Alta Affidabilità

Shibboleth @ Unipi

Samuele Tognini

Utenti

Totale: ~ 120.000

Alumn: ~ 60.000

- Ex-Studenti: servizi post-laurea

Personale T/A: ~1400

Personale Ricerca : ~4500

- Ordinari, Associati, Ricercatori, Assegnisti, Dottorandi, Specializzandi

Studenti in corso: ~55000

Servizi Saml (accessi Gennaio 2014)

Federati: ~2500

- Riviste Elettroniche, Wifi Federato, FileSender Garr

Interni: ~25000

- 16356: Esami (Applicazione rivolta agli studenti e docenti per la registrazione degli esami)
- 7732: Titulus (Protocollo Informatico dell'Ateneo)
- 1000: Altri servizi di vario genere (moodle + Webfax di ateneo + infostudente, applicazione rivolta a studenti e ex-studenti)

Servizi critici => Continuità servizio/affidabilità dati => Cluster

Cluster Shibboleth

Documentazione ufficiale:

- https://wiki.shibboleth.net/confluence/display/SHIB2/IdPClusterIntro

Idp è stateful

- informazioni di stato e le sessioni utente in memoria

Cluster Stateful

- Replica memoria fra più nodi
- Failover trasparente: sessione utente mantenuta
- Mantenute tutte le funzionalità shibboleth
- Terracotta

Cluster Stateless

- No replica memoria
- Failover non trasparente: sessione utente persa -> deve riautenticarsi
- Mancano alcune funzionalità: SAML V1, transientID, artifact bindings

Estensioni

- Contributi esterni
- Storage sessione utente per cluster stateless: memcache, mysql
- Risolvono problema perdita sessione utente durante failover
- Non risolvono altre mancanze: supporto Saml v1, transientId, artifact binding

Considerazioni

Terracotta

- Complesso e pesante, non esente da malfunzionamenti critici. Sconsigliato.

