza, la concorrenza, la gestione della memoria.
Ad ogni web application è associato un file di configurazione in formato XML che prende il nome di Web Application Deployment Descriptor (DD) che rispetta un determinato schema.
La più piccola web application funzionante e quindi riusabile costituisce un modulo web e può contenere:

- classi di utilità lato server come JavaBeans
- classi lato client come Applet



Struttura di una web application
La directory root dell'applicazione contiene risorse statiche, la sottodirectory /WEB-INF/ invece contiene:

- il file web.xml che costituisce il Web Application Deployment Descriptor
- i tag library descriptor files
- la cartella classes che contiene classi di utilità lato-server
- la cartella tags che contiene tag files
- la cartella lib che contiene librerie di classi

Una web application viene generalmente compressa in un file JAR rinominato in **WAR (Web ARchive)** in maniera tale da poter essere riutilizzata o installata su un altro web container conforme alle Java Servlet Specification (deployment).

2.2.1 Deployment Descriptor

Il Deployment Descriptor contiene tutte le informazioni e le impostazioni di configurazione relative alla web application alla quale fa riferimento: la sua presenza permette di rendere molto più leggero il processo di deployment (installazione) di nuove web application all'interno di un web container (si parla infatti di **Pluggables Web Components**).

In particolare permette di poter specificare:

- le icone per la rappresentazione grafica dell'applicazione
- la descrizione dell'applicazione
- i parametri di inizializzazione
- i nomi delle servlet in esecuzione ed i relativi parametri di configurazione
- i filtri associati alle diverse servlet
- i listener di cui fa utilizzo la web application
- il timeout delle sessioni
- il mapping delle servlet con i relativi indirizzi
- il welcome file list ovvero la sequenza di file da utilizzare in risposta ad una richiesta client
- le regole per la gestione degli errori e della sicurezza

Tutte queste impostazioni di configurazione sono memorizzate sottoforma di un documento XML che fa utilizzo di determinati tag tutti contenuti all'interno del tag <web-app> che rappresenta la radice del documento stesso.

2.2.2 Apache Tomcat

Tomcat è un apllication server della Apache Software Foundation capace di eseguire servlet Java e rendere pagine che includono codice JSP. Descritto come una implementazione di riferimento per le Servlet Java le specifiche JSP, Tomcat è il risultato, come quasi tutti i prodotti Apache, della collaborazione globale di numerosi sviluppatori ed è disponibile gratuitamente dal sito Web della Apache.

Oltre alle funzionalità già descritte, Tomcat offre sofisticate funzionalità di clustering e replica delle sessioni, rendendo le web apps e l'intera architettura altamente scalabili.

Anche se al suo interno è presente un connettore HTTP che lo trasforma in un server HTTP a tutti gli effetti, il suo utilizzo più consolidato, specie in scenari di produzione, è quello che lo vede in abbinamento con un server HTTP vero e proprio.
Dato che Apache HTTP Server è il server più utilizzato in internet, l'abbinamento di Tomcat con Apache è probabilmente il più diffuso.

2.2.3 L'architettura a connettori di Tomcat

Tomcat dalla versione 4.0.x completa e migliora la sua implementazione basata su un modulo centrale al quale è possibile agganciare vari tipi di connettori.
Senza tali connettori il server sarebbe praticamente inutilizzabile dato che non sarebbe possibile comunicare con lui dall'esterno, ovvero non sarebbe possibile eseguire web application basate su pagine JSP o servlet.
I connettori possono essere di tre tipi:

- Connettori che parlano il protocollo HTTP (mod_http, mod_jk, coyote, mod_proxy) e che consentono di mettere in comunicazione il browser direttamente con Tomcat, senza intermediari. In questo caso le invocazioni per servlet e pagine JSP sono intercettate da Tomcat e rimappate su componenti in grado di eseguire le servlet e le pagine invocate. Si tenga presente che mod_proxy pur essendo un connettore che parla HTTP in gene-re viene utilizzato per realizzare un proxy HTTP fra il server web e Tomcat. Potrebbe però essere utilizzato per attivare la comunicazione diretta browser-server.
- Connettori come mod_jk e coyote che parlano il protocollo di invocazione Apache Jakarta Protocol (AJP nelle varie versioni 1.2, 1.3) per web applications. In questo caso Tomcat non è raggiungibile direttamente da un browser ma deve essere abbinato ad un server HTTP. In genere tutte le chiamate per servlet e pagine JSP vengono associate a determinati URL pattern, che a loro volta sono routati dal server HTTP a Tomcat. Tutte le risorse statiche (pagine HTML, immagini GIF o JPG) sono invece servite direttamente dal server web. Discorso particolare per le immagini e risorse statiche contenute all'interno di una web application. La specifica ed il buon senso indicano di far servire questi elementi dal server Tomcat, ma niente impedisce di estrapolarle e metterle sotto il controllo del server HTTP. E' bene ribadire che tale soluzione è fortemente sconsigliata se si desidera mantenere le applicazioni ed il sito dinamico facili da gestire e mantenere.
- Connettori di management che in un modo o nell'altro si agganciano al framework Java Management Extention (JMX). Essi permettono di gestire Tomcat dall'esterno effettuando tutte le principali operazioni di management.

2.2.4 I connettori HTTP

Prima di passare ad analizzare i connettori AJP, è utile fare una breve considerazione sui connettori per HTTP.
Quando uno di questi connettori viene abilitato, Tomcat diviene a tutti gli effetti un server HTTP in grado di fornire pagine HTML e risorse statiche direttamente al browser, così come di restituire il risultato dell'esecuzione di una servlet o di una pagina JSP.
Questo uso, sebbene molto utile in fase di sviluppo e di test, è fortemente sconsigliato in uno scenario di produzione per svariati motivi:

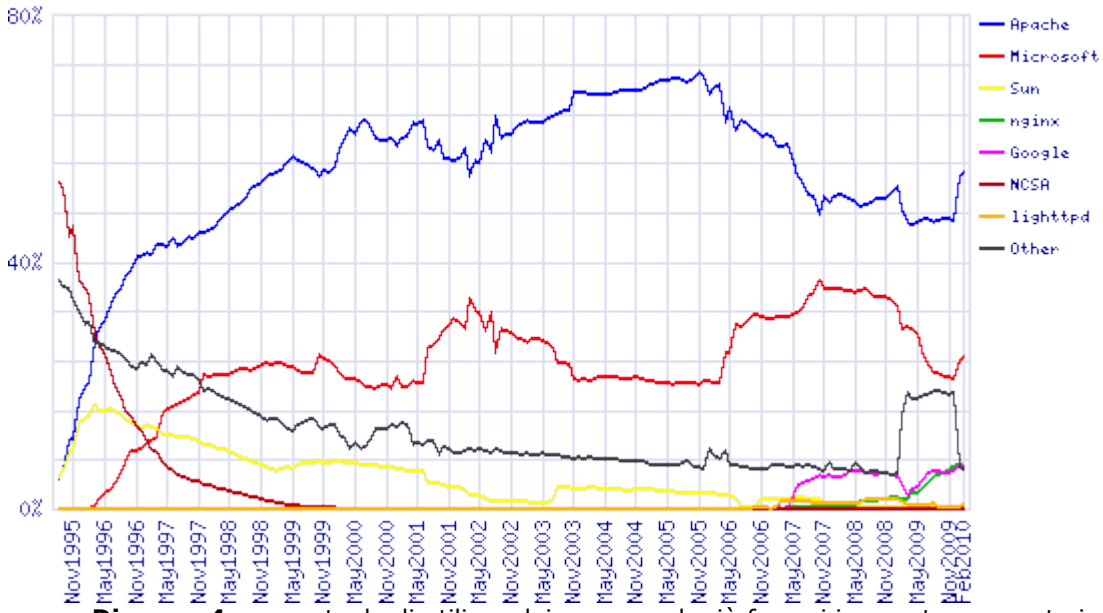
- Il grado di flessibilità e le possibilità di configurazione che si possono ottenere sfruttando gli strumenti messi a disposizione da Tomcat sono certamente minori di quanto possa fare ad esempio Apache che di mestiere fa il server HTTP.
- Legato al punto precedente c'è da considerare il fatto che utilizzare Tomcat per le risorse statiche è probabilmente un inutile spreco di risorse. Una pagine HTML o una immagine GIF non hanno bisogno di un web container per essere inviate al browser che ne fa richiesta.
- Dato che Tomcat funziona come container web, è presumibile che al suo interno si svolga della business logic. Fornire accesso al client web (browser) a Tomcat significa pubblicare direttamente su internet i servizi di business logic, cosa che può risultare rischiosa ed in alcuni casi sconsigliabile.
- Se Tomcat funziona come server HTTP è ovvio che in un modo o nell'altro viene associato direttamente a un indirizzo internet. In questo modo non sarà possibile sfruttare le tecniche di clustering e bilanciamento di Tomcat che, invece, funzionano in maniera impeccabile se la linea dei Tomcat si trova al di sotto di un server web.

Disegno 3: Se Tomcat viene utilizzato anche come server HTTP l'architettura che ne risulta è difficilmente manutenibile, meno sicura, e non scalabile.

Queste poche ma significative considerazioni dovrebbero far comprendere quanto sia importante l'utilizzo di Tomcat in abbinamento con un server HTTP. Eventualmente ci si potrebbe chiedere quale sia la scelta migliore per il server HTTP: attualmente la scelta che in genere si compie è fra due prodotti: Apache Web Server ed Internet Information Server (IIS) di Microsoft.

I connettori per Tomcat forniti da Apache Group sono in grado di funzionare bene con entrambi i prodotti anche se in passato si erano riscontrati diversi problemi per l'inoltro delle credenziali di autenticazione in HTTPS nel caso di utilizzo di IIS come server web.

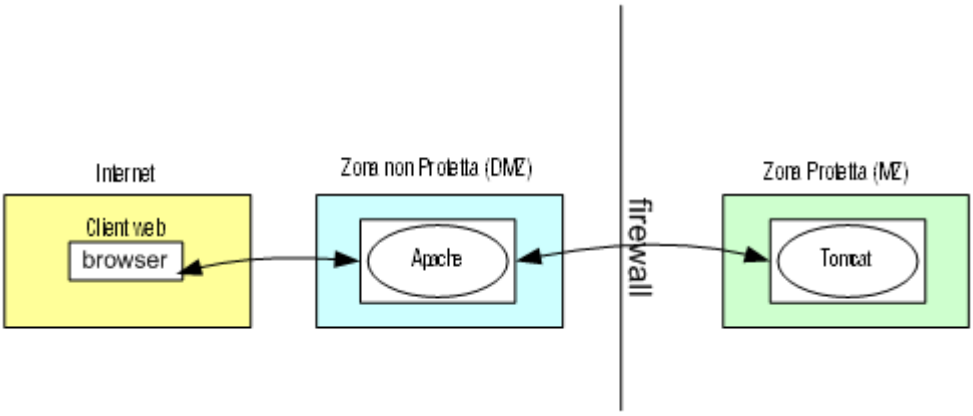
Non ci addentreremo oltre in analisi e paragoni fra i due prodotti, limitandoci ad analizzare lo scenario attuale che vede IIS in netta minoranza rispetto ad Apache, come raffigurato nel disegno 4.



Disegno 4: percentuale di utilizzo dei server web più famosi in questo momento in internet

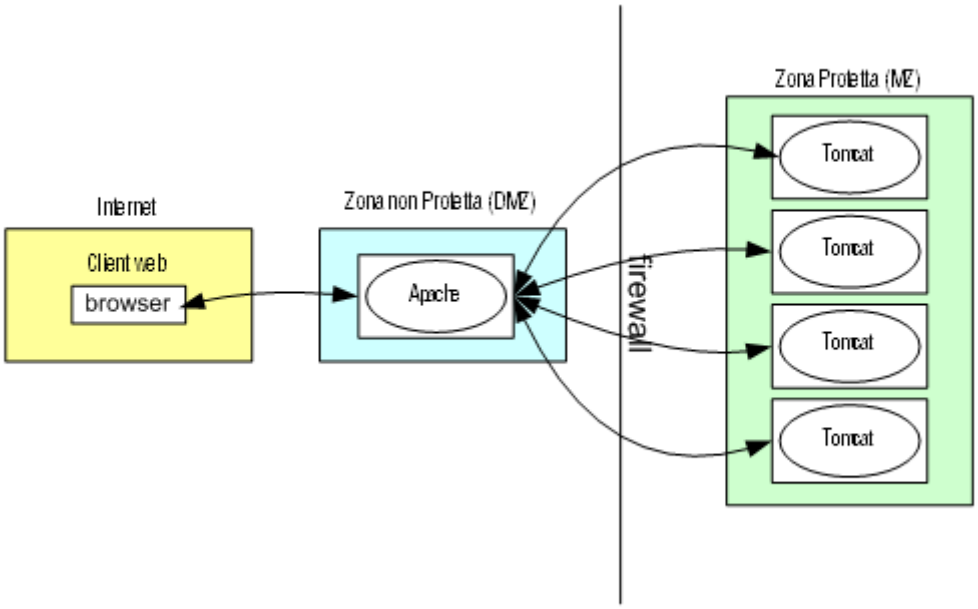
2.2.5 Architetture eterogenee con Tomcat e Apache Server

Una volta compreso che Tomcat da solo non basta, viene da chiedersi quali benefici si possano trarre dall'utilizzo in accoppiata con un server HTTP. I due più importanti sono certamente quelli legati alla sicurezza ed alla scalabilità.
Sicurezza perché a questo punto si può mettere tutto l'application server su un host protetto da un firewall in una zona protetta della rete, inaccessibile da client remoti, e rendere pubblico il solo server HTTP: i due comunicano a questo punto tramite protocollo AJP. Tutte le invocazioni per risorse statiche verranno servite dal server HTTP, mentre le richieste per servlet e pagine JSP verranno inoltrate verso Tomcat che le esegue.
Questa suddivisione dei compiti permette di ottimizzare fortemente le risorse del sistema e di bilanciare il carico di lavoro.



Disegno 5: L'utilizzo di un connettore AJP permette di mettere un server HTTP di fronte a Tomcat: in questo modo lo si può isolare dal resto del mondo, rendendo più sicura l'architettura complessiva. Il server HTTP inoltre offre maggiori potenzialità di configurazione ed ampliamento (fonte: www.mokabyte.it)

Oltre a questo avere un solo punto di entrata permette (se il connettore lo consente) di realizzare architetture in cluster, dove a fronte di un solo indirizzo internet di entrata si possono associare più server Tomcat in maniera del tutto trasparente e scalabile.



Disegno 6: L'utilizzo di un connettore AJP consente di associare ad un solo server HTTP (quindi un solo indirizzo internet) più server Tomcat, dando vita in modo semplice ad una configurazione cluster facilmente scalabile e quindi di servire un numero maggiore di clienti (fonte: www.mokabyte.it)

Per connettere Apache e Tomcat sono disponibili diverse alternative:

- mod_proxy: permette la connessione fra apache e Tomcat tramite una semplice conessione di tipo HTTP proxy. In alcune configurazioni questa soluzione può apparire non performante o eccessivamente semplicistica. La sintassi utilizzata per la configurazione delle due componenti (Apache e Tomcat) è relativamente semplice.
- mod_jk: questo connettore si basa sul protocollo AJP 1.2 (Apache Jakarta Protocol) o AJP 1.3 ed è stato introdotto come evoluzione del mod_proxy. E' particolarmente seguito e sviluppato dal gruppo di lavoro di Apache- Jakarta.
- mod_webapp: questo connettore è stato di recente rilasciato dal team di Apache-Jakarta come strumento espressamente pensato per mettere in comunicazione Apache e Tomcat secondo la modalità web application. Utilizza il protocollo WARP, che prendendo spunto dalla famosa velocità warp della astronave Enterprise, doveva nelle intenzioni dei progettisti fornire delle prestazioni nettamente migliori rispetto agli altri connettori. Putroppo per manifestati problemi di stabilità e funzionamento è stato poco adottato.

2.2.6 mod_proxy VS mod_jk

Dei tre connettori visti nel paragrafo precedente, i due più performanti sono il mod_proxy e il mod_jk.

A partire dalla 2.0 di Apache, il mod_proxy viene distribuito a corredo del Server web e dalla 2.2 prende il nome di mod_proxy_ajp e viene accompagnato dai mod_proxy_http e mod_proxy_balancer, rispettivamente un connettore evoluto per l'HTTP e il sistema che implementa il load balancing.

