A/I Orange Book (1.0)

An *how-to* for the realization of a resilient network of self-managed servers

A/I Orange Book (1.0): An *how-to* for the realization of a resilient network of self-managed servers

This document describes in detail the configuration and management of a network composed of geographically distributed servers, considering the requisites of the servers and their objectives. The mechanism described in this page uses CFengine[1] and other free software tools to increase the automation involved in the process.

To check out for other version of this document, read Appendice C, Sezione 1

"Sharing knowledges, without founding powers."

—Primo Moroni

Table of Contents

Chapter 1. Introduction

1.1. First Thoughts

As early as in 2003, some members of the Autistici/Inventati collective started wondering what effects the upcoming wave of repression would have on our lives.

At that time it was clear to us that the growing importance of the digital communication media would call for the repression forces' attention, especially if connected to radical political contexts. We were far too optimistics: since then we have seen the increasing global paranoia pushing forward orwellian and panopticon ideologies towards total control, not only aiming at the restricted area of social and political dissent, but targetting the society at large.

We considered the main weak spots of the services we offered and wondered what compromises we could reach in balancing our political needs, the personal energies available in our collective and the responses required by the forecoming threats. We managed to sketch out three main problems:

- On one side, since our server was hosted by commercial providers, we could not guarantee for their physical integrity and for the privacy of the data stored on their disks. On the other side, trying to move our server to other locations (private houses or social spaces) would force us to defend a physical place or an object. This second option is not self-sustainable for us and in the end does not allow for a different outcome compared to the first one. Since costs did not make it possible to deploy better solutions (like cabling some private bunker), we had to resign to considering our servers embedded in a hostile and untrustworthy environment.
- The partial success of our project (at least considering the widespread use of our services) had put us in the uncomfortable position of hosting on a single box a far too large number of personal resources (as mailboxes, for example). This of course is a problem only when combined with our relative failure in spreading the use of privacy and self-protection tools as gpg, or, more generally, with our inability to encourage the protection of everyone's fundamental liberties. And with such a big number of users thinking of encrypting the server hard disk would have put us in the even worse position of having a single key for thousands of people's data. Therefore we needed to consider a more decentralized model for our services, in order to decrease the average number of users of any single box.
- The national and international legal scenerio seemed to make more and more unlikely the possibility to ensure anybody's data integrity, especially considering the cases of digital data seizing. We could only assure people that *they would still be able to communicate*, no matter what.

We drew many conclusions from these considerations, and this document is one of them: the technical design of an infrastructure aimed at anonymous communications and based on a number of servers distributed in various parts of the world. Since we think that every physical server can be easily compromised, we have to consider each box as a disposable node of the network.

1.2. Guidelines

The configuration described in this document has been thought of and written along guidelines that we believe respond to the needs we have just stated above. We have defined such guidelines as follows:

- Many of our replication schemes require that one server takes on the role of *master*, however we want to be able to quickly change this master server with any other box: this is why the general server configuration (e.g.: the files in the `/etc` directory, the users database, the mailbox/ftp/website database, etc.) is replicated on each box. Defining the "master" is thus simply equivalent to labelling a server as such, which allows for a fast switch in case something bad happens.
- Sensitive data (essentially mailboxes and mailing lists) have to be distributed among the various servers. The split-up criteria may be random or load-bound, but since our focus is not on a load-balanced situation, it is up to the server managers to decide. In any case, the association between each mailbox and its specific server needs only to be changed in the database, so that it is possible to relocate mailboxes very quickly if necessary. Saving the data lost in the compromised box is *not* considered a priority.
- If such a network is to work correctly, the different servers need to be substantially alike, configured in the same way and freely exchangeable. This implies that they have to be managed centrally, possibly by the same group of people.

This document describes essentially three different but related things:

1. the configuration of the various communication services (mail, web, mailing lists), developed in order to distribute our users among different servers without assigning them different virtual addresses (i.e.: all mail addresses will be something like @domain.org and not @server1.domain.org, @server2.domain.org, and so on...). This will keep in our users the feeling of always being in contact with a single organization.
2. the implementation of a mechanism to centrally manage all these configurations -- this mechanism permits to handle configurations not on each single server but with a higher level of abstraction (so that managing the whole network should be easier than managing N single servers).
3. the tricks to make all communications anonymous while using this infrastructure.

Of course one can find several ways to do all the things described above, and we have just chosen one amongst many: we do not imply this is the best solution ever, or even that this is an example of good practice. Generally speaking, we are just explaining why we have chosen something, based on our collective experience and on our greater familiarity with some tools than with others.

All the software used in this document is Free Software (the whole infrastructure is based on Debian[1]).

Each of the following chapters will introduce the configuration of a part of the infrastructure.

Notes

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

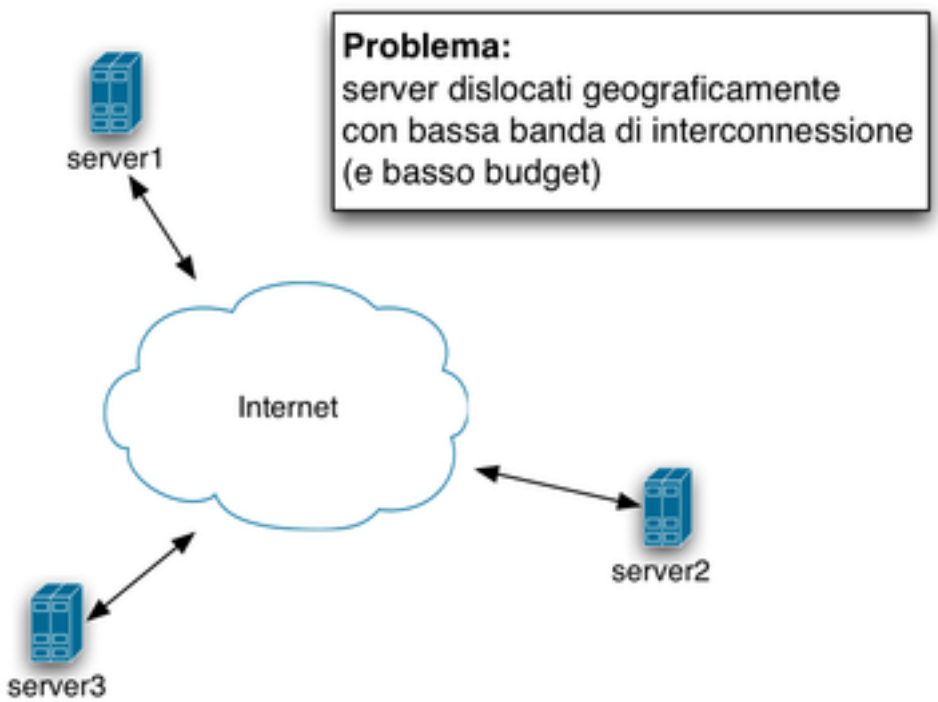
Chapter 2. Network

This chapter briefly describes the network configuration defining our *virtual organization*. The way this network is built, and its characteristics, have crucially determined the choices we made afterwards, so it is necessary to start from here to understand the rest of the infrastracture.

The network configuration is mantained on each server using CFengine (see chapter 6).

2.1. Network Structure

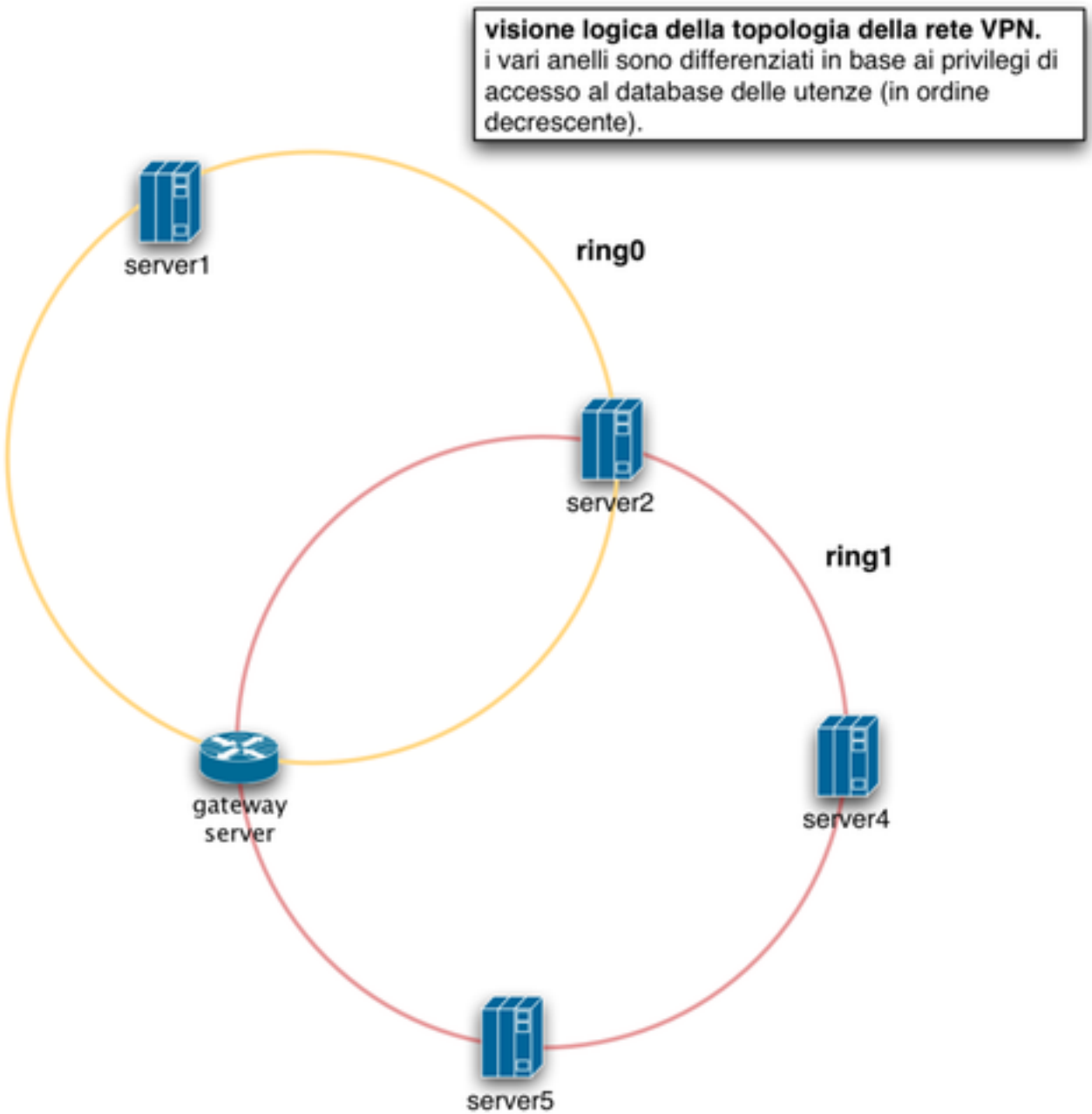
The various server are phisically located in different countries and each one has its own public IP address. One of the characteristics of this configuration is the low bandwidth available for server intercommunication: the geographical distribution of the servers means that the server-to-server communication channel shares the same connection of the normal communication, the one on which people use the various services (see Appendix B for a more in-depth study). This is the single most important difference between the model we are describing and traditional clustering and high-availability solutions, that generally imply the existence of some high-speed local interconnection between the servers.