Sessione Utente

- Preferibile continuità di servizio alla persistenza della sessione utente

HA vs Failover

- Preferibile alta disponibilità del servizio all'alta affidabilità dei dati

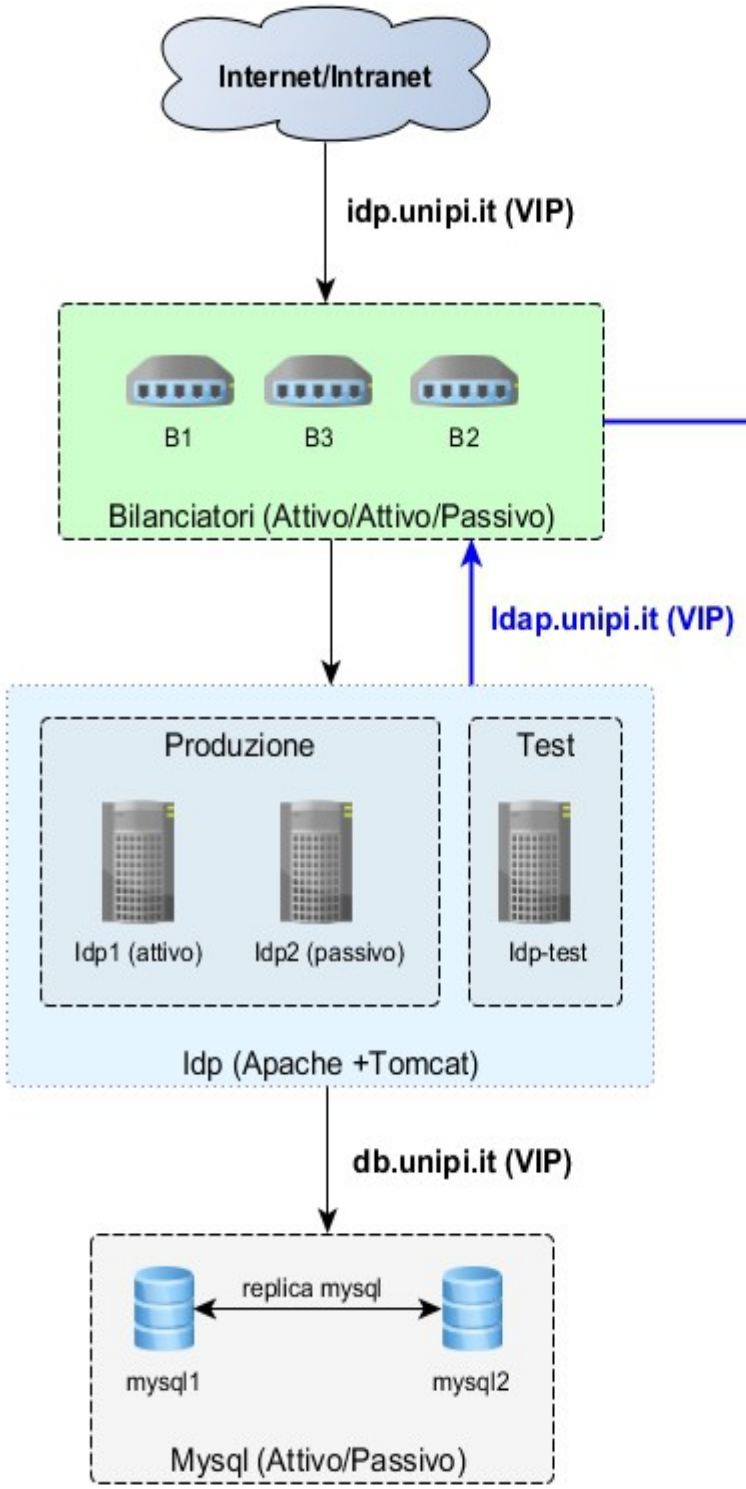
Scalabilità

- Load balancing al momento non necessario

Architettura HA

- già esistente e testata

Cluster Stateless Attivo/Passivo

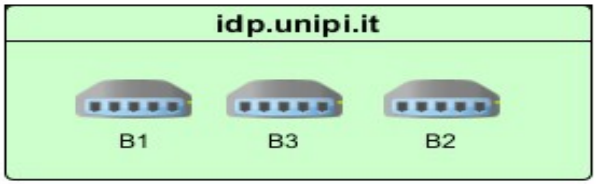


Scelta:

- Stateless: più semplice
- Supporto a tutte le funzionalità di shibboleth
- Flessibile: Ogni componente gestito in modo autonomo
- Tempi di failover generali di pochi secondi
- Failover: perdita sessione utenti
- No load balancing

Alternativa:

- Corosync/pacemaker e macchina virtuale
- Storage condiviso ISCSI o DRBD
- No flessibilità
- Tempi di failover nell'ordine dei minuti



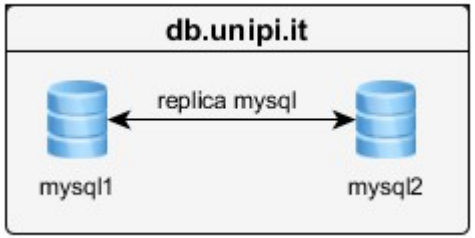
Bilanciatore

LVS

- Sistema linux per creare un bilanciatore layer 4
- Modulo kernel: robusto e performante
- Prende IP dell'Idp (VIP) e ridireziona richieste SAML ad una istanza Idp
- Ldirectord effettua failover su idp in standby (check url 'profile/status')
- Ambiente di test tramite fwmark e iptables
- In cluster HA con corosync/pacemaker
- Possibilità di affinità di sessione per client IP (load balancing)

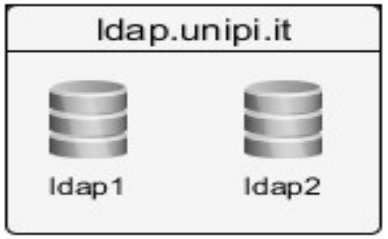
Reverse Proxy (alternativa per load balancing):

- Bilanciatore layer 7: affinità sessione per cookie e offloading SSL
- Valutazione rimandata a quando i contributi saranno ufficializzati



Mysql

- Dati persistenti (persistentID e uApprove)
- Inizialmente: macchina virtuale su ISCSI gestita da corosync/pacemaker
- Ora: soluzione Percona Replication Manager.
  https://github.com/percona/percona-pacemaker-agents
  Resource Agent Pacemaker
- Due o più repliche Mysql e un VIP monitorate da Ra Percona
- RA attiva un master su replica funzionante e gli assegna il VIP
- Tempi di failover: 5-10s
- Problematiche di split-brain (affidabilità dei dati minore)



Ldap

- Backend dati utente
- Pool di consumer Ldap
- Gestito dal bilanciatore (VIP)
- Ambiente di test con fwmark e iptables
- Alternativa: shibboleth configurato con ldap multipli

Alta Affidabilità

Shibboleth @ Unipi

Samuele Tognini

Utenti

Totale: ~ 120.000

Alumn: ~ 60.000

- Ex-Studenti: servizi post-laurea

Personale T/A: ~1400

Personale Ricerca : ~4500

- Ordinari, Associati, Ricercatori, Assegnisti, Dottorandi, Specializzandi

Studenti in corso: ~55000

Servizi Saml (accessi Gennaio 2014)

Federati: ~2500

- Riviste Elettroniche, Wifi Federato, FileSender Garr

Interni: ~25000

- 16356: Esami (Applicazione rivolta agli studenti e docenti per la registrazione degli esami)
- 7732: Titulus (Protocollo Informatico dell'Ateneo)
- 1000: Altri servizi di vario genere (moodle + Webfax di ateneo + infostudente, applicazione rivolta a studenti e ex-studenti)

Servizi critici => Continuità servizio/affidabilità dati => Cluster

SERVIZI SAML

L'università ha iniziato ad utilizzare shibboleth dal 2012 cioè da quando siamo entrati in IDEM e l'utilizzo dei suoi servizi, per quanto non sia dei più elevati, è stato comunque sempre in crescita.

Quindi per quanto riguarda i Service Provider federati:

Sono costituiti principalmente dall' accesso alle riviste elettroniche ma anche da wifi-federato *(unito e cnr)* e filesender del Garr

In numeri stiamo parlando di circa 2500 accessi nel mese di Gennaio 2014

Server Provider interni:

Oltre a servizi federati abbiamo deciso di puntare su SAML anche per servizi web interni e nell'ultimo anno abbiamo iniziato a mettere sotto autenticazione SAML alcune applicazioni, di cui alcune anche critiche.

Sempre prendendo come riferimento Gennaio 2014

16356 | Esami (Un applicazione rivolta agli studenti e docenti per la registrazione degli esami)

7732 | Titulus (Il Protocollo Informatico dell'Ateneo)

1000 | Altri servizi di vario genere (moodle + Webfax di ateneo + infostudente, applicazione rivolta a studenti e ex-studenti)

in tutto si tratta di circa 25000 accessi

Cluster Shibboleth

Documentazione ufficiale:

- https://wiki.shibboleth.net/confluence/display/SHIB2/IdPClusterIntro

Idp è stateful

- informazioni di stato e le sessioni utente in memoria

Cluster Stateful

- Replica memoria fra più nodi
- Failover trasparente: sessione utente mantenuta
- Mantenute tutte le funzionalità shibboleth
- Terracotta

Trovandoci a gestire dei servizi interni critici per noi è stato perciò da subito fondamentale trovare una soluzione che in caso di malfunzionamenti di hardware, di software o della rete garantisse una certa continuità di servizio del nostro Idp, e per far questo il nostro punto di partenza è stata la documentazione di introduzione al cluster shibboleth:
https://wiki.shibboleth.net/confluence/display/SHIB2/IdPClusterIntro

Qui si trovano diverse informazioni interessanti riguardo all'argomento e una di queste è che:
L'attuale implementazione dell' idp shibboleth è stateful, cioè mantiene le informazioni di stato e le sessioni utente in memoria.

Da questo derivano due metodi principali per clusterizzare il servizio:

CLUSTER STATEFUL

In questa implementazione la memoria dell'idp viene replicata in tempo reale fra più nodi.
In questo modo si ottiene prima di tutto un failover trasparente del servizio, nel senso che il guasto ad un nodo per esempio non inficia sulla sessione di un utente che non perderà il single sign on.
Il solo meccanismo di questo tipo raccomandato da shibboleth è Terracotta.

CLUSTER STATELESS
Se non si necessitano di alcune funzionalità di Shibboleth è possibile in alternativa creare un cluster stateless, cioè un cluster i cui dati della memoria non vengono replicati.
In caso di failover cè la perdita della sessione utente che quindi al successivo accesso ad un SP deve riautenticarsi.

