

cluster -- Clustered Architectures for Mailing Systems and the PigeonAir Project

Carlo Contavalli

ccontavalli at masobit.net

Purpose of this document is to introduce users, developers or administrators to clustered architectures for mail servers, with particular regard to the PigeonAir and PigeonDeliver infrastructure.

1. Before starting

This document was written as part of the documentation of the PigeonAir Project to provide help and support to users, system administrators or developers.

While every effort has been made to ensure that the information is accurate at the time of publication, this document may contain errors, omissions, incongruences or wrong technical details. No liability for damages is accepted by the Author/Authors, the publishers or any other organization or person providing the information, arising from any errors or omissions that may appear, however caused.

In case you find an error, you would like to propose better solutions than those discussed in this document or you would like to discuss an idea regarding this document or its content, we would be glad to hear from you and please feel free to contact us by writing to the <pigeon-dev at ml.pigeonair.net> mailing list or by directly contacting one of the authors.

1.1. Intended Audience

This document is meant to introduce users, administrators and developers with the problems involved with building mail clusters. It describes some clustered architectures that can be used to deploy mail clusters, how they can be employed in PigeonAir Clusters and to the terminology and main problems encountered when working on PigeonAir Cluster.

This document is meant to be an informal introduction to the PigeonAir system, for those approaching its installation or infrastructure for the first time. It will not discuss technical details nor detailed studies about the described architectures and about the employed solutions. This kind of discussion is left to other documents.

1.2. Copyright Notice

This document was written by Carlo Contavalli <ccontavalli at masobit.net> and is thus Copyright (C) Carlo Contavalli 2003, 2004 and the PigeonAir Project.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts and no Back-Cover Texts.

Any example of program code available in this document should be considered protected by the terms of the GNU General Public License.

You should have received a copy of the GNU General Public License along with this document; if not, write to the Free Software Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

Trademarks are owned by their respective owners.

2. From Mail Servers to Mailing Systems

Setting up Mail Servers is quite a common task for those working on a day by day basis on the internet infrastructures.

We all know what we usually mean by mail server: some sort of computer or computing system, connected to the internet, with some softwares installed able to speak the SMTP protocol and often the IMAP/POP3 protocols.

Most of the times, however, when a system administrator is asked to setup a Mail Server he is also asked to provide a web interface for users to access their emails and some sort of “panel” or integration with existing applications to allow any customer to choose his own settings and administrators to add/remove users or add/remove domains.

Moreover, in the ISP world, a Mail Server is not just something able to send or receive emails: it is often a service being sold which must offer a very good image of itself, rich of features to make it more attractive to customers’ eyes.

A Mail Server thus becomes something more complex than we are used to think: it becomes something not just able to exchange emails with other servers, but rich of features such as antiviruses, mail filters, cleaners, spam filters, mailing lists, sms gateways, pager support, palm synchronization and so on... with attractive, powerful yet easy to use interfaces.

A simple Mail Server thus become a complex Mailing System, something much more complex to setup, even in non clustered environments: more components to integrate, more resources necessary, more code to deal with when looking for bugs or when trying to perform troubleshooting and more stuff to backup when backing up the system.

The PigeonAir (<http://www.pigeonair.net>) project was not started just to build up Clusters of Mail Servers. It is not that complex today using common technologies, even if the result may look way too complex or too fragile... it was meant for building up fully featured clustered Mailing Systems, complete of all the nice features a provider would like to have on its own systems.

3. Introducing mailing technologies

If we temporarily forget about advanced services like antiviruses or antispam filters, a mail server is usually made of:

a SMTP Server

Able to speak the SMTP protocol with other servers (to receive emails and to deliver emails remotely) and to deliver mails coming from PPP clients (aka, to behave as a relay host).

As you probably know, when some other server wants to send an email to one of your users, it first looks up the domain name of your user into the DNS looking for a MX record. MX records indicate the server in charge to receive emails for a specific domain, and indicate a priority for the given record. The sender fetches all the MX records, sorts them out from the lowest priority to the highest, and then contacts all the servers in the resulting order. If more than one MX record have the same priority, then the server will round robin over all the given records, for each connection, in order to provide a trivial way of load balancing.

Whenever our server receives a mail, it verifies if the recipient is really one of its own users or not. If it is, it probably will accept the mail and pass it over to a Mail Delivery Agent (MDA) that will take care of storing the email on one of the hard disks (or, more generically, on the storage) or of performing any other needed operation on the email being delivered.

If it is not, our server will verify to be a relay host for the given client. If it is, the mail will be taken, inserted in a queue, and forwarded to the correct host. If it is not, the mail will be rejected.

Given that a SMTP server works as described above, it must now be clear that in order to work, it needs:

- to know who the users of the systems are, which domains are handled by the mailing system and which preferences the user/domain expressed about the delivery of their own emails.
- to know where users data is kept on the storage itself and to have some kind of access to the storage itself.

IMAP/POP3 Server

An IMAP/POP3 Server allows users to fetch their emails stored on the disks (storage). Users connect to the given IMAP/POP3 server, prove their own identity using some kind of authentication mechanism, and finally access their emails.

Comparing the needs of an IMAP/POP3 server with those of a SMTP Server, it needs:

- to know users' authentication data (to be able to authenticate users), and to know where users data is stored.

- to have some kind of access to the physical storage, to fetch the emails and send them back to the user.

An administrative interface

Usually a web interface, used by administrator/power users to add, remove, rename, disable, reconfigure users or to add, remove, rename, disable, reconfigure domains or single mail server features.

Compared to the other system services, an administrative interface needs:

- to know where users information are kept and to be able to modify them.

While a Web Mail has almost the same needs of an IMAP/POP3 server, from the discussion above we can deduce that any kind of mailing system or mail cluster needs to deal with at least two kinds of data in order to be able to correctly perform its duty:

control data

data used by the mailing system to know, for example, if a user or a domain exists, where its data is stored, the options the MDA should use for the delivery...

mail data

the email themselves, that need to be transported and stored on the storage.

Now, it may seem a dumb remark, but this means that the SMTP server and IMAP/POP3 server need a common way to store and read control data. The administrative interfaces also need to be able to access and modify control data, while only the IMAP/POP3 server and the SMTP server needs to read/store emails on a shared storage in a common format.

4. Introducing mail clusters

The basic idea behind mailing cluster is not that difficult to understand: if the load is too high for a single machine to handle it, we try to split the load over multiple machines, or, if we need a very reliable service, we try to put up many machines able to take over in case one of them fails.

Actually, the first kind of cluster is also known as “high scalability cluster”, while the second kind is known as “high availability cluster” (HA -- sometimes the terms “vertical cluster” or “horizontal cluster” are also being used, but in some environments they take the meaning of how nodes are added, or of how power is added to the system, talking about “vertical scalability” and “horizontal scalability”). The most aimed result for a ISP is usually a very scalable and very reliable cluster, something in between the two, probably one of the hardest results to achieve.

Now, let’s say we want to build up some kind of cluster for mailing systems. The only thing we can be sure of is that the cluster will be made of multiple computers (or, at, least it will look like).

As being made by multiple machines, we will probably split up services among those machines, and we will find some way to make those services coexist and share information happily. Yes, because in order to work, they need to share lots of data...

4.1. Mail Cluster Delivery process

So, let's say we have a single machine handling the load of a whole mail server. Let's say this machine is not able to handle the load anymore, and we want to split it up over a cluster of machines.

4.1.1. Avoiding Clusters by adding servers

We have one, very simple solution at no cost: configuring a second machine as a mail server, and move part of the domains over this second machine. However, this solution has a few disadvantages:

- first of all, a single domain can be handled only by a single machine. If we have a very large domain, we cannot split it over more than one machine, so it doesn't scale very well...
- it requires us to choose on which machine we want to store each domain. Even in case there is some automatic procedure, we will still have to check that the load is well balanced.
- if at a certain time a given domain grows very large, we need to move it from one server to another, not always quite easy to do...
- we would need to give some customers some configurations to download their emails, while others would have different configurations, making customer care a pain.
- in case the load grows even more, we are still over.

We thus won't discuss this kind of solution any further. What we want to achieve is something (probably a group of machines) the users would envision as a single machine providing the needed services, and able to split domains up over many machines almost automatically.

Keep also in mind that the delivery process with the described configuration is not changed at all. It is just like having many mail servers to administer and configure, while taking an eye over load and balancing.

4.1.2. Real Clusters and the delivery process

Ok, so what is a real mail cluster? Well, while there are many discording opinions, it is usually meant to be a group of machines handling the load of a given service "transparently", that may look like as a single entity both to end users and administrators, where it is easy to add/remove nodes (machines) and where each node, even if we have few users/domains, concurs in handling the load of the whole cluster.

So, how do I build up a mail cluster? Well, there are a few things you will need for sure:

a balancer

ok, if users, in a way or another, see my own cluster as a single machine, well, all connections will go to that machine. So, there will be some kind of device/appliance/extraterrestrial technology able to distribute (balance) the connections over the cluster of machines. There are *many* technologies to achieve this result,

from the simple DNS round robin balancing (one record that points to many servers) to more complex setups using Linux IPVS (<http://www.linuxvirtualserver.org/>) or expensive appliances.

something to allow machines to share control data

Well, if we have a couple machines handling the load, they all need to know who the users/domains being managed are, so, for sure, control data will be shared.

something to allow machines to access the same storage

We have seen before that both the SMTP and IMAP/POP3 server need to access the storage. The first one, in order to store new emails, while the second one to give emails back to users. So, in a mail cluster, the SMTP and IMAP/POP3 server will still need to access a shared storage. To make things harder, SMTP and IMAP/POP3 are now split over many machines, and still need to access a shared storage. Well, pay attention, by accessing a shared storage I don't necessary mean accessing a shared file system. There are many alternative ways to make a storage accessible to many machines.

Well, once we find some way to share control data and some way to let all the services that need it to access the storage, we should (almost) be done: both the web mail and the administrative interface should be fine too, since they don't need anymore than what has already been discussed. The only thing we may have to think about is finding a good way to balance web sessions. Yes, because web interfaces use sessions, and sessions are hard to share among different machines (more about this will be discussed later).

So, what happens when new mails come in? Well, the remote server tries to establish a connection to our own server. While opening the connection, some way or another, the balancer gets in the middle and sends the connection to one of the clustered servers.

And when a customer tries to connect to the IMAP/POP3 server or to the web interface? Same thing as above.

5. Architectures for Mail Clusters

Ok, in the previous section we looked at clusters from a theoretical point of view. In this section we will try to go into further technical details, trying to answer many of the questions the previous section should have raised, and describing into details some of the solutions which can be employed to "share control data" among machines or on how to allow many machines to "access storage".

5.1. About Control Data

It is important to note that most SMTP/IMAP/POP3 servers keep control data either in a database or on the same storage as mail data.

In the first case, building a good clustered environment will probably mean that all the nodes will access the same shared database, which will be replicated (for reliability) or be scaled (by adding more database servers) using some database specific mechanism.

In the second case, the same mechanism used to share mail data can be used, without worrying to much, to share control data.

It is also very important to note that the database actually being used depends:

- on the databases supported by all the elements of the mail cluster. If we use a given software as SMTP server, another software as IMAP/POP3 server and one more software to provide an Administrative Interface, they all need to support the same kind of database.
- assuming that a given database can be used for all the components of the mail cluster, we will need to ensure that stored records are kept (or can be fetched) in the same format. To be clear, if the Administrative Interface needs a field named “PostMaster” in the db containing the username of a domain administrator, while the SMTP expects a field named “PostMaster” containing the ip address of hosts authorized by the postmaster to connect, things won’t work quite happily.
- the database is a central point of the delivery process. Each received email, each attempt of a user to login requires one or more lookups to be performed on the database. It is thus very important for the database to be able to scale well and to be very reliable (or support redundant mechanisms).

For mail systems, it is very common to use something like LDAP as a database, or Lightweight Data Application Protocol.

Another very common database is MySQL or any SQL variant.

5.2. About mail data

Ok, there are mainly two ways to share mail data among a cluster of mail servers:

- use some kind of shared file system
- use a proxy & redirect mechanism, in order to avoid the necessity of sharing at all

In the first case, we talk about “Shared Storage” clusters, while in the second we talk about “Distributed Storage”.

5.2.1. Shared Storage Clusters

In shared storage mail clusters, all nodes (machines) of the cluster that need physical access to the mail data use some kind of shared file system.

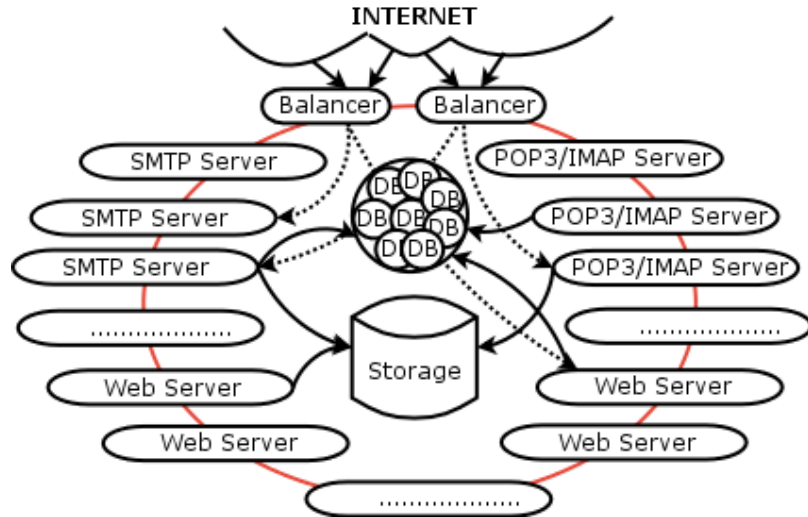
At time of writing, there are actually many solutions which can be used:

- Some kind of SAN (Storage Area Network) with an adequate file system (like Sistina GFS). This allows many machines to mount the same hard disks, connected using technologies like Fiber Channel or similar.
- NFS Servers/Appliances exporting a file system for all nodes to mount. This is the simplest solution and probably the best known.
- NFS crossmounts, where each server exports a portion of the file system for others to mount. Each node will have n-1 (where n is the number of nodes) NFS mounted partitions.

- Some other kind of networked file system, like samba or alike. Well, if you want to hurt yourself...

A shared storage cluster thus introduces one more component to the cluster architecture: a storage server/system.

If we thus try to draw a scheme with all the components involved with the cluster and how they interact with each other, we may obtain something like:



Shared Storage Architecture

Keep in mind that many components can reside on a single machine. You don't have to have a thousand machines to install PigeonAir, and it is not uncommon to setup a single machine with all the above indicated components.

Ok, now let's talk about the above scheme: as you can see connections come from the internet to the balancers. The balancers, depending on the connection kind (IMAP/POP3/SMTP), take one connection at a time and redirects it to one of the machines providing the necessary service, trying its best to balance the load. That machine accesses control data using what it believes to be a single database, while it accesses mail data using the shared storage.

Obviously, to share a common storage, we need to have operating system support and rely on operating system facilities.

5.2.2. Distributed Storage Clusters

Using this architecture every server (well, those servers we decide to) part of the cluster keeps part of the mail data stored on its own file system.

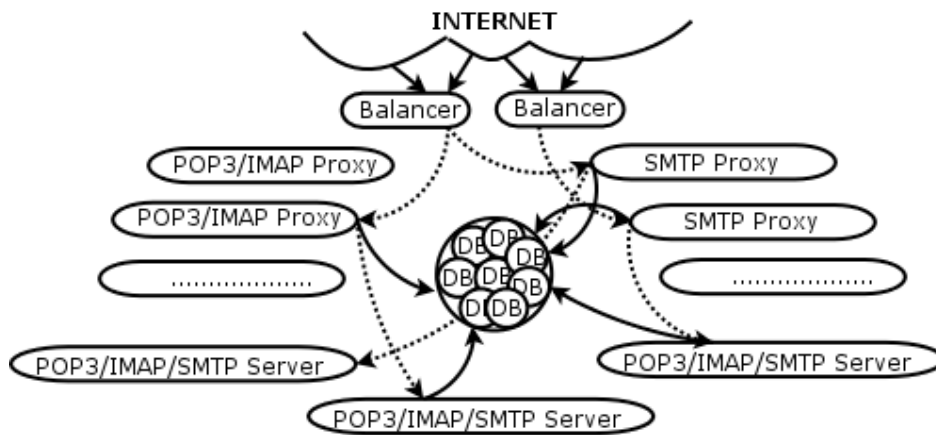
This means that when a mail is received, it is simply redirected on the right server, without any need for the operating system to support strange/unreliable shared file systems.

The same thing happens with IMAP/POP3: after authentication, the user is redirected on the server where his own data is stored, without need for further authentication.

This kind of architecture introduces two new elements to the cluster structure: a SMTP “proxy” (this term is not necessarily correct, since a “normal” SMTP server can be configured as a “proxy”), and an IMAP/POP3 proxy, something able to redirect connections to the correct server.

From a technical point of view, a SMTP proxy is easily achievable by using aliases/rewriting rules on most of the known SMTP Servers, while an IMAP/POP3 proxy is a very simple software that performs users authentication and then redirects connections to the server in charge.

If we repeat the same exercise as in the previous section, we may get something like:



Distributed Storage Architecture

Where the balancers receive connections from the outside world. The balancers redirect each connection to some protocol proxy, which performs some basic operations to understand where a particular mail needs to be stored or where an user account is actually stored, and redirects the connection to the correct server.

In this case, to build up a solution like this, we would need no particular operating system support but we will have to install some kind of software able to redirect connections (we will need the proxies).

5.2.3. Comparing the two models

The PigeonAir Project allows administrators to use any of the previously described models and any of the described technical solutions, even the crossmounts one.

However, before going on some considerations needs to be made:

- we just talked about scalability, but not much about reliability. One important point in building up a cluster is to avoid SPF, or Single Points of Failure. A Single Point of Failure is something that in case of failure takes the whole system down.

- in both models, we have a couple of possible SPF:
 - the database, if no redundancy technologies are being used
 - the shared storage, in the shared storage model. In this case it is much harder to provide a redundancy mechanism without hardware/operative system support.
- for the whole cluster to scale, it is important for both the database and the shared storage to be able to scale. If we want to scale the cluster without taking it offline, both the database and the shared storage needs to be able to scale while online. While it is probably easy to scale a database without taking it offline (or taking it offline only a few seconds), it is much harder for a shared storage, where some day or another we will hit the physical limit on the number of disks we can attach on it and where it is much more risky to resize an array or volume of disks and thus a file system than it is to scale a database.

From this perspective, it is important to note that a redirectors based mechanism does not introduce any more SPF and removes one: the shared storage. It also allows easier scaling: when a new server is added, connections just need to start be redirected to that server.

6. PigeonAir Architecture

Right now, it is probably time to start talking about the PigeonAir Project and PigeonDeliver.

There are 5 main characteristics that make PigeonAir easy to be used to build up mail clusters:

- All PigeonAir components share natively the same database structure. This means they are all able to easily share control data. They also share a database abstraction layer, that will allow them to support many kinds of different databases (datatree layer). Right now, however, only LDAP is supported. So we will probably have a LDAP Database in our cluster.
- PigeonDeliver natively supports clustering. This means that it natively provides a POP3/IMAP proxy and that it is able to redirect mails natively, without using aliases or rewriting tables.

Actually, PigeonDeliver is much more powerful: as being made of modules, it allows each module to reside on a different machine. This is the mechanism used to deliver emails on different machines: the mailStore module (which takes care of storing emails) simply resides somewhere else, and the abstraction layer calls a remote module to perform delivery.

- All administrative operations can be performed by just modifying the shared control data. This means that the administrative interfaces just need to perform operations on the database to add/remove users, add/remove domains or change user/domains configurations. This may not seem such a great feature, but keep in mind that most administrative interfaces out there need to execute commands on the server(s) in order to perform administrative operations, to store data on the file system (which make them very hard to distribute), or to connect to remote ports using weird protocols just to inform the server that new users have been added/removed.

And yes, if we need a mail cluster to handle all the emails, probably we also need a cluster of web interface/administrative interfaces to let all users access those services.

This also means that if you want to build your own applications to manage a PigeonAir Cluster, you just need to write code to access the database and perform operations on the database, *nothing more*.

- PigeonDeliver born to virtualize mail services. As such, it doesn't require users to be added to the passwd file, but still supports:
 - each user to have its own UID. On Linux, this means you can have up to 4G of users on one servers, a little more than half the users of our planet.
 - deliver emails using the correct UID, avoiding most security risks.
 - enforce quota using the file system support.

Keep in mind that PigeonDeliver is very configurable, and doesn't force you to use any particular configuration. If you don't want uids to be used, just disable them.

- All PigeonDeliver modules share a common way to fetch/retrieve user configurations, avoiding atomicity problems at all and allowing a very flexible and expansible interface.

When building clustered mailing systems, it is usually a pain to add components that need to fetch users configuration from a database or use some administrator defined configuration.

That said, by using PigeonDeliver you are free to use any kind of clustered model you may like. You may even decide to mix the two models shown in the previous sections. You could, for example, make the cluster use a shared storage until it suffices and start to use redirects in order to make the cluster use a second shared storage as soon as the first one is not sufficient anymore. The magic is mainly performed by the mailStore module and by the embedded IMAP/POP3 Proxy, which create a layer of abstraction over the real storage.

If we need to translate the schemes shown in the previous sections into something practical, we actually need to have a very good knowledge of the software infrastructure, of which databases are supported and how data is stored on disks.

In the next section, we will try to go into further technical details about PigeonAir concepts and we will try to translate the logical schemes into something practical.

7. PigeonAir in practice

In the following sections, we will try to build up a very simple clustered architecture to be used with PigeonAir.

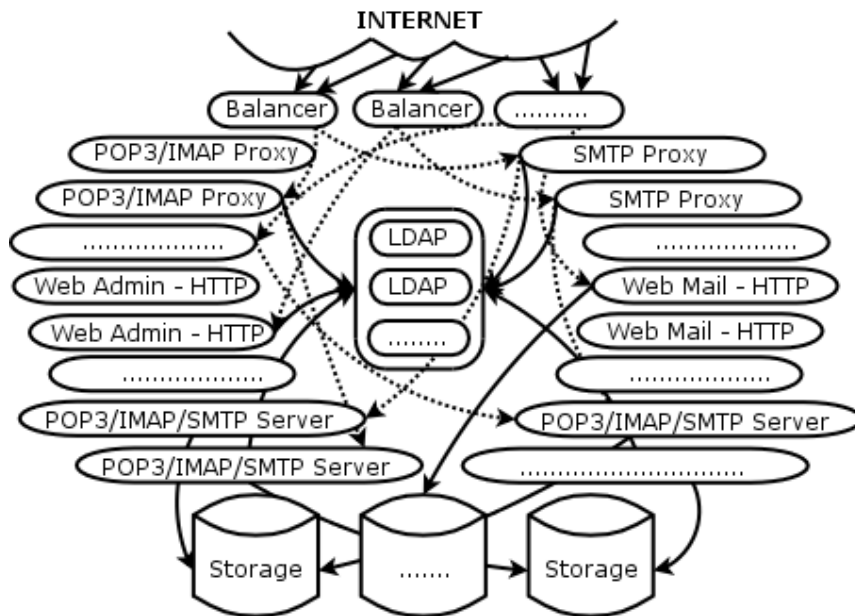
If you take a look to the various modules documentation, you can realize it is very easy to build up very different solutions.

Ok, in the previous sections we talked a lot about the various components of a mail clusters. Let's say we want to build a PigeonAir cluster, and let's say we decided to use LDAP as database backend, and let's say we want to have a clear picture of the whole structure.

From a logical point of view, the task of building a PigeonAir cluster mainly involves a decision of how many components of a single type to use, on which machine to put them, which components we don't want to use and which components should share the same hardware resources.

As you should know by now, PigeonDeliver uses modules to deliver emails. You can also decide to distribute those modules among different machines, even if that is not usually necessary, and won't be discussed in this document.

So, let's start by drawing a scheme of all the components we may make use of.



All components we can use, in practice

Note that broken lines represent “redirections”, while unbroken ones represent simple connections or services usage. Not all connections or redirections have been represented.

In more descriptive words, we need to complete a puzzle and all the pieces we have are:

- one or more balancers, to split the load.
- one or more LDAP servers, to safely keep control data.
- one or more SMTP/IMAP/POP3 proxy, to redirect connections to the correct servers.
- one or more shared storages, to keep users' data and to split the load.
- one or more web servers for the Web Mail.
- one or more web servers for the Administrative Interface (Web Admin).
- the real SMTP/IMAP/POP3 servers.

Ok, depending on our needs, we will combine pieces differently, and configure PigeonAir accordingly.

7.1. Spaghetti arithmetic for hardware requirements

Ok, now a very simple example. We were just asked by a brand new ISP to build up his own Mail Server. The ISP himself doesn't care to have a cluster or a single machine, he just wants something able to scale well in case things go well, and he hopes things will go very well. So, we are wondering about how to configure his server...

At a first glance, we could just configure a normal mail server and be happy with that. However, the provider told us that he hopes things will go real well in a very short time, so he is probably planning to grow... additionally, we really like the interface those folks of the PigeonAir project have developed, and we'd like to use them...

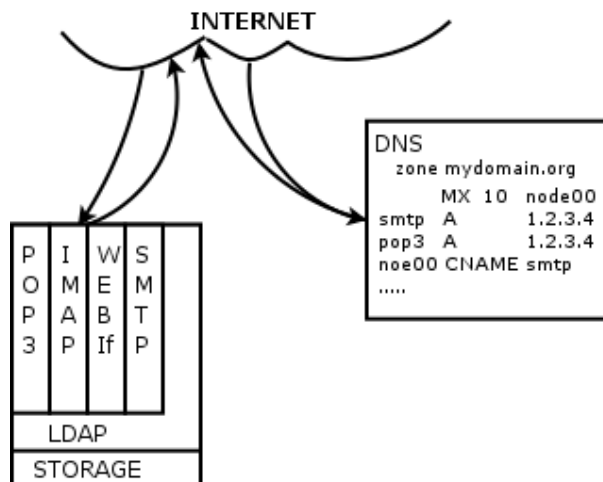
Ok, let's try to write down our hardware shopping list to build up our mail server:

- uhm... balancer... do we need this? No, since we have a single server, we don't need it. And we are always in time to change a bounce of DNS records or to put a balancer in the middle in case we need.
- LDAP. We do need a LDAP server. So 1 LDAP server.
- IMAP/POP3 Proxy. We don't need it right now, but we may need it later.
- Shared storage... what's that? We don't really need it... in case of growth, we'll use the provided proxy.
- Well, ok, now I still need a web server for the web mail, one for the web admin and one for the real SMTP/IMAP/POP3 server.

Ok, so, do we need 4 servers? No. 1 is enough. We can put all the services on a single machine, and as long as we have some couple thousands users/domains, things should just get along fine.

Well, ok, and if I wanted to start from the beginning with a cluster with two nodes? Did I need like 100 machines to build it up? No. Two machines would still suffice. You use the DNS as balancer, and you put all services on all machines. Probably not the best solution, but splitting up services is really useful when the load gets real high, in order to simplify the configuration of every single machine and to allow administrators to fine tune hardware requirements and to optimize the load by adding/removing machines specifically configured to perform their own task.

Ok, so we have to install all the needed software, and finally install PigeonAir and everything else in a pretty straight setup. Not much to say, PigeonAir should support something like this almost out of the box, obtaining something like:



Hardware setup of a single machine cluster

7.2. Our small cluster grows...

So, let's say the provider who used our server actually grew as expected. We were thus contacted to prepare the service for new users to join in.

Ok, so we finally need to expand our own cluster, we basically have to add one node, without interrupting the service.

So, let's first prepare the second machine, connect it to the network, and slightly modify the original setup in order to make it working.

Before going on, let's rethink a bit about our previous service distribution. Since the load increased, we could also start wondering about reliability of the services and stuff like that.

Basically, we need the same services as above. Since the load is quite low, it still doesn't make much sense to keep, for example, web services on a machine and let the other handle SMTP/POP3, so lets sneak with the same distribution as above.

The only weird thing about this setup would be the LDAP server. We would end up with one LDAP database and two servers. While I believe there would be no problem in term of CPU load, it would become for sure a SPF. A failure of the first server would bring both servers down. Additionally, the first one would have access to a local database, while the second one would access a remote database, over the network.

This may seem a little overhead, but consider that:

- the PigeonDeliver database structure requires few queries but transfers lot of data
- the caching layer in the datatree handling is not complete yet
- in the average, there will be few changes to the database compared to the number of accesses (a few accesses for every received email/POP3/IMAP connection, compared to a few writes for every administrative operation on the database).

So it could make sense to setup a Slave LDAP server on the second node of the cluster, in order to have a more reliable service, give both nodes access to a local database and let only changes to the database travel among the network.

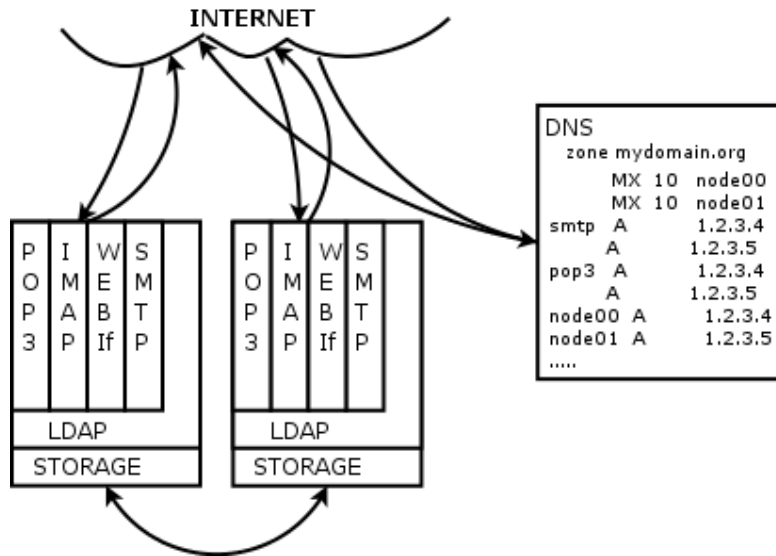
We should also consider that the LDAP database is probably the most important data stored on our servers. Well, I mean, if we loose emails, it's bad... but if we loose the list of our users or their passwords, it is even worse, and for sure we are in *big troubles*.

So, how do we configure this second node? It is enough to enable the POP3/IMAP Proxy on the first node, replicate the configuration of the first one over this second, change a couple configurations (hostname, IP address, ...), setup the LDAP Server to behave as a Slave, and finally tell the balancer it can start to redirect connections to this node too.

Nothing more. That way, you can add as many nodes as you want. You just need to setup another machine with the same configurations, make sure the model still does make sense, and configure the balancer to connect to this machine too. The information in the shared control data (LDAP stored informations), will be enough for the second node to correctly talk with the first one (given the correct authentication key) and for the first one to know when to contact the second.

Translate all of this in command lines is left as an exercise to the reader. (At time of writing, PigeonDeliver code is accessible only using TLA/CVS. So it doesn't make much sense describing the practical steps to install/configure something that still uses a developers-only installation procedure -- %TODO%).

Ok, well... the final setup may look something like:



Hardware setup of a single machine cluster

Note that in this setup it may be a good idea to provide the two nodes with a second network interface to connect them using a private network, to allow an higher load to be handled without overloading the public network.

Using the above described setup the load of the internal network will *always* be less than that of the external network, and as long as both networks have the same bandwidth, it will not become a bottleneck. This sentence cannot be stated in case a shared storage is being used, and on most setup of a certain size NFS *does become* a bottleneck, unless mixed solutions are used.

7.3. About balancers

At time of writing, the PigeonAir (<http://www.pigeonair.net>) project does not provide any load balancer.

However, a load balancer is very easy to build and configure using existing technologies.

As a first choice, you could even decide to buy an appliance to handle the load balancing. Usually very efficient and very reliable... but take a look to your own dealer catalog to know more.

A second choice could be to setup a Linux IPVS (<http://www.linuxvirtualserver.org/>). IPVS is mainly a kernel module allowing you to setup a Linux BOX as a network balancer, acting at a low level of the IP Stack. IPVS is also *very* reliable, and with some work you can also configure the whole cluster to use more than one balancer, for take over and for improved scalability. To know more about setting up Linux IPVS for PigeonAir, take a look at %TODO%.

As a last resort, you could simply use the DNS. The DNS is very easy to configure to balance requests over a cluster of mail servers, but requires you to add many records in each of the virtual domains handled by PigeonAir and is not very configurable in splitting the load (eg, if something bad happens, it takes a long time for the DNS to stop sending it connections and requires manual intervention). However, it is extremely reliable and under normal conditions perform a fairly good job. Keep also in mind that I wouldn't suggest to use the DNS for any cluster with more than 2/4 nodes. Take a look at %TODO% to know how to configure a DNS to balance the load over a PigeonAir cluster.

7.4. When a mail arrives...

Ok, let's see what happens when a remote server needs to send a mail to one of our users:

1. First of all, the remote server searches the domain of the recipient (our user) in the DNS, looking for a MX record.
2. Given the sender can find the MX record, the remote server will contact the first server found with lowest priority. In case more than one MX record have the same priority, the DNS will take care of mixing them all before returning the answer to the remote server.

In case an appliance or IPVS server is used as balancer, the MX record will contain the IP address of the balancer. The remote server will thus contact the balancer that will take care to redirect the connection to one of the nodes of the cluster.

3. One of the nodes SMTP servers, probably Postfix, will receive the connection. The SMTP server will lookup the user in the database, to verify it is its duty to deliver emails to the given user. If it is not, it will either drop the connection returning an error, or accept it anyway if configured as a relay. If it is, the mail will be queued for local delivery.
4. PigeonDeliver will be called to perform the local delivery. For more details about the delivery, please refer to %TODO%.
5. PigeonDeliver will call the modules involved with the delivery that can be executed directly by the local server (for example, the mailAntivirus or mailForward module). All the modules will be contacted using the server_map and service_map. Those maps allow single modules to reside on different machines.
6. Once the email gets to the mailStore module, depending on the clustered model being used and on user configurations, it will either:
 - forward the email to the mailStore service of the server where all other user emails are kept
 - store the email directly on the shared storage

7.5. When a POP3/IMAP connection arrives...

Now, let's see what happens when a user tries to download his own emails:

1. First of all, his client will look for the IP address of the POP3/IMAP server in the DNS.
2. If DNS balancing is being used, the DNS server will return the list of all POP3/IMAP servers available, with the server listed in a different order for each request, in order to provide simple balancing.
3. The client will connect to the IP of the IMAP/POP3 server. If there is a balancer on the way, the balancer will be configured to redirect the connection to one of the POP3/IMAP Proxies.
4. The PigeonDeliver IMAP/POP3 Proxy will authenticate the user, lookup its home directory in the database, and redirect the connection to the real IMAP/POP3 server. This server will be configured to accept connections only from redirectors, and will relay on authentication data and configurations provided by the Proxy, in order to avoid unnecessary lookups. Proxy/Server connections are obviously authenticated using a simple challenge response mechanism.
5. PigeonDeliver will call the modules involved with the delivery that can be executed directly by the local server (for example, the mailAntivirus or mailForward module). All the modules will be contