A/I Orange Book (1.0)

Un *how-to* sulla replicazione e distribuzione di una rete resistente di server autogestiti

A/I Orange Book (1.0): Un *how-to* sulla replicazione e distribuzione di una rete resistente di server autogestiti

Questo testo descrive un meccanismo di configurazione e gestione di server geograficamente distribuiti, partendo dall'analisi dei requisiti e le loro motivazioni. Il meccanismo descritto utilizza CFengine[1] e altri tool free per aumentare il livello di automazione in modo supervisionato.

Per altre versioni di questo documento consultare l' Appendice C, Sezione 1

"Socializzare saperi, senza fondare poteri."

—Primo Moroni

Sommario

Capitolo 1. Introduzione

1.1. Riflessioni

Alcuni di noi già nel 2003 cominciarono, all'interno del collettivo Autistici/Inventati, a immaginarsi l'ondata repressiva prossima ventura, e a riflettere sulle modalità che avrebbe assunto.

Ci parve allora che la crescente importanza dei mezzi di comunicazione digitali, specie all'interno di contesti politici radicali, avrebbe presto attirato l'attenzione delle forze repressive. Eravamo ottimisti: da allora abbiamo visto la sempre crescente paranoia globale dare nuova spinta alle ideologie orwelliane e panopticiste del controllo totale, che ormai mirano non solo ai comunque ristretti ambiti del dissenso, ma alla società intera.

Ci chiedemmo quali potevano essere i "punti deboli" dei servizi che gestivamo, e quali potevano essere i compromessi da compiere tra le nostre necessità politiche, le energie a disposizione, e il tipo di risposta da preparare di fronte alle minacce prossime venture. Identificammo dunque questi principali problemi:

- La collocazione di server presso provider commerciali non permetteva più di garantire l'integrità e la riservatezza dei dati che vi fossero memorizzati - d'altronde l'alternativa (la collocazione in case o in altri spazi sociali) ci avrebbe prima o poi costretto alla difesa fisica di un luogo o di un oggetto, una posizione non sostenibile e che comunque alla fine non produce un risultato diverso. Dato che i costi impedivano altre soluzioni (come ad esempio cablare con banda sufficiente il proprio bunker personale), bisognava dare per scontata l'insicurezza fisica delle macchine.
- Il parziale successo del nostro progetto fino ad allora (almeno relativamente alla diffusione) ci aveva messo nella posizione imbarazzante di accentrare su di un'unica macchina un numero eccessivo di risorse personali, come le caselle di posta ad esempio (questo è un problema solo se appunto valutato alla luce del parziale fallimento del nostro progetto sui temi della protezione individuale dei propri dati riservati, ad esempio con gpg, e più in generale della formazione a una cultura dell'autodifesa delle proprie libertà fondamentali). Lo stesso elevato numero di utenti sconsigliava l'adozione di tecniche di crittografia dei dischi, che ci avrebbero messo nella scomoda posizione di detenere un'unica chiave per l'accesso ai dati di migliaia e migliaia di persone diverse. Era perciò evidente la necessità di una decentralizzazione, anche come meccanismo per ridurre il numero medio di utenti su una singola macchina.
- L'evoluzione dello scenario legale, nazionale ed internazionale, specie rispetto ai casi di sequestro di materiale informatico, sembrava dunque mettere sempre più a rischio la possibilità di garantire l'integrità dei dati di chicchessia. Rimaneva solo da cercare di garantire alle persone la *possibilità di comunicare*, comunque ed in ogni caso.

Questa riflessione ha prodotto molti risultati, uno dei quali è questo documento: la descrizione tecnica di un'infrastruttura per la comunicazione anonima utilizzando un numero di server dislocati in tutto il mondo, in una situazione in cui si dia per scontata l'insicurezza fisica delle macchine, e ciascuno dei "nodi" di questa Rete possa essere considerato sostituibile.

1.2. Linee guida

La configurazione descritta in questo documento è stata progettata e realizzata in funzione di linee guida che si crede possano rispondere con successo alle necessità appena indicate. Queste sono precisamente:

- Deve essere possibile sostituire rapidamente la macchina che ha il ruolo di "master" con un'altra equivalente: per questo motivo tutta la configurazione "statica" (i files in `/etc` per esempio, ma anche i database delle utenze e delle mailbox, i siti web, gli archivi ftp) è replicata su tutti i server, uno dei quali viene dichiarato essere il principale. La presenza di una copia aggiornata dappertutto permette di sostituire questa macchina con una modifica molto rapida.
- I dati particolarmente "sensibili" (come le mailbox, o le mailing list) vanno invece suddivisi tra le varie macchine, anche casualmente, il criterio in realtà non è molto importante dato che non è il load balancing che si vuole ottenere. Deve comunque essere possibile modificare, nei database, l'assegnazione di una mailbox ad una particolare macchina, nel caso in cui quest'ultima non sia più operativa, nel tempo più breve possibile. Recuperare i dati perduti sulla prima macchina *non* è considerata una priorità.
- Perché un modello di questo genere sia efficace, è necessario che tutte le macchine coinvolte siano fondamentalmente simili, configurate allo stesso modo, e intercambiabili (e, presumibilmente, gestite centralmente dalle stesse persone).

Questo documento descrive tre cose sostanzialmente differenti, benché profondamente correlate tra loro:

1. la configurazione dei vari servizi di comunicazione (posta, web, mailing list) in modo da supportare la suddivisione degli utenti per macchina mantenendo un'unica "presenza" virtuale (gli indirizzi di posta, ad esempio, saranno @dominio.org anziché @server1.dominio.org, @server2.dominio.org, etc.) di modo che l'esperienza dell'utente sia comunque quella di aver a che fare con un oggetto "unico";
2. l'implementazione di un meccanismo per gestire centralmente tutte queste configurazioni e manipolarle non più al livello della singola macchina ma utilizzando una maggiore astrazione (affinché la complessità di gestione risulti semplificata rispetto alla gestione diretta di N singole macchine);
3. gli accorgimenti che è necessario adoperare per rendere anonime le operazioni effettuate mediante tale infrastruttura.

Ovviamente ci sono molti modi per realizzare quanto descritto, quello da noi prescelto è solamente uno tra i tanti: non intendiamo proporlo come migliore, o neanche come esempio di pratiche corrette. In generale, ci siamo fatti guidare nelle nostre scelte dalle considerazioni esposte qua sopra, dalla nostra esperienza collettiva e eventualmente dalla maggiore familiarità con alcuni strumenti piuttosto che altri.

Tutto il software utilizzato è Free Software (l'infrastruttura si basa su sistemi Debian[1]).

I prossimi capitoli introdurranno ciascuno la configurazione di una parte del software.

Note

1. http://www.debian.org/

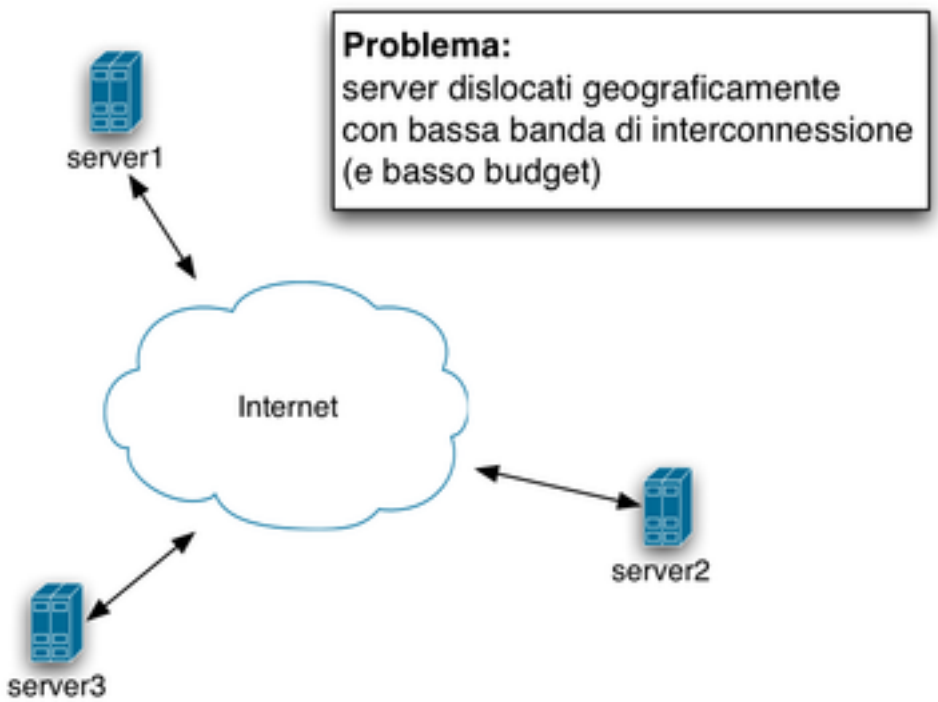
Capitolo 2. Rete

Questo capitolo illustra brevemente la configurazione della rete che definisce l'*organizzazione virtuale*. La conformazione della rete, e le caratteristiche che ne conseguono, hanno determinato in modo fondamentale le scelte che sono state fatte successivamente, dunque è necessario partire da qui per comprendere il resto dell'infrastruttura.

La configurazione di rete è mantenuta sui vari server utilizzando CFengine (vedi il Capitolo 6).

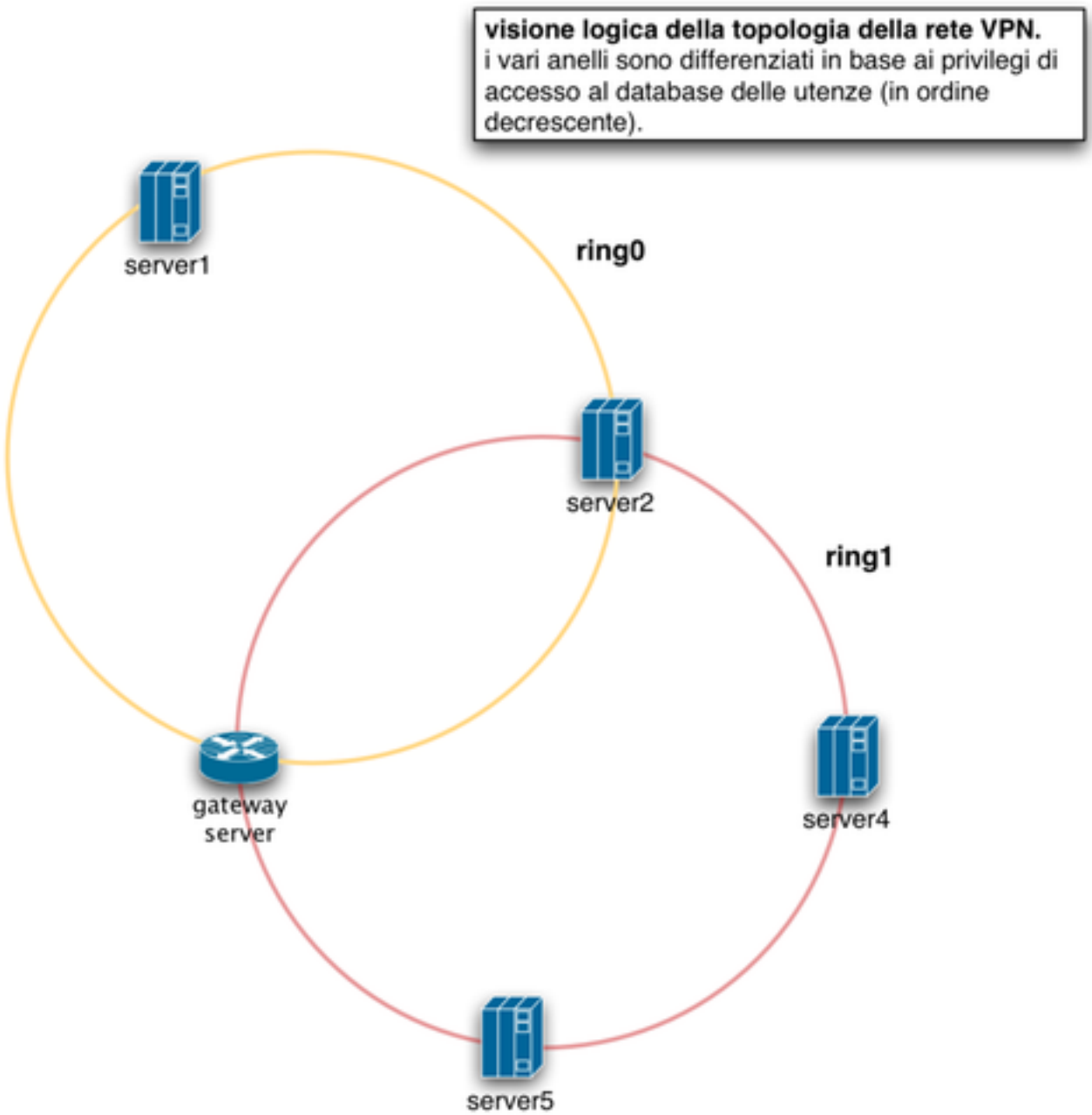
2.1. Struttura

I vari server sono dislocati geograficamente in luoghi differenti, e sono dunque raggiungibili ciascuno con il suo specifico IP pubblico. Caratteristica determinante di questa configurazione è la bassa banda di interconnessione tra i vari server: la dislocazione geografica fa sì che il canale di comunicazione interno competa per le risorse con la comunicazione normale (si veda l'Appendice B per uno studio più approfondito). Ciò rappresenta la maggior differenza tra il modello discusso in questo documento e le soluzioni tradizionali di clustering e high availability, che (generalmente pensate per un ambito più "aziendale"), presuppongono invece una capacità di interconnessione arbitrariamente elevata (locale).



Struttura reale della rete

Il canale di comunicazione interno è realizzato con una VPN (utilizzando il software Tinc), che collega i vari nodi permettendo una comunicazione cifrata senza l'overhead di una negoziazione SSL per ciascuna connessione.



Schema delle connessioni logiche tra i vari server

La VPN è considerata *trusted*: la sola presenza sulla VPN è indice di affidabilità, inoltre le singole connessioni su questo anello non hanno strettamente bisogno di essere cifrate (già provvede tinc ad un livello più basso). Molti servizi amministrativi interni (oltre ai meccanismi di sincronizzazione) sono disponibili solo su queste interfacce.

Lo schema completo della rete comprende anche svariate altre macchine, organizzate in *anelli* concentrici e intercomunicanti, corrispondenti a vari gradi (decrescenti) di affidabilità, per cui, tanto per fare un esempio, già le macchine dell'anello 1 (la numerazione parte da 0) hanno un accesso ristretto al database delle utenze LDAP. Queste macchine

sono utilizzate per servizi di minore importanza, per i backup, e altri compiti ausiliari come ad esempio fare da gateway per le connessioni ssh e VPN degli amministratori (l'avere un singolo punto di entrata non pubblico e il traffico successivo instradato attraverso le vpn in modo non tracciabile permette di mantenere un relativo anonimato per gli amministratori medesimi).

Nei punti di interconnessione tra i vari anelli sono presenti dei firewall per regolare i flussi tra un anello e l'altro. Un'altra possibilità che non abbiamo ancora esplorato è quella di avere delle macchine dedicate a questo scopo (il "gateway server" nella figura) anziché addossare questo ruolo agli stessi server. I nodi di connessione dovrebbero poter essere molteplici: stiamo studiando il modo migliore per instradare correttamente i pacchetti tra i due anelli anche se uno dei nodi non dovesse essere più raggiungibile.

2.1.1. VPN

Tinc[1] è un demone VPN con routing automatico in grado di mantenere connessi N host con una serie di connessioni punto-punto gestite automaticamente. Inoltre effettua l'autenticazione e la crittazione basandosi su un suo set autonomo di chiavi RSA, che quindi vanno distribuite opportunamente.

È stato scelto, rispetto ad altre soluzioni, perché è molto facile da installare, e permette di nascondere le varie connessioni punto-punto tra ciascuna macchina: presenta al sistema operativo direttamente un'unica interfaccia di rete, gestendo il routing in proprio in maniera trasparente (per essere più chiari, è come se le macchine fossero tutte collegate in rete locale - anzi meglio perché si occupa automaticamente di cercare percorsi alternativi nel caso la connessione diretta tra due macchine venga a mancare).

Abbiamo scelto per questo, nei file di configurazione di esempio, la rete 172.16.0.0/16, assegnando i seguenti indirizzi IP (che saranno poi aggiunti al file `/etc/hosts` distribuito centralmente):

```
test1     172.16.1.1
test2     172.16.1.2
test3     172.16.1.3
```

Per comodità, è utile scegliere dei nomi per questi IP che siano chiaramente differenti rispetto a quelli usati per gli IP pubblici delle stesse macchine: per esempio utilizzando un sottodominio (`test1.vpn.dominio.org`) o un suffisso comune (`test1-vpn.dominio.org`).

Per cominciare, è stata creata la configurazione base per una rete, di nome *ring0* (che corrisponde all'anello più interno dello schema di rete privata basato sul trust), in `/configfiles/ring0/common/tinc/ring0`, in particolare (a parte `tinc-up` e `tinc-down`) il file di configurazione principale `tinc.conf`:

```
# Sample tinc configuration file

# The name of the machine
# Name = test1

# Connections to be done
ConnectTo = test1
ConnectTo = test2
ConnectTo = test3

# The tap device to use
Device = /dev/net/tun
```

Nota: i nomi dei computer usati in questo file non devono necessariamente essere validi nel DNS, ma corrispondono ai files nella directory `ring0/hosts`. E' per la comodità di gestire la cosa con CFengine che i nomi scelti sono uguali a quelli degli host.