ESTENSIONI
Esistono contributi esterni interessanti che possono essere utilizzate per risolvere il problema della perdita di sessione di un cluster stateless, tipicamente permettendo di utilizzare un database esterno (esempio memcache o mysql) come storage delle sessioni:

Queste estensioni non risolvono tutte le mancanze di questo tipo di cluster, ad esempio le query dell'attributo transiente e degli artifact resolution (che rimangono a carico della memoria dell'idp) e al supporto alla vecchia versione di saml.

Cluster Stateless

- No replica memoria
- Failover non trasparente: sessione utente persa -> deve riautenticarsi
- Mancano alcune funzionalità: SAML V1, transientID, artifact bindings

Estensioni

- Contributi esterni
- Storage sessione utente per cluster stateless: memcache, mysql
- Risolvono problema perdita sessione utente durante failover
- Non risolvono altre mancanze: supporto Saml v1, transientId, artifact binding

CONSIDERAZIONI

Terracotta: per noi sconosciuto, leggendo i commenti in giro ci è sembrato uno strumento molto complesso e pesante, e anche poco gestibile (ad esempio è molto legato a certe versioni di tomcat e java), e soprattutto non è esente da malfunzionamenti che rischiano di rendere in alcuni casi il servizio inutilizzabile.

Sessioni utente: Per le nostre esigenze la perdita della sessione utente durante un failover per un guasto hardware o software o per uno switch programmato per manutenzione e' accettabile se comunque il servizio rimane disponibile.
Se in futuro ci sara' un supporto ufficiale alle estensioni di terze parti di cui abbiamo accennato prima sarà comunque semplice implementarle.
per , questo per evitare che il loro utilizzo diventi poco gestibile nel tempo ed evitare per esempio di trovarci in produzione un estensione non più sviluppata.

Scalabilità: Per il carico attuale del nostro Idp inoltre, il load balancing non e' ancora necessario e quindi la scalabilita' del servizio, anche se può essere fatta con l'affinità di sessione, per il momento è stata rimandata.

Avevamo gia' a disposizione un'architettura robusta e flessibile di alta disponibilita' in cui integrare il servizio idp

Considerazioni

Terracotta

- Complesso e pesante, non esente da malfunzionamenti critici. Sconsigliato.

Sessione Utente

- Preferibile continuità di servizio alla persistenza della sessione utente

HA vs Failover

- Preferibile alta disponibilità del servizio all'alta affidabilità dei dati

Scalabilità

- Load balancing al momento non necessario

Architettura HA

- già esistente e testata

ARCHITETTURA

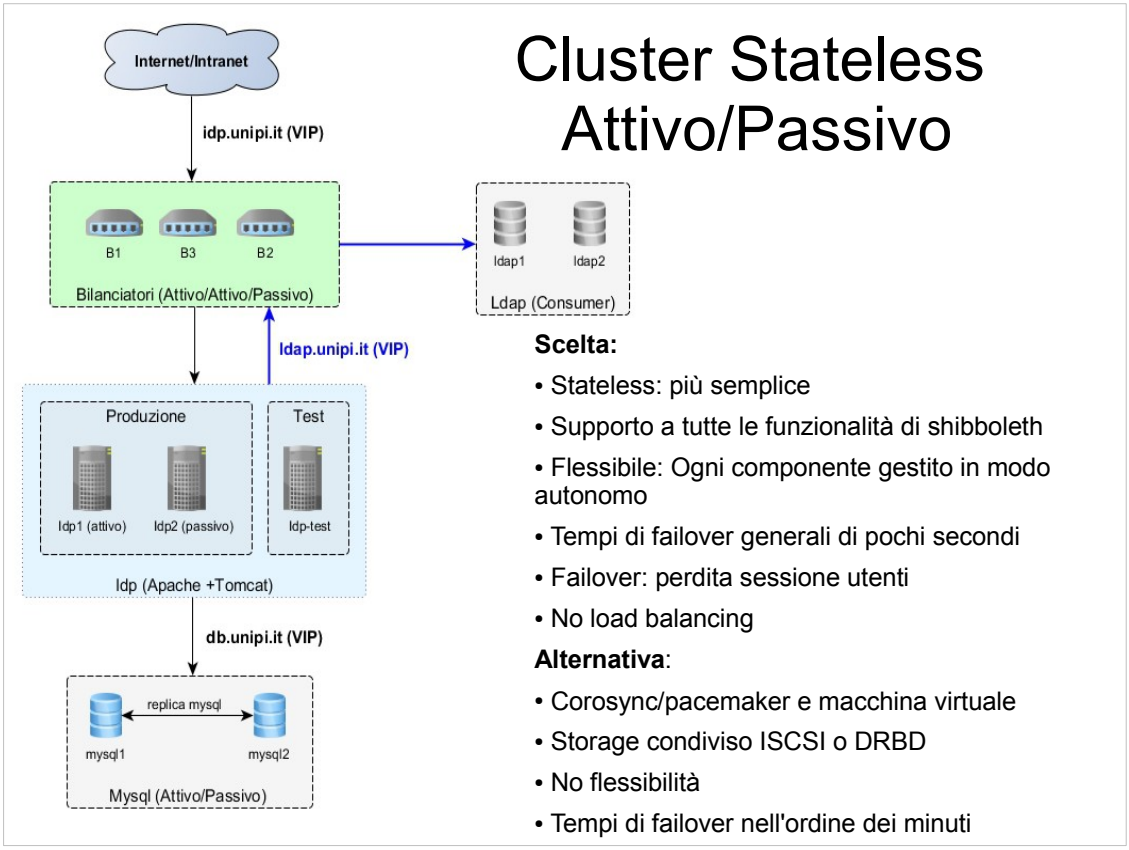
Ovviamente alla fine abbiamo scelto la via per noi piu' semplice, in pratica un cluster stateless in modalita' attiva/passiva rinunciando ad alcune funzionalità in questo momento non necessarie