The actual network

The internal communication channel uses a VPN (implemented through Tinc) connecting the various nodes of the network. This configuration allows for encrypted communications without the overhead of the cryptographic handshake

on each connection (as SSL does).



Scheme detailing the logical connection between the various servers

We consider the VPN a *trusted* connection: the simple presence on the VPN means the server can be trusted as part of the network; the single connections on this ring do not need to be encrypted (since tinc already provides for it on a lower layer). Many internal administrative services (as well as the synchronizing mechanisms) are available only on the VPN interfaces.

The full structure of the network includes several other servers, organized in concentric and connected *rings*, defining the level of "reliability" of the servers. For instance we can consider that nodes on ring 1 (the ring numbering starts from 0) have only a restricted access to the users LDAP database. These boxes are in fact used only for minor services, for backups and other side-tasks as relay connection for the ssh and vpn access of the network administrators (a single non-public point of access to the network with the ensuing traffic properly routed anonymously allows the administrator to mantain a sufficient degree of privacy :).

In the node connecting the different rings, firewalls are set up to manage the flows of data between one ring and the other. Another possibility we still haven't checked out thoroughly is to set up servers with this specific inter-connecting role (gateway servers in the picture) instead of burdenind already overbusy servers with this function. The nodes connecting different rings should be multiple, but we are still studying the best way to route the traffic between the rings even if one of the nodes is unreachable.

2.1.1. VPN

Tinc[1] is a VPN daemon with automated routing functions able to mantain N hosts connected with point-to-point connections. It also manages authentications and encrypting on an internal autonomous set of RSA keys that have then to be distributed accordingly to the software needs on the various servers.

It has been chosen among other solutions since it's very easy to install and allows to hide point-to-point connections between the single servers: it presents the OS with a single network interface, transparently managing the routing with its own internal procedures (e.g.: it is as though the boxes were all connecting to one another on a local network, or even better, since the software looks for alternate traffic routes if the direct connection between two nodes is failing).

Accordingly, we have chosen for our internal connections the network 172.16.0.0/16, distributing the ip addresses as follows (the addresses will be then added to the `/etc/hosts` file distributed through CFengine):

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

To avoid confusion, it can be useful to choose for these addresses a domain different from the one that will be used for public IP addresses: for example a subdomain (`test1.vpn.domain.org`) or a common name-suffix (`test1-vpn.domain.org`) could be used.

As regards the configuration, we have created base configuration files for a network called *ring0* (corresponding to the innermost ring in the trust-based VPN scheme) in `/configfiles/ring0/common/tinc/ring0`. Apart from the basic `tinc-up` and `tinc-down` files, the `tinc.conf` file is the main configuration location and it contains the following directives:

```
# 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
```

Note that the names of the servers used in this file need not be valid dns fully qualified hostnames, but need to correspond to the files in the `ring0/hosts` directory. It's only for simplicity with CFengine use that the names we have chosen are identical to the names assigned to the hosts.

The `/configfiles/ring0/common/tinc/ring0/hosts` directory should contain two files for each server, identified with their names with no domain suffix: `NAME` and `NAME.priv`. The RSA keys can be generated with the command:

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

We then need to edit the file with the public key to add some options, as in the following example:

```
# 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. DNS Organization

For such a plan to work, a correct organization of the DNS service is crucial.

In the DNS organization it is clear that we need to separate the management level and the users' services level: the network described above needs a clear and unambiguous naming of the various addresses connected to the different interfaces, so as to make it clear which connection we are referring to with a specific name. On the other hand, from the point of view of the users, the less specific the naming system is, the better they will be able to understand that they are dealing with a single "virtual organization". That is to say that the whole infrastructure needs to stay hidden from the eyes of the users that connect to www.domain.org and do not really want to know exactly which server they are connecting to, or whether their mailbox is in Japan or in Brazil.

2.2.1. Infrastructural Domain

We have chosen to use a specific domain for the infrastructure part of the network (infra.org in our case) to resolve the address of the various servers and of the non-public services. Since on this domain we do not offer any service to users, we do not need this domain to be public (or in any case publicly known). This separation is by no means an absolute necessity for the system to work, but could be handy to keep things a bit clearer.

This is more or less the structure of our infra.org zone dns record:

```
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
```

It’s fairly easy to add aliases to this zone for specific services or roles the server fulfills:

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

2.2.2. Other Domains

The DNS structure of "normal" domains needs to be standardized (i.e. all zones are identical in terms of their dns record), and it substantially implements all the round-robin mechanisms with which the load and traffic are approximately balanced over the various servers.

The common part of the dns record of all the domains is as follows:

```
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
```

This practically means that each mail is directed randomly to one of the three mail servers, and that http requests are distributed at random among the three webservers. DNS round robin is a quite rough way of performing load distribution, but it is enough for our purposes.

The only domains that have different dns records are those for which we need to identify the single boxes. This applies both to subsites (www.domain.org/something) specifically assigned to a single box (particularly because of the difficulty of multi-master mysql replication), and to some https services (as webmail). For these reasons the A record of these domains is as follows:

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

Last but not least, concerning the DNS organization (as we will detail more in depth in the chapter on Apache service), we have decided to centralized all the SSL services on a single domain (e.g. www.domain.org), so as to keep the coeherence of the name detailed on SSL certificates (that is one and one only for each server).

Notes

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

Chapter 3. Filesystem

The filesystem organization is, for the most part, common to all servers. This allows for example to move easily services and other content from one server to the other without having to change paths all over the structure, or to put a path directly in the users database without the fear of having to change everything, when moving things around (e.g.: the users homedir or the website document root).

3.1. Disks Organization

Devising a common partitioning scheme (which can be of course adapted to specific cases) makes it easier to manage the network and to install new boxes.

When possible, it is better to configure the servers with RAID and LVM, so as to manage the diskspace with the best level of flexibility (we generally use RAID1 or RAID5 and LVM on top of it). What follows is a trivial example of partitioning scheme we tried to stick to (but the scheme can vary very much depending on your hardware, disk space and economic needs).

```
/               - 500M
/tmp            - 500M noexec,nosuid
/usr            - 3G
/var            - 6-10G
/home/admins    - 4G, home of the administrators
/home/mail      - >20G, mailboxes
/home/users     - >20G, websites and ftp accounts
```

Aside from that, a number of filesystem optimizations had to be implemented, since we are expecting high levels of traffic. We will not go much into details here since it does not make very much sense to speak about performance in a general case: numbers will vary according to specific hardware and software characteristics, and benchmarking is really the only way to tell the useful from the useless solutions. There are, anyway, a few tips that might be valid: for example, you may want to issue the following command on each server:

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

This command allows the full journaling of data and metadata of the ext3 filesystem in the postfix spool directory (which helps preserving the spool contents in case of a filesystem crash), as well as setting the *noatime* flag, which improves performance by avoiding updates of the access time on inodes. (this example comes from: “Postfix on an ext3 filesystem”, R. Hildebrandt[1]).

Another very useful thing is to setup Amavis[2] to use a temporary directory in a *tmpfs* filesystem, when we need to save a MIME attachment file in its different parts. This further decreases the disk usage. You can do this with the following lines in `/etc/init.d/amavis` file:

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

and modifying the `/etc/amavis/amavisd.conf` file as follows:

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

3.1.1. Encrypted Partitions

The use of encrypted partitions has been excluded for what concerns the users data, not just for performance reasons, but since we thought quite unwise to be the only repository of a single key for thousands of different doors. Nevertheless, it is a valid strategy to protect the sensible data of the system as a whole: in our case the SSL certificates and the matching encrypting keys are a good place to start.

This mechanism implies the need to fill in a password when the server boots and is needed to avoid the unauthorized copy or modification of the data. We also considered the possibility of mounting the partition when starting each service, unmounting it just after it has started fully: this could protect the box also from intrusions when the box is up and running, but it seems to be only partially useful since at that level of compromise of your box it's very difficult to know what level of security your password will have while you are typing it.

Our implementation of encrypted partitions is based on dm-crypt. This more out of simplicity than for other reasons: if we had to encrypt larger partitions or partitions enduring higher levels of I/O, we would have probably chosen loop-AES (you can find interesting benchmarks and howtos here: http://deb.riseup.net/storage/encryption/benchmarks/).

To make it simple, this is how we create an encrypted image `private.img` and how we configure it:

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

We now need to create the mapper device to create the encrypted partition and insert the passphrase for the simmetric encryption:

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

We can now create the ext2 filesystem and mount it on `/mnt`:

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

3.2. Filesystem Synchronization

Some data (as for example the `/opt` directory we described before holding the content not connected to a specific user) have to be synchronized on the various servers. Since this is not a small amount of data and since it does not need

to be modified specifically for every single box, we have chosen *not* to distribute it through CFengine. Rather, we have chosen easier systems. At the moment we did not find a solution to actually have a truly shared filesystem, since the particular situation of our network (non-local boxes with low bandwidth connecting to each other) is not compatible with the standard design of distributed filesystems (which are generally targeted to local high-speed connections).

To copy all of the data simultaneously, we use then a `rsync` server, non encrypted, available on every single server on the VPN. This solution allows for a decent transfer speed, with a low overhead. The side effect of this solution is that it breaks the simmetry of the servers: one of them needs to be defined as the master and the others as the slaves the copies are distributed to, although as usual the "master" role can be switched in no time very easily.

The same mechanism will be used to move a single website or mailbox from one server to the other. To avoid too much confusion (and the risk of tragic errors) with misspelled commands, the rsync connections are read-only, so every server can only *pull* contents from the others.

Notes

1. http://www.stahl.bau.tu-bs.de/~hildeb/postfix/postfix_ext3.shtml
2. http://www.ijs.si/software/amavisd/
3. http://deb.riseup.net/storage/encryption/benchmarks/

Chapter 4. The users database

4.1. Introduction

This short chapter summarizes the structure of the database holding the user system data.

LDAP is a protocol to access *"directory based"* systems. It was born as a gateway for the OSI data repository standard called X.500: originally LDAP defined in fact only the transport (TCP/IP) and the format of the messages used by client to access directory based systems which followed the X.500 standard, and which implied the use of the full OSI stack (too complex and expensive to be implemented in the smaller LAN). Substantially the LDAP server was the gateway between a TCP/IP environment and a OSI environment, collecting client requests (TCP/IP) and forwarding them to a X.500 server using the OSI stack. Since OSI stack and X.500 were too complex, as time went by the developers thought of providing LDAP with a native storage directory system so as to make it independent and to make it possible to use it not only as X.500 gateway but on its own, since it was far slimmer than the known OSI standard.

Our choice of using LDAP for the management of the users data came as a consequence of evaluating the following characteristics:

- LDAP is optimized for a load consisting of very frequent read operations and very rare modifications
- LDAP can be very easily replicated and allows for redundancy of the whole database.

When we first wrote this document we did not find any other free and complete LDAP suite apart from OpenLDAP[1], so we have decided to use it, notwithstanding the fact that it's far from perfect and still seems to have several flaws. Later on, other solutions have become available (as for example Redhat Fedora Directory Server[2]) that need to be further tested, considering the pivotal role of the LDAP server in the structure we are describing.

4.2. Database structure

LDAP stores its data as objects (entries), each characterized by a certain number of attributes. Each attribute has a well-defined ID.

Objects are mantained hierarchically along their *distinguished name* (`dn` in short). Children objects inherit the dn of their parent as a suffix to their own dn. The ensuing structure is therefore a tree structure.

With a tree like the following:

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

the object dn would be:

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

Each object can have different attributes depending on the *classes* it belongs to. The inheritance in LDAP can be multiple: this means an object can belong to several classes at the same time. Each class defines mandatory and optional attributes that each object of that class needs to (or could) have. This structure is described in the so-called LDAP *schemas*.

4.2.1. Database Content

In the structure we have adopted, we can isolate our data from other virtual organizations data, collecting all our system data under the object `dc=infra, dc=org, o=Anarchy` (this is a standard naming convention).

A database includes several kinds of users and objects, which are stored in different branches of the LDAP tree under `dc=infra, dc=org, o=Anarchy`. For example, these are the main branches we use:

- `ou=Admins`: this branch includes the servers administrators, that is the users that have an ssh access to the boxes. *Note*: the administrator tasks that do not imply the use of an ssh access (e.g. the web administration panel) authenticate using objects identical to the ones stored in this branch; anyway these objects are located under the branch `ou=People`, and have their own set of password. This means that SSH access is completely separated from all the other type of access for what concerns authentication.
- `ou=People`: this branch includes all website, email, ftp users, and so on. This is the "virtual" part of the LDAP database.
- `ou=Operators`: this branch includes "system" users that allow the software to handle parts of the LDAP database without requiring the full access privileges of the LDAP administrator. It is mainly used by daemons and software agents accessing the LDAP database.
- `ou=Domains`: this branch includes an object for each "public" virtual domain managed by the organization. Postfix for example looks in this branch to decide whether a domain is local or not (to build up its transport tables).

4.2.2. Virtual Users

Throughout the following chapter we will mean "contact" by the word "user", that is an entity with access to various services, be it an organization, a person, or anything else. We introduce here this concept to better abstract the administration tasks and therefore the database structure.

Each user is represented in the database by an object identified through the `uid` key. We can choose whatever naming system for these objects, but in our case we have decided to name them after their main email addess (e.g. `uid=phasa@domain.org`). If we want to group the services of a whole organization, instead of a single user, we can choose to create an object called after a domain name as `uid=organizzazione.it` or anything else we deem meaningful (once a scheme has been laid out and chosen).

Each of these objects can be associated to different services, that will be stored under the "user" object. The type of these objects will correspond to the type of service. For example, these are some possible structures:

```
* a single user with a single mailbox

  ou=People
  \-- uid=user@domain.org                    shadowAccount
      \-- mail=user@domain.org               virtualMailUser

* a user with a mailbox and a website (therefore an ftp account)
```

```
  ou=People
  \-- uid=user@domain.org                    shadowAccount
      +-- mail=user@domain.org               virtualMailUser
      +-- alias=usersite                     subSite
      \-- ftpname=user                       ftpAccount

* an organization with more accounts, a website and a domain of its own

  ou=People
  \-- uid=organiz.org                        shadowAccount
      +-- mail=user1@organiz.org             virtualMailUser
      +-- mail=user2@organiz.org             virtualMailUser
      +-- cn=www.organiz.org                 virtualHost
      \-- ftpname=organiz                    ftpAccount
```

On the right of each object you can see the class it belongs to. Particularly, the user object belongs to the class `shadowAccount` (a POSIX standard) that will allow to resolve NSS lookups and quota management (allowing to specify a uid name and not only a numeric id for each user), even if it will never be used in authentication.

4.2.3. Authentication

In the structure described above, each service requiring authentication matches the requested credentials directly against the child object of the relative service. So the authentication credentials of the `shadowAccount` object identifying the user, even if present as requested by the scheme, are never directly used.

By using appropriate query filters, we can then make sure that for example the IMAP daemon matches the authentication of users against a `virtualMailUser` object, with its own password. This allows, in case, to have different passwords for the different services. This possibility is included in the *policy* that needs to be implemented through the use of administration tools: in fact, LDAP does not have an internal mechanism to specify particular relationships between objects (apart from the hierarchical relationship), a thing most other relational database can do natively. This is the reason why it is impossible to tell LDAP that an object needs to have the same password of the parent object. This of course implies a certain redundancy of the data in the scheme (but not too much).

We will now see how the different services have been configured (relatively to the LDAP authentication). For each service we specify the configuration files concerning the authentication and what is the query that they are set to do on the LDAP server, composed of a *base* limiting the scope of the query and a *filter* that selects one or more specific objects:

ssh

ssh authenticates through PAM with its own configuration file, selecting only users in the *ou=Admins* branch of the LDAP database

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

the FTP daemon authenticates using PAM but with another configuration file looking for objects, including the *objectClass=ftpAccount* attribute, and matching the *host* attribute with the name of the machine we are trying to authenticate on.

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=server1)(status=active)
pam_password crypt
```

dovecot

Dovecot is the IMAP daemon and authenticates mail users. It does not use PAM, but its own mechanisms.

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 authenticates users sending mail through SMTP. Saslauthd is the daemon actually carrying on the authentication. The configuration of this software is different from the others since it's possible to skip PAM and authenticate directly using an LDAP query. This unfortunately requires a different format for the configuration file. Furthermore we need to remember the `-r` option for the software since we authenticate users with both a name AND a domain (forgetting this option will cause the daemon to match only the user name and not the domain). We also need to remember to create the saslauthd socket in the Postfix chroot.

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
```

Note: all the queries above match the provided password with the one stored in the `userPassword` attribute. The passwords are stored in the LDAP database with different encoding mechanisms, detailed by a prefix that you can find in the attribute before the encoded password in brackets. E.g.:

```
{crypt}dlkj8h23dU9j1
```

Apart from *crypt* (which uses the `crypt` system function), it is possible to use the *MD5* or the *SSHA* encoding systems, both of which simply use a hash of the password. Many common commands (as `ldappasswd`) set the password using one of this mechanisms, considering them more secure. This is relevant when we think that there is not a single way to verify authentication: some services use the *bind LDAP* system, i.e. try to access the LDAP database using the provided credential: this means that it is the LDAP server itself that manages the password verification. But in other cases (like dovecot) applications behave differently: they get the password from the LDAP server and verify it on their own. The problem is that generally this particular services are *not able* to correctly check passwords unless they are stored using the *crypt* mechanism! So be careful when using administration tools.

4.3. The LDAP scheme

The following scheme defines the classes and attributes used in our example database. We added some comments to better explain the choices we made and the meaning of the various sections.

```
#
# infra.scheme
#

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

these are macros defining the OIDs for the classes and attributes we will describe. OID 1.1 is used for tests. One should register his/her own OID to avoid overlapping with other schemes defined by somebody else.

```
# short forms for very common data types
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



### Attributes

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

the *status* attribute has been introduced to allow objects to exist in different states (e.g. to specify in what phases of creation the object is, or to enforce a password change before users can access their newly created services, or to disable an account without deleting it). All of the filters we use for virtual services include a part verifying that the `status` of the object is set to *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 'website an apache alias is referring to'
        EQUALITY caseExactIA5Match
        SYNTAX String SINGLE-VALUE )
```

these attributes have been created particularly to store values for virtual hosts and subsites configuration of Apache server.

```
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 )
```

these last three attribute concern the MySQL database objects. Mysql is not able to authenticate users on an LDAP database but it's useful nonetheless to keep track of this informations from an administrative point of view.

```
### Classes

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

These is the base class from which all of the objects of virtual services are derived. In this way it's possible to have a common set of attributes relative to all virtual services objects (as attribute relatively to the management of the user or the box on which the services is hosted through the `host` attribute).

```
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 )
        )
```

These are the classes corresponding to the different virtual services. The only exception is the ftpAccount object, defined as a addon class to shadowAccount, so that the FTP server can authenticate using PAM. In the end there discrepancies are not important since the management software hides them.

4.4. LDIF

LDIF (LDAP Data Interchange Format) is a conventional format to describe the entries of an LDAP database. An LDIF file is simply a text file with a series of matching pair of attribute/value with the following syntax:

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

LDIF files are used to import, modify and export the database entries (they are clearly very useful considering they are plain and simple ascii). Here follows the logical structure and underneath it some examples of ldif commented accordingly:

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

Here comes the corresponding LDIF (directly from the migration script we used to move all of our users to the new system)

```
dn: uid=phasa@domain.org, ou=People, dc=infra, dc=org, o=Anarchy
shadowMax: 99999
uid: phasa@domain.org
cn: phasa@domain.org
homeDirectory: /var/empty
uidNumber: 13468
objectClass: top
objectClass: person
objectClass: posixAccount
objectClass: shadowAccount
objectClass: organizationalPerson
objectClass: inetOrgPerson
```

```
shadowWarning: 7
gidNumber: 2000
gecos: phasa@domain.org
shadowLastChange: 12345
sn: Private
userPassword: {crypt}x
givenName: Private
loginShell: /bin/false

dn: mail=phasa@domain.org, uid=phasa@domain.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@domain.org
userPassword: {crypt}$1$fa0c5a13$VVqsukrQmdr79LZg2xvnM.
mailMessageStore: domain.org/phasa/

dn: alias=phasa, uid=phasa@domain.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. LDAP Database Replication

Once the users database were created, and mailboxes had been evenly split among different servers, we still had to solve a major problem: where do we keep the database? On which server? Well, of course on ALL of them! :) Actually OpenLDAP allows us to split the database in how many parts we like, and to replicate the different parts in different ways, but at this stage of the project we thought it was better to just replicate the full database on each server.

OpenLDAP replication can be deployed in many ways: the easiest one implies a master and several slaves. The database updates are made on the master server and they get forwarded to all of the slaves; at the same time the request of new activation or updates can be made at any given slave, and it will know how to communicate to the master server the request that will then provide for the update. If the master server is attacked by a fleet of grasshoppers or some

other kind of major shit happens to it, the configuration of the other servers will allow us to select immediately a new master server with a one-line change in the configuration: in this way the whole system will continue to work.

4.6. `Slapd` Configuration

First of all, you need to install slapd (the server part of the free OpenLDAP suite). The server configuration is defined in the `/etc/ldap/slapd.conf` file. The very same file it's used also by slurpd (the daemon managing the replicatoin mechanism).

There are three different type of directives in the configuration file of the server:

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

The first set of directives set generic parameters for `slapd`, like the address to bind to, where to put logs, ACLs, and so on...

The second set of options defines parameters needed by the database backend. We decided to use BDB notwithstanding the reported consistency problems (a quick search on the web suggests daily backup of the data), since the other OpenLDAP-supported backend (LDBM) is even officially deprecated...

Last but not least, the third set of options defines parameter to configure the databases (in our case it's a single database), allowing you to specify what indexes to create, where to store teh database, and other options that it's better to check closely. It is absolutely necessary to create an index for each attribute that could be present in a filter. Note that you can create the index at any given moment, so if you forget something you can always add it later (using the **slapindex** command). However if you missed an index on a frequently used search attribute you will notice it very soon, since a missing index can almost bring a busy server to a complete halt!

4.6.1. Fine Tuning

We will not enter into the detail of the basic slapd configuration that anybody can find in Chpater 6: "The slapd configuration file"[3] of the "OpenLDAP Administration Guide". In this chapter we would like instead to cover the fine tuning we deployed for increased stability and performance. Nearly any operation on the servers, in fact, implies an LDAP request: this lets you understand how important the efficiency of slapd is to the overall system.