La directory `/configfiles/ring0/common/tinc/ring0/hosts` deve contenere due file per ciascun server, identificati con il loro nome senza suffisso di dominio, `NOME` e `NOME.priv`. Le chiavi RSA si possono generare con

```
$ tincd -n ring0 -K
```

Bisogna poi editare il file con la chiave pubblica per aggiungervi delle opzioni, questo è un esempio:

```
# The real IP address of this tinc host. Can be used by other tinc hosts.
Address = 192.168.1.21

# Portnumber for incoming connections. Default is 655.
Port = 655

# Subnet on the virtual private network that is local for this host.
Subnet = 172.16.1.2/32
# Connection with the other ring
# Subnet = 172.17.0.0/16

#IndirectData = Yes
#TCPOnly = Yes

# The public key generated by 'tincd -n example -K' is stored here
-----BEGIN RSA PUBLIC KEY-----
MIGJAoGBAJGedJTd4GnIe1VssM+ROBwsMzRXbdI/reZvkLmji3YK0HJcyDIKnRZ2
/ikPJNyH1bKSWlqds28j4eG6ENM5ZjaWDETztW6OyNOt4vDxAXEQRF50WLBL5BOk
e6bXlPoOtXWrVK/ZpBiDl86XpEdm0DHhETB2Cit9KNAXcW2ajnabAgMBAAE=
-----END RSA PUBLIC KEY-----
```

2.2. Organizzazione del DNS

Per il funzionamento di uno schema come quello che stiamo qui descrivendo, una delle parti più essenziali è proprio il servizio DNS.

Nell'organizzazione del DNS è evidente la separazione tra il livello di gestione e i servizi destinati agli utenti: la struttura di rete sopra descritta richiede una nomenclatura efficace degli indirizzi assegnati alle varie interfacce, in modo che sia sempre chiaro a quale connessione si fa riferimento. D'altra parte, dal punto di vista degli utenti è invece auspicabile avere meno specificità possibile, per mantenere l'impressione di aver a che fare con un'unica "organizzazione virtuale" (vale a dire che tutta l'infrastruttura si colloca sotto la soglia della percezione dell'utente che si collega a www.dominio.org e non vuole davvero sapere a quale macchina specifica si stia collegando, o se la sua posta si trova in Brasile o in Malesia).

2.2.1. Dominio infrastrutturale

La scelta per la parte infrastrutturale è stata di usare un dominio apposito (nel nostro caso infra.org), su cui appoggiare la risoluzione degli IP delle varie macchine e di altri servizi che eventualmente non dovessero essere pubblici. Dato che su questo dominio non vengono offerti servizi agli utenti, non è neanche necessario che tale dominio sia pubblico (o meglio, pubblicamente conosciuto). Questa separazione ovviamente non è assolutamente necessaria, ma può aiutare a mantenere chiarezza nell'organizzazione dei servizi.

Questa più o meno è la struttura dei record DNS di questa zona:

```
infra.org                   NS      ns1.infra.org
                                    ns2.infra.org
                                    ns3.infra.org
ns1.infra.org               A       1.2.3.4
ns2.infra.org               A       2.3.4.5
ns3.infra.org               A       3.4.5.6
mx1.infra.org               A       1.2.3.4
mx2.infra.org               A       2.3.4.5
mx3.infra.org               A       3.4.5.6
server1.infra.org           A       1.2.3.4
server2.infra.org           A       2.3.4.5
server3.infra.org           A       3.4.5.6
server1-vpn.infra.org       A       172.16.1.1
server2-vpn.infra.org       A       172.16.1.2
server3-vpn.infra.org       A       172.16.1.3
```

È abbastanza facile aggiungere a questa zona degli alias per alcuni servizi o ruoli particolari, per esempio:

```
ldap-master.infra.org       CNAME   server1
```

2.2.2. Altri domini

La struttura del DNS dei domini "normali" invece è standardizzata (tutte le zone cioè sono fatte nello stesso modo), ed essenzialmente implementa tutti i meccanismi di round-robin con cui il carico viene approssivatimanente distribuito tra i vari server.

La parte comune a tutti i domini è fatta in questo modo:

```
public.org                NS      ns1.infra.org
                             NS      ns2.infra.org
                             NS      ns3.infra.org
public.org                  MX      10 mx1.infra.org
                             MX      10 mx2.infra.org
                             MX      10 mx3.infra.org
www.public.org               A       1.2.3.4
                             A       2.3.4.5
                             A       3.4.5.6
```

Che significa, brevemente, che la posta viene indirizzata con eguale probabilità ad uno qualsiasi dei server di posta, e le richieste web vengono mandate in modo ugualmente casuale ai vari server web.

Gli unici domini che si differenziano dagli altri in questo senso sono quelli per cui è necessario identificare singolarmente ciascuna macchina. Questa situazione la si riscontra sia nel caso dei sottositi (www.dominio.org/qualcosa), che sono assegnati ad una macchina specifica per via della difficoltà / infattibilità della replicazione MySQL multi-master, sia nel caso di alcuni servizi forniti tramite https (la webmail, ad esempio). Per questo motivo i record A del dominio dominio.org sono fatti così:

```
www.dominio.org              A       1.2.3.4
                             A       2.3.4.5
                             A       3.4.5.6
www1.dominio.org             A       1.2.3.4
www2.dominio.org             A       2.3.4.5
www3.dominio.org             A       3.4.5.6
```

Un'ultima cosa da dire riguardo all'organizzazione del DNS (cosa che verrà approfondita nel capitolo dedicato ad Apache) è che è stato deciso di accentrare tutti i servizi web forniti su SSL su di un unico dominio (in questo caso www.dominio.org), per mantenere la coerenza con il nome indicato sul certificato SSL (che comunque è unico, per ciascun server).

Note

1. http://www.tinc-vpn.org/

Capitolo 3. Filesystem

L'organizzazione del filesystem è almeno in parte comune a tutte le macchine. Ciò permette per esempio per poter facilmente spostare servizi ed altro da una macchina all'altra senza dover cambiare tutti i path, oppure di memorizzare alcuni path specifici direttamente nel database, come le home degli utenti o le root dei siti web.

3.1. Organizzazione dei dischi

Identificare uno schema comune di partizionamento, ovviamente modificabile in casi specifici, semplifica la gestione e velocizza l'installazione di nuove macchine.

Dove possibile è opportuno configurare le macchine in modalità RAID e, a livello superiore, usare LVM per gestire lo spazio con maggiore flessibilità. Sotto potete trovare un banale esempio di partizionamento a cui ci siamo adeguati (ma lo schema è molto dipendente dalle vostre situazioni in termini di hardware, necessità di spazio disco e disponibilità economiche).

```
/               - 500M
/tmp            - 500M noexec,nosuid
/usr            - 3G
/var            - 6-10G
/home/admins    - 4G, home degli amministratori
/home/mail      - >20G, caselle di posta
/home/users     - >20G, siti web ed ftp
```

Trattandosi poi di server che principalmente gestiranno un notevole traffico di posta, è anche consigliabile applicare alcune ottimizzazioni, ad esempio, a Postfix:

```
chattr -R -S +j +A /var/spool/postfix/
```

che abilita il journaling completo dei dati e dei metadati del filesystem ext3 sulla directory di spool di Postfix, oltre a impostare il flag *noatime* (esempio tratto da "Postfix on an ext3 filesystem", R. Hildebrandt[1]).

Un'altra cosa utile è far sì che Amavis[2] utilizzi una directory temporanea in un filesystem *tmpfs*, quando ha necessità di salvare un file con allegati MIME nelle sue diverse parti. Questo riduce ancora l'impatto sui dischi. In `/etc/init.d/amavis` si può mettere:

```
mkdir /dev/shm/amavis
test -e /var/lib/amavis/tmp \
  || ln -s /var/lib/amavis/ /dev/shm/amavis
```

e modificare `/etc/amavis/amavisd.conf`:

```
$TEMPBASE = "$MYHOME/tmp";
```

Infine, alcuni programmi devono essere compilati, la strategia migliore in questo caso è creare pacchetti debian binari, anche per agevolare la distribuzione sulle diverse macchine (non sarà necessario compilare sempre tutto dappertutto). Sempre per questo motivo, potrebbe essere comodo trovare dei percorsi standardizzati per mettervi programmi, librerie, script, o files non-binari che magari possono essere poi facilmente distribuiti con rsync. Una possibilità è di usare `/opt` per questo.

3.1.1. Partizioni crittografate

L'utilizzo di partizioni crittografate è stato scartato relativamente ai dati degli utenti, dato che, aldilà delle considerazioni di performance, non è comunque saggio mettersi nella condizione di possedere un'unica chiave per migliaia e migliaia di differenti porte. In compenso rimane una strategia valida per proteggere quelli che sono i dati sensibili del sistema nel suo complesso: nel caso di una struttura di questo tipo tanto per cominciare sono i certificati SSL e le relative chiavi crittografiche.

Questo tipo di meccanismo richiede l'inserimento della password della partizione all'avvio del computer, e serve essenzialmente a prevenire la copia o la modifica non autorizzata dei dati contenuti. Si può anche pensare di montare la partizione con i certificati SSL solo al momento dell'avvio del servizio, e smontarla successivamente: questo dovrebbe rappresentare una protezione anche rispetto ad intrusioni quando la macchina è accesa, ma in misura solo parziale dato che comunque a quel punto diventa anche difficile controllare la sicurezza dell'inserimento della password.

La nostra implementazione prevede l'utilizzo di dm-crypt. Questo più che altro per la semplicità, dovendo cifrare partizioni più grandi o con più I/O avremmo probabilmente utilizzato loop-AES (si trovano dei benchmark interessanti a questa url: http://deb.riseup.net/storage/encryption/benchmarks/).

Per esemplificare la metodologia utilizzata, creiamo l'immagine crittata `private.img` e configuriamola:

```
$ dd if=/dev/urandom of=private.img bs=1024 count=512
$ export CALOOP=`losetup -f`
$ losetup $CALOOP private.img
```

Creiamo ora il mapper device dove creare la partizione cifrata ed inseriamo la passphrase utilizzata per la cifratura simmetrica:

```
$ cryptsetup -v -y -c aes -s 256 -h ripemd160 create private $CALOOP
```

Ora creiamo il filesystem (ext2) e montiamolo su `/mnt`:

```
$ mkfs -t ext2 -v /dev/mapper/private
$ mount  -t ext2 /dev/mapper/private /mnt
```

3.2. Sincronizzazione dei filesystem

Alcuni dati (come ad esempio la gerarchia `/opt` descritta sopra che conterrebbe i contenuti non legati a singoli utenti) vanno sincronizzati tra le varie macchine. Dato che questi dati hanno dimensioni non piccole, e soprattutto non necessitano di modifiche specifiche per ciascuna macchina, si è scelto di *non* usare CFengine per gestirli, ma sistemi più semplici. Purtroppo al momento della realizzazione di questo documento, non abbiamo trovato una soluzione che ci permettesse di usare un vero filesystem condiviso - le peculiarità della nostra situazione (macchine non locali, con bassa banda di interconnessione) infatti non sono facilmente compatibili con il design normale di un filesystem distribuito.

Per copiare questi dati al momento si usa quindi un server `rsync`, non crittato, disponibile su ciascuna macchina all'interno della VPN, soluzione che permette una certa velocità di esecuzione e trasferimento, con basso overhead. Il punto dolente di questa soluzione è che rompe la simmetria delle varie macchine: una infatti deve assumere il ruolo di "originale" da cui poi vengono propagate automaticamente le copie.

Lo stesso meccanismo potrà essere usato per spostare da una macchina ad un'altra un singolo spazio utente o una mailbox. Per evitare di poter fare troppa confusione con un comando sbagliato, gli accessi tramite `rsync` sono read-only, dunque è permesso solo il *pull*.

Note

1. http://www.stahl.bau.tu-bs.de/~hildeb/postfix/postfix_ext3.shtml
2. http://www.ijs.si/software/amavisd/

Capitolo 4. Il database degli utenti

4.1. Introduzione

Questo breve capitolo descrive sinteticamente la struttura del database che mantiene i dati delle utenze sui vari server.

LDAP è un protocollo per accedere a sistemi *"directory based"*, è nato come gateway per lo standard di data repository OSI denominato X.500: infatti LDAP inizialmente definiva soltanto il trasporto (TCP/IP) e il formato dei messaggi utilizzati dai client per accedere a sistemi di directory conformi a X.500, che richiedevano l'intero stack OSI (troppo complesso e costoso per essere implementato su piccole LAN). In sostanza il server LDAP faceva da tramite (gateway) tra un eviroment TCP/IP e uno OSI, raccoglieva le richieste dai client (TCP/IP) e le inoltrava a un server X.500 usando lo stack OSI. A causa della complessità dell'OSI stack e di X.500, col passare del tempo si è pensato di dotare LDAP di un suo sistema di storage directory in modo da renderlo autonomo, e non utilizzarlo solo come gateway per X.500, essendo più snello del noto standard OSI.

La nostra scelta di usare LDAP per la gestione delle utenze email è stata determinata dalle seguenti caratteristiche:

- LDAP è ottimizzato per amministrare dati che richiedono un numero accessi in lettura molto maggiore del numero accessi in scrittura
- LDAP si replica con molta facilità e permette quindi di ridondare velocemente l'intero database.

Al momento della realizzazione iniziale di questo documento non esistevano altre suite LDAP complete e free, quindi abbiamo deciso di utilizzare OpenLDAP[1], benché sia un software che innegabilmente presenta numerosi problemi. In seguito anche altre soluzioni sono diventate disponibili (come ad esempio il Fedora Directory Server[2] di RedHat) e riteniamo che valga senz'altro la pena investigarle dato il ruolo centrale che il server LDAP riveste nella struttura che stiamo descrivendo.

4.2. Struttura del database

Le informazioni in LDAP sono conservate sotto forma di oggetti (entries) caratterizzati da attributi, ogni attributo ha un id ben definito ed univoco detto OID.

Gli oggetti sono mantenuti in modo gerarchico secondo il loro *distinguished name* (abbreviato in `dn`). Gli oggetti "figli" ereditano il dn del genitore ponendolo come suffisso al loro stesso dn. La struttura che ne deriva è dunque disposta ad albero.

Con un albero di questo tipo

```
o=Anarchy
 \-- dc=org
     \-- dc=infra
```

il dn dell'oggetto investici risulta dunque essere

```
dc=infra, dc=org, o=Anarchy
```

Ciascun oggetto può avere differenti attributi a seconda delle *classi* a cui appartiene. L'ereditarietà in LDAP può essere multipla (cioè un oggetto può appartenere a più classi contemporaneamente). Ciascuna classe definisce *attributi*, necessari od opzionali, che devono (o possono) essere presenti nell'oggetto. Queste strutture sono descritte negli *schemi* di LDAP.

4.2.1. Contenuto del database

Nella struttura che abbiamo adottato, per isolare l'albero con i nostri dati da quelli di eventuali altre virtual organizations come la nostra, i dati relativi al nostro sistema saranno interamente collocati al di sotto dell'oggetto `dc=infra, dc=org, o=Anarchy` (questa è una nomenclatura standard).

Il database comprende varie tipologie differenti di utenze e di altri oggetti, memorizzati in rami differenti dell'albero LDAP collocati al di sotto di `dc=infra, dc=org, o=Anarchy`:

- `ou=Admins`: questo albero contiene gli amministratori delle macchine, cioe' gli utenti che hanno un accesso in ssh. *NOTA*: i compiti amministrativi che non comportano l'uso della shell (per esempio il pannello di amministrazione via web) sono autenticati su degli altri oggetti identici a questi ma con password diversa, collocati sotto l'albero `ou=People`. Questo vale a dire che l'accesso tramite SSH è separato da tutti gli altri per quanto riguarda l'autenticazione.
- `ou=People`: sono tutti gli utenti email, ftp. etc .. questi costituiscono la parte "virtuale" del db LDAP.
- `ou=Operators`: contiene gli utenti "di sistema" per far manipolare parti del database ldap al software senza dover necessariamente avere tutti i privilegi del manager LDAP. E' l'utente prevalentemente utilizzato dai demoni che accedono al database.
- `ou=Domains`: contiene un oggetto per ciascun dominio virtuale gestito dalla struttura. Postfix, ad esempio, legge questo sottoalbero per determinare se un dominio è locale oppure no (nella costruzione delle tabelle di transport).

4.2.2. Utenze virtuali

Intendiamo qui di seguito "utente" nel significato di "contatto", concetto che introduciamo per poter raggruppare i servizi che fanno capo ad un'unica entità, sia persona che organizzazione o altro, scelta in modo da agevolare l'amministrazione dei servizi.

Ciascun utente è rappresentato nel database da un oggetto identificato dalla chiave `uid`. Possiamo scegliere per questi oggetti la nomenclatura che preferiamo: nel nostro caso facciamo l'esempio di nominare gli utenti con il loro indirizzo email principale (es: `uid=phasa@dominio.org`) Volendo raggruppare i servizi relativi ad esempio ad un'organizzazione intera, anziché un singolo utente, potremmo scegliere di creare un oggetto con un nome di dominio `uid=organizzazione.it` o qualunque altra cosa ci sembri significativa (una volta scelto uno schema).