Questa e' l'architettura attualmente utilizzata:

[Immagine]

Qui si vede che il servizio e' composto da quattro componenti:
1) Un cluster di bilanciatori
2) Delle istanze virtuali di Idp (Apache+Tomcat)
4) Il backend ldap con le informazioni utente
3) Il backend mysql utilizzato da shibboleth per salvare alcuni dati persistenti (come il PersistentID o i dati di uApprove)

ALTERNATIVA:
Un alternativa sempre stateless attiva/passiva, e che si basa su un'architettura che già utilizziamo, per noi sarebbe potuta essere quella di utilizzare una soluzione di failover di una macchina virtuale basata su pacemaker e uno storage condiviso come quello di una SAN o di DRBD ma ci e' sembrato molto piu' flessibile suddividere il servizio in piu' componenti da gestire separatamente, applicando per ognuno la soluzione che piu' ci sembrava consona e potendolo fare solo nel momento in cui si rendesse necessario.



BILANCIATORE

Il bilanciatore è implementato via software con LVS (Linux Virtual Server), un software OpenSource integrato nel kernel linux che viene utilizzato come alternativa economica ma più flessibile rispetto ad un bilanciatore hardware:
http://www.linuxvirtualserver.org

LVS è un bilanciatore di livello 4 OSI, cioè redireziona le richieste tcp/udp che arrivano su determinate porte.
Un ottima alternativa a LVS che un giorno potremmo utilizzare è un bilanciatore layer 7, come il reverse proxy di apache o nginx, che in aggiunta può fare l'offloading ssl (liberando gli idp dal carico ssl) e la persistenza sul cookie di sessione.

Per il momento non ne abbiamo la necessità, ance perchè LVS, per la sua semplicità e perchè compilato come modulo del kernel, è molto più robusto e performante.
Vi rimando comunque all' ottima presentazione dello scorso anno di Fabio Spelta per maggiori dettagli a riguardo all' uso del reverse proxy.

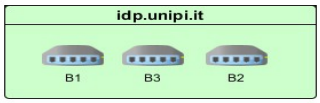
FUNZIONAMENTO
Per quanto riguarda il funzionamento:
Al bilanciatore viene assegnato l'ip pubblico del nostro idp (chiamato in questo caso VIP, virtual ip), in modo che riceva le richieste destinate all'idp.

Al bilanciatore, per ogni vip presente, è inoltre associato un pool di realserver, nel nostro caso delle macchine virtuali, su cui è installato il servizio idp vero e proprio, a cui verranno routate le richieste per la porta https.

La gestione del pool dei realserver viene fatta in automatico da ldirectord, un demone che gira sul bilanciatore e che, verificando la raggiungibilità del servizio erogato dai singoli realserver, rimuove dal pool quelli non funzionanti o ri-aggiunge quelli che tornano a funzionare.

Nel nostro caso, non avendo uno storage comune della memoria, abbiamo deciso di utilizzare un realserver sempre attivo, e un realserver di fallback, utilizzato in caso di malfunzionamento del primo.

Ldirectord per controllare la raggiungibilità del servizio effettua ad intervalli prestabiliti dei controlli sui vari realserver, questi controlli possono essere semplici come il check della porta https, ma anche più complessi.
Nel nostro caso, abbiamo abilitato sull'idp il controllo interno dello stato tramite la url "profile/status".



Bilanciatore

LVS

- Sistema linux per creare un bilanciatore layer 4
- Modulo kernel: robusto e performante
- Prende IP dell'Idp (VIP) e ridireziona richieste SAML ad una istanza Idp
- Ldirectord effettua failover su idp in standby (check url 'profile/status')
- Ambiente di test tramite fwmark e iptables
- In cluster HA con corosync/pacemaker
- Possibilità di affinità di sessione per client IP (load balancing)

Reverse Proxy (alternativa per load balancing):

- Bilanciatore layer 7: affinità sessione per cookie e offloading SSL
- Valutazione rimandata a quando i contributi saranno ufficializzati

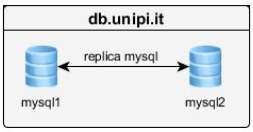
AMBIENTE DI TEST

Tramite il meccanismo del bilanciatore abbiamo inoltre creato un ambiente di test trasparente, nel senso che all'utente non è richiesta nessuna modifica sul proprio computer per utilizzarlo, e questo ci è servito per permettere oltre a noi amministratori anche a utenti esterni di testare le modifiche all'idp prima che fossero messe in produzione.

Per fare questo abbiamo utilizzato la funzionalità FWMARK di LVS e le regole iptables di marcatura dei pacchetti per ridirezionare le richieste generate da specifici indirizzi IP su l'idp di test.

HA BILANCIATORE

Infine il bilanciatore stesso, per evitare che diventi il Single Point of Failure è stato clusterizzato con corosync e pacemaker.
Dato che oltre all'idp abbiamo altri servizi bilanciati e in failover come l'ldap con le informazioni utente, il cluster è stato creato in modo da avere più bilanciatori attivi allo stesso momento e ognuno con la possibilità di prendersi in carico le risorse di un altro bilanciatore che non funzionante.



Mysql

- Dati persistenti (persistentID e uApprove)
- Inizialmente: macchina virtuale su ISCSI gestita da corosync/pacemaker
- Ora: soluzione Percona Replication Manager.
  https://github.com/percona/percona-pacemaker-agents
  Resource Agent Pacemaker
- Due o più repliche Mysql e un VIP monitorate da Ra Percona
- RA attiva un master su replica funzionante e gli assegna il VIP
- Tempi di failover: 5-10s
- Problematiche di split-brain (affidabilità dei dati minore)

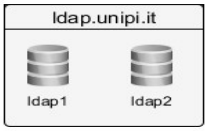
MYSQL

[IMMAGINE]

Nel nostro caso mysql contiene le informazioni persistenti come il persistentID e quelle sul consenso degli attributi gestiti da uApprove.
Inizialmente l'HA mysql era stata fatta sempre con corosync/pacemaker in cui la risorsa gestita era una macchina virtuale su drbd.
In questo caso il cluster era meno affidabile in quanto non aveva visione del servizio mysql.

Ultimamente siamo passati ad una soluzione basata sempre su corosync/pacemaker in cui è il servizio mysql stesso ad essere gestito dal cluster.
Per far questo abbiamo utilizzato un Resource Agent (ocf) di Percona chiamato Percona Replication Manager:
https://github.com/percona/percona-pacemaker-agents

in cui il cluster gestisce sia il VIP del servizio, sia una o più repliche mysql.
Il servizio è monitorato dal cluster che assegna il VIP e promuove a master una replica funzionante.



Ldap

- Backend dati utente
- Pool di consumer Ldap
- Gestito dal bilanciatore (VIP)
- Ambiente di test con fwmark e iptables
- Alternativa: shibboleth configurato con ldap multipli

infine, avremmo potuto far gestire a shibboleth l'alta affidabilità di ldap assegnandogli due consumer ldap da interrogare, ma abbiamo preferito utilizzare nuovamente il bilanciatore anche per poter testare con il solito meccanismo dell'FWMARK degli ldap di test.