Le differenze tra i due connettori sono notevoli e si possono riassumere nel seguente schema:

mod_proxy

- pro:
  1. mod_proxy, mod_proxy_ajp, mod_proxy_http e mod_proxy_balancer vengono distribuiti con Apache2.2, dunque non c'è bisogno di compilare e mantenere un modulo separato.
  2. Capacità di utilizzare i protocolli HTTP, HTTPS e AJP anche all'interno dello stesso bilanciatore.
  3. Sistema di configurazione notevolmente più semplice rispetto a mod_jk.
- contro:
  1. mod_proxy_ajp non supporta pacchetti di dati superiori a 8K.
  2. Sistema di load balancing basilare.
  3. Non supporta il clustering del modello di dominio

mod_jk

- pro:
  1. Sistema di load balancing evoluto.
  2. Rilevamento avanzato della caduta dei nodi.
  3. Supporta pacchetti di dati superiori a 8K.
- contro:
  1. Va mantenuto in un modulo separato.

L'Apache Software Foundation sta puntando molto sullo sviluppo del mod_proxy, ma, per il momento, date le distanze che lo separano dal mod_jk nella gestione di aspetti

avanzati quali il load balancing e il node fault, aspetti centrali nella progettazione della nostra architettura, abbiamo preferito utilizzare mod_jk come connettore AJP.

2.3 DBMS

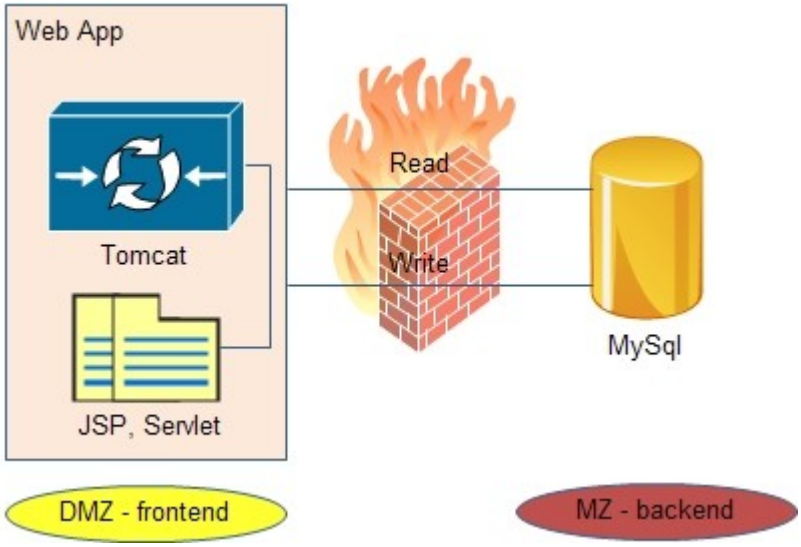
Il DBMS utilizzato nell'architettura è MySql 5.1, il DBMS relazionale più famoso al mondo. MySql viene distribuito in due versioni: la Community, rilasciata con licenza GPL e completamente gratuita e la Enterprise, una versione a pagamento che include il MySQL Enterprise Server, ovvero la versione più sicura, affidabile e aggiornata del DBMS, il MySQL Enterprise Monitor, che tiene costantemente sotto controllo il database e informa in modo proattivo su come implementare le best practice MySQL e un servizio di assistenza attivo 24hx7.

Per la nostra architettura abbiamo preferito affidarci alla versione Community che, dopo le opportune modifiche alla configurazione del MySQL server e l'integrazione di altri strumenti open source quali *Tungsten Replicator* per la replica dei db e *Jet Profiler for MySQL* per il monitoring, si è avvicinato di molto alla più blasonata versione Enterprise.

Per quanto riguarda MySQL 5, tra le novità introdotte rispetto alle versioni precedenti troviamo:

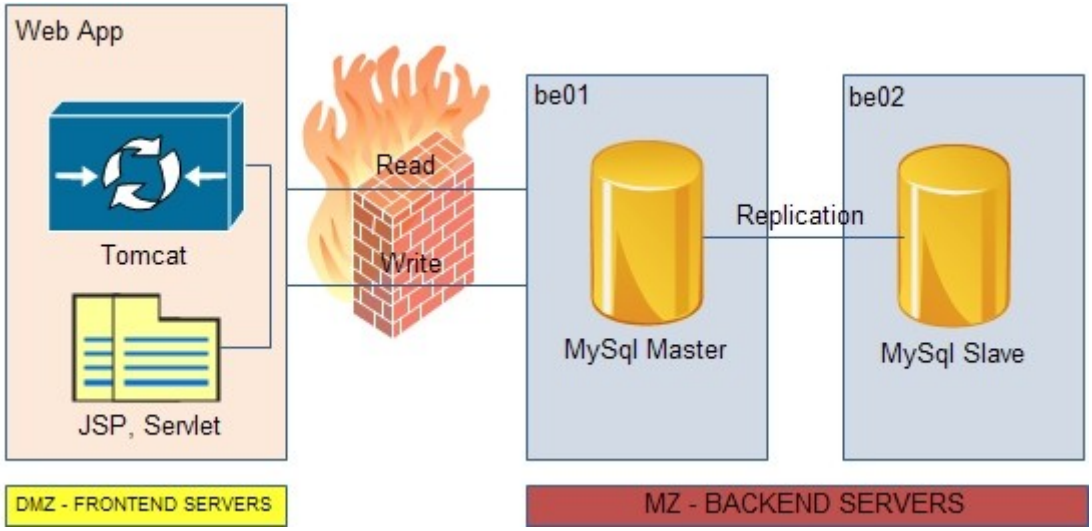
- il partizionamento delle tabelle
- un'API per scrivere nuovi parser per le *ricerche FULLTEXT*
- gli eventi
- replica basata sui dati (anziché sulle query)
- i log possono essere scritti in un database, oltre che nei file di testo
- supporto per Xpath
- campi AUTOINCREMENT e varie ottimizzazioni per le tabelle ARCHIVE
- ClusterDB ora può scrivere i dati su disco, oltre che conservarli nella RAM
- ALTER TABLE, CREATE INDEX e DROP INDEX sono molto più performanti

Le macchine in cui risiedono i database devono collocarsi in uno strato protetto dell'architettura (MZ). In questo modo, attraverso le opportune tecniche di firewalling, è possibile consentire l'accesso ai database alle sole applicazioni nella DMZ e agli sviluppatori tramite un mysqlproxy nella DMZ.



Disegno 7: Collegamento tra una webapp nella DMZ e una macchina DBMS nella MZ

Una volta messi al sicuro i dati, resta un problema logico nella disponibilità degli stessi. Immaginiamo che nell'ambiente di produzione, per un qualche motivo, la macchina DBMS dovesse cadere o il servizio andare in fault o, peggio ancora, si perdessero in maniera definitiva tutti i dati nei db. Con una sola macchina db, senza alcuna politica di backup, si giungerebbe a un blocco del servizio anche a tempo indeterminato. Per evitare uno scenario così catastrofico, nell'architettura è stata inserita una seconda macchina di backend che, attraverso la replica immediata dei dati presenti nel DBMS attivo, ha in qualunque istante una copia integrale di tutti i database. Così facendo, i dati non risiedono in un solo nodo, ma esistono copie di tabelle su più server. In questo modo oltre alla certezza dell'integrità dei dati, modificando l'indirizzo di connessione ai database nelle webapps, si avrebbe in breve tempo un recupero di tutti i servizi. Lo strumento utilizzato per replicare i dati è Tungsten Community Replicator, un'applicazione open source della Continuent, la cui configurazione è trattata nel settimo capitolo.



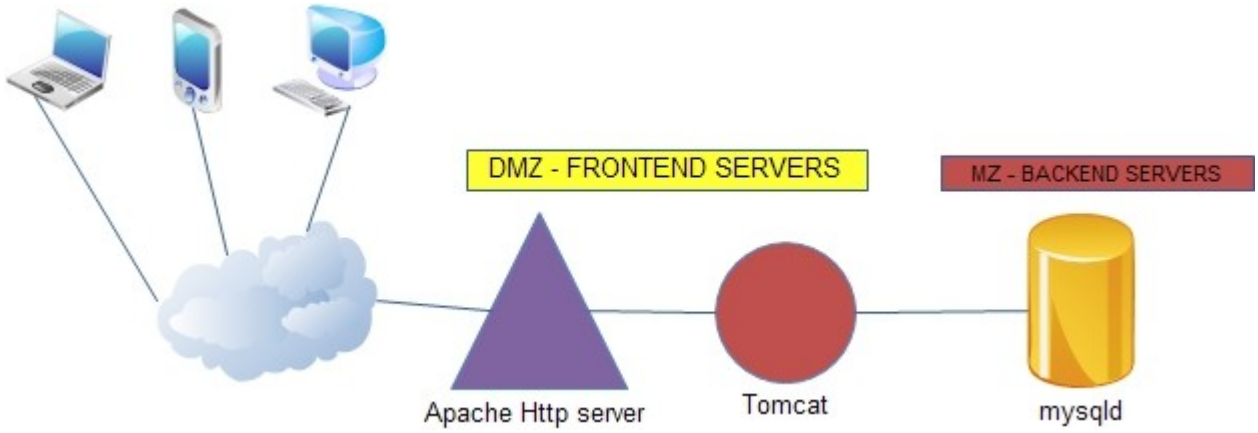
Disegno 8: Collegamento tra una webapp nella DMZ e una macchina DBMS nella MZ i cui dati sono replicati in una seconda macchina di backend

2.4 Modelli architetturali

In linea astratta, l'architettura può essere strutturata verticalmente su tre livelli differenti:

1. Server Web
2. Server applicativi
3. Macchine DBMS

In una prima schematizzazione, epurandola di concetti sofisticati quali clustering e scalabilità, potremmo trovare una struttura di questo tipo.



Disegno 9: Architettura non scalabile

E' ovvio che con un modello di questo tipo il grado di attività dei servizi deve essere massimo perchè, nel caso in cui cadesse uno solo dei tre nodi principali, l'intera architettura non sarebbe più in grado rendere alcun servizio all'esterno.

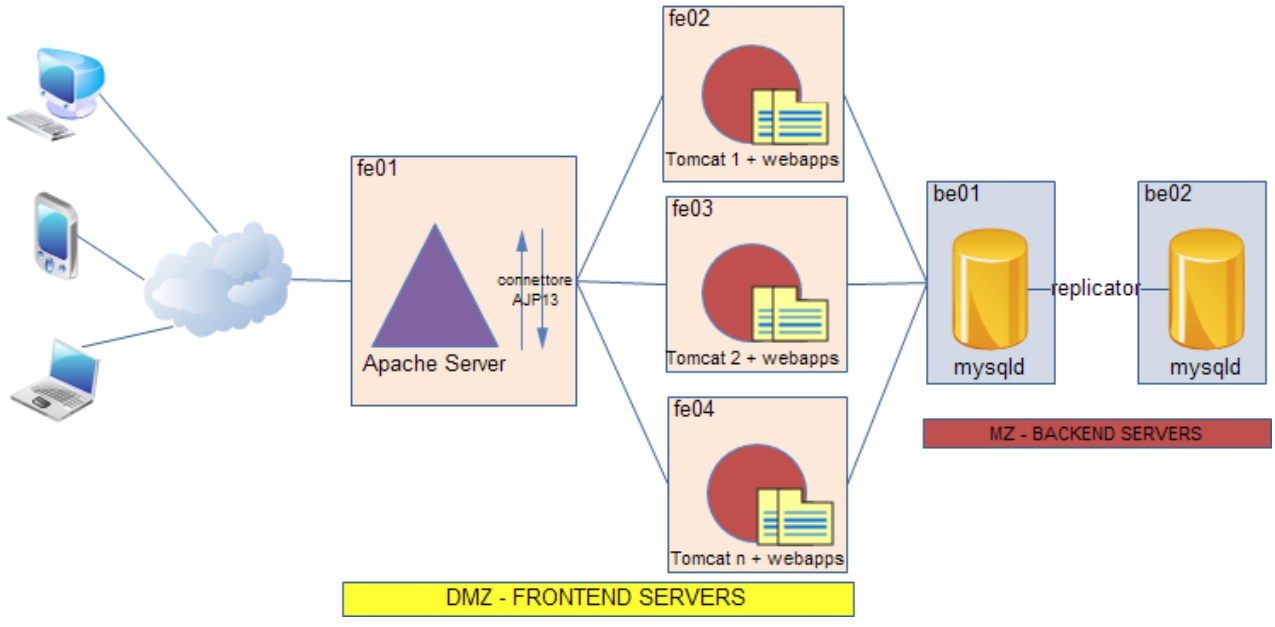
2.4.1 Modello scalabile

Questo problema si risolve rendendo l'architettura scalabile e tollerante ai guasti (fault tolerance) e i servizi “altamente disponibili” (high avalaibility).

Con il termine scalabile ci si riferisce, in termini generali, alla capacità di un sistema di “crescere” o “decrescere” in funzione delle necessità o delle disponibilità. La scalabilità può essere verticale o orizzontale. Quella verticale si ottiene incrementando le risorse hardware di un calcolatore (essenzialmente RAM, CPU e/o dischi). In questo contesto l’aumento della capacità computazionale non cresce linearmente all’aumentare delle risorse, ma è comunque limitato dall’hardware e dai programmi in esecuzione. La scalabilità orizzontale si realizza con un cluster di calcolatori. L’aumento delle esigenze computazionali, derivante dall’incremento delle richieste dei servizi, viene soddisfatto aggiungendo nuovi nodi al cluster.

Con il concetto di "alta disponibilità", in inglese high avalaibility, si intende la capacità di un sistema di garantire la continuità nell’erogazione dei servizi. Nell’ambito di un cluster, nel caso in cui un nodo si blocchi, il carico delle richieste che il nodo inattivo non può più processare viene reindirizzato verso gli altri nodi del sistema. La ridondanza dei nodi definiti in un cluster ha come obiettivo l’eliminazione dei punti deboli del sistema (i così detti "single point of failure") attraverso procedure automatiche che mantengono i nodi sincronizzati tra loro. In un tale contesto la tolleranza ai guasti (fault tolerance) è un aspetto che assume un'importanza rilevante. Maggiore è il numero degli elementi di un sistema, maggiore è la probabilità che ciascuno di essi possa andare in fault, determinando un’interruzione del servizio. Ciò rende la tolleranza ai guasti un requisito fondamentale al crescere dei sistemi distribuiti. Le tecniche sottostanti alla creazione di un cluster (alta disponibilità, bilanciamento e scalabilità) permettono a un sistema distribuito di garantire elevati livelli di fault tolerance.

A questo punto l'idea astratta di architettura potrebbe essere la seguente:

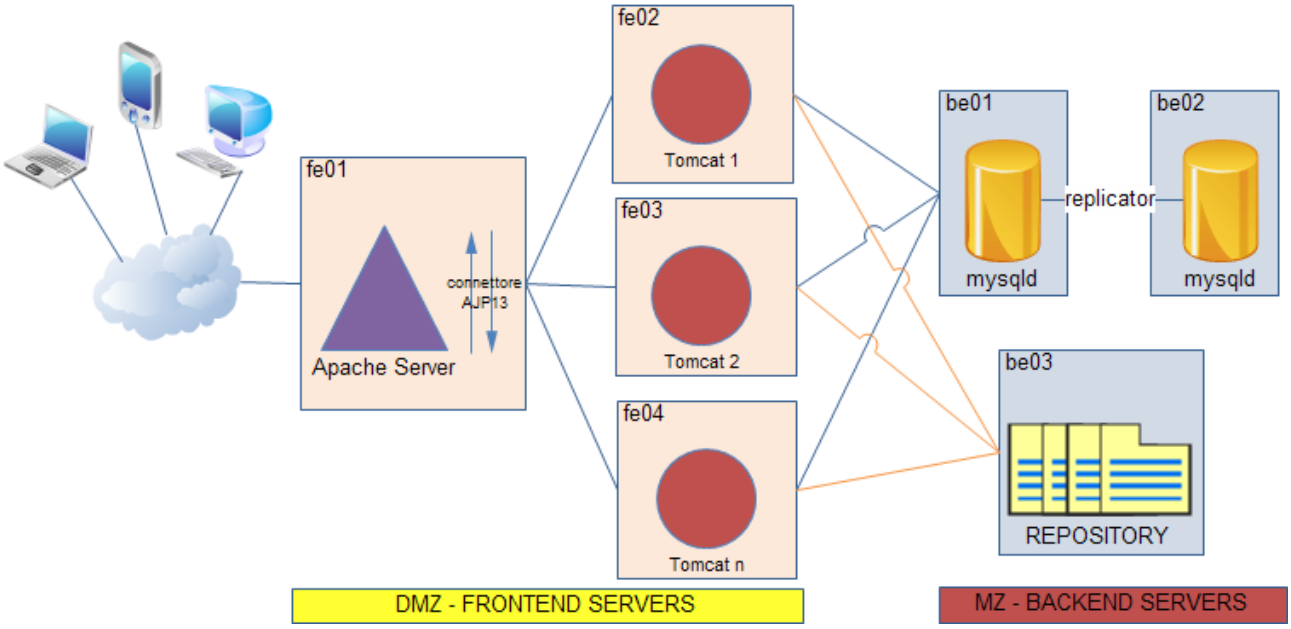


Per il momento, pur non entrando nel merito delle tecniche di clustering e bilanciamento del carico, possiamo affermare che un modello così strutturato garantisce un'elevata scalabilità dei servizi.

Volendo migliorarlo ulteriormente, si potrebbe pensare di ottimizzare la distribuzione delle applicazioni all'interno dell'architettura. Guardando il diagramma precedente, ci si accorge che a ogni Tomcat corrisponde un insieme di applicazioni, ovvero che ogni applicazione è replicata per ciascun Tomcat del cluster. Ciò oltre a introdurre un'elevata ridondanza, rende estremamente macchinose le operazioni di deploy, in quanto si ha la necessità di replicare manualmente qualunque operazione per tutti i Tomcat, pena il disallineamento delle informazioni nel sistema. Se è vero che uno scenario di questo tipo può essere considerato tollerabile in un piccolo ambiente di produzione, è altrettanto vero che in un'architettura con un elevato numero di nodi o applicazioni, diventa estremamente complicata qualunque operazione di manutenzione delle webapps.

L'idea, dunque, è quella di centralizzare le applicazioni, creando un vero e proprio repository da cui i tomcat possano attingere. Con un sistema di questo tipo, inserendo, per esempio, una versione aggiornata di una webapp all'interno del deposito unico, le modifiche sarebbero immediatamente disponibili per tutti i Tomcat. L'unico problema di questo modello è che, mentre i tomcat continuano a rispettare i principi di scalabilità, le applicazioni rischiano di essere indisponibili completamente nel caso in cui dovesse passare in fault la macchina di repository. Tuttavia il problema si risolve eleggendo come macchina di repository una all'interno della MZ, quindi un nodo altamente affidabile dell'architettura.

Il modello astratto finale è il seguente:



3 Architettura Applicativa

una volta disposti i vari componenti middleware, il passo successivo è stato progettare e implementare l'architettura logica e applicativa. Oltre alle scelte architetturali, è stato necessario frazionare le risorse e disporle in base alle esigenze tecnologiche e di traffico delle applicazioni da sviluppare. In questo capitolo e per ogni suo paragrafo, verranno chiariti i concetti di **scalabilità, sicurezza, disponibilità e affidabilità** e di come questi si siano realizzati attraverso politiche di bilanciamento del carico, clustering, riciclo delle sessioni, replica e backuping dei database e trasporto sicuro delle informazioni (HTTPS). Verrà, inoltre, illustrata l'idea di sleeper applicativo secondo cui in un cluster composto da n server applicativi ve n'è almeno uno inattivo che entra in funzione soltanto quando almeno uno di quelli attivi non passa in stato di fault, garantendo sempre e comunque una ripartizione equa del carico tra i componenti. Nel resto del capitolo questi concetti, insieme a quelli del capitolo precedente, serviranno a illustrare l'architettura implementata per le applicazioni sviluppate. In particolare sono state tracciate due distinte geometrie applicative, la prima volta a ospitare l'IMS e la seconda i siti istituzionali di nuova e vecchia implementazione.

3.1 Protocollo AJP13 e tecniche di bilanciamento

3.2 Cluster IMS

3.3 Cluster Siti Istituzionali

3.4 DBMS Replicator e Database Server Backup

4 Tuning Apache HTTP Server

Apache HTTP Server è la più diffusa piattaforma server Web che realizza le funzioni di trasporto delle informazioni via HTTP e HTTPS, di internetwork e di collegamento.
Nella nostra architettura sono state utilizzate due istanze di Apache 2.2, la prima per

5 Tuning Apache Tomcat

Apache Tomcat è un web container open source sviluppato dalla Apache Software Foundation. Implementa le specifiche JSP e Servlet di Sun Microsystems, fornendo una piattaforma per l'esecuzione di applicazioni web sviluppate in Java. Tomcat è uno strumento altamente dinamico e affidabile che può essere utilizzato anche come server web. Tra le sue caratteristiche più interessanti vi è la possibilità di predisporne più istanze in cluster, implementando sofisticate tecniche di bilanciamento, replica e riciclo delle sessioni. Il modello architetturale definito per l'ambiente di produzione ci ha spinti ad approfondire e adottare tali funzionalità e in questo capitolo verranno approfonditi questi aspetti, riprendendoli anche dal capitolo precedente, per ripercorrere i passaggi più importanti dello sviluppo del cluster applicativo.

5.1 mod_jk e configurazione dei workers

5.2 Clustering, riciclo delle sessioni e load balancing

5.3 Sleepers

5.4 Configurazione del connettore Https nel cluster IMS

5.5 Configurazioni aggiuntive

6 Tuning JVM

Uno degli aspetti più critici nella configurazione di un Container J2EE quale Tomcat è l'ottimizzazione della JVM e dell'utilizzo che fa della memoria. Difatti uno dei motivi di blocco più frequenti per un container è la saturazione della Heap Memory utilizzata dalla JVM con conseguente irreperibilità delle applicazioni deployate. In questo capitolo verranno illustrati i concetti cardine di Java Heap Memory e Garbage Collecting, nonchè le tecniche adottate per prevenire problemi derivanti da una cattiva gestioni degli stessi.

6.1 Java Heap Memory

6.2 Garbage Collector e algoritmi di garbage collecting

6.3 JVM monitoring

6.4 Heap Dump

7 DBMS

Altro aspetto fondamentale dell'architettura è quello legato alla persistenza dei dati. La scelta del DBMS è ricaduta su MySql Community 5.1, in assoluto uno dei più noti e utilizzati RDBMS. Al termine dell'installazione, MySql si presenta con due componenti, un client testuale accessibile da locale e un server che rappresenta il motore del DBMS. La versione Community, tuttavia, richiede maggiore attenzione nella configurazione, essendo predisposta per scenari produttivi di piccole dimensioni. In questo capitolo verranno illustrate le modifiche apportate alla configurazione del server MySql (mysqld) per consentirgli sia di sostenere pienamente il traffico di dati all'interno del sistema sia di soddisfare le esigenze applicative delle webapps. Altro aspetto fondamentale nella gestione di un DBMS è la salvaguardia dell'integrità dei dati. Potrebbe infatti capitare che il DBMS o la macchina ospitante passino in stato di fault e che i tempi di ripristino siano abbastanza lunghi. Se è vero che uno scenario del genere può essere considerato accettabile per un server applicativo in quanto le richieste vengono comunque servite dagli altri containers nel cluster, è altrettanto vero che per un DBMS un lungo periodo di inattività può significare il blocco totale di qualunque servizio web fino al momento del ripristino del sistema. Data tale criticità, si è pensato di utilizzare un secondo dbms su un'altra macchina, mappando i db in

produzione e replicando in tempo reale tutti i dati inseriti, eliminati o aggiornati sul dbms principale. Per fare ciò si è utilizzato Tungsten Replicator, un componente open source che sincronizza due o più DBMS in configurazione Master-Slave, consentendo di avere sempre e in qualunque istante una copia aggiornata dei database in produzione.

7.1 Tuning MySql

7.2 Tungsten Replicator

8 Monitoring

Un'architettura complessa richiede un costante monitoraggio dei componenti fisici e applicativi. Ciò può essere fatto manualmente, ma in uno scenario produttivo, per mantenere un alto livello di efficienza del sistema, sarebbe più opportuno affidare tale attività a un sistema automatico. Pertanto si è ritenuto opportuno utilizzare Nagios per espletare i compiti di monitoraggio. **Nagios** è una nota applicazione open source per il monitoraggio di computer, risorse di rete e processi. La sua funzione base è quella di controllare nodi, reti e servizi specificati, avvertendo quando questi non garantiscono il loro servizio o quando ritornano attivi. Un'altra peculiarità di Nagios è la possibilità di riattivare automaticamente una risorsa quando questa passa in stato di fault. Nagios è sviluppato in maniera modulare, dando così l'opportunità di utilizzare plugins, comandi e addons di terze parti. Nel nostro caso abbiamo utilizzato una serie di commands esterni per il monitoring di risorse specifiche, un addon, pnp4nagios, per la generazione di grafici dei servizi in formato web e pdf e un componente complesso, check_mk, per il monitoring remoto delle risorse.

8.1 Introduzione a Nagios

8.2 Host, servizi, comandi e event handlers

8.3 pnp4nagios

8.4 check_mk e check_mk_agent

8.5 Un caso d'uso: monitoring di tomcat

8.6 Compatibilità e dipendenze

9 Test di carico (Stress Test)

Lo “stress test” ha lo scopo di provare a “rompere” il sistema, sovraccaricando o sottraendo le sue risorse. Lo scopo principale di queste attività è verificare che il sistema, una volta in produzione, sia in grado di reggere un'elevata quantità di traffico. Nel caso di fault, poi, in congiunzione con le politiche di monitoring, il sistema dovrà essere capace di ripristinare autonomamente le funzionalità dei processi. Per i

test abbiamo utilizzato Jmeter, un progetto dell'Apache Software Foundation che può essere utilizzato come tool per test di carico, capace di analizzare e misurare le prestazioni di un'ampia varietà di servizi, con un focus particolare sulle applicazioni web.

9.1 Test prestazionali sui siti istituzionali

10 Analisi dei siti

Altro elemento di assoluta importanza nel periodo di post produzione di un sistema web è la presenza di un tool di analisi del traffico. Vagliando i vari strumenti offerti dal mondo free e open source, la scelta è ricaduta su Google Analytics. Basato su Urchin, altro prodotto della famiglia Google, Analytics è un servizio gratuiti che genera le statistiche di traffico per siti web. Il software web-based è in grado di monitorare gli accessi a un sito, estrapolandone tutta una serie di informazioni utili a generare report dettagli e di varia natura. Analytics può offrire informazioni sui browser utilizzati, le sorgenti di traffico (motori di ricerca, link, accessi diretti e altro), i domini e le reti di provenienza, le località geografiche e molto altro ancora. Il software offre inoltre la possibilità di personalizzare il monitoring, dando così l'opportunità di mantenere sotto osservazione solo quegli aspetti ritenuti di maggiore importanza e, conseguentemente, di generare report ad hoc, inviandoli poi a uno o più amministratori tramite posta elettronica. Nel resto del capitolo verrà illustrata nel dettaglio l'attività di Google Analytics sui portali dell'infrastruttura.

10.1 Google Analytics sui siti istituzionali

11 Tavole

11.1 Elenco delle applicazioni (repository e deploy)

11.2 Databases e utenti

11.3 Elenco degli script principali

11.4 Utenti SO e permessi

11.5 Utenti applicativi

11.6 Servizi di Nagios

11.7 Event Handlers di Nagios