A ciascuno di questi oggetti possono essere associati diversi servizi, che saranno oggetti collocati al di sotto dell'oggetto "utente". Il tipo di questi oggetti corrisponderà al tipo di servizio. Per esempio queste sono alcune strutture possibili:

```
* un singolo utente con casella di posta

  ou=People
  \-- uid=utente@dominio.org                    shadowAccount
```

```
      \-- mail=utente@dominio.org              virtualMailUser

* utente con casella di posta e sito web (e dunque account ftp)

  ou=People
  \-- uid=utente@dominio.org                   shadowAccount
      +-- mail=utente@dominio.org              virtualMailUser
      +-- alias=sitoutente                     subSite
      \-- ftpname=utente                       ftpAccount

* organizzazione con piu' account, sito web e dominio proprio

  ou=People
  \-- uid=organiz.org                          shadowAccount
      +-- mail=utente1@organiz.org             virtualMailUser
      +-- mail=utente2@organiz.org             virtualMailUser
      +-- cn=www.organiz.org                   virtualHost
      \-- ftpname=organiz                      ftpAccount
```

Sulla destra di ciascun oggetto è specificata la classe cui appartiene. In particolare gli oggetti utente appartengono alla classe `shadowAccount` (uno standard POSIX) che, seppure non venga mai utilizzata per l'autenticazione, permette di risolvere i lookup NSS e dunque gestire le quote più facilmente (potendo specificare direttamente il nome dell'utente anziché solo la uid numerica).

4.2.3. Autenticazione

Nella struttura sopra descritta ciascun servizio che necessita autenticazione la effettua direttamente sull'oggetto "figlio" del tipo relativo. Dunque le credenziali di autenticazione presenti nell'oggetto `shadowAccount` che identifica l'utente, benché presenti perché richieste dallo schema, non sono mai utilizzate direttamente.

Tramite opportuni filtri di query dunque si fa in modo che per esempio il demone IMAP effettui l'autenticazione degli utenti su un oggetto di tipo `virtualMailUser`, con la sua propria password. Questo meccanismo permette, volendolo, di avere password differenziate per i vari servizi. La possibilità di fare ciò rientra nelle *policy* che vanno imposte attraverso gli strumenti di amministrazione, in quanto LDAP, al contrario di un database relazionale, non ha dei meccanismi propri per specificare, appunto, particolari relazioni tra oggetti diversi (tranne quella gerarchica): per questo è impossibile (o estremamente contorto) specificare in LDAP concetti come il fatto che un oggetto debba avere la stessa password di quello che lo contiene; questo fatto porta naturalmente ad una certa ridondanza dei dati presenti nello schema (non elevata, però).

Vedremo ora a grandi linee come sono configurati i vari servizi (relativamente all'autenticazione sul database LDAP). Per ogni servizio sono specificati quali sono i files di configurazione relativi all'autenticazione, e qual è il tipo di query effettuata - costituita da una *base* che limita la ricerca e una *query* (detta anche *filtro*) che cerca un oggetto specifico:

ssh

ssh si autentica tramite PAM con un file specifico di configurazione che seleziona solamente il ramo *ou=Admins* del database ldap.

query: `&(uid=*)(objectClass=posixAccount)`

base: `ou=Admins, dc=infra, dc=org, o=Anarchy`

files:

/etc/pam.d/common-auth

```
auth  sufficient  pam_ldap.so config=/etc/pam_ldap_admin.conf
auth  required    pam_unix.so use_first_pass
```

/etc/pam_ldap_admin.conf

```
host 127.0.0.1
base ou=Admins,dc=infra,dc=org,o=Anarchy
ldap_version 3
rootbinddn cn=manager,o=Anarchy
pam_password crypt
```

vsftpd

il demone FTP utilizza sempre PAM ma con un altro file di configurazione, che cerca oggetti con attributo *objectClass=ftpAccount* e *host* uguale al nome della macchina.

query: `&(ftpname=*)(objectClass=ftpAccount)(host=server1)`

base: `ou=People, dc=infra, dc=org, o=Anarchy`

files:

/etc/pam.d/vsftpd

```
auth  required  pam_ldap.so config=/etc/pam_ldap_ftp.conf
```

/etc/pam_ldap_ftp.conf

```
host 127.0.0.1
base ou=People,dc=infra,dc=org,o=Anarchy
ldap_version 3
rootbinddn cn=manager,o=Anarchy
scope sub
pam_login_attribute ftpname
pam_filter &(host=test1)(status=active)
pam_password crypt
```

dovecot

dovecot e' il demone IMAP, ed autentica gli utenti di posta. Non usa PAM ma dei meccanismi propri.

query: `&(objectClass=virtualMailUser)(host=server1)`

base: `ou=People, dc=infra, dc=org, o=Anarchy`

files:

/etc/dovecot/dovecot-ldap.conf

```
ldap_version = 3
scope = subtree
dn = cn=dovecot,ou=Operators,dc=infra,dc=org,o=Anarchy
dnpass = blablabla
base = ou=People,dc=infra,dc=org,o=Anarchy
user_attrs = mail,mailMessageStore,mailMessageStore„uidNumber,gidNumber
user_filter = (&(objectClass=virtualMailUser)(status=active)(mail=%u))
```

```
pass_attrs = mail,userPassword
pass_filter = (&(objectClass=virtualMailUser)(status=active)(mail=%u))
```

saslauthd

SASL serve ad autenticare gli utenti che spediscono i loro messaggi con SMTP. saslauthd è il demone che effettua l'autenticazione. La configurazione di questo programma è differente dalle altre in quanto è possibile saltare PAM ed effettuare direttamente una query LDAP, questo però richiede un formato diverso del file di configurazione. Inoltre dato che autentichiamo utenti con nome E dominio, bisogna ricordarsi di dare al programma l'opzione `-r` (per non scartare il *realm*), e di creare il socket nella chroot di Postfix...

query: `&(mail=*)(objectClass=virtualMailUser)`

base: `ou=People, dc=infra, dc=org, o=Anarchy`

files:

/etc/default/saslauthd

```
START=yes
MECHANISMS="ldap"
PWDIR=/var/spool/postfix/var/run/saslauthd
PIDFILE="$PWDIR/saslauthd.pid"
PARAMS="-r -m $PWDIR"
```

/etc/saslauthd.conf

```
ldap_servers: ldap://127.0.0.1/
ldap_bind_dn: cn=ring0op,ou=Operators,dc=infra,dc=org,o=Anarchy
ldap_password: blablabla
ldap_search_base: ou=People,dc=infra,dc=org,o=Anarchy
ldap_filter: (&(status=active)(objectClass=virtualMailUser)(mail=%u))
ldap_auth_method: custom
```

Nota: tutte le query sopra descritte confrontano la password fornita con quella memorizzata nell'attributo `userPassword`. Anche su questo è il caso di precisare una cosa che può risultare utile: le password possono venir memorizzate nel database LDAP codificate in modi differenti, contraddistinti da un prefisso posto prima della password tra parentesi graffe, ad esempio

```
{crypt}dlkj8h23dU9j1
```

Altri metodi oltre a *crypt* (che utilizza la funzione `crypt` di sistema) sono *MD5* e *SSHA*, che utilizzano semplicemente un hash della password. Molti comandi comuni (ad esempio `ldappasswd`) impostano la password con uno di questi ultimi meccanismi, considerati più sicuri. Questo è rilevante dal momento che c'è più di una possibilità per verificare l'autenticazione: molti servizi utilizzano il cosiddetto *bind LDAP*, ovvero semplicemente tentano l'accesso al database LDAP usando le credenziali fornite: in questo caso è il server LDAP medesimo a verificare la password, ma esistono altri servizi (un esempio per tutti è dovecot) che fanno le cose in modo differente: ottengono la password dal server LDAP e la verificano autonomamente, e bisogna fare molta attenzione al fatto che generalmente *non* sono in grado di verificare correttamente le password memorizzate con meccanismi diversi da *crypt*!

4.3. Lo schema LDAP

Questo che segue è lo schema che definisce le classi e gli attributi utilizzati nel nostro database di esempio, con qualche commento aggiunto per spiegare le varie sezioni:

```
#
# infra.schema
#

# OIDs di base
objectIdentifier infraOID 1.1
objectIdentifier infraLDAP infraOID:2
objectIdentifier infraAttributeType infraLDAP:1
objectIdentifier infraObjectClass  infraLDAP:2
```

queste sono delle macro che definiscono gli oid per le classi e gli attributi che andiamo a descivere. L'oid 1.1 è utilizzato per test e prove, sarebbe opportuno registare i propri oid per evitare sovrapposizioni con altri elementi.

```
# abbreviazioni per tipi di dato molto comuni
objectIdentifier String   1.3.6.1.4.1.1466.115.121.1.26
objectIdentifier Boolean  1.3.6.1.4.1.1466.115.121.1.7
objectIdentifier Date     1.3.6.1.4.1.1466.115.121.1.26
objectIdentifier Counter  1.3.6.1.4.1.1466.115.121.1.27



### Attributi

attributetype ( infraAttributeType:7 NAME 'status'
        DESC 'Status of an object'
        EQUALITY caseIgnoreIA5Match
        SYNTAX String SINGLE-VALUE )
```

l'attributo *status* è stato introdotto per mantenere nel database LDAP oggetti con stati differenti (ad esempio in fasi diverse della creazione, o per indicare l'obbligatorio cambio di password al primo utilizzo, o per disabilitare un account senza cancellarlo). Tutti i filtri di query che vengono usati per i servizi virtuali contengono una parte che verifica se `status` è *active*.

```
attributetype ( infraAttributeType:1 NAME 'documentRoot'
        DESC 'The absolute path to the document root directory'
        EQUALITY caseExactIA5Match
        SYNTAX String SINGLE-VALUE )

attributetype ( infraAttributeType:2 NAME 'serverAlias'
        DESC 'Apache server alias'
        EQUALITY caseExactIA5Match
        SYNTAX String )

attributetype ( infraAttributeType:3 NAME 'options'
        DESC 'Comma separated string of options'
        EQUALITY  caseExactIA5Match
```

```
        SYNTAX String SINGLE-VALUE )

attributetype ( infraAttributeType:5 NAME 'alias'
        DESC 'apache alias'
        EQUALITY caseExactIA5Match
        SYNTAX String SINGLE-VALUE )

attributetype ( infraAttributeType:6 NAME 'parentSite'
        DESC 'sito a cui si riferisce un alias apache'
        EQUALITY caseExactIA5Match
        SYNTAX String SINGLE-VALUE )
```

questi sopra sono attributi creati appositamente per memorizzare valori relativi ai virtual host e sottositi di Apache.

```
attributetype ( infraAttributeType:8 NAME 'ftpEnabled'
        DESC 'Machine enabled to use the network '
        EQUALITY caseIgnoreIA5Match
        SYNTAX String  SINGLE-VALUE )

attributetype ( infraAttributeType:12 NAME 'creationDate'
        DESC 'Accont creation date'
        EQUALITY caseIgnoreIA5Match
        SYNTAX Date SINGLE-VALUE )

attributetype ( infraAttributeType:20 NAME 'dbuser'
        DESC 'username of datbase'
        EQUALITY caseIgnoreIA5Match
        SYNTAX String SINGLE-VALUE )

attributetype ( infraAttributeType:21 NAME 'dbname'
        DESC 'database name'
        EQUALITY caseIgnoreIA5Match
        SYNTAX String SINGLE-VALUE )

attributetype ( infraAttributeType:22 NAME 'clearPassword'
        DESC 'cleartext password'
        EQUALITY caseExactIA5Match
        SYNTAX String SINGLE-VALUE )
```

questi ultimi tre sono attributi relativi agli oggetti database MySQL. Ovviamente MySQL non è in grado di autenticare gli utenti con LDAP, ma è comunque utile dal punto di vista amministrativo mantenere questi dati nel database (perlomeno possiamo sapere facilmente a quale utente fanno riferimento).

```
### Classi

objectclass ( infraObjectClass:1 NAME 'infraObject'
        SUP top ABSTRACT
        DESC 'Basic administrable object'
        MAY ( creationDate $ host $ status )
        )
```

questa è la classe base da cui poi derivano tutti gli oggetti che corrispondono ai servizi virtuali; in questo modo è possibile avere un set di attributi comuni a tutti questi oggetti (come ad esempio degli attributi relativi alla gestione, oppure - vedi `host` - si può specificare su quale macchina è basato il servizio).

```
objectclass ( infraObjectClass:2 NAME 'virtualHost'
        SUP infraObject STRUCTURAL
        DESC 'Apache virtual host'
        MUST ( documentRoot $ cn $ status )
        MAY ( serverAlias $ emailAddress $ options )
        )

objectclass ( infraObjectClass:3 NAME 'virtualMailUser'
        SUP infraObject STRUCTURAL
        DESC 'Virtual mail user'
        MUST ( mail )
        MAY ( mailMessageStore $ userPassword $
              mailAlternateAddress $ mailForwardingAddress $
              gidNumber $ uidNumber )
        )

objectclass ( infraObjectClass:4 NAME 'subSite'
        SUP infraObject STRUCTURAL
        DESC 'Apache sub-directory of our main sites'
        MUST ( alias $ parentSite $ documentRoot )
        )

objectclass ( infraObjectClass:8 NAME 'dataBase'
        SUP infraObject STRUCTURAL
        DESC 'MySQL database info'
        MUST ( dbname )
        MAY  ( dbuser $ clearPassword )
        )

objectclass ( infraObjectClass:10 NAME 'ftpAccount'
        SUP infraObject AUXILIARY
        DESC 'an FTP account'
        MUST ( ftpEnabled )
        )
```

queste sono le classi corrispondenti ai vari servizi virtuali. L'unica eccezione è l'oggetto ftpAccount, definito come una classe "aggiuntiva" a shadowAccount, di modo che il server FTP si possa autenticare tramite PAM; in definitiva queste però non sono differenze importanti dato che comunque vengono nascoste dal software di gestione.

4.4. LDIF

Per descrivere le entry di un database LDAP viene utlizzato un formato convenzionale chiamato LDIF (LDAP Data Interchage Format). Un file LDIF è semplicemente un file di testo con una serie di coppie attributo/valore con questa sintassi:

```
dn: <distinguished name>
objectclass: <object class>
...
...
<attribute type>: <attribute value>
<attribute type>: <attribute value>
...
```

I file LDIF vengono utilizzati per importare modificare ed esportare le entry del database (sono decisamente comodi essendo ascii puro). Riporto la struttura logica e successivamente alcuni esempi di ldif commentati:

```
dc=infra, dc=org, o=Anarchy, ou=People
      |
      |_ uid=phasa@dominio.org
            |
            |_ alias=phasa
            |
            |__mail=phasa@dominio.org
```

Esempio dell'LDIF corrispondente (proveniente direttamente dallo script di migrazione):

```
dn: uid=phasa@dominio.org, ou=People, dc=infra, dc=org, o=Anarchy
shadowMax: 99999
uid: phasa@dominio.org
cn: phasa@dominio.org
homeDirectory: /var/empty
uidNumber: 13468
objectClass: top
objectClass: person
objectClass: posixAccount
objectClass: shadowAccount
objectClass: organizationalPerson
objectClass: inetOrgPerson
shadowWarning: 7
gidNumber: 2000
gecos: phasa@dominio.org
shadowLastChange: 12345
sn: Private
userPassword: {crypt}x
givenName: Private
loginShell: /bin/false

dn: mail=phasa@dominio.org, uid=phasa@dominio.org, ou=People,
  dc=infra, dc=org, o=Anarchy
```

```
mailAlternateAddress: phasa@public.org
mailAlternateAddress: phasa@public2.org
mailAlternateAddress: phasa@public3.org
status: active
uidNumber: 13468
objectClass: top
objectClass: virtualMailUser
host: server1
gidNumber: 2000
creationDate: 2002-05-07
originalHost: server1
mail: phasa@dominio.org
userPassword: {crypt}$1$fa0c5a13$VVqsukrQmdr79LZg2xvnM.
mailMessageStore: dominio.org/phasa/

dn: alias=phasa, uid=phasa@dominio.org, ou=People, dc=infra,
  dc=org, o=Anarchy
parentSite: public.org
status: active
objectClass: top
objectClass: subSite
documentRoot: /home/users/phasa/html
host: server1
originalHost: server1
alias: phasa
```

4.5. Replicazione del database LDAP

Una volta creato il database di tutti gli utenti e suddivise le mailbox in modo omogeneno sugli N server ci rimane da risolvere un problema e cioè dove mettere il database, su quale degli N server? Beh ovviamente su tutti :), in realtà OpenLDAP ci permette anche di partizionare il database in quante parti vogliamo e di replicare le varie parti in modo differente, ma per questa prima implementazione abbiamo pensato, per semplicità, di mantere un unico database replicato su tutti gli N server.

La replicazione di OpenLDAP può avvenire in diversi modi: il più semplice prevede un master e N slave. Gli aggiornamenti al database vengono effettuati sul master, inoltrando tutti i cambiamenti agli slave, ma le richieste di nuove attivazioni o di aggiornamenti possono essere fatte indifferentemente agli slave o al master: nel caso degli slave saranno essi stessi a comunicare tali richieste al server master che poi provvederà a svolgere l'aggiornamento. Nel caso in cui il server master dovesse essere assaltato da cavallette e/o da sfiga la configurazione degli altri server ci permetterà di selezionare un nuovo master immediatamente con una semplice modifica di configurazione: in tal modo il sistema continuerà a funzionare.

4.6. Configurazione di `slapd`

Per prima cosa è ovviamente necessario aver installato slapd (la componente server di OpenLDAP, una suite LDAP free). La configurazione del server è completamente contenuta nel file `/etc/ldap/slapd.conf`. Anche slurpd (il demone che gestisce la replicazione) utilizza lo stesso file di configurazione.

Ci sono tre tipi di direttive di configurazione del server:

- *global*
- *backend specific*
- *database specific*

Le prime specificano parametri generici di `slapd`, come gli indirizzi da bindare, dove mettere i log, le ACL, etc...

La seconda serie di opzioni è configura parametri specifici del backend utilizzato per i database. Noi usiamo BDB nonostante tutti i suoi problemi di consistenza (sono consigliati backup quotidiani dei dati) visto che l'altro backend supportato da OpenLDAP (LDBM) è addirittura sconsigliato ufficialmente...

Infine, le opzioni relative ai database (che nel nostro caso è uno solo) specificano ad esempio quali indici ottimizzati creare, dove memorizzare il database, e altri parametri che è opportuno controllare. In genere è necessario (se si desidera un minimo di performance) creare un indice per ciascun attributo che potrebbe comparire in qualche filtro, e comunque è possibile creare gli indici anche in un secondo momento (con il comando **slapindex**).

4.6.1. Ottimizzazioni

Non è nostra intenzione, qui, illustrare la configurazione base di slapd, per la quale si rimanda alla "OpenLDAP Administration Guide", in particolare al Capitolo 6: "The slapd configuration file" [3]. Intendiamo invece approfondire le modifiche apportate ad una tipica configurazione base per ottenere maggiore stabilità e performance: quasi ogni operazione sui server comporta una richiesta LDAP, dunque si intuisce facilmente che il funzionamento complessivo del sistema dipende strettamente dall'efficacia di slapd.

Il problema può essere affrontato con due approcci differenti, che non si escludono ma si complementano: da una parte, si può tentare di ridurre il più possibile il numero delle richieste (ad esempio utilizzando meccanismi di *caching*), dall'altra è necessario ottimizzare attentamente la configurazione di slapd in modo da ottenere prestazioni sufficienti.

Come esempio della prima strategia,a si cerca di alleggerire il server dalle (numerosissime) richieste NSS: anzitutto conviene specificare *files* prima di *ldap* in `/etc/nssswitch.conf` per risolvere subito almeno gli utenti di sistema; l'altra condizione fondamentale è l'esecuzione del demone di caching NSS nscd.

Per quel che riguarda l'ottimizzazione di slapd invece, queste sono le considerazioni che abbiamo fatto:

- è assolutamente *necessario* creare, nel database LDAP, un indice per ciascuno degli attributi su cui si prevede di fare una query. Questo ad esempio è il frammento rilevante di `slapd.conf`, nella sezione *database*:

  ```
  index objectClass,status eq
  index sn,uid,mail,alias,cn eq,pres
  index host,uidNumber eq,pres
  index mailForwardingAddress eq,pres
  index mailAlternateAddress eq,pres
  ```

  Quando viene effettuata una query con un filtro che comprende un attributo per cui non è stato specificato un indice, il server deve esaminare singolarmente ciascun oggetto nel database: questa situazione è da evitare a tutti i costi.

- può servire aumentare il più possibile la dimensione della cache degli oggetti di slapd, un valore indicativo (che consente di mantenere l'intero database in memoria, se non è troppo grande) può essere appunto il numero di direttive presenti nel database (ottenibile approssimativamente con `slapcat | wc -l`):

  ```
  cachesize 50000
  ```

- aumentare il numero di threads e configurare un timeout per chiudere le connessioni che dovessero rimanere "appese" può essere utile se il carico sul server LDAP dovesse essere tale da determinare dei timeout sui client:

  ```
  threads 64
  idletimeout 300
  ```

  Un'altra cosa che si può voler fare in questo caso, nell'eventualità che le connessioni LDAP locali utilizzino i socket TCP anziché UNIX, è ampliare il range delle porte effimere, ad esempio ponendo in `/etc/sysctl.conf`

  ```
  net/ipv4/ip_local_port_range=20000 65535
  ```

  (cosa questa che complessivamente è un bene sui server in generale).

- Infine si possono modificare alcuni parametri specifici di Berkeley DB creando un file denominato `DB_CONFIG` nella directory dove risiedono i database (su Debian, `/var/lib/ldap`). Anche qui conviene aumentare le dimensioni della cache dei valori, su sistemi moderni ad esempio 50Mb non sono un problema:

  ```
  set_cachesize 0 52428800 0
  ```

  Si può poi specificare che la creazione di files temporanei avvenga su un filesystem tmpfs:

  ```
  set_tmp_dir /dev/shm
  ```

  Ma forse la cosa più importante, anche se leggermente rischiosa, è disabilitare il log delle transazioni in scrittura sui server slave:

  ```
  set_flags DB_TXN_NOSYNC
  ```

  Il log delle transazioni garantisce l'integrità dei dati al prezzo di un certo impatto sulle prestazioni: sulle macchine slave possiamo pensare di sacrificare questa sicurezza, avendo comunque a disposizione la copia principale se si verificassero problemi.

4.6.2. ACL

La parte forse più complessa della configurazione del server LDAP è la creazione delle ACL (liste di controllo di accesso) necessarie a permetterne il funzionamento. Ciò in parte è dovuto al fatto che la forma con cui bisogna scrivere queste regole è particolarmente contorta... Per un dato *uid* richiesto e un dato uid richiedente, OpenLDAP considererà la prima regola (in ordine di apparizione nel file di configurazione) che si applica all'oggetto richiesto, per poi applicare la prima ACL che viene specificata (ancora una volta, in ordine di apparizione) che sia compatibile con il richiedente. Ad esempio, questa è la ACL per controllare l'accesso all'attributo *userPassword*:

```
access to attrs=userPassword
  by dn="cn=manager,o=anarchy" write
  by group/groupofnames/member=\
        "cn=admins,ou=Group,dc=infra,dc=org,o=Anarchy" write
  by dn="cn=ring0op,ou=Operators,dc=infra,dc=org,o=Anarchy" read
  by dn="cn=ring1op,ou=Operators,dc=infra,dc=org,o=Anarchy" read
  by dn="cn=dovecot,ou=Operators,dc=infra,dc=org,o=Anarchy" read
  by anonymous auth
  by self write
  by * none
```

La modifica è sempre e comunque permessa al manager e al gruppo di amministratori, mentre gli operatori possono solo leggerla per l'autenticazione.

Le ACL vanno dunque inserite nel file `slapd.conf` a partire dalla più specifica (per la parte *access to*) alla meno specifica.

Note

1. http://www.openldap.org/
2. http://directory.fedora.redhat.com/
3. http://www.openldap.org/doc/admin23/slapdconfig.html

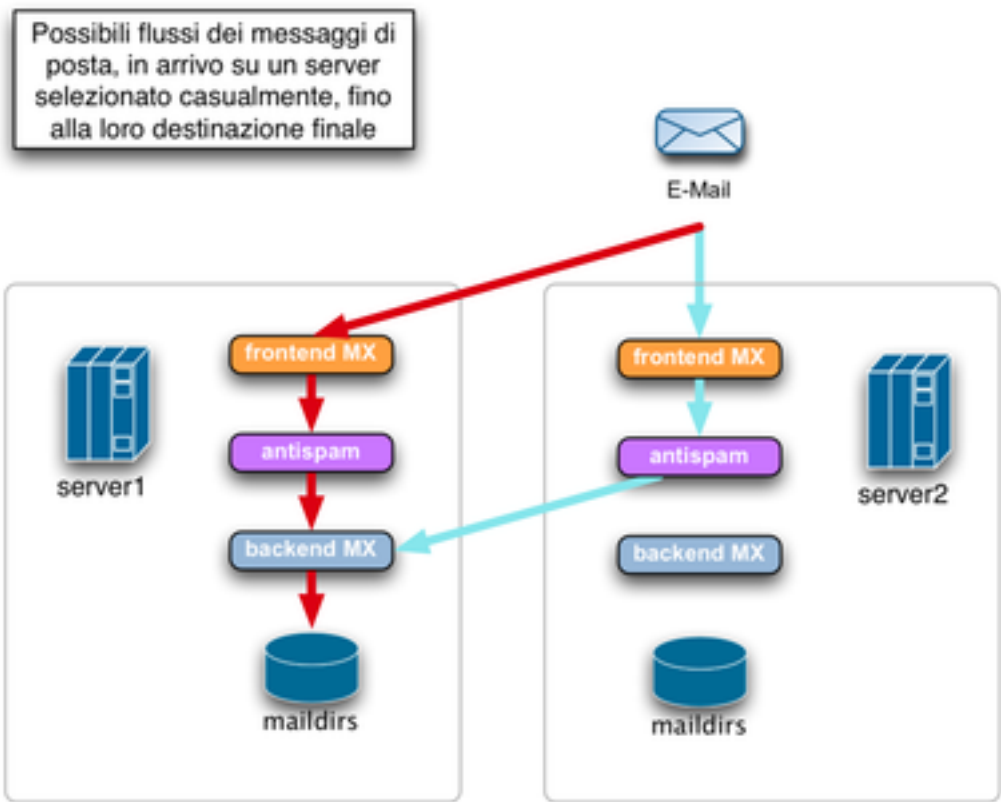
Capitolo 5. Servizi di posta

Si è detto nell'introduzione che uno degli obbiettivi principali da realizzare era la suddivisione delle mailbox (e quindi dei servizi correlati, SMTP, IMAP, webmail) sulle N macchine. Fortunatamente il protocollo SMTP e soprattutto la flessibilità della configurazione di Postfix permettono di implementare quanto voluto senza troppe complicazioni.

La scelta fatta per la configurazione dei servizi SMTP, conseguentemente con le linee guida identificate al Capitolo 1, è stata questa:

1. assegnare la posta di ciascun utente ad una sola delle N macchine (in modo intercambiabile ma univoco);
2. avere tutte le N macchine come MX del dominio (o dei domini, che saranno probabilmente più di uno), con eguale probabilità, ed effettuare poi la redirezione interna al server di destinazione finale.

Essendo il database degli utenti replicato su tutte le macchine, ciascun server sa qual'è il server destinatario (*finale*) di un particolare indirizzo email, ed è quindi possibile costruire le tabelle di transport necessarie.



Flusso delle connessioni SMTP

La figura precedente illustra il percorso seguito, all'interno della nostra rete, dalle mail in arrivo. Anzitutto, il server SMTP remoto sceglie ogni volta casualmente un server MX tra quelli elencati con la stessa priorità, dunque il messaggio arriva su uno qualsiasi dei server. La figura mostra due dei percorsi possibili, corrispondenti al caso in cui il server sia quello che ospita la casella di posta (la destinazione finale della mail) oppure no. Nel primo caso il messaggio

non viene ritrasmesso e raggiunge direttamente la maildir dell'utente; nell'altro caso invece viene rimessa in coda, instradata però verso il server di destinazione (attraverso la VPN). In entrambi i casi il messaggio viene sottoposto al controllo antispam un'unica volta.

In relazione al punto 2 qui sopra, si potrebbe considerare che questa soluzione possa non essere la migliore in termini di consumo di banda; e in effetti, facendo un rapido calcolo, in media la probabilità che un messaggio in arrivo su un server non sia destinato lì (e vada quindi ritrasmesso al server finale) è (N-1)/N, che tende a 1 con un certo numero di server[1]. Va però considerato che questa soluzione offre un'elevata resistenza alle interruzioni momentanee di rete, oltre a garantire la suddivisione delle mailbox, il che forse è un vantaggio sufficiente considerato anche che il traffico totale di mail non è particolarmente elevato (le statistiche attuali ci danno una media di 1 messaggio al secondo, con dei picchi di 10 nei momenti di maggiore traffico).

5.1. Configurazione di Postfix

Postfix è configurato, su ciascuna macchina, in modo da accettare la posta in arrivo per tutti gli utenti virtuali, consultando poi una opportuna tabella di *transport* che gli dirà se consegnare il messaggio localmente, oppure se instradarlo nuovamente verso la macchina di destinatazione finale.

Potendo effettuare l'instradamento finale attraverso la VPN ci si può anche risparmiare un'ulteriore negoziazione SSL tra i server medesimi. Anche la scansione antivirus può essere effettuata solo una volta, al momento della ricezione dall'esterno. Per questo si possono configurare due istanze differenti di `smtpd` che siano in ascolto sulle due interfacce, esterna e VPN, aggiungendo qualche riga in `/etc/postfix/master.cf`:

```
# smtp su interfaccia esterna
192.168.1.1:smtp inet    n   -   -   -   -   smtpd -o opzioni...
# smtp sulla vpn
172.16.1.1:smtp inet     n   -   -   -   -   smtpd -o opzioni diverse...
```

e differenziando opportunamente le opzioni fornite. Per esempio, se il filtro antispam fosse abilitato in `main.cf` l'istanza sulla VPN potrebbe avere le seguenti opzioni, per disabilitare la crittazione TLS e il filtro antispam, e non ripetere due volte i controlli sul messaggio:

```
-o smtpd_use_tls=no
-o content_filter=
-o local_recipient_maps=
-o smtpd_helo_restrictions=
-o smtpd_client_restrictions=
-o smtpd_sender_restrictions=
-o smtpd_recipient_restrictions=permit_mynetworks,reject
-o mynetworks=172.16.0.0/16
```

Un'altra possibilità è quella di avere due istanze di Postfix differenti (e dunque due serie di code differenti) per l'interno e l'esterno, duplicando `/etc/postfix` e `/var/spool/postfix`[2]. Lo schema rimane in ogni caso quello di un sistema a due livelli (o tre, se si vuole considerare anche l'antivirus che è un sottosistema SMTP autonomo), con il livello esterno incaricato di gestire la posta entrante e uscente, di effettuare il controllo degli indirizzi validi e dello spam, e di applicare tutti gli eventuali filtri, mentre il livello interno deve occuparsi solamente della delivery finale alla mailbox. La gestione speciale delle code di *relay* nelle ultime versioni di Postfix dovrebbe però garantire un buon throughput interno anche con un'unica istanza, perfino quando la coda esterna è intasata[3].

5.1.1. Mappe LDAP

Le mappe LDAP devono permettere il riconoscimento di tutti gli utenti virtuali (in local_recipient_maps), il loro instradamento verso il giusto server di replicazione (in transport_maps), infine la delivery alla mailbox corretta (virtual_mailbox_maps) e la risoluzione degli alias (virtual_alias_maps).

Vediamo come sono composte:

`ldaptransport`

in `transport_maps` - &(mail=%s)(objectClass=virtualMailUser)(!(host=server1)) -> *host*

questa mappa funziona perché sono definiti (in `master.cf`) dei transport SMTP con i nomi delle varie macchine e destinazioni hard-codate. Per esempio, il transport `server1`, presente su tutte le macchine tranne quella, sarà un transport SMTP con destinazione prefissata (server1-vpn), permettendo customizzazioni specifiche come la regolazione fine della concorrenza, TLS disabilitato, e altre ottimizzazioni. Naturalmente su "server1" le connessioni provenienti dalla VPN non avranno antivirus e antispam abilitati, così il controllo verrà fatto una volta sola.

```
ldaptransport_server_host = localhost
ldaptransport_server_port = 389
ldaptransport_scope = sub
ldaptransport_bind_dn = "cn=manager, o=anarchy"
ldaptransport_bind = no
ldaptransport_lookup_wildcards = no
ldaptransport_search_base = ou=People, dc=infra, dc=org, o=anarchy
ldaptransport_query_filter = (&(mail=%s)(!(host=amnistia)))
ldaptransport_result_attribute = host
ldaptransport_result_filter = relay:[%s-vpn]
```

Qui è usata anche l'opzione *result_filter*, che permette di modificare il risultato della query LDAP dopo averla ricevuta dal server.

`ldapmailbox`

in `virtual_mailbox_maps` - &(mail=%s)(host=server1)(objectclass=virtualMailUser)(mailMessageStore=*) -> *mailMessageStore*

mappa degli indirizzi di posta verso la loro mailbox di destinazione. I path ritornati dalla query sono considerati relativi a `/home/mail` e generalmente comprendono una directory con il nome del dominio e una con il nome della casella.

```
ldapmailbox_server_host = localhost
ldapmailbox_server_port = 389
ldapmailbox_scope = sub
ldapmailbox_bind_dn = "cn=manager, o=anarchy"
ldapmailbox_bind = no
ldapmailbox_lookup_wildcards = no
ldapmailbox_search_base = ou=People, dc=infra, dc=org, o=anarchy
ldapmailbox_query_filter =
  (&(mail=%s)(objectclass=virtualMailUser)(mailMessageStore=*))
ldapmailbox_result_attribute = mailMessageStore
```

`ldapalias`

in `virtual_alias_maps` - &(mailAlternateAddress=%s)(objectclass=virtualMailUser) -> *mail*

mappa degli alias locali verso l'indirizzo email di destinazione finale.

```
ldapalias_bind_dn = cn=manager,o=anarchy
ldapalias_bind = no
ldapalias_search_base = ou=People, dc=infra, dc=org, o=anarchy
ldapalias_query_filter =
  (&(mailAlternateAddress=%s)(objectclass=virtualMailUser))
ldapalias_result_attribute = mail
ldapalias_lookup_wildcards = no
```

5.1.2. Antispam/Antivirus

I servizi antispam sono molto semplici per il momento, anche se piuttosto efficaci. Su tutte le macchine che ricevono la posta è installato **Postgrey**[4], un sistema di *greylisting*: ogni messaggio che arriva per la prima volta (caratterizzato dalla tripletta server smtp / mittente / destinatario) viene respinto con un errore temporaneo che permane per 60 secondi. Dopo *N* messaggi consegnati con successo per una particolare tripletta, questa viene inserita nella *whitelist* così da evitare ulteriori controlli per un certo lasso di tempo. Per i server SMTP normali questo errore temporaneo non rappresenta un problema, il messaggio viene messo in coda e la consegna viene tentata di nuovo dopo poco tempo. Il meccanismo funziona perché la stragrande maggioranza dei virus e dei meccanismi usati dagli spammer non utilizzano un vero server SMTP e non hanno la possibilità di mantenere messaggi in coda, dunque generalmente non riprovano la consegna una seconda volta. Certo, dal punto di vista dell'utente questo significa che la consegna dei messaggi non è "immediata" ma richiede più tempo (si parla comunque di minuti), in ogni caso ci pare un buon compromesso, vista l'efficacia del greylisting.

Postgrey è installato come *policy_service* nell'istanza di Postfix che accetta posta dall'esterno (sull'IP pubblico):

```
smtpd_recipient_restrictions = ...,
                  check_policy_service inet:127.0.0.1:60000,
                  ...
```

È da diverso tempo che usiamo Postgrey e non abbiamo riscontrato problemi di sorta, anzi, come protezione antispam è risultata (per il momento) molto efficace. È utile in ogni caso controllare `/etc/postfix/whitelist_clients`, ed eventualmente inserirvi i server di posta con cui si comunica più frequentemente, sia per velocizzare le consegne che per impedire che il database temporaneo di Postgrey cresca troppo di dimensione.

In aggiunta a questo, il meccanismo principale antispam è **amavis-ng**[5], che utilizziamo con la scansione antivirus disabilitata (che è eccessivamente pesante sulla CPU delle macchine, dato il traffico gestito) e che dunque è usato solamente come *wrapper* SMTP per il modulo di analisi di SpamAssassin[6]. Si compensa parzialmente alla mancanza dell'antivirus con una serie di espressioni regolari da controllare nel corpo dei messaggi, implementate direttamente dentro Postfix in `mime_header_checks` e `body_checks`: queste regexp sono mantenute regolarmente da www.securitysage.com[7] e contengono fingerprint dei virus più comuni, come anche molte altre regole utili per respingere buona parte dello spam prima ancora di dover inoltrare il messaggio ad Amavis.

Amavis è installato come `content_filter` nell'istanza di Postfix che accetta la posta dall'esterno (sull'IP pubblico), ed è configurato per inoltrare i messaggi approvati con successo al server di back-end (quello in ascolto su *localhost*).

5.2. POP / IMAP

La collocazione fisica delle singole caselle di posta sui vari server è determinata dal parametro `host` presente nell'oggetto LDAP corrispondente. Questa collocazione va *nascosta* all'utente, nel senso che bisogna avere la possibilità di spostare le mailbox in modo trasparente, senza che l'utente si accorga di nulla, per esempio in caso di problemi hardware oppure semplicemente per ricollocare caselle che attirano molto traffico. Per questo motivo è necessario fornire agli utenti dei parametri di connessione che non cambiano mai.

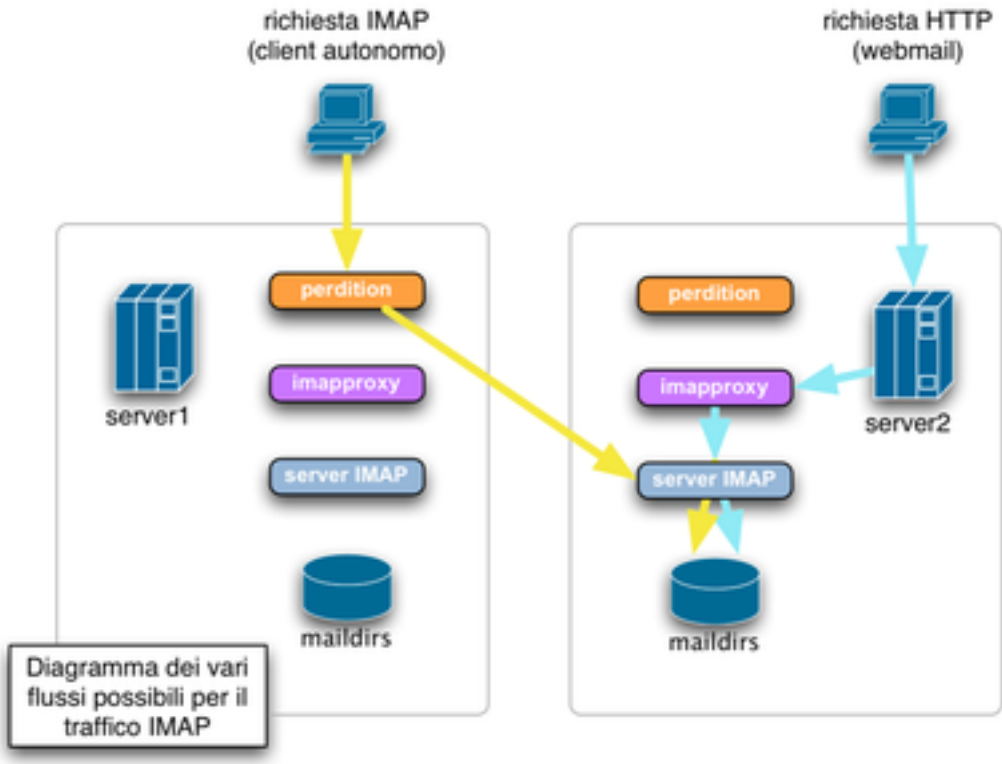
Il problema però è che i protocolli di lettura della posta non permettono la redirezione (i redirect per IMAP sono in un RFC che, credo, nessun client implementa). La soluzione iniziale che avevamo pensato prevedeva un CNAME per ciascun utente, `UTENTE.users.dominio.org`, che puntasse al server di posta giusto e che l'utente potesse usare come server IMAP nella configurazione del suo client di posta. Purtroppo questa soluzione, che altrimenti sarebbe stata ottimale, *non* funziona appena entra in gioco l'SSL: la verifica tra il nome del server e il CN del certificato, infatti, non andrà mai a buon fine e l'utente si troverebbe di fronte a continui avvisi di sicurezza.

Dunque la scelta è tra perdere in flessibilità dicendo agli utenti di configurare i loro client con la macchina specifica che ha la loro posta (e però in questo modo diventa praticamente impossibile spostare una mailbox da una macchina all'altra), oppure utilizzare un proxy POP/IMAP come Perdition, e accollarsi il trasferimento interno dei dati (valgono le stesse analisi numeriche fatte per il traffico SMTP). Per ora stiamo esplorando questa seconda possibilità, sorvegliando il consumo di banda: nel caso si rivelasse praticamente insostenibile, bisognerà ritornare alla prima opzione, anche se insoddisfacente.

I parametri di configurazione saranno dunque fatti in questo modo per tutti gli utenti:

```
POP/IMAP host:    mail.dominio.org
SMTP host:        smtp.dominio.org
```

Con questa configurazione i client si connettono al front-end server (Perdition[8]) su una macchina qualsiasi, e la loro connessione viene poi instradata internamente (attraverso la VPN) verso il server di back-end sulla macchina corretta. Abbiamo scelto dovecot[9] per implementare il backend perché tra tutte le possibilità ci è parso il software più leggero e performante.



Flusso delle connessioni IMAP

La figura mostra il flusso della connessione IMAP secondo le due tipologie standard di accesso:

- Gli utenti che utilizzano un client IMAP autonomo sul proprio computer (come qualunque client di posta) si connettono a Perdition che li redirige direttamente (attraverso la vpn) al server dovecot sulla macchina dove risiede la loro casella di posta;
- L'accesso alla posta tramite la webmail, invece, avviene direttamente sulla macchina "giusta" (dato che è possibile redirigere la connessione con un redirect HTTP). L'applicazione della webmail si connette (tassativamente su *localhost*) ad un proxy di connessioni IMAP denominato up-imapproxy[10] che serve solamente per mantenere una cache di connessioni persistenti: questo è molto utile considerando il fatto che l'applicazione web (in PHP) è costretta ad eseguire una nuova connessione IMAP al caricamento di ogni nuova pagina, utilizzando il proxy locale si evita una nuova connessione con relativa autenticazione ad ogni accesso. Infine, il proxy locale si connette al server IMAP vero e proprio.

Questa tabella mostra i parametri di configurazione essenziali che sono stati impostati per implementare lo schema appena descritto:

perdition (porta 143 e 995)

in `perdition.conf`:

```
map_library /usr/lib/libperditiondb_ldap.so.0
map_library_opt "ldap://127.0.0.1/ou=People,dc=infra,\
          dc=org,o=Anarchy?mail,host?sub?(&(mail=%s)\
          (status=active))"
outgoing_port 10143
ssl_ca_file /etc/ssl/certs/imap.pem
ssl_key_file /etc/ssl/private/imap.key
```

Per far corrispondere all'attributo *host* l'IP che vogliamo (in questo caso l'IP interno alla VPN, corrispondente a *host*-vpn), dato che purtroppo il backend LDAP di perdition non supporta la riscrittura del risultato (come invece fa Postfix), abbiamo installato il demone in una *chroot* dove la risoluzione dei nomi è controllata da un apposito `/etc/hosts`. Questo demone inoltre è l'unico a ricevere connessioni SSL dall'esterno.

up-imapproxy (port 11143)

in `imapproxy.conf`:

```
server_hostname 127.0.0.1
server_port 10143
listen_address 127.0.0.1
listen_port 11143
```

In questo modo il proxy è istruito per instradare le connessioni esclusivamente verso il server IMAP locale.

dovecot (porta 10143)

in `dovecot.conf`:

```
imap_listen = 127.0.0.1:10143
ssl_disable = yes
```

La configurazione dell'autenticazione degli utenti (in `dovecot-ldap.conf`) è stata già affrontata nel capitolo precedente.

Quando c'è bisogno di spostare una casella da una macchina all'altra (tranne nel caso di down catastrofico della prima, in cui i dati vecchi non sono più disponibili) è possibile usare rsync, dopo la modifica del db LDAP, per spostare la vecchia casella sulla nuova (che in quel momento starà già ricevendo nuovi messaggi).

5.3. Webmail

La posta via web, molto più semplicemente degli altri servizi, demanda il controllo dell'autenticazione al server IMAP (che è sempre *localhost*, di modo che è necessario che gli utenti utilizzino la webmail della macchina dove hanno la mailbox); di conseguenza non è neanche necessario configurare il supporto LDAP. Il software che abbiamo scelto è IMP[11] (il client di posta del framework Horde, scritto in PHP), principalmente perché è, in questo momento, l'unico che ci permette di sperimentare con l'integrazione PGP.

5.4. Configurazione di Mailman

Il gestore delle liste può essere configurato anch'esso copiando la configurazione di tutte le liste su tutti i server, e abilitando su ciascuno di questi gli alias adatti in ricezione.

Una complicazione è dovuta al fatto che in Mailman ci sono due agenti che modificano la configurazione della lista: il client lanciato dal MTA in corrispondenza degli alias di lista, e il `qrunner` che svolge i compiti periodici. Associare questi solo alle liste specifiche di una particolare macchina non è complesso, ma richiede a quel punto un contorto schema di replicazione delle configurazioni.

Anche volendo mantenere il gestore di mailing list è operativo su una sola macchina, è comunque necessario generare perlomeno le mappe di transport per inoltrare al server corretto la posta per le liste:

```
cat virtual-mailman | awk '/^[^#]/ {print $1 " relay:[test2-vpn]" }'
```

sarà sufficiente per generare una mappa adatta a comparire in *transport_maps* sugli altri server.

Note

1. Si consiglia, come ulteriore elemento di riflessione, la lettura dell'articolo *High Capacity Email* (disponibile presso `http://www.vergenet.net/linux/mail_farm/`), che contiene anche utili considerazioni sulle misure migliori per ridistribuire il carico sui server pesando opportunamente mailbox con alto e basso traffico.
2. A questo proposito si può leggere la trattazione completa svolta su `http://advosys.ca/papers/postfix-instance.html`, e la documentazione ufficiale (per Postfix 2.1) presso `http://www.postfix.org/FILTER_README.html`
3. Si veda la discussione presente su `http://www.postfix.org/ADDRESS_CLASS_README.html`
4. http://isg.ee.ethz.ch/tools/postgrey/
5. http://sourceforge.net/projects/amavis
6. http://spamassassin.apache.org/
7. http://www.securitysage.com/files/
8. http://www.vergenet.net/linux/perdition/
9. http://dovecot.org/
10. http://www.imapproxy.org/
11. http://www.horde.org/imp/

Capitolo 6. Configurazione

La gestione centralizzata della configurazione di più macchine non è un problema banale, se implementata in modo sbagliato può dare l'impressione di "perdere" il controllo delle macchine e di quello che fanno: se gli automatismi sono mal definiti ed eccessivamente opachi c'è il rischio che prendano il sopravvento e frustrino i tentativi di comprensione da parte degli amministratori. D'altra parte però è anche un'ottima possibilità per gestire una struttura complessa impegnando le energie (che sono in quantità finita) in modo intelligente.

Questo capitolo illustra la soluzione che abbiamo implementato, e in che modo questa interagisca col sistema. Sapere questo è fondamentale per capire in che modo modificare la configurazione per ottenere i risultati voluti.

6.1. Overview

La configurazione dei vari server è centralizzata, prevedendo più di un meccanismo per le customizzazioni specifiche. In particolare, si è utilizzato CFengine[1] per gestire i files di configurazione "semplici", cioè quelli che non dipendono dai dati presenti nel database delle utenze, e degli script customizzati (i binding LDAP per i vari linguaggi di scripting sono piuttosto semplici da usare) per gestire tutto il resto.

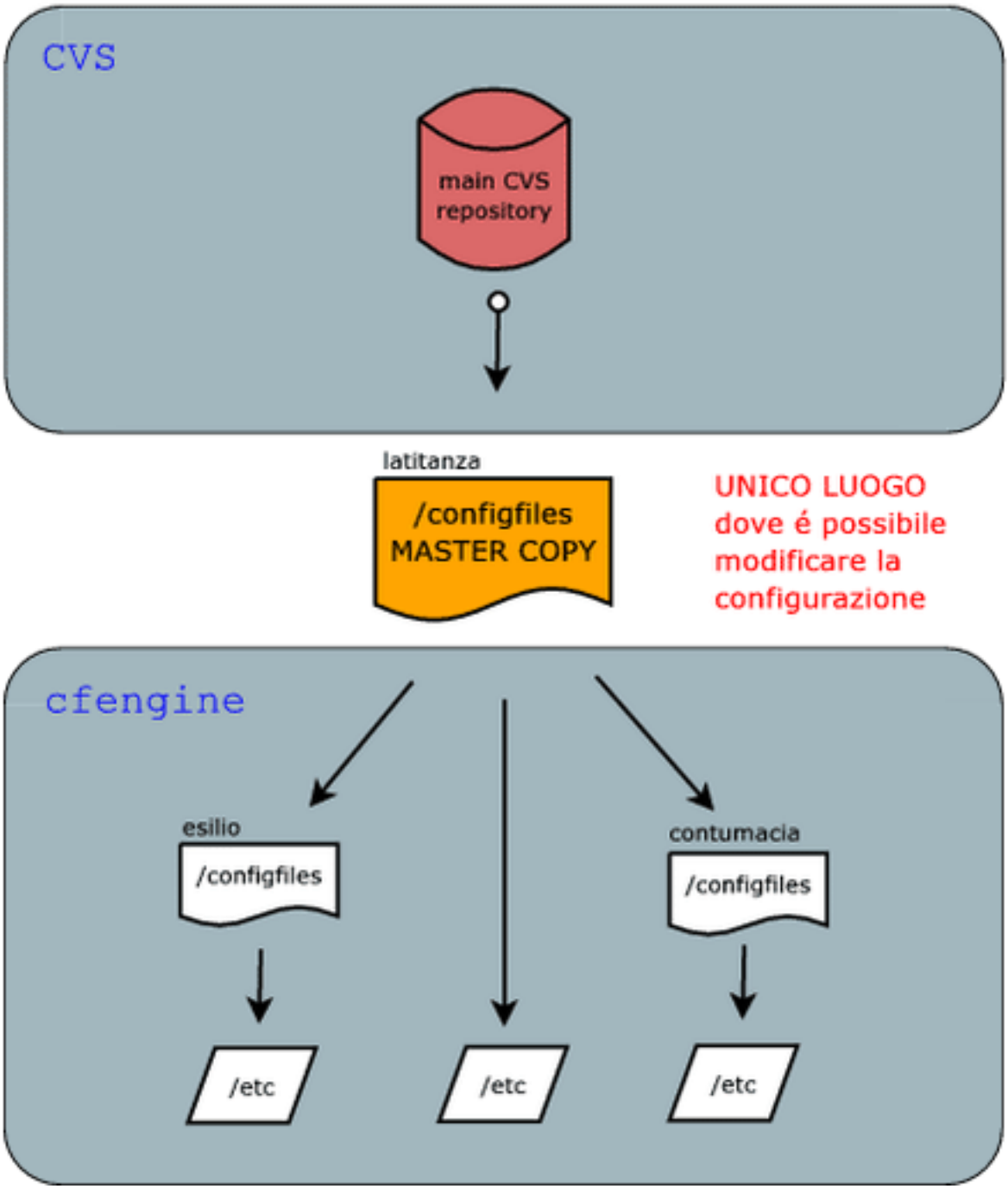
6.1.1. Configurazioni che non dipendono dal database

CFengine si è rivelato uno strumento molto utile, permettendo di realizzare un modello di gestione basato sul principio della minima differenziazione: le configurazioni sono uniche, fin dove possibile, dopodiché si procede ad editarle per semplici sostituzioni, eventualmente se necessario si possono mantenere files completamente differenti per una o tutte le macchine (vedremo più sotto in dettaglio come ciò viene realizzato). Questo modello rende molto conveniente gestire, ad esempio, files molto grandi con opzioni identiche tranne poche, che dipendono dalla macchina specifica, come può essere un indirizzo IP per il binding di un servizio o un hostname.

La gestione centralizzata dei files di configurazione deve inoltre permettere di tener traccia delle varie modifiche e di chi le abbia effettuate, in modo reversibile.

Per questo abbiamo deciso di combinare un sistema di controllo di versioni (come CVS, nel nostro caso usiamo Subversion[2]) con cui gestiamo un repository che viene poi a sua volta replicato su tutte le macchine: questo significa che è necessario effettuare le modifiche in un *unico* luogo; le modifiche si propagheranno poi automaticamente fino alla destinazione corretta, in modo determinato dalla configurazione di CFengine. Allo stesso tempo, una copia della configurazione originale completa rimane su ciascuna macchina, così che sia possibile, in caso di necessità, cambiare molto rapidamente la macchina che svolge il ruolo di server principale. Come ulteriore forma di backup, il server Subversion si trova su di un'altra macchina ancora.

L'illustrazione qua sotto spiega, con un diagramma, qual è il punto adatto alla modifica dei files, e con quali differenti meccanismi questi vengano distribuiti verso le varie destinazioni.



Flusso dei dati di configurazione.

Conviene che i files che vengono gestiti in questo modo siano solamente quelli che in qualche modo sono diversi rispetto ad un'installazione standard Debian, non è ovviamente necessario gestire l'intero albero `/etc`.

Una spiegazione dettagliata del contenuto della directory `/configfiles` è compresa nel la Sezione 6.2.1>.

6.1.2. Configurazioni che dipendono dal database

Alcuni servizi hanno una configurazione che dipende dal contenuto del database degli utenti (quello in LDAP), in genere perché è impossibile o poco pratico far sì che leggano dinamicamente la loro configurazione dal database. Un esempio è il server web: in questo caso si è scelto di semplificare la struttura dei dati dentro il database (evitando soluzioni tipo mod_config_ldap, che leggessero direttamente le direttive di Apache dentro complessi oggetti LDAP) per facilitarne la gestione, rendendo però necessaria la creazione del file di configurazione in qualche altro modo.

Questi files vengono dunque generati automaticamente con degli *script*, presenti su ciascuna macchina in `/configfiles/scripts` (che possiamo spostare in un posto più furbo se ci viene in mente qualcosa di pratico). Questi script leggono i dati presenti nel database LDAP e generano i file di configurazione necessari. Anche altre azioni sono possibili, come per esempio controllare la presenza e i permessi sul filesystem delle directory delle caselle di posta, o altri compiti amministrativi di questo tipo.

L'idea di fondo è che il database LDAP sia la fonte autoritativa per la configurazione del sistema, e che il ruolo degli script sia dunque di plasmare la configurazione effettiva in modo da renderla conforme ai dati contenuti nel database. Questo implica che per ottenere delle modifiche alla configurazione dovrò necessariamente modificare il database LDAP (si vedrà poi con quali strumenti ciò è possibile).

6.2. CFengine

CFengine è utilizzato sia per la distribuzione dei files dal server centrale, sia per alcuni compiti di monitoraggio e checkup del sistema come il controllo dei permessi, la pulizia dei files vecchi e il monitoraggio di quelli nuovi (implementazione di *policies*).

CFengine offre un insieme omogeneo di funzionalità già fornite da altri programmi, da rsync a netsaint/nagios, e dunque permette molti approcci differenti alla configurazione del sistema. Per la configurazione descritta in questo howto abbiamo scelto di implementare principalmente due modalità:

distribuzione

> copia in /etc di interi files, sia comuni a tutti i server, sia specifici per ciascuno di essi. Questo è il principale metodo di distribuzione del contenuto statico, si appoggia ad un protocollo specifico di `cfservd`, con una sua struttura autonoma di autenticazione, e non ha bisogno quindi di installare chiavi ssh o di abilitare rsh.

editing

> modifica di files di configurazione. CFengine possiede delle direttive piuttosto potenti per modificare il contenuto di files di testo: è possibile per esempio commentare linee selezionate in base a regexp, e aggiungere testo a piacere.

modalità che possono essere combinate a piacere, potendo quindi programmare modifiche specifiche per ciascuna macchina su dei files di configurazione distribuiti centralmente. Gli stessi files di configurazione di CFengine sono gestiti in questo modo. Di seguito si vedrà come, seguendo passo per passo degli esempi concreti.

CFengine si configura con dei files contenenti una serie di regole che si applicano a seconda delle "classi" dinamiche attive in quel momento. Le regole sono divise in sezioni, che vengono applicate in ordine, e con sintassi differenti: per esempio ci sono sezioni per controllare la presenza (e il checksum) dei files, per ripulire le directory dai files vecchi, per eseguire delle modifiche a dei files di testo, etc... Sono definite classi per svariati attributi specifici della macchina: il nome, il tipo di sistema operativo, anche il giorno e l'ora.

Per approfondire lo studio della configurazione di CFengine si raccomanda senz'altro la lettura dei seguenti testi:

- CFengine Tutorial - http://www.cfengine.org/docs/cfengine-Tutorial.html
- CFengine Reference Manual - http://www.cfengine.org/docs/cfengine-Reference.html

6.2.1. Struttura della configurazione di CFengine

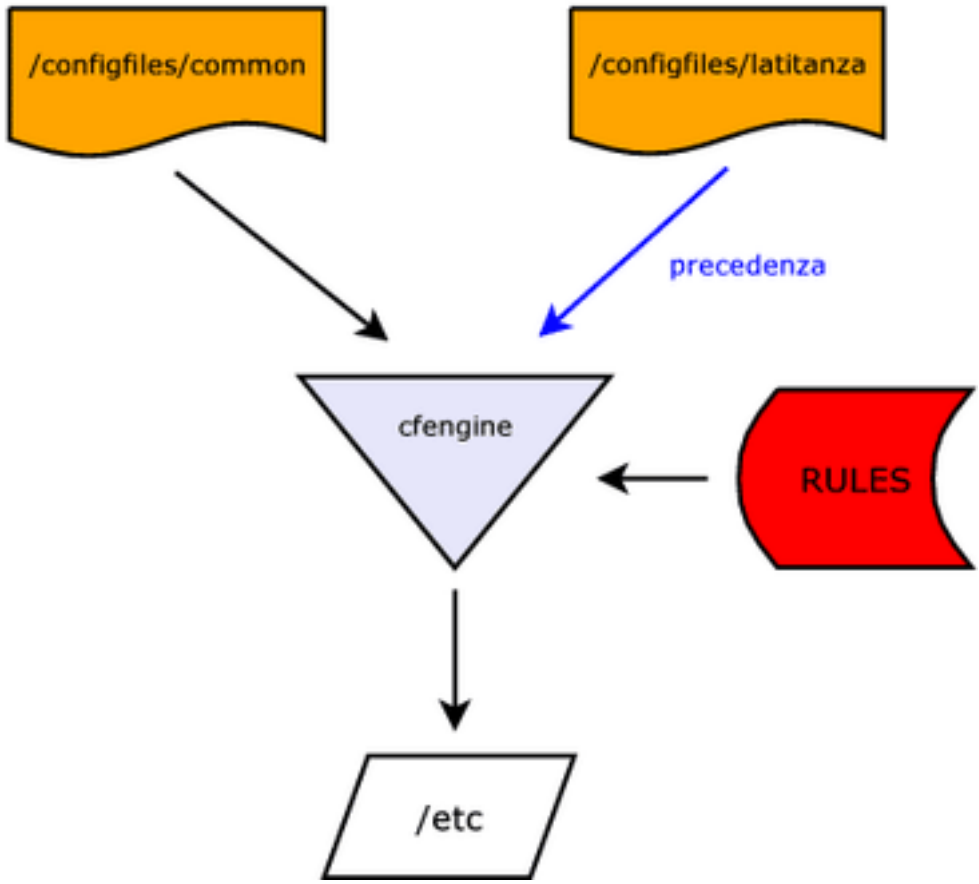
La configurazione delle varie macchine che compongono la rete viene gestita centralmente, modificando una serie di files in un unico punto (il *repository* centrale della configurazione). Ciascuna macchina mantiene poi una copia locale della versione più aggiornata di questo repository, in modo da poter sapere "cosa fare" anche in caso di problemi a raggiungere il server principale; questo permette anche di cambiare molto rapidamente la macchina che ha il ruolo di server principale.

Le copie dei files di configurazione sono, in questo esempio, installate sotto `/configfiles`. Le macchine di test si chiamano `test1`, `test2` e `test3`.

Se si guarda alla directory `/configfiles` si vede che è strutturata in questo modo:

```
\--+ configfiles
   |--+ cfengine
   |  \--- inputs
   |  \--- ppkeys
   |--+ ring0
   |  |--+ common
   |  |  \--- ...
   |  |--+ test1
   |  |  \--- ...
   |  |--+ test2
   |  |  \--- ...
   |  \--+ test3
   |     \--- ...
   \--+ ring1
      \--+ common
         \--- ...
```

Queste directory (a parte `cfengine`) contengono files che sono copiati (ricorsivamente se comprendono sottodirectory) nella `/etc` del sistema: la directory `common` contiene i files comuni a tutti i server, mentre le directory `testN` contengono ciascuna files specifici per la macchina dallo stesso nome. Se un file è presente sia nella directory generica che in quella specifica quest'ultima ha la precedenza. La suddivisione tra `ring0` e `ring1` è dovuta al fatto che tra i due differenti livelli di privilegio sono più le differenze che le somiglianze.



Come i dati di configurazione vengono generati per le varie macchine.

La directory `cfengine` contiene la configurazione di CFengine ed è gestita separatamente. I files in `cfengine/inputs` controllano completamente l'operato di CFengine e vanno modificati a mano (sulla copia principale!) per cambiare un po' di parametri specifici dei server. Mediante questi files si possono anche impostare delle *policies* sui vari sistemi, per automatizzare alcuni compiti di routine per esempio.

6.2.2. Esempio di configurazione di un servizio

Mentre è abbastanza semplice, dato lo schema qua sopra, capire come sia possibile gestire un servizio che ha solamente bisogno di configurazione statica (cioè copiando i suoi files di configurazione direttamente in `/configfiles/ringN/common/qualcosa` o `/configfiles/ringN/HOST/qualcosa`), può essere utile mostrare come sia possibile configurare CFengine per gestire un servizio che ha bisogno di una configurazione specifica per ciascun sistema piuttosto complicata.

6.2.2.1. Configurazione di CFengine per Tinc

Si è già introdotto il software Tinc[5], che implementa le reti private virtuali corrispondenti ai vari anelli basati sul trust. La configurazione di base è stata introdotta nel Capitolo 2.2: VPN>.

E' poi necessario effettuare un po' di modifiche a questi file di configurazione di base, modifiche dipendenti dalla macchina e che quindi è particolarmente comodo assegnare a CFengine.

La configurazione di CFengine che è stata realizzata già si occupa di riprodurre una copia di `/configfiles/ringN/common/tinc` su tutte le macchine (vedi il commento al file `cf.sysconfig` più sotto). Per realizzare le modifiche ulteriori sono stati creati dei file in `/configfiles/cfengine/inputs` di nome `cf.tinc.ringN` (uno per ciascun anello VPN). Uno di questi è riprodotto qui di seguito, con commenti:

```
control:

    vpn_ring0_name = ( ring0 )
    vpn_ring0_cf_dir = ( /etc/tinc/$(vpn_ring0_name) )
```

Questa è la sezione "control", dove si impostano variabili globali relative a tutta la configurazione.

```
links:

        $(vpn_ring0_cf_dir)/rsa_key.priv ->! $(vpn_ring0_cf_dir)/hosts/$(host).priv
```

La sezione "links" permette di specificare i link simbolici che vogliamo creare. Qui si dice a CFengine di creare un link alla chiave privata (`/etc/tinc/ring0/rsa_key.priv`), che per ciascuna macchina punterà ad un file differente.

```
files:

        $(vpn_ring0_cf_dir)/hosts/*.priv mode=400 o=root action=fixplain
        $(vpn_ring0_cf_dir)/tinc-up mode=755 o=root action=fixall
        $(vpn_ring0_cf_dir)/tinc-down mode=755 o=root action=fixall
```

La sezione "files" comprende vari controlli che possiamo fare sui files. Qui controlliamo che i due files specificati abbiano i permessi necessari per essere eseguiti, e siano di proprietà dell'utente specificato (action=fixall causa la correzione dei permessi se quelli riscontrati effettivamente sono differenti da quelli richiesti). Inoltre le chiavi private non devono essere leggibili da altri utenti a parte `root`.

```
editfiles:

        { /etc/tinc/nets.boot
          AutoCreate
          AppendIfNoSuchLine "$(vpn_name)"
        }
```

Nella sezione "editfiles" si istruisce CFengine a manipolare i files di testo che in genere contengono la configurazione di un sistema. In questo caso il file `/etc/tinc/nets.boot`, almeno sui sistemi Debian, contiene semplicemente un elenco delle VPN da attivare al boot: il comando AppendIfNoSuchLine serve ad aggiungere a questo file una

linea contenente il nome della nostra VPN, nel caso non fosse gia' presente. Se il file non esistesse verrà creato (AutoCreate).

```
{ $(vpn_ring0_cf_dir)/tinc.conf
  CommentLinesContaining "ConnectTo = $(host)"
  DeleteLinesStarting "Name ="
  AppendIfNoSuchLine "Name = $(host)"
}
```

Le istruzioni per manipolare il testo hanno in genere dei nomi piuttosto espressivi. Questa serie qui serve per essere sicuri che il file di configurazione contenga solamente una linea con il nome della macchina, e che su ciascuna macchina il server VPN non cerchi di connettersi a sé stesso.

```
{ $(vpn_ring0_cf_dir)/tinc-up
  BeginGroupIfNoSuchLine "# added by cfengine; do not edit"
    Append "# added by cfengine; do not edit"
    Append "/sbin/ifconfig $(dollar)INTERFACE $(vpn_ring0_ip)
            netmask 255.255.0.0 up"
  EndGroup
}
```

Qui infine creiamo il file che attiva l'interfaccia IP corrispondente alla VPN, una volta che la connessione sia stata stabilita (l'ultima Append è una sola linea, spezzata per esigenze tipografiche, attenzione!). L'editing è "protetto", nel senso che il file non verrà editato due volte grazie al controllo sulla presenza del commento.

```
!ring0_gw::
     { $(vpn_ring0_cf_dir)/tinc-up
       AppendIfNoSuchLine "/sbin/ip route add 172.17.0.0/16 via 172.16.1.1"
     }
```

Questo segmento aggiunge una route per la VPN ring1 su tutte le macchine tranne quelle che appartengono alla classe dei gateway tra ring0 e ring1 (`ring0_gw`), che non ne hanno bisogno.

6.2.3. Configurazione attuale di CFengine

I files presenti nella directory `/configfiles/cfengine/inputs` controllano completamente il comportamento di CFengine. Sono strutturati in maniera modulare, di modo che ciascun file contenga tutte le direttive relative ad un particolare sottosistema (sperando così di fare in modo che le direttive strettamente collegate tra loro siano raccolte nello stesso file).

Abbiamo già intuito come questi files siano divisi in "sezioni" (che specificano tipologie differenti di azione) e siano pilotati da un sistema di "classi" dinamiche (che saranno differenti su ciascuna macchina). Può essere utile a questo punto capire a cosa servono i vari moduli presenti, ed esaminarne la funzionalità:

`cfagent.conf`

Principale file di configurazione per `cfagent`. Contiene la definizione di alcune variabili fondamentali (come il dominio e l'indirizzo email dell'amministratore), ed include tutti i file di configurazione successivi (nella sezione *import*). Un parametro fondamentale contenuto in questo file è la sequenza in cui le varie "sezioni" vanno eseguite:

```
actionsequence =
    (
    resolve
    copy
    directories
    files
    links
    editfiles
    shellcommands
    tidy
    disable
    )
```

`cf.cfengine`

File che si occupa di mantenere la copia locale di /configfiles, e di installare correttamente tutte le chiavi pubbliche conosciute per CFengine.

`cf.linux`

File con la configurazione specifica di Linux, viene eseguito solo se l'host appartiene alla classe "linux". Non contiene nulla di particolarmente intelligente, solo qualche esempio.

`cf.sysconfig`

File che implementa la copia della configurazione in /etc prima dalla directory generica (`/configfiles/common`) e poi da quella specifica (`/configfiles/NAME`). I files sono copiati dal repository principale mantenendone i permessi. Per capire come è semplice programmare un'azione del genere con cfengine può valer la pena di guardare la parte di questo file che codifica questo compito:

```
control:

    any::

        rconfigdir = ( "$(configdir)/$(ring)" )
        configpath = ( "$(rconfigdir)/common:$(rconfigdir)/$(host)" )

copy:

    any::

        $(configpath)           dest=/etc/
                                recurse=inf
                                timestamps=preserve
                                backup=timestamp
```

Che copia tutti i file (per via del recurse=inf) dalle due directory specificate in configpath in `/etc`, preservandone permessi e timestamp, mantenendo inoltre, nel caso dovesse effettuare qualche modifica, per esempio per rimuovere un file differente trovato localmente, una copia di backup dei file modificati in `/var/lib/cfengine2/repository`. La linea inform=true permette di notificare gli amministratori ogni volta che CFengine modifica un file.

Inoltre mantiene un database dei checksum (in stile Tripwire) dei file binari di sistema.

`cf.tinc.ring0`
`cf.tinc.ring1`

Configurazione di Tinc (vedi il paragrafo 2.2.3).

`cf.apache`
`cf.mysql`
`cf.postfix`
`cf.bind`
`...`

Configurazione degli altri servizi. In relazione a questi un'altra cosa che è utile notare è che per capire se bisogna riavviare un servizio, e dunque rilevare quando vi sia stato un cambiamento effettivo dei files di configurazione che vengono editati, bisogna appoggiarsi ad una directory temporanea, dove copiare il file da `/configfiles` per poi modificarlo e confrontarlo con quello in `/etc`: in questo caso se vi è una differenza si può definire una classe che riavvia il servizio.

6.2.4. Aggiornamento automatico

Per configurare il sistema in maniera da permettere l'aggiornamento remoto delle configurazioni (e il trasferimento dei files dalla copia principale a quelle locali) è necessario abilitare `cfservd` e `cfexecd` su tutti i server. L'autenticazione degli host avviene mediante un sistema autonomo di chiavi RSA, di cui è necessario verificare la distribuzione iniziale per permettere il bootstrap del sistema (vedi Appendice A>).

Altra cosa da controllare è la configurazione delle access list in `/configfile/cfengine/inputs/cfservd.conf`.

CFengine funziona secondo un modello *pull*, per cui sono i client a richiedere al server principale l'aggiornamento della configurazione. `cfexecd` permette l'esecuzione automatica di `cfagent` a intervalli di tempo determinati (invece che da crontab). Di default questo aggiornamento avviene ogni ora.

Nel caso in cui si voglia forzare un'aggiornamento istantaneo (*push*) su tutte le macchine, per esempio dopo una modifica urgente della configurazione, è disponibile il comando `cfrun`, che richiede a tutte le macchine l'esecuzione immediata di `cfagent`.

6.2.5. Sicurezza

A margine delle riflessioni presenti nel Tutorial di CFengine[6], si può notare che CFengine è soggetto ai problemi dovuti all'implementazione di un proprio modello di trust e dunque dipende dal controllo sulla distribuzione delle chiavi. Questo è comunque più o meno il meglio che è possibile avere in ogni caso, volendo mantenere una sufficiente flessibilità.

Sotto un altro aspetto, essendo il comportamento di CFengine in genere non distruttivo e ampiamente monitorabile (con qualche --verbose si ottiene una quantità quasi molesta di informazioni), questo software rappresenta probabil-

mente uno strumento in grado di *aumentare* la sicurezza complessiva del sistema, in quanto è in grado di imporre sui server una *policy* comune e controllata.

6.3. Script

Gli script che configurano il sistema in base ai contenuti del database sono fatti in modo da effettuare sempre le minime modifiche possibili, evitando di modificare il sistema se già si trova nello stato desiderato: queso permette di eseguirli anche molto spesso, e di accorgersi quando un servizio va riavviato perché è cambiata la sua configurazione.

Alcuni degli script che sono più pesanti sul sistema (ad esempio perché devono controllare permessi e presenza di migliaia di files o directory) supportano due modalità di esecuzione, una *veloce* (che serve essenzialmente a creare nuovi oggetti che non siano presenti sul sistema), e una *completa* che effettua tutti i controlli necessari. Questo permette, differenziando i periodi di esecuzione, di avere una rapida risposta all'inserimento di nuovi oggetti nel database senza dover caricare eccessivamente il sistema.

Note

1. http://cfengine.org/
2. http://subversion.tigris.org/
3. http://www.cfengine.org/docs/cfengine-Tutorial.html
4. http://www.cfengine.org/docs/cfengine-Reference.html
5. http://www.tinc-vpn.org/
6. http://www.cfengine.org/docs/cfengine-Tutorial.html#Security%20and%20cfengine

Capitolo 7. Certification Authority

Per gestire le connessioni SSL di tutti gli utenti su tutti i server, abbiamo ritenuto opportuna la creazione di una propria Certification Authority, che possa firmare tutti i certificati SSL utilizzati dai vari server, facendo sì che essi possano riconoscersi tra di loro e dagli utenti.

Siccome è stato necessario implementare delle cose non banali (come ad esempio la possibilità di avere certificati validi per più nomi differenti di uno stesso server), abbiamo aggiunto questo capitolo con un breve *how-to* che altri potrebbero trovare utile.

7.1. Configurazione iniziale

Come directory dove mantenere e gestire i certificati utilizzeremo `/opt/ca`:

```
$ export CADIR=/opt/ca
$ mkdir -p $CADIR
```

creiamo la struttura delle directory ed alcuni file necessari alla CA:

```
$ mkdir $CADIR/certs
$ mkdir $CADIR/conf
$ mkdir $CADIR/crls
$ mkdir $CADIR/ext
$ mkdir $CADIR/newcerts
$ mkdir $CADIR/private
$ chmod g-rwx,o-rwx private
$ echo '01' > serial
$ > index
```

`serial` conterrà il prossimo seriale da utilizzare, mentre `index` è un database contenente i riferimenti ai certificati firmati dalla CA.

Ora creiamo una configurazione per la nostra CA. Quella utilizzata di default da OpenSSL nella distribuzione Debian *sarge* è `/etc/ssl/openssl.cnf`.

Per la nostra CA utilizzeremo il file di configurazione `/opt/ca/conf/ca.conf`. Per farlo utilizzare correttamente ad openssl, senza specificarlo nella linea di comando, basterà esportare il percorso nella variabile OPENSSL_CONF.

```
$ export OPENSSL_CONF=$CADIR/conf/aica.conf
```

Il contenuto di `ca.conf`:

```
RANDFILE               = $ENV::CADIR/.random

[ ca ]
default_ca             = CA_default
```

```
[ CA_default ]
dir                     = $ENV::CADIR
certs                   = $dir/certs
crl_dir                 = $dir/crl
database                = $dir/index
new_certs_dir           = $dir/newcerts
certificate             = $dir/ca.pem
serial                  = $dir/serial
crl                     = $dir/crl.pem
private_key             = $dir/private/ca.key
x509_extensions         = certificate_extensions
email_in_dn             = no
default_days            = 3643
default_crl_days        = 31
default_md              = sha1
preserve                = yes
policy                  = policy_match

[ policy_match ]
countryName             = supplied
organizationName        = supplied
organizationalUnitName  = optional
commonName              = supplied
emailAddress            = supplied

[ policy_anything ]
countryName             = optional
organizationName        = optional
organizationalUnitName  = optional
commonName              = supplied
emailAddress            = optional

[ req ]
default_bits            = 4096
default_md              = sha1
default_keyfile         = privkey.pem
distinguished_name      = req_distinguished_name
attributes              = req_attributes
x509_extensions         = v3_ca
string_mask = nombstr

[ req_distinguished_name ]
countryName                     = Country Name
countryName_default             = IT
countryName_min                 = 2
countryName_max                 = 2
0.organizationName              = Organization Name
0.organizationName_default      = Intra.Org
organizationalUnitName          = Organizational Unit Name
organizationalUnitName_default  =
commonName                      = Common Name
commonName_max                  = 64
emailAddress                    = Email Address
```

```
emailAddress_max                = 60
emailAddress_default            = ca@infra.org
SET-ex3                         = SET extension number 3

[ req_attributes ]

[ certificate_extensions ]

[ v3_ca ]
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid:always,issuer:always
basicConstraints = critical, CA:true
keyUsage = cRLSign, keyCertSign
nsCertType = sslCA, emailCA, objCA
nsComment = "InfraCA"
subjectAltName=email:copy
issuerAltName=issuer:copy
```

7.2. Creazione CA

Il primo passo da fare per creare una CA è quello di creare un certificato di root, quel certificato con il quale firmeremo tutti gli altri certificati:

```
$ openssl req -new -x509 -keyout $CADIR/private/cakey.pem \
    -out $CADIR/cacert.pem -days 3643
```

vediamo il nostro buon certificato e la relativa chiave privata:

```
$ openssl x509 -text -noout -in $CADIR/cacert.pem
$ openssl rsa -noout -text -in $CADIR/private/cakey.pem
```

aggiorniamo il file ‘serial‘:

```
$ openssl x509 -in $CADIR/cacert.pem -noout -next_serial \
    -out $CADIR/serial
```

7.3. Certificati

Ora creiamo le richieste di certificato che ci interessano (ad es. per il servizio web):

```
$ openssl req -new -keyout $CADIR/certs/web_key.pem \
```

```
    -out $CADIR/certs/web_req.pem -days 1097 -nodes
```

durante la creazione ci verranno chiesti l'*OU* ed il *CN*, da riempiere rispettivamente con la tipologia del servizio ed il dn a cui fa riferimento il servizio.

I *subjectAltName*, quelle simpatiche estensioni di X.509v3, verranno specificate nei file extension (presenti in `/opt/ca/ext`) e verranno aggiunti quando la richiesta verrà firmata utilizzando il certificato di root della CA.

L'opzione `-nodes` ci permette di non cifrare la chiave, così da non dover inserire nessuna password durante la gestione e l'utilizzo della stessa. Nel caso venga fatta una richiesta senza l'opzione `-nodes`, per eliminare la password dalla chiave basterà eseguire il comando

```
$ openssl rsa -in $CADIR/certs/web_key.pem \
    -out $CADIR/certs/web_key.pem
```

Vediamo la richiesta di certificato:

```
$ openssl req -text -noout -in $CADIR/certs/web_req.pem
```

Firmiamo il certificato appena creato:

```
$ openssl ca -days 1097 -policy policy_anything \
    -out $CADIR/certs/web_cert.pem \
    -extfile $CADIR/ext/webserver.ext \
    -infiles $CADIR/certs/web_req.pem
```

Ora sarà possibile eliminare `web_req.pem` e portare (nel caso del certificato per il server web) il certificato `web_cert.pem` e la rispettiva chiave `web_key.pem` su ogni server.

Vediamo come è stato modificato il nostro certificato:

```
$ openssl x509 -text -noout -in $CADIR/certs/web_cert.pem
```

Possiamo notare l'aggiunta di alcune estensioni (definite nel file `/opt/ca/ext/webserver.ext`), prima non presenti. Questo è il contenuto del file:

```
basicConstraints        = CA:false
nsCertType              = server
keyUsage                = nonRepudiation, digitalSignature, \
                          keyEncipherment, dataEncipherment
extendedKeyUsage        = serverAuth
nsComment               = "web services"
subjectKeyIdentifier    = hash
authorityKeyIdentifier  = keyid, issuer:always
subjectAltName          = @subject_alt_name
issuerAltName           = issuer:copy
nsCaRevocationUrl       = http://dominio.org/crl/cacrl.crl
```

```
nsRevocationUrl         = http://dominio.org/crl/cacrl.crl
crlDistributionPoints   = @cdp_section


[ subject_alt_name ]
DNS.1=dominio.org
DNS.2=www.dominio.org
DNS.3=altro.dominio.org
email=copy

[ cdp_section ]
URI.1=http://dominio.org/crl/cacrl.crl
```

Per creare i certificati per tutti gli altri servizi basterà cambiare opportunamente i nomi delle richieste, delle chiavi e dei certificati.

7.4. Revoca e CRL

Per revocare il certificato `web_cert.pem` basterà eseguire il comando:

```
$ openssl ca -revoke $CADIR/certs/web_cert.pem
```

ed aggiornare la lista delle revoche (CRL versione 1):

```
$ openssl ca -gencrl -out $CADIR/crl/cacrl.crl
$ openssl crl -in $CADIR/crl/cacrl.crl \
    -out $CADIR/crl/cacrl.crl -outform DER
```

il secondo comando serve a convertire la CRL dal formato PEM (un file in codifica base64) al formato DER, dato che alcuni software non accettano revoche in formato PEM.

Capitolo 8. Il Web

Questo capitolo contiene informazioni su come, e perché, sono configurati i servizi web. Per affinità si parla anche di MySQL ed FTP dato che sono servizi direttamente (ed esclusivamente) collegati alla gestione dei siti web degli utenti (adesso che il database delle utenze è in LDAP, MySQL può essere dedicato solamente a questo scopo).

È importante sottolineare come in effetti le configurazioni contenute in questo capitolo siano da considerarsi un semplice esempio dei diversi modi in cui è possibile configurare il servizio web di un server che si voglia avvalere delle potenzialità del meccanismo di distribuzione che viene descritto in questo documento

8.1. Apache

8.1.1. Struttura del web

Nel pensare la configurazione dei vari server web sulle diverse macchine, è stato necessario anche ripensare all'organizzazione dei domini "infrastrutturali" che serviamo, particolarmente in relazione al protocollo https, che dà in effetti alcuni problemi se gestito nel modo in cui abbiamo fatto per anni: cioè fornendo i vari servizi privati (webmail, servizio di cambio della password, "gatti" per le password perdute, accessi mysql, etc...) su domini di secondo livello differenti: webmail.dominio.org, squirrel.dominio.org, passwd.dominio.org e molti altri.

Sfortunatamente il certificato SSL che il server web utilizza contiene un solo hostname (nell'attributo CN), con cui il browser confronta la URL e in caso siano differenti visualizza un messaggio allarmante di avviso all'utente, anche se questo ha installato il certificato della CA. Questo vanifica in un certo modo il fatto stesso di spingere gli utenti ad installare la chiave della CA e ad utilizzare la struttura di PKI in modo corretto.

Questo problema si può risolvere in almeno due modi: il primo comporta l'utilizzo dei cosiddetti certificati *wildcard*, ovvero ad esempio un certificato per **.dominio.org*, che sarà valido per tutti gli host compresi in dominio.org. Questo trucco, benché paia godere di uno status un po' oscuro negli RFC, funziona con la stragrande maggioranza dei browser.

Noi però, anche per approfittare di questa occasione per fare ordine nell'organizzazione stessa dei siti, abbiamo preferito un altro approccio: si è infatti deciso di accorpare i vari sottodomini su di un unico dominio, www.dominio.org. Inoltre, dato che è necessario comunque raggiungere ciascuna singola macchina in modo indipendente, ogni macchina risponde anche ad un indirizzo www*N*.dominio.org, dove *N* è un numero. Fortunatamente un'estensione molto popolare delle specifiche X509 (subjectAltNames) ci permette di aggiungere, nel certificato del web server, qualche (il numero è limitato) altro nome di dominio, per cui si può creare un certificato, ad esempio per www.dominio.org, www1.dominio.org, www2.dominio.org eccetera, in modo che in ogni caso tutte le transazioni su protocollo HTTPS possano essere correttamente validate con la semplice installazione del certificato della CA.

8.1.2. Configurazione

La configurazione di Apache 2 è stata progettata per inserirsi nello schema modulare utilizzato da Debian: dunque è strutturata in questo modo: (la directory `apache2` qua sotto è quella che si trova in `/configfiles/ring0/common`)

- `apache2/apache2.conf` - file di configurazione principale di apache2, direttive generiche sul funzionamento del server

- `apache2/virtualhost-template` - questo file serve da modello per la generazione automatica della configurazione dei virtual host: tutti i siti presenti nel database vengono generati sostituendo alcuni valori specifici in questo file (il nome dell'host ed il percorso del file di log, ad esempio) ed eventualmente integrando i relativi frammenti di configurazione dai files in `apache2/conf.d/extra`.
- `apache2/include/` - pezzetti di configurazione ripetuti molto spesso, che fa comodo raggruppare autonomamente: per esempio le direttive per configurare PHP per un sito, oppure per bloccare i robot rompiballe, etc:
  - `include/dominio-auth.conf` - limita l'accesso agli utenti del dominio dominio.org, includendo questo file nelle parti di configurazione desiderate è possibile restringere l'accesso a parti del sito (preferibilmente sotto https) ai soli amministratori:

    ```
    <Location "blahblah">
      Include /etc/apache2/include/dominio-auth.conf
    </Location>
    ```

  - `include/cache.conf` - abilita la cache dei contenuti (per le location dove uno dovesse fare da reverse proxy)
  - `include/coral.conf` - pezzo di configurazione da abilitare quando eventualmente dovessimo morire di eccessive richieste, redirige tutto sulla cache distribuita Coral
  - `include/php-common.conf` - configurazione standard di PHP
  - `include/stop-robots.conf` - blocca alcuni robots particolarmente rompicoglioni
- `apache2/mods-enabled/` - pezzi di configurazione che abilitano i vari moduli da caricare (`.load`) e eventualmente ne configurano le caratteristiche generali (`.conf`): invece di manipolarli con `a2enmod` questi li gestiamo con CFengine
- `apache2/sites-available/` - configurazione dei virtualhost strutturali, questi vanno poi abilitati sulle varie macchine con il comando a2ensite (questo a mano, che può darsi che non su tutte le macchine debbano andare gli stessi virtualhost): il che equivale semplicemente a fare un link al file relativo nella directory `sites-enabled`
- `apache2/conf.d/` - roba che viene inclusa dagli altri file nei posti opportuni
  - `conf.d/ssl/` - vari moduli di configurazione per i vari servizi che girano sotto https su questa macchina (cioè su www*N*.dominio.org)
  - `conf.d/virtualhosts/` - virtualhost degli utenti (generati automaticamente)
  - `conf.d/subsites-dominio/` - sottositi di dominio.org (generati automaticamente)
  - `conf.d/subsites-public/` - sottositi di public.org (generati automaticamente)
  - `conf.d/extra/` - in questa directory stanno files con eventuali direttive di configurazione che si vogliono aggiungere a mano per qualche sito particolare (il file deve avere il nome del dominio per i virtualhost, e il nome dell'alias per i sottositi, più l'estensione `.conf`; verrà inserito in un punto opportuno del file di configurazione)

L'obbiettivo che si è cercato di raggiungere era di avere una distribuzione approssimativamente uniforme delle richieste degli utenti sui vari server utilizzando il *round-robin* DNS (cioè la possibilità di fornire al client gli indirizzi IP in un ordine casuale all'atto della risoluzione). Questo approccio va bene per il contenuto statico, o comunque in sola lettura, perché per quanto riguarda le applicazioni web, nel momento in cui si effettua un login e si genera una sessione è auspicabile che l'utente venga indirizzato sempre verso la stessa macchina (anche se in genere può non avere importanza *quale*), così che l'accesso ai dati di sessione sia sempre locale (dunque veloce).

Per fare questo, dopo il round-robin DNS si utilizzano delle redirezioni HTTP, strutturate in questo modo:

- Un file `/etc/apache2/servers_map` definisce l'insieme dei server su cui è disponibile una certa applicazione. Contiene linee del genere (una per ciascun differente insieme di server che si vuole definire):

  ```
  pool    www1|www2|www3
  ```

- Nella configurazione del virtual host principale (www.dominio.org) sono presenti queste direttive:

  ```
  RewriteEngine On
  RewriteMap servers rnd:/etc/apache2/servers_map
  RewriteRule ^(/app.*)$ https://${servers:pool}.dominio.org$1 [R,L]
  ```

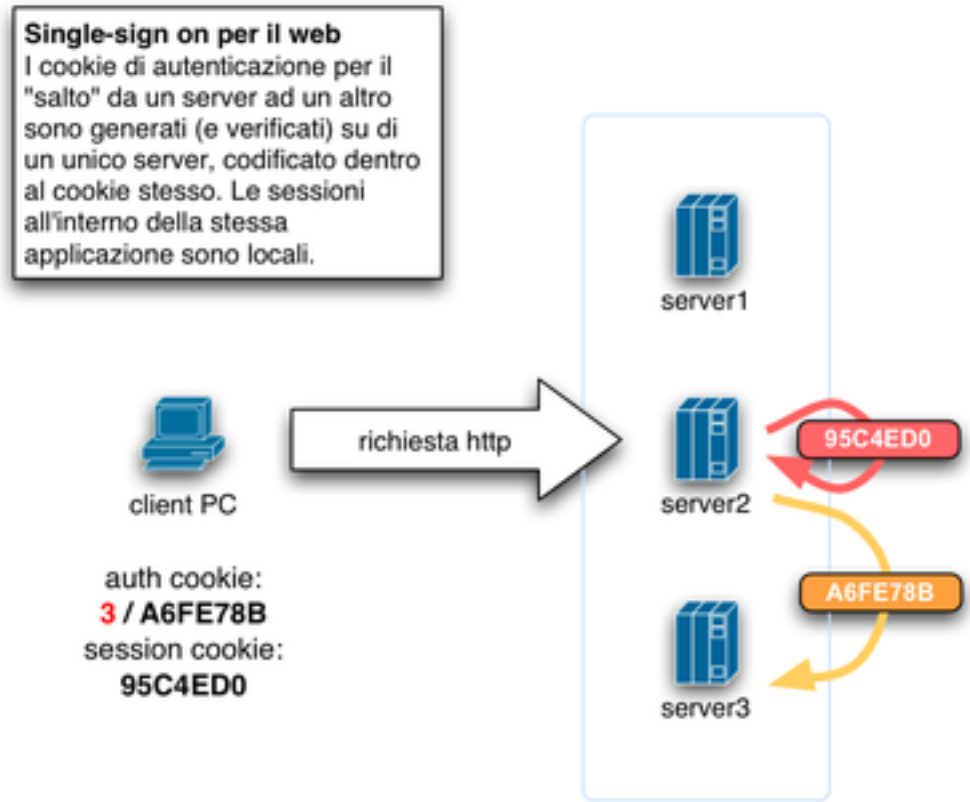
  che hanno l'effetto di redirigere su un server a caso (preso dall'insieme denominato *pool*) tutte le richieste per `/app`. In questo modo l'utente viene rediretto ad un server specifico per tutto il resto dell'interazione con l'applicazione.

8.1.3. Applicazioni distribuite

Vista la possibilità, abbiamo considerato senz'altro interessante installare alcuni applicativi (ad esempio il pannello di controllo per gli utenti, essenzialmente un'interfaccia al database LDAP, oppure il blog) contemporaneamente su tutti i server, per garantirne la funzionalità anche in caso una macchina non dovesse essere raggiungibile per brevi periodi.

Il software "normale", per essere utilizzato in questo modo, ha bisogno di alcune modifiche, principalmente la separazione delle query di scrittura e di lettura sui database MySQL (supponendo che il database sia replicato in modalità single-master), e la possibilità di "trasferire" le sessioni da una macchina all'altra, necessità questa dovuta al fatto che in ogni caso alcuni servizi devono essere localizzati con precisione su una specifica macchina che può essere diversa da quella su cui l'utente effettua il login iniziale.

Per questo secondo problema, la soluzione più comune (un database SQL centralizzato, o replicato) non ci avrebbe soddisfatto per via dell'elevato consumo di banda interno (non è pensabile che nelle condizioni date si possa effettuare una query mysql remota ad ogni accesso ad una pagina web), dunque è stato sviluppato un meccanismo apposito, che utilizza dei cookie di autenticazione che vengono consultati, come un meccanismo di autenticazione "trasparente", solo nel passaggio da una macchina all'altra. Intercettando le funzioni di login delle varie applicazioni, quando si riscontra un cookie di autenticazione valido viene ricreata una nuova sessione sul server di destinazione.



Meccanismo di single-sign on

8.2. MySQL

MySQL è usato principalmente per i vari database dei siti, e dunque, non ha bisogno di configurazioni particolari a parte l'esistenza di uno script in grado di leggere dal database LDAP quali db MySQL devono essere presenti su una determinata macchina ed eventualmente crearli.

Sono però presenti alcuni database che è utile condividere tra le varie macchine, e dato che sono prevalentemente sbilanciati verso la lettura (un esempio è un blog delle attività dell'organizzazione che gestisce i server) ha senso utilizzare le funzionalità di replicazione di MySQL.

8.2.1. Replicazione

Nella nostra situazione attuale l'unica soluzione attuabile è quella di una replicazione master-slave. il master su cui avvengono letture-scritture, e lo slave che replica (in tempo reale, o in differita se vogliamo). Lo slave ha la sola funzione di backup, in caso di caduta del master contiene già tutti i dati.

Il progetto di avere una soluzione, non solo di backup, ma anche di distribuzione del carico, significherebbe avere i server in modalità master-master, ove entrambi permettono lettura-scrittura, in equilibrio col carico; questa soluzione richiede tanta banda, e sopratutto banda stabile. Per server geograficamente lontani come i nostri non è attuabile.

Fate riferimento a "MySQL Reference Manual: Chapter 6, Replication in MySQL"[1].

I passi da seguire per mettere su la replicazione sono:

1. configurazione `my.cnf` sul master:

   ```
   log-bin
   #binlog-do-db=(se si vuole limitare a un singolo db)
   server-id       = 1
   ```

   questo abilita il logbin, è un file in `/var/lib/mysql` (può essere grosso). in debian va modificato anche `/etc/mysql/debian-log-rotate.conf` aumentando KEEP_BINARY_LOGS= - bisogna tenerne un numero sufficiente alto, in caso di caduta della replicazione lo slave per ricominciare deve avere a disposizione i vecchi log.

2. configurazione `my.cnf` sullo slave:

   ```
   server-id = 2
   master-host = xxx.xxx.xxx.xxx
   master-user = repl
   master-password = xxxxxxxx
   master-port = 3306
   #replicate-do-db =
   #skip-slave-start
   ```

   l'utente `repl` va creato sul master, con i seguenti permessi:

   ```
   GRANT REPLICATION SUPER RELOAD ON *.* TO REPL@IP_SLAVE IDENTIFIED BY 'PASSWORD';
   ```

   I privilegi SUPER e RELOAD servono nel caso si voglia usare dallo slave il comando `LOAD TABLE FROM MASTER` per il caricamento dello snapshot del db. Per grandi database questo metodo è assolutamente sconsigliato.

3. copia dello snapshot del db:

   a. interrompere temporaneamente le scritture sul master:

      ```
      FLUSH TABLES WITH READ LOCK;
      ```

   b. lasciando aperta la console mysql (con i processi di scrittura bloccati) si può procedere alla copia del db:

      ```
      cd /var/lib/mysql
      tar cvf /tmp/database.tar ./  --exclude mysql
      scp /tmp/database.tar utente@slave
      ```

   c. si prende nota dello status del master al momento dello snapshot: nome del *file* e *position*

      ```
      SHOW MASTER STATUS;
      +----------------+-----------+--------------+------------------+
      | File           | Position  | Binlog_do_db | Binlog_ignore_db |
      +----------------+-----------+--------------+------------------+
      | FILE-bin.011   | 117646022 | borsa        |                  |
      +----------------+-----------+--------------+------------------+
      ```

   d. rimozione del lock:

      ```
      UNLOCK TABLES;
      ```

4. copia dello snapshot sullo slave:

```
cd /var/lib/mysql
tar xvf database.tar
```

5. si informa lo slave sulla posizione da cui inziare la replicazione:

```
mysql> CHANGE MASTER TO MASTER_LOG_FILE='FILE-BIN.00X', MASTER_LOG_POS=XXXXXXXXX;
```

6. si lancia lo slave:

```
SLAVE START;
```

7. per monitorare lo stato della replicazione:

```
SHOW MASTER STATUS \G;
SHOW MASTER LOGS
```

Note

1. http://dev.mysql.com/doc/refman/4.1/en/replication.html

Capitolo 9. Anonimizzazione dei log

Uno dei temi a cui teniamo di più è la possibilità di *non* mantenere registrazioni degli accessi degli utenti (*log*), così da proteggere la scelta di anonimato dei nostri utenti. Questa possibilità è attualmente sotto grave minaccia, e ciò è una delle ragioni che ci hanno spinto a scrivere questo documento.

Per questo motivo, grazie anche al lavoro di altri attivisti in tutto il mondo, utilizziamo alcuni semplici accorgimenti per fare in modo che qualsiasi log in grado di associare accessi ai nostri servizi e numeri IP (che potrebbero portare all'identificazione dell'utente) non sia *mai* presente sui nostri sistemi neanche temporaneamente.

Oltre ai vari accorgimenti (dalla rimozione di `wtmp` all'utilizzo di relay per le connessioni ssh, a Tor[1]) per proteggere l'identità degli amministratori, vediamo come sono stati configurati i vari servizi:

9.1. Apache

La modifica più facile è quella per anonimizzare i log di Apache. E' infatti sufficiente creare una nuova tipologia di log in `apache2.conf`, diciamo "anonymous":

```
LogFormat "127.0.0.1 %l %u %t \"%r\" %>s %b
 \"%{Referer}i\" \"%{User-Agent}i\"" anonymous
```

(tutto su una riga).

Così si potrà definire un file di log di accesso anonimo per ciascun virtualhost:

```
CustomLog /var/log/apache2/access_log anonymous
```

9.2. syslog-ng

I restanti log di sistema, quelli che passano attraverso *syslog* (e ciò comprende i log della posta, di IMAP, degli accessi ssh di amministrazione...) vengono filtrati degli indirizzi e-mail ed IP direttamente nel demone syslog.

Per questo esiste una patch per syslog-ng[2], già presente di default in Debian, che permette di applicare un filtro mediante regexp (in `/etc/syslog-ng/syslog-ng.conf`:

```
filter f_strip { strip(ips); };
filter f_stripemail { strip("([0-9a-zA-Z]+[-._=+&])*[0-9a-zA-Z]+@([-0-9a-
zA-Z]+[.])+[a-zA-Z]{2,4}"); };
filter f_stripdomain { strip("([-0-9a-zA-Z]+[.]){2,}+([a-zA-Z]{2,4})"); };
```

Dopodiché è sufficiente inserire il filtro all'interno delle definizioni dei flussi:

```
filter f_proftpd { program("proftpd"); };
destination d_proftpd { file("/var/log/proftpd.log"); };
log {
        source(s_all);
```

```
filter(f_proftpd);
filter(f_strip);
destination(d_proftpd);
flags(final);
};
```

Questo tipo di riscrittura dei log non altera la possibilità di usare tool di analisi statistica, se non per la possibilità di identificare connessioni provenienti da uno stesso utente riconoscendolo appunto dall'IP.

9.3. Postfix

Anche quando i log di Postfix sono stati anonimizzati mediante syslog, rimangono comunque delle informazioni sensibili sui messaggi che è necessario rimuovere: in particolare, nei messaggi inviati attraverso i nostri server SMTP dagli utenti normalmente Postfix inserisce un header *Received* che segnala l'IP dell'utente.

Per rimuovere questo header è necessaria una patch (molto semplice per la verità) al codice di Postfix. Una versione sempre aggiornata alla distribuzione *unstable* di Debian è disponibile aggiungendo questa riga in `/etc/apt/sources.list`:

```
deb http://deb.riseup.net/debian/ unstable main
```

Note

1. http://tor.eff.org/
2. http://www.balabit.com/products/syslog_ng/

Appendice A. Installazione su una nuova macchina

Per l'implementazione di prova di questo howto si è scelta una distribuzione Debian[1] *sarge* che forniva pacchetti per tutto il software necessario ed era sufficientemente stabile da poter essere utilizzata su un server.

A.1. Installazione dei pacchetti Debian

Quindi il primo passaggio, per ciascuna macchina, dopo una normale installazione del sistema, consiste nell'installazione dei pacchetti necessari ad effettuare il bootstrap della configurazione:

```
$ apt-get install cfengine2
```

A.2. Distribuzione iniziale delle chiavi RSA

Dopodiché bisogna fare in modo che il client `cfagent` su questa macchina sia autorizzato a connettersi al server (che mantiene la copia principale della configurazione). E' dunque necessario generare le chiavi RSA per il nuovo server, per poi copiarle sulla macchina di configurazione principale e lasciare intanto che si propaghino sui server già configurati:

```
$ cfkey
```

`cfkey` genera le parti pubblica e privata della chiave RSA in

```
/var/lib/cfengine2/ppkeys/localhost.{pub,priv}
```

Bisogna poi copiare questi files sul server che mantiene la copia principale della configurazione, nella directory `/configfiles/ppkeys/`, la chiave pubblica in `root-NOME.pub` e `root-IP.pub`; la chiave privata invece in `root-NOME.priv` (dove `NOME` e `IP` sono rispettivamente il nome qualificato e l'ip del server). Per completare la distribuzione iniziale delle chiavi (che in seguito sarà effettuata automaticamente da CFengine) bisognerà anche copiare le chiavi pubbliche almeno del server principale, sempre nella directory `/var/lib/cfengine2/ppkeys`.

A.3. Prima configurazione

Il server dovrebbe adesso essere pronto per ricevere la configurazione con

```
$ cfagent --verbose
```

Si consiglia di esaminare l'output di questo comando (molto dettagliato), alla ricerca di eventuali errori. Se la configurazione iniziale è stata fatta correttamente dovrebbe essere presente la copia locale dei files di configurazione in `/configfiles`.

Note

1. http://www.debian.org/

Appendice B. Scalabilità

Questa appendice è dedicata ad un'analisi matematica, seppure approssimativa, della scalabilità di un modello di gestione della posta come quello descritto in questo documento. Tale modello viene utilizzato per una rete geograficamente distribuita di servizi che non supportano la redirezione del client (nel nostro caso SMTP e POP/IMAP).

In questo modello, i messaggi arrivano con eguale probabilità su ciascuno dei server disponibili, e vengono ritrasmessi al server di destinazione finale attraverso un canale interno (VPN).

L'ipotesi di distribuzione geografica (non-locale) è che questo canale privato utilizzi le stesse risorse di banda dei messaggi in arrivo. E' anche ragionevole supporre che questo canale abbia una qualche possibilità di comprimere il traffico che vi transita, e denoteremo con *c* il suo fattore di compressione.

Supponiamo allora di avere un traffico di messaggi in ingresso pari a *i* (in qualche unità di banda), e sia *N* il numero dei server: se la probabilità che il messaggio debba essere ritrasmesso è

$$\frac{(N-1)}{N}$$

allora per ciascun server il traffico in ingresso sarà

$$i_s = \frac{i}{N}$$

e genererà un corrispondente traffico in uscita (tenendo conto della compressione)

$$\mathrm{o}_s = c i_s \frac{N-1}{N}$$

sul canale privato.

Ciascun server riceverà un traffico o_s dagli altri *N - 1* server, dunque il totale del traffico in ingresso (contando anche il traffico che entra attraverso il canale privato) sarà:

$$\mathrm{i}_{tot} = i_s \left(1 + c\frac{N-1}{N}\right)$$

Per *N* che tende all'infinito il valore di i_{tot} tende a

$$(i_{tot})_{N\to\infty} = \frac{i}{N}(1+c)$$

Il significato di questa equazione è che un modello di traffico di questo tipo (cioè con reinstradamento interno) è linearmente scalabile con il numero *N* di server installati: infatti capovolgendo l'ultima equazione si ha che, fissata la disponibilità di banda di ciascun server, la capacità totale del sistema (in termini di traffico) risulta direttamente

proporzionale ad N. Dato poi che si parla, per il caso in questione, di traffico SMTP, il coefficiente di compressione c può assumere valori piuttosto bassi (elevata comprimibilità), minimizzando l'overhead della ritrasmissione.

Un'altro modo di vedere quest'ultimo risultato è che, considerato che la banda a disposizione di ciascun server è generalmente fissata (dalle disponibilità tecniche e dai costi), questa si suddivide, per ciascuna macchina, nel caso limite di N che diventa molto grande, a metà tra traffico "interno" e traffico proveniente dall'esterno.

Appendice C. Altre versioni di questo documento

L'originale di questo documento è scritto in Docbook (http://www.docbook.org/), da cui vengono generate automaticamente le versioni nei vari formati di presentazione (HTML, PDF, RTF).

C.1. HTML, PDF, RTF

Questo documento è disponibile nelle seguenti versioni:

HTML - https://www.autistici.org/doc/orangebook/
PDF - https://www.autistici.org/doc/orangebook.pdf