The main problem can be faced from two complementary perspectives: on one side we can try to reduce the number of request to the minimum (for example using various sorts of *caching* mechanisms); on the other side, we need to optimize the slapd configuration so as to get better performances.

As an example of the first perspective we can relief the server of the very frequent NSS requests. Many unnecessary requests can be avoided by specifying the *files* directive before the *ldap* directive in `/etc/nssswitch.conf`, so as to resolve system users without querying the LDAP database right away. The NSS caching daemon nscd can reduce further these queries to the bare minimum.

Concerning the optimization of the slapd configuration, you can find some thoughts below:

- It is absolutely *necessary* to create indexes in the LDAP database for every attribute used in the various service filters. This is the example you can find in our `slapd.conf` configuration file, in the database-specific section:

```
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
```

When a query is done with a filter depending on an attribute that has not been indexed, the server needs to examine individually each object in the database. This is to be avoided at all costs.

- Increasing as much as possible the slapd object cache could be handy. A value that could be used to keep the whole database in the cache (if it's not too big) is the amount of directives in the database. You can get this value running the `slapcat | wc -l` command.

  ```
  cachesize 50000
  ```

- Increasing the number of threads the server can spawn and configure a decent timeout for hanging connections can be very useful especially if the load on the LDAP server can lead to client timeouts:

  ```
  threads 64
  idletimeout 300
  ```

  If local LDAP connection uses up TCP sockets instead of UNIX socket, it could come in handy to widen the range of available ephemeral ports, editing for example the `/etc/sysctl.conf` file as follows:

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

  It has to be noted that this sort of tuning does good to the overall health of your server even independently of the LDAP suite and server.

- Last but least you can modify some parameter of the Berkley DB backend creating a file named `DB_CONFIG` in the directory where the database are stored (in Debian the directory is `/var/lib/ldap`). Here also it is better to increase the cache size up to at least 50 Mb (that in modern system is not a problem):

  ```
  set_cachesize 0 52428800 0
  ```

  It is possible then to specify that the creatoin of temporary files is to be carried on in a tmpfs filesystem:

  ```
  set_tmp_dir /dev/shm
  ```

  But most important of all, even if a bit risky, is to disable the write transaction log of the slave servers:

  ```
  set_flags DB_TXN_NOSYNC
  ```

  This log provides for data integrity but impacts the performance of the slapd server: on slave servers we can think of giving this security away knowing we can recover a full copy of the database from the master server in the blink of an eye.

4.6.2. ACL

The most complex part of the LDAP server configuration is the creation of the ACL (Access Control Lists) needed to make it work properly. This is due partly to the particularly deranged syntax of the ACL themselves... For each requested *uid* and for each given requesting uid, OpenLDAP will check the first ACL rule (scanning the file from the top down) that it can apply to the requested uid. Then it will check for the first ACL rule it can apply to the requesting uid. For example the following ACL is needed to controll the access to the *userPassword* attribute:

```
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
```

Modification of the attribute is in any case allowed to ldap manager and to the servers administrators, while operators can only read the attribute to use it for authentication.

The order of ACLs in the configuration files is very important: they need to be written starting with the most specific to the least specific (concerning the *access to* approach).

Notes

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

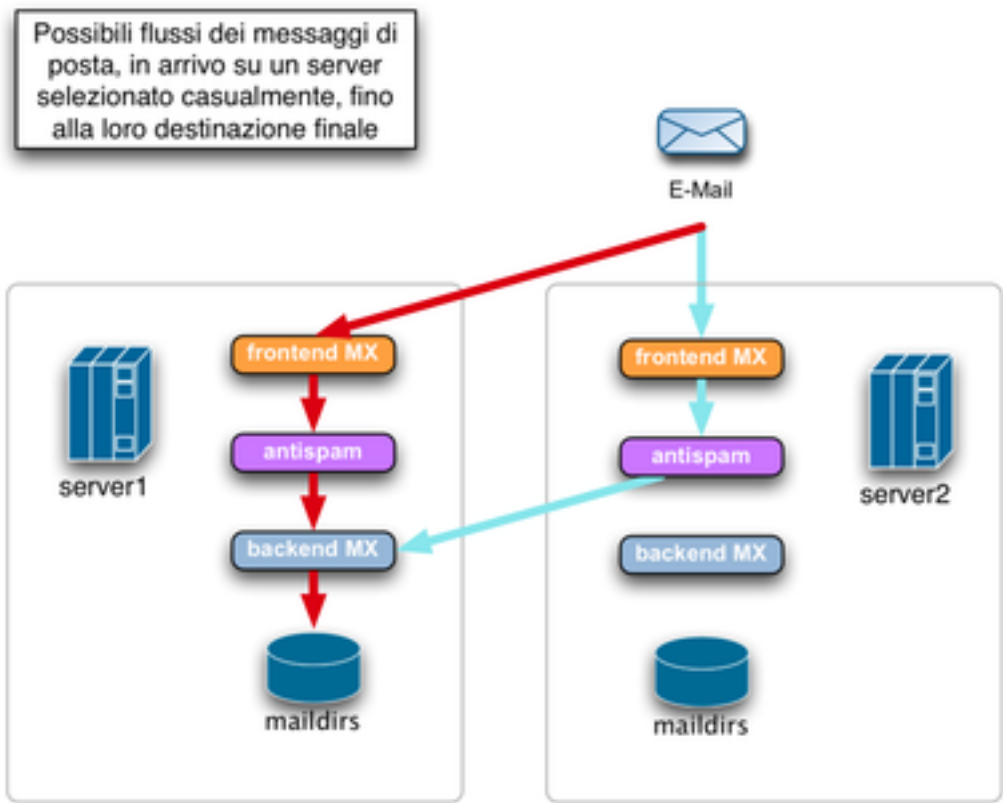
Chapter 5. Mail Services

In the introduction we stated that one of the main aims of this project was to split up mailboxes on N servers. This essentially means splitting the mailbox-related services as SMTP, IMAP and webmail. Luckily the SMTP protocol and the flexibility of Postfix configuration allow us to implement this feature without too many complications.

The choices we made for the SMTP services configuration were the following (along the guidelines described in chapter 1):

1. storing each user's mailbox in only one of the N servers (we can change this location but it has to be one and only one);
2. the N servers have to be set as MX of the domain (or domains) with the same priority as they will redirect the traffic to the correct server internally.

Since the users database is replicated on every server, each one of them knows which is the *final* target server of a particular email address. Through this information it is possible to build the appropriate transport tables.



SMTP Connections Flow

The preceding picture shows the flow of incoming mail inside our network. First of all the remote SMTP client chooses randomly one of the MX servers with the same probability. The incoming mail is therefore directed and received by one of the N servers. The incoming mail can then follow two different patterns, whether the receving server is the host

where the mailbox is located (the final target server) or not. In the first case the message is not forwarded to other servers and is delivered in the user local maildir. In the second case the mail is queued and forwarded to the final target server (through the VPN). In both cases the message is checked by antispam controls only once.

As regards the second case above detailed, one could thing this solution is not very wise in terms of banwidth usage. In fact the average probability of a message arriving to a server it is not destined to (and then forwarded to the correct final target server) can be represented as N-1/N, tending to 1 with N sufficiently high. [1]. However, it must be considered that this solution offers a high resistance to temporary network lapses and ensures an even splitting of mailboxes on the various servers. This could be a sufficient advantage, balancing out the problem explained above, also considering that the mail traffic is not particularly intense (our yearly statistics of some time ago gave us a figure of 1 message a second, with peaks up to 10 messages a second).

5.1. Postfix Configuration

Postfix is configured on each server to accept incoming mail for all of the virtual users. It then checks out an appropriate *transport* table and decides whether to deliver the message locally or to forward it to the final target server.

Since we can forward the message through our VPN, we can also spare another SSL negaotiation between the ring servers. Equally, the antivirus scanning can be carried out only once, when the message drops on the doorstep of the first server of our network. This is why we can configure two different instances of the `smtpd` listening on both interfaces (the external one and the VPN one, adding a couple of lines to `/etc/postfix/master.cf`:

```
# smtp on the external interface
192.168.1.1:smtp inet    n   -   -   -   -   smtpd -o options...
# smtp on the VPN interface
172.16.1.1:smtp inet     n   -   -   -   -   smtpd -o different options...
```

By properly changing the options, we can fine tune the smtp service. For example if the antispam filter were to be setup in the `main.cf`, the VPN instance of the smtpd could have the following options to disable TLS communications and antispam filtering, since both tasks are already carried out by the smtpd on the external interface and we don't need to check out messages twice.

```
-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
```

Another possibility would be to have two different postfix daemons (and two different queues) for internal and external traffic, doubling the `/etc/postfix` and `/var/spool/postfix` directories [2] The scheme is the same: a two-layers mechanism (three if we consider the antispam filter as an autonomous SMTP system) where one layer handles incoming and outgoing mail (and possibly antispam and other filters) and the other only handles local delivery. The particular *relay* queue handling by Postfix latest versions should ensure a good enough internal throughput also with a single instance, even if the external queue is stuffed with messages. [3]

5.1.1. LDAP Maps

LDAP maps should allow to recognize all virtual users (in local_recipient_maps), to forward them to the correct final target server (in trasport_maps) and finally to deliver the message into the correct mailbox (in virtual_mailbox_maps) and perhaps alias resolution (in virtual_alias_maps).

Let's see how maps are actually defined:

`ldaptransport`

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

this map works because in `master.cf` one can define SMTP transports with the name of the different servers and hard-coded destinations. For example the `server1` transport that is defined in all servers apart from server1 itself will be the SMTP transport with a predefined destination (server1-vpn). In this way we can fully customize it with fine tuning of the concurrency, disabled TLS and other optimizations. Of course on "server1" the connection coming from the VPN will have the antispam filter disabled so that messages are only checked once.

```
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]
```

Here we use also the *result_filter* option that allows us to modify the result of the LDAP query after having received it from the LDAP server.

`ldapmailbox`

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

the mailbox address map forwarding them to the destination mailbox. The paths returned by the query are considered relatively to the `/home/mail` directory and generally include a directory with the domain name and one with the mailbox name.

```
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*

the local alias map specifying which destination mailbox the alias address has to be forwarded to.

```
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

Antispam services are fairly easy and quite effective for the time being. We installed **Postgrey**[4] on all servers managing the mail service. It is a *greylisting* system, that works as follows: each message received for the first time (characterized by the smtp server / sender / destinatary triple information) is bounced with a temporary delivery error for 60 seconds; after *N* messages delivered successfully for a particular triple this is passed onto a "whitelist" so that for a certain period of time there will be no further delays. For normal SMTP servers this temporary error is not a problem and the message is queued for a second attempt at delivery shortly after. The mechanism works because most viruses and spam tools do not use a true SMTP server and do not have the possibility to hold messages in a queue, so they generally try to deliver messages only once. Of course this means that the delivery of the message is not "instantly" carried out, but it needs a small period of time to be processed (we are talking of minutes anyway). In any case we think this system is a good compromise in terms of performance, considering the efficiency that greylisting has showed.

Postgrey is installed as a *policy_service* in the Postfix instance accepting mail from the outside (on the public IP):

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

We have been using Postgrey for some time now and we have not noticed any problem, while it has proven excellent as an antispam protection (for the moment). In any case it could turn out useful to check out the `/etc/postfix/whitelist_clients` file and add in it the mail servers that most frequently contact you, so as to make delivery quicker and to avoid that the temporary database used by Postgrey grows too much.

Summing up to Postgrey, our main antispam mechanism is **amavis-ng**[5], that we use disabling its internal antivirus scanning system (it is far too heavy on our CPUs shoulders). Therefore, this software is for us only a SMTP *wrap-*

per of the SpamAssassin[6] analysis module. We put up with the absence of an antivirus with a series of regexp that checks out the body of the incoming messages, a system directly integrated into Postfix `mime_header_checks` and `body_checks`: this set of regexp is regularly update on the site www.securitysage.com[7] and includes the most common virus fingerprints as well as other very useful rules to bounce back a good part of the spam even before it gets examined by Amavis.

Amavis is installed as `content_filter` in the Postfix instance managing mail coming from the outside (the public IP) and it is configured to forward the approved messages to the backend server (the one listening on *localhost* interface).

5.2. POP / IMAP

The physical location of the single mailboxes on the various servers is determined by the `host` attribute of the corresponding LDAP object. This location is *hidden* for users since we have to be able to move the mailboxes transparently without the users being aware of it, for example when the hardware of one server fails or simply to balance traffic among the various servers. This is why we need to provide users with connection parameters that never change, independently of the physical box location.

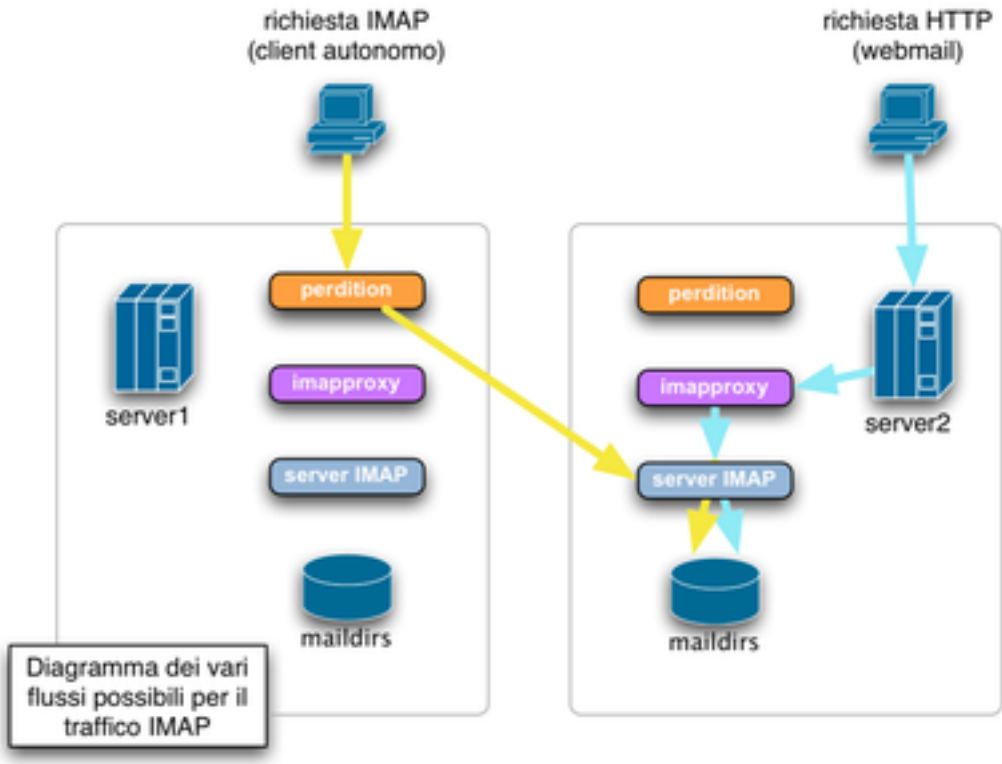
The problem is that mail reading protocols do not allow redirection easily (IMAP redirection is included in an RFC that no client actually implements). The initial solution we had sorted out defined for each user a CNAME that pointed at the appropriate mail server and could be used by users in their client configuration (something like `UTENTE.users.dominio.org`). Unfortunately this solution was not very functional, since it *does not* work properly when SSL comes into play: in fact, the validation of the name and certificate CN would never happen in such a system and the user would have to click on a security warning each other second.

The choice had then to be made between losing flexibility giving the user the specific server they had to configurate in their client (but in this way moving mailboxes around servers when it comes to be needed becomes practically impossible) and using a IMAP/POP proxy like Perdition, handling independently to move the data between servers (you can watch out for the same numerical analysis done for SMTP traffic). For the moment being we are trying out this second option monitoring the traffic and bandwidth consumption: if it turns out to be not sustainable then we will have to switch to the previous option.

The configuratoin parameter for the users' client will then be all the same:

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

With these configurations the client connect to the front-end server (Perdition[8]) on any of the boxes and their connection is then forwarded internally (through the VPN) to the back-end server of the appropriate box. For the back-end server we chose dovecot[9] because among all the different possibilities it seemed to us the lightest and most performing software.



IMAP connections flow

The picture shows the IMAP connection flow considering the two different access hypothesis:

- The users connection through an autonomous IMAP client on their computer (as any mail client for example) will connect to Perdition that redirects them directly (through the VPN) to the dovecot server on the server where their mailbox stands.
- The users checking their mail through the webmail instead will be forwarded directly to the "correct" server (since the connection can be forwarded playing with HTTP redirect). The webmail application connects to an specific imapproxy (up-imapproxy[10]) on the server itself. This proxy is needed to keep track of persistent connections and it's very useful since the web php application has to do a new IMAP connection for each new page. With the local imapproxy we avoid this unpleasant waste of connections. Finally the local proxy connect to the true IMAP server and serves the mail to the user.

This table shows the configuration parameter we used to implement the mechanism described above:

perdition (ports 143 and 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
```

To have the host attribute correspond to the IP we need (the internal VPN address in this case, resolved by the *host*-vpn hostname), we installe perdition in a appropriate *chroot* with its own `/etc/hosts` file. We had to do this since perdition LDAP backup does not support the rewriting of the result as Postfix does. This daemon is the only one receving SSL connections from the outside.

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 this way the proxy knows it has to forward the connections only towards the IMAP local server.

dovecot (port 10143)

in `dovecot.conf`:

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

The configuration of the users' authentication (in `dovecot-ldap.conf`) has been already described in the previous chapter.

When a mailbox will need to be moved from one server to the other (apart a very bad scenario in which the first box is completely crashed and data are not accessible anymore) it will be possible to use rsync after having edited LDAP database objects to move the old mailbox physically on the new server (that will be already receiving incoming messagges by then).

5.3. Webmail

Webmail much more easily then other services leaves the control of the authentication to the IMAP server (that is always *localhost*, and that's why users need to be redirected to the apache of the box where their mailbox resides to check webmail). Consequently we don't even need to configure the LDAP support on the webmail software. The software we chose is IMP[11] (the php mail client of the Horde framework) since in this moment it is the only one supporting some sort of integration with PGP.

5.4. Mailman Configuration

The mailing list software can be configured copying the configuration of all the lists on all of the server and setting up the appropriate aliases on each single server.

A major complicatoin is dued to the structure of Mailman that includes two agents modifying the list configuration: the client used by the MTA where the list alias is available and `qrunner` dealing with periodic tasks. Tie this two

agents to the specific lists on a single box is not very complex but implies a very complex mechanism to replicate the configuration on the various boxes.

Even keeping the mailing list software on a single box implies the need to generate transport maps to forward the list traffic to the correct server: the following command should be enough to generate a appropriate map to be moved to the other servers' *transport_maps*

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

Notes

1. We advise to read as a further in depth study the article *High Capacity Email* that can be found at `http://www.vergenet.net/linux/mail_farm/`. The article also details the best ways to redistribute the load on several servers weighing the mailboxes with high and low traffic.
2. You can read further on the topic at `http://advosys.ca/papers/postfix-instance.html`, and in the official Postfix 2.1 documentation at `http://www.postfix.org/FILTER_README.html`
3. Check also the debate at `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/

Chapter 6. Configuration

The centralized managing of the configuration of different servers is not a trivial problem to be solved and if implemented in the wrong way, it could make administrators think they have lost control of the servers and of what they do: if automatisms are ill defined and not too transparent, the risk is they hinder the administrators' attempts to interpret any possible problems. On the other hand, the automatic managing of the servers configuration is also an excellent possibility to manage a complex structure using energies in an intelligent way (since they are not endless).

This chapter details the solution we have implemented and the way this solution interacts with the overall system. Understanding this mechanism is crucial if you want to know how to modify the servers configuration so as to have services do what you want them to.

6.1. Overview

The servers configuration is centralized and it is connected to different mechanisms allowing specific customizations. In particular we have decided to use CFengine[1] to manage the simpler configuration files, that is files not depending on the users' data stored in the LDAP database, and customized scripts (the LDAP bindings for the various scripting languages are rather simple to be used) for the rest.

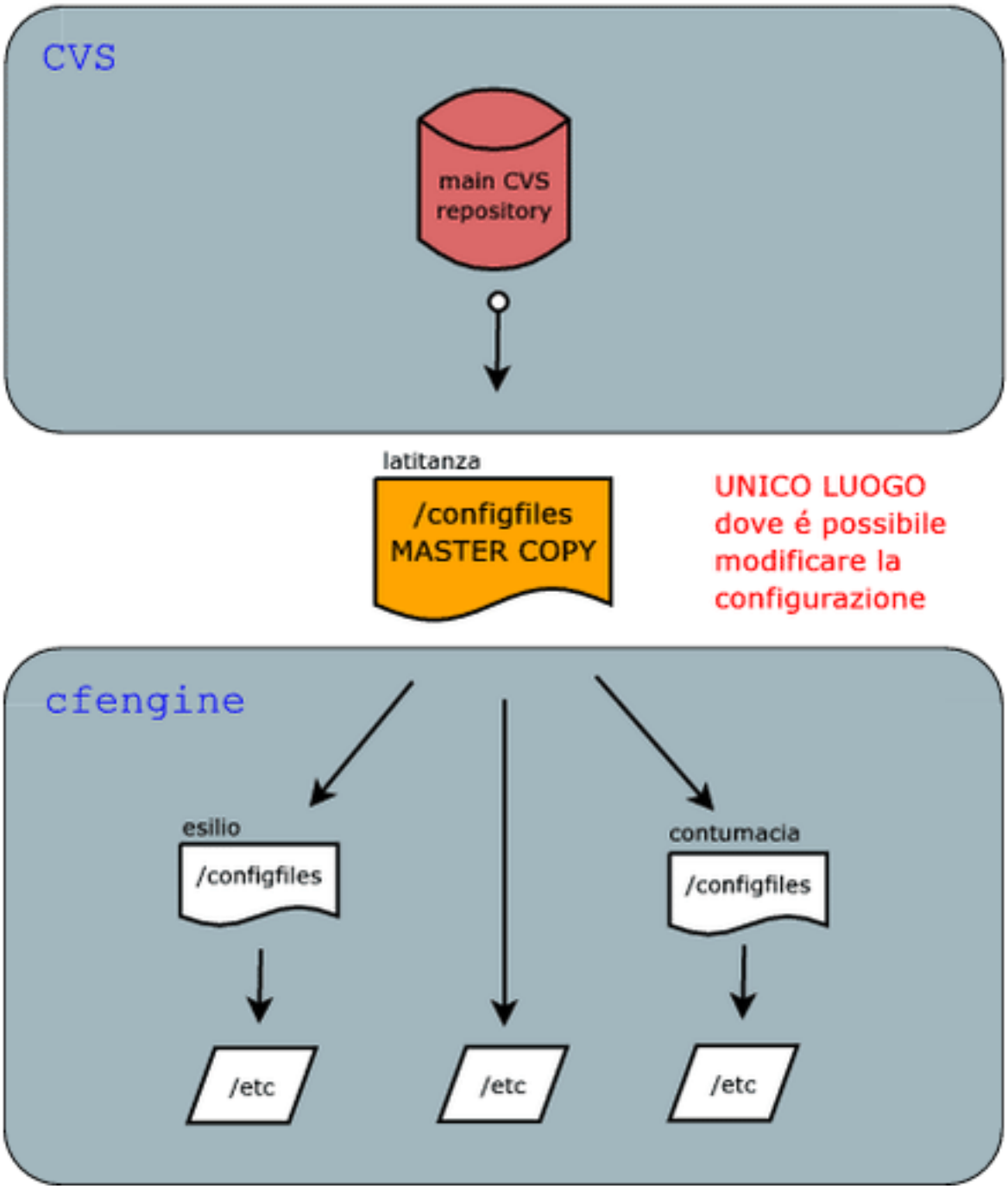
6.1.1. Database-independent configuration

CFengine has turned out to be a very useful tool, allowing us to create a management model based on the principle of the least differentiation possible among servers: configuration files are identical on each server, but it is possible to edit them with a series of simple substitutions . It is even possible to keep entirely different files for one or all boxes, as we will see below in this same chapter. This management model makes it quite easy to use very big configuration files with a lot of identical options and some options that need to be specified for each single server (for example the IP address bound to a service or to a hostname).

The centralized management of configuration files has to allow to keep track of the various modifications and of who made them, and it has to make them reversible.

That is why we have chosen to combine a version control system (as CVS, we are using Subversion[2]) with which we manage a common repository that is then replicated on all servers: this means that one needs to modify files in a single place and that they will then be spread all over the servers to the right final destination, as determined by the CFengine configuration. At the same time, a copy of the original configuration is kept on each box so that it is possible to change very quickly the main server if needed. As a further backup, the Subversion server is also located on another box.

The picture below details through a diagram which is the exact place to modify files and what are the different mechanism the modifications are distributed through.



Configuration Data Flows

It is definitely better that files managed this way are the ones that in some way are different from a standard debian install, since it is unnecessary to manage the whole `/etc` tree.

A detailed explanation of the `/configfiles` directory is included in Section 6.2.1>.

6.1.2. Database dependent configurations

Some services have a configuration that depends on the data included in the users' LDAP database because it is impossible (or unpractical at least) for them to read the database content dynamically. An example is the web server: in this case we have chosen to simplify the database data structure (avoiding the use of mod_config_ldap that reads the apache directive directly from complex LDAP objects) in order to make the management of the data easier. However, this solution implies the creation of the Apache configuration file in some other way.

These files are then generated automatically with some *script* that can be found on each server in the `/configfiles/scripts` directory (we can also put them somewhere else, if we find a more intelligent place in the filesystem). These scripts read the LDAP database data and create the configuration files accordingly. Other actions are possible too: for example one could check the existence of directories in the filesystem and their permissions, or the same thing for mailbox Maildirs, or other similar administrative tasks.

The basic idea is that the LDAP database is the authoritative source for the system configuration and that the scripts simply translate this source of information in actual data in the filesystem and configuration files. This implies that, in order to modify the configuration files, it will necessarily modify the LDAP database (we will see later on in this document how it is possible to do such a feat!).

6.2. CFengine

CFengine is then used both to distribute files from the central server to the other servers and to carry out some monitoring tasks and some system checkups such as filesystem permissions scheme, old files and directories cleaning up and creation of new files and directories (*policies* implementation).

CFengine offers a homogeneous array of functions other softwares already provide (rsync, netsaint/nagios, ecc.), thus allowing for many different ways to configure the system. Concerning the configuration described in this howto, we have chosen to implement two of them:

distribution

: the copying of whole files in the /etc directory, both common to all of the servers and specific to only one of them. This is the main way the static content gets spread out and works through a specific procol of the `cfservd` daemon, including its own authentication mechanism (that let us avoid the installation of ssh keys or rsh enabling).

editing

: the modification of configuration files. CFengine includes some very powerful directives allowing to modify the content of text files: it is possible for example to comment specific lines in a file depending on a series of regexp or to include specific paragraphs in some files.

These two ways can be combined as preferred, making it possible to schedule specific event for each server on a set of files centrally distributed. The very same CFengine configuration files are managed in the same way. We will see below how to do it, with some practical examples.

CFengine is configured through some files including a series of rules that are applied depending on the dynamic "classes" active at a specific moment. The rules are divided into sections that are applied one after another and with different syntaxes: for example, some sections only check whether a file exists or not (and its checksum), some other

only clean directories from old files, or modify text files, etc. Classes are defined for every other attribute specific to the servers: the name, the operative system, the day and time.

To study more in-depth CFegning configuration, we recommend you to read these documents:

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

6.2.1. CFengine configuration structure

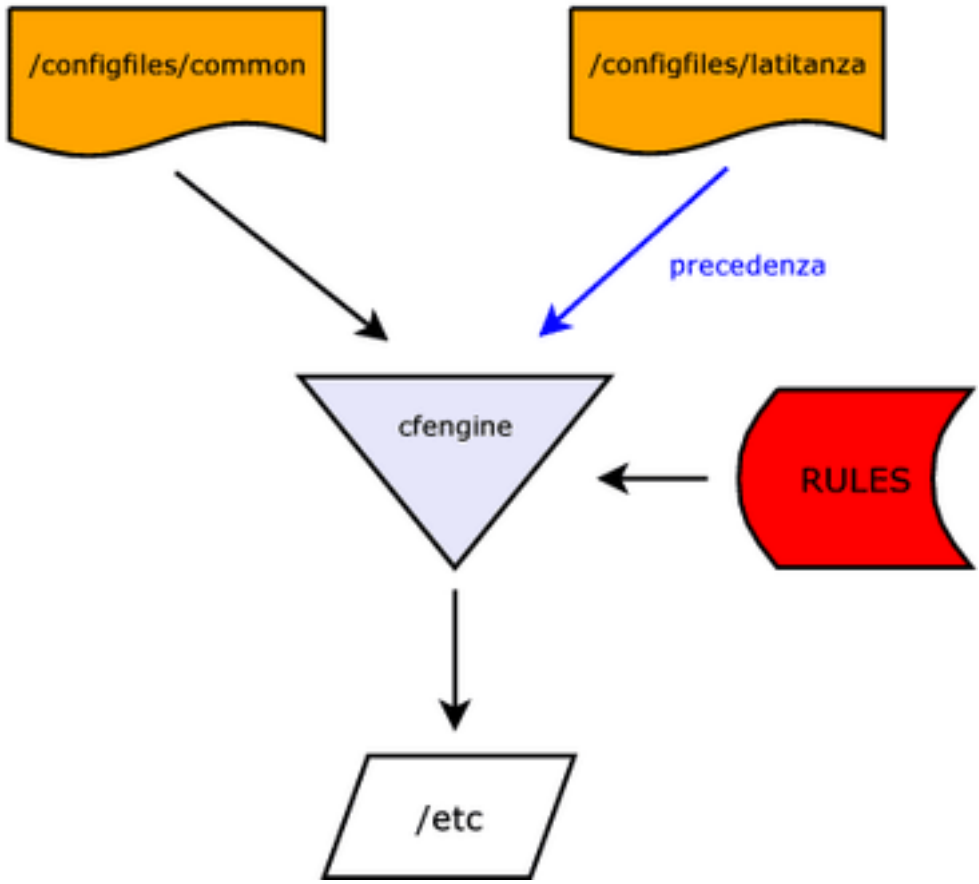
The configuration of the different servers in the network is managed centrally, by modifying a series of files in a single location (the central *repository* of the configuration). Each server keeps locally a copy of the most updated version of this repository, so that it knows "what to do" even in case it has problems connecting to the main server. This also allows for a quick change of the server working as main repository.

In this example, the configuration files copies are installed under the `/configfiles` directory. The servers are called `test1`, `test2` and `test3`.

If you take a look at the `/configfiles` directory you can see it is structured like this:

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

These directories include files that are copied (recursively if necessary) in the `/etc` directory of the system: the `common` directory contains the files that are common to all servers, while each `testN` directory contains files specific to the namesake server. If a file is present both in the generic directory and in the specific directory, the latter is the one that gets copied. The division into a `ring0` and a `ring1` is due to the fact that between one layer and the the other there are more differences than similarities.



How configuration data are generated for the various servers

The `cfengine` directory contains CFengine configuration and is handled separately. Files in `cfengine/inputs` check completely the way CFengine works and have to be modified by hand (on the main repository!) so as to change server specific parametres. Through these files you can also set the *policies* on the various systems, in order to make some routine tasks automatic.

6.2.2. An example on how to configure a service

While it is fairly easy to understand how to configure a service that only has static configuration files (i.e. copying its config files in the `/configfiles/ringN/common/something` or `/configfiles/ringN/HOST/something` directories directly), it could be useful to show how to configure CFengine for a service needing a specific configuration for each host.

6.2.2.1. Tinc CFenging configuration

We have already discussed about the Tinc[5] software used to implement the VPNs corresponding to the differnet trust rings. The basic configuration has already been discussed in Capitolo 2.2: VPN>.

After this basic configuration, we need to do a bunch of modifications to these basic configuration files, depending on the host, a task best suited to CFengine.

The CFengine configuration we have implemented handles the copying of the `/configfiles/ringN/common/tinc` file on every single box (check the comment in the `cf.sysconfig` file below). In order to manage the further modifications needed, we have created a `cf.tinc.ringN` file in the `/configfiles/cfengine/inputs` directory for each VPN ring. One of these is presented below commented as needed:

```
control:

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

This is the "control" section handling the global variables specific to the whole configuration of the service.

```
links:

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

The "links" section allows us to specify the symbolic link we want to create. Here we tell CFengine to create a different link to the private key (`/etc/tinc/ring0/rsa_key.priv`) for each single box.

```
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
```

The "files" section includes various checks that are to be done on files. For example here we check that the two specified files have the appropriate permissions and that they are owned by the specified user (action=fixall implies that they are to be fixed if needed). We also tell CFengine to ensure that the private keys can be read only by `root`.

```
editfiles:

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

The "editfiles" section tells CFengine to edit the text files generally containing the system configuration. In this case the `/etc/tinc/nets.boot` file in Debian systems includes a list of the VPN to be activated at boottime: the command AppendIfNoSuchLine adds a line including our VPN if it is not already in the file. If the file does not exist it will be created (AutoCreate).

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

The commands to handle text files usually have very verbose names. The previous series of command ensures that the configuration file includes only one line with the server name and that in each server the VPN service won't try to connect to itself.

```
{ $(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
}
```

Here we create the file activating the IP interface of the VPN, once the connection has been established (the last Append is a single line broken up for the need of this document layout). The editing of the file is "protected", that is to say the file won't be edited twice through the checking of comments.

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

This last part of the file adds a route for the ring1 VPN to every server apart from the one working as a gateway between the two rings (`ring0_gw`), which does not need this route.

6.2.3. Current CFengine configuration

The files in the `/configfiles/cfengine/inputs` directory completely control the behaviour of the CFengine software. They are fully modular, so that each file includes all the directive concerning a particular subsystem (hoping to gather all of the directives specifically connected in a single file).

We have already suggested how these files are divided into "sections" (specifying different types of actions to be carried out) and how they are managed by a system of dynamic "classes" (different on each server). It will now be useful to check how the various modules work and what they are needed for:

`cfagent.conf`

This is the main `cfagent` configuration file. It includes the definition of some crucial variables (as the domain and the admin email address) and all the following configuration files (in the *import* section). A crucial parametre included in this file is the sequence by which the various sections are executed:

```
actionsequence =
```

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

`cf.cfengine`

This file mantains a local copy of the /configfiles directory. It also installs correctly all the public keys needed by CFengine.

`cf.linux`

This file includes the specific configuration for Linux system and it is executed only if the host is included in the "linux" class. It does not contain anything particularly cunning, but it is rich in examples.

`cf.sysconfig`

This file implements the copying of configuration files in /etc, starting from the common directory (`/configfiles/common`) and reaching each specific directory (`/configfiles/NAME`). Files are copied from the main repository keeping their permissions. To understand how easy it is to program such a task in cfengine, it can make sense to check the part of the file handling it.

```
control:

    any::

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

copy:

    any::

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

The lines specify that CFengine has to copy all the files (thanks to the recurse=inf directive) of the two directories specified by the configpath variable in the `/etc` directory, keeping permissions and timestamps, as well as backup modified or deleted files in the `/var/lib/cfengine2/repository` directory. The inform=true line allows for notification to the administrators for each modification CFengine does to a file.

CFengine also keeps a checksum database (Tripwire style) of the binary files on the system.

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

These files detail Tinc configuration (see paragraph 2.2.3).

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

These files detail the configuration of other services. It is useful to know that if you want to understand when to restart a service, you should keep a temporary directory where to store the `/configfiles` content, change it and compare it to the `/etc` files: if the files have been modified one can then define a class to restart the service.

6.2.4. Automatic updates

In order to configure a system for remote configurations update (and for transferring files from the main repository to the local ones), we need to enable the `cfservd` and `cfexecd` software on every server. The host authentication is handled by an autonomous RSA key system that we need to check when it is first setup (see Appendice A>).

Another thing to check is the access list configuration in the `/configfile/cfengine/inputs/cfservd.conf` file.

CFengine works using a *pull* model. It is the clients that request the main server to update their configurations. `cfexecd` allows the automatic execution of `cfagent` at regular times (it basically replaces crontab). By default the time span is one hour.

In case we want to force an instant configuration update on all servers (*push*), for example because we have just changed a configuration file and we want it to be spread out, we can use the `cfrun` command that tells each server to run `cfagent` immediately.

6.2.5. Security

As a sidenote to the aspects detailed in the CFengine Tutorial[6], we can see how CFengine is subject to the huge problems of implementing an appropriate trust system and depends heavily on the control of the key distribution for security. This is in any case the best possible solution we can develop, without losing flexibility.

In any case, CFengine behaviour is non-destructive and heavily monitored (any --verbose option will produce an almost obnoxious amount of monitoring information), and it generally generates an increase in the overall security of the system, since it forces a common and verified *policy* for all servers participating in the network.

6.3. Script

The scripts configuring the system depending on the content of the LDAP database are made to implement the least modification possible, and to avoid editing if the system already has the desired status: this allows them to be run

fairly often and to understand when we have to restart a service.

Some scripts are heavier on the system (for example the one responsible of checking permissions and the existence of thousands of files or directories). These scripts can be run in two different modes, a *quick* one, only creating new files or directories as needed, and a *complete* one, checking permissions as well. This helps to differentiate the time of the day when to execute which, so as to have a quick compliance of the system with the database content in terms of file existence.

Notes

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

Chapter 7. Certification Authority

For the management of the various SSL connection of users on servers, we have considered it useful to create our own Certification Authority: with it we can sign all the SSL certificates used by the servers and make them easily verifiable by users and servers alike.

Since we needed to implement some non-trivial features (as the possibility of having certificates valid for several different names on a single server), we have added this chapter to the documentation, detailing a short *how-to* others could find useful.

7.1. Initial Configuration

We will use the `/opt/ca` directory as a place where to keep and manage certificates:

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

Then we have to create the directory structure and some files needed by a 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
```

The `serial` file will include the next serial to be used, while the `index` file is a database of all the references to the certificate signed by the CA.

Now we have to create a basic configuration for our CA, for example the one used as a default by OpenSSL in our Debian *sarge* distribution in `/etc/ssl/openssl.cnf`.

Our CA will use the `/opt/ca/conf/ca.conf` configuration file. To have openssl use it correctly without constantly having to specifying it in the command line, we can simply export it in the OPENSSL_CONF variable.

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

The content of the `ca.conf` is as follow:

```
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. CA creation

The first step to create a CA consists in creating a root certificate, the one with which we will sign all the other certificates:

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

After this, we can check that our certificate exists, along with its own private key:

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

We can then update the serial file:

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

7.3. Certificates

Now we can create our certificates request (for example the web certificate):

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

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

During the creation process, we will be asked the *OU* and the *CN* that are to be filled with the type of service and the dn the service refers to.

The *subjectAltName* (nice X.509v3 extensions) will be specified in the `/opt/ca/ext` file and will be added to the certificate when the request will be signed with the CA root certificate.

The `-nodes` option allows us to avoid the encryption of the key, so that we do not need to fill in a passphrase during the management and use of the key itself. In case we give a request without this option, the passphrase deleting from the key can be achieved by typing the following command:

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

Let's see how we ask for a certificate:

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

We can now sign it:

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

Now we can delete the `web_req.pem` file and move the `web_cert.pem` certificate and its key to all servers.

Let's see how the certificate has been changed:

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

We can see that some extensions have been added (as defined in the `/opt/ca/ext/webserver.ext` file). This is the content of the 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://domain.org/crl/cacrl.crl
nsRevocationUrl         = http://domain.org/crl/cacrl.crl
```

```
crlDistributionPoints   = @cdp_section


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

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

In order to create certificates for all other services, you just need to change appropriately request, key and certificate names.

7.4. Revoking and CRL

To revoke the `web_cert.pem` certificate you only need to type the command:

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

Then you have to update the list of revoked certificate (CRL version 1):

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

The second command is needed to convert the CRL from the PEM format (a 64base coded file) to a DER format, since some software does not accept PEM revocations.

Chapter 8. The Web Services

This chapter offers information on how web services are configured (and why). We also discuss FTP and MySQL considering how much they are related to each other with respect to the user.

It's important to stress that the configurations described in this chapter are primarily meant as examples: in fact, there are many possibilities for configuring the web service on a server aimed at fully exploiting the potentiality of the distribution mechanism we have just described in the previous chapters.

8.1. Apache

8.1.1. Web service structure

When we had to choose the configuration of the different web servers on the various boxes, we needed to take into account also the reorganization of the infrastructural domains that we serve on the web.

This was particularly true in the case of the https protocol, which gives some problems if managed as we have been doing until now, i.e. with third level domains for each service: webmail.domain.org, squirrel.domain.org, passwd.domain.org, etc.. The SSL certificate unfortunately allows just a single hostname (in the CN attribute) to match the browser URL request and if there is more than one certificate, the browser shows an alarming message, even if the user has already installed the CA certificate. In a way, this vanquishes the effort to have users install the CA certificate or to have them use the PKI structure correctly.

This problem can be solved in at least two different ways: the first implies the use of *wildcard* certificates, i.e. a certificate for something like **.domain.org* that will be valid for all hosts included in domain.org namespace. While this workaround is a bit shady in the RFC, it seems to work on most browsers.

However, we have favoured a different solution (also giving us the chance to rearrange a bit our websites): we have put all the SSL subdomains under a single domain (www.domain.org). The servers have to be reached independently though, so we have also set up a www*N*.domain.org address where *N* is a number. Luckily a very popular extension of the X509 specifics (subjectAltNames) allows us to add some other domain name to the web server certificate. In this way we can create an ad hoc certificate that allows for all HTTPS connections to be correctly validated with the simple installation of the CA certificate.

8.1.2. Configuration

The Apache 2 configuration has been planned to be fit into the modular scheme used by Debian systems. It is then structured as follows: (the `apache2` directory below is the one you can find in `/configfiles/ring0/common`).

- `apache2/apache2.conf` - this is the main apache2 configuration file where you can find the generic directives to make the server work.
- `apache2/virtualhost-template` - this file is used as a template for the automatic generation of virtual host configurations: the LDAP database is parsed by a periodic script and a virtual host configuration is created for each hosted site, substituting some values in this file (the hostname and the logfile paths for example). In some cases the information included in `apache2/conf.d/extra` directory is included in the virtual host configuration.

- `apache2/include/` - this directory contains some pieces of configuration which are often repeated so that they can be included and changed with the touch of a single file instead of multiple editing. E.g.: standard php directives for user sites, robot blocking, etc.:
  - `include/dominio-auth.conf` - this file limits the access to the users of the domain.org domain. By including this file in the section desired you can make part of the site accessible only to administrators (this part of the website should better be in https)

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

  - `include/cache.conf` - this file enables the caching of contents (for locations where one has to do reverse proxying).
  - `include/coral.conf` - this piece of configuration enables the coral distributed cache (it might be useful when a specific site or server is overwhelmed by requests).
  - `include/php-common.conf` - standard PHP configurations
  - `include/stop-robots.conf` - this configuration blocks some particularly nasty robots.
- `apache2/mods-enabled/` - this directory contains parts of configuration enabling the various modules to be loaded (`.load`) by apache and their configurations (`.conf`). We do not handle this with `a2enmod` but rather with CFengine.
- `apache2/sites-available/` - this directory includes the configuration of the structural virtualhosts. This has to be enabled using the a2ensite command (run by hand on each server, since it's possible that not all servers share the same virtualhosts). The command simply builds a link to the file in the `sites-enabled` directory
- `apache2/conf.d/` - this directory contains configurations that are included by other files in the appropriate places.
  - `conf.d/ssl/` - here you can find various modules to configure https services of the server (i.e. on www*N*.domain.org)
  - `conf.d/virtualhosts/` - users virtualhosts (automatically generated)
  - `conf.d/subsites-domain/` - domain.org subsites (automatically generated)
  - `conf.d/subsites-public/` - public.org subsites (automatically generated)
  - `conf.d/extra/` - in this directory you can find files to list additional directives to be included in some particular site (the file has to be named as the virtualhost or as the alias of the LDAP database followed by the `.conf` suffix; it will be automatically entered in the appropriate configuration file).

The main aim of this configuration is to have a more or less homogeneous distribution of the users requests on the different servers using a *round-robin* DNS record (i.e. the possibility to return to clients asking for a domain name a series of IP addresses randomly). This approach is perfect for static content or for pseudo-dynamic content that only needs to be read. In fact, regarding web applications, when you login and generate a session it is a good idea to redirect the user always to the same server (even if it's not necessarily important to which server): this way session data are accessed locally (and quickly).

To do this, after the round-robin DNS, we use HTTP redirections structured as follows:

- A file in `/etc/apache2/servers_map` defines where a specific application is available (on which server). It includes several lines, each defining a different combination:

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

- In the main virtualhost configuration (www.domain.org) you can already find the following directives:

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

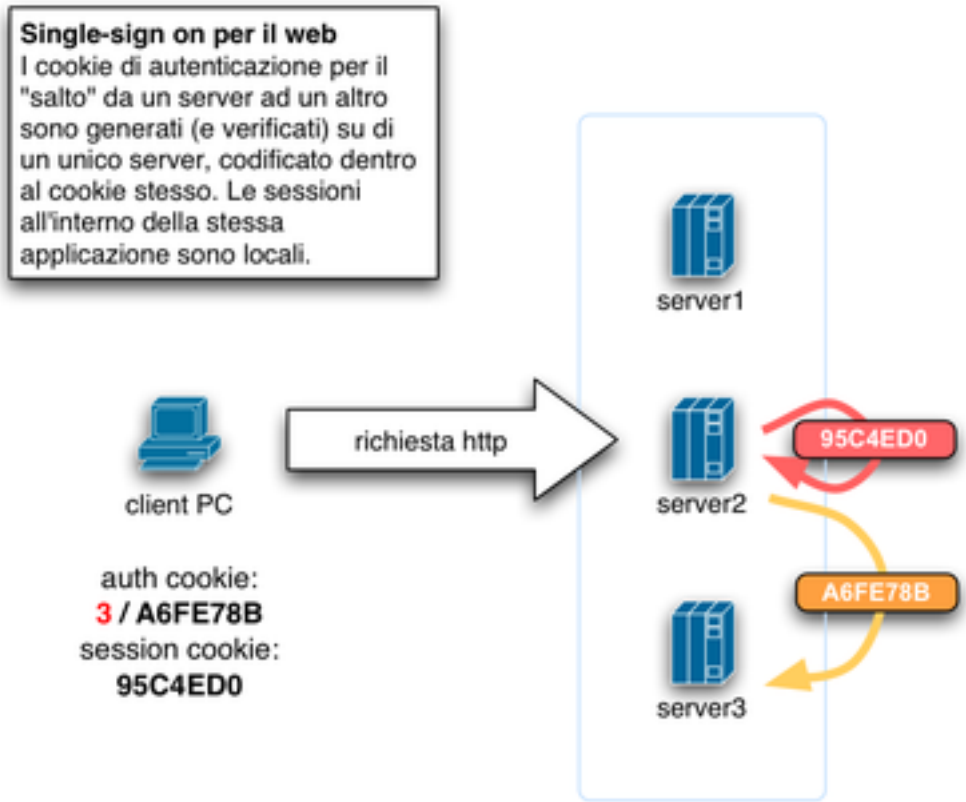
  this is used to redirect (from the *pool* list) all requests for a specific `/app` on a random server. This way the user is locked onto a specific server for the rest of the session.

8.1.3. Distributed Applications

Since we had this possibility, we decided to install certain applications on every server so that if one of them is off-line it will be possible to connect to the others (the user control panel, or our blog).

In order to be used in this way, normal "software" needs to be modified, particularly concerning the separation of the read and write queries to the MySQL database (supposing the database is replicated in single-master mode) and the possibility to tranfer sessions from one box to the other. The latter is needed because services might be located on a specific server that can be different from the one where the user logs in.

To avoid this problem, the most common solution (a centralized SQL database or a replicated one) was not satisying for us since it would imply a high internal bandwidth consumption (it’s not thinkable to do a remote SQL connection for each access to a webpage). We therefore developed a specific mechanism using authentication cookies that are read as a transparent authentication system while the user passes from the front server to the final server. By intercepting the login functions of the various applications, a new session on the new server is created when the authentication cookie matches.



Single sign-on mechanism

8.2. MySQL

MySQL is the main database used by user websites and it does not need particular configurations. We only had to find a way to read in the LDAP database which MySQL databases have to be on a specific server and to create them when necessary.

However, there are some databases that it is useful to share on the different servers and that are normally unbalanced towards read queries (our blog for example). Considering this unbalance, the MySQL replication functionalities can be the answer to our problem.

8.2.1. Replication

In our present situation the only MySQL replication possible is the master-slave one: the master, where write operations can be performed, and slaves replicating the database (in near-realtime). The slave has also a backup function if the master fails.

The idea of having a single solution not only for backup but also for load distribution would imply setting up the servers in master-master replication, with each server allowing for reading and writing on the database balancing load out. This solution implies high and overall stable bandwidth, a quality our servers do not have.

For further documentation refer to “MySQL Reference Manual: Chapter 6, Replication in MySQL”[1].

To setup the replication system:

1. this is the configuratoin of the `my.cnf` on the master server:

   ```
   log-bin
   #binlog-do-db=(if we want to limit the configuration of the single db)
   server-id       = 1
   ```

   this enables the binary log (a file in `/var/lib/mysql` that can become quite big). In debian it is also necessary to modify the `/etc/mysql/debian-log-rotate.conf` to increase the KEEP_BINARY_LOGS= option. If the replication fails the slave can use these old logs to rebuild the database.

2. this is the configuration of the `my.cnf` on the slave server:

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

   the user `repl` is to be created on the master server with the following permissions:

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

   The SUPER and RELOAD privileges are used when the slave needs to import a table from the master with the `LOAD TABLE FROM MASTER` command. This method is highly deprecated for big databases.

3. this is how you can make a snapshot of the db:

   a. first and foremost stop writing on the master:

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

   b. then leave the mysql console open and copy the db:

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

   c. Take note of the master status when the snapshot is made (both *file name* and *position*)

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

d. Remove the lock

```
UNLOCK TABLES;
```

4. Copy the snapshot on the slave server:

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

5. Inform the slave server on where to begin replication from:

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

6. Launch the slave:

```
SLAVE START;
```

7. Monitor the replication status:

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

Notes

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

Chapter 9. Log anonymization

A theme we are particularly caring for is the possibility of *not* keeping any record of the users access (*log*), so as to protect the choice of anonymity our users have done. This possibility is presently heavily threatened and this is one of the reasons why we wrote this document in the first place.

Thus, thanks to the work of many other activists all over the world, we can use some simple tricks to make sure that on our servers there is *no* log (not even temporarily) that will allow to connect accesses to our services to IP addresses (leading to the identification of users).

Apart from various tricks (from `wtmp` deletion, to the use of ssh relays for connections, to Tor[1]) that we use to protect the identity of our administrators, let's see how services have been configured:

9.1. Apache

This is by far the easiest thing to do. To make Apache anonymous, you need to create a new type of logs in the `apache2.conf` file that you can call "anonymous":

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

(tutto su una riga).

This way you can define an anonymous access log for each virtualhost:

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

9.2. syslog-ng

The remaining system logs, passing through the *syslog* service (including mail log, IMAP logs, ssh access logs, and so on) are filtered, with email address and IP deletion, directly by the syslog daemon.

To implement this, we use a syslog-ng[2] patch already included in the default Debian package, that allows the filtering of logs wih a regexp (included 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})"); };
```

Once this is done, we just need to insert the filter in the flow definitions:

```
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);
};
```

This kind of log rewriting does not alter the possibility to use statistics tools apart from the fact you cannot aggregate connections coming from the same IP address.

9.3. Postfix

Even if Postfix logs are anonymized by syslog configurations, some sensible information is still kept in the messages. To remove it, we need to edit off the *Received* header that includes the user IP from the messages sent through our SMTP servers.

To do that, we have to patch Postfix. A constantly updated version of the patched Postfix package for Debian can be downloaded by including the following line to the `/etc/apt/sources.list` file:

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

Notes

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

Appendix A. Installation of a new server

For the test implementation of this howto we have chosen a *sarge* Debian[1] distribution, which provides packages for all the needed software and is sufficiently stable to be used on a server.

A.1. Debian packages installation

The first step consists in installing a normal Debian distribution on each box and then adding up the needed packages to bootstrap the configuration:

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

A.2. RSA key initial distribution

After this initial step, we need the `cfagent` client to be authorized to connect to the main repository server (where the configurations copy resides). It is then necessary to generate RSA keys for the new server and to copy them to the main server. In the meantime they will be spread on all the servers already belonging to the network:

```
$ cfkey
```

`cfkey` generates the public and private part of an RSA key in

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

We then have to copy these files to the main repository server in the `/configfiles/ppkeys/` directory, renaming the public key `root-NAME.pub` and `root-IP.pub`, while the private key has to be renamed as `root-NAME.priv` (where NAME and IP are respectively the qualified hostname and the ip address of the server). In order to complete the initial distribution of the keys (that afterwards will be carried on automatically by CFengine) we have to copy the public key of the main server in the `/var/lib/cfengine2/ppkeys` directory.

A.3. First configuration

The server will now be able to receive the configuration by launching the following command:

```
$ cfagent --verbose
```

We suggest to examine the output of this command closely to look for any errors. If the initial configuration has been carried out correctly, you should find a local copy of the configuration files in `/configfiles`.

Notes

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

Appendix B. Scalability

This appendix is dedicated to a mathematical analysis, even if approximative, of the scalability of a mail management model such as the one we describe in this document. Such model is used on a geographically distributed network of services that do not support client redirection (in our case SMTP and POP/IMAP).

In this model, messages (connections) arrive with the same probability on each of the servers available, and are redirected to the final destination server through an internal channel (in our case, a VPN).

The hypothesis of geographical (non local) distribution is that this private channel uses up the same resources as the incoming traffic. It is reasonable to think that this channel has some possibility to compress the traffic that goes through it, and we will note as *c* the compression factor.

Let us then suppose we have an incoming traffic of *i* (the figure should be some bandwidth unit) and let us name *N* the number of servers: if the probability of the message to be redirected is

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

then for each server the incoming traffic will be

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

and it will generate an outgoing traffic of

$$o_s = c i_s \frac{N-1}{N}$$

on the private channel

Each server will receive a traffic of o_s from the other *N - 1* servers, so that the total incoming traffic (including the incoming traffic from the private channel) is:

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

For *N* tending to infinite, the value of i_{tot} tends to:

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

The meaning of this equation is that such a traffic model (i.e. with internal redirection) is linearly scalable with the number of installed server *N*: in fact reverting the equation we find out that once we fix the total availability of bandwidth of each server, the total bandwidth capacity of the system is directly proportional to *N*. In the case, for

example, of mail traffic, the compression coefficient c can assume very low values, thus minimizing the redirection overhead.

Another way to see the last result is that, considering the bandwidth of each server as generally set (technically and economically), this is split up (in the limit of N very big), on each box, to half traffic for internal redirections and half traffic for incoming external traffic.

Appendix C. Where to find other versions of this document

The original version of this document has been written in Docbook (http://www.docbook.org/), and we have then automatically generated the other versions (HTML, PDF, RTF).

C.1. HTML, PDF, RTF

You can find this document in the following versions:

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

Notes

1. http://www.docbook.org/
2. https://www.autistici.org/doc/orangebook/
3. https://www.autistici.org/doc/orangebook.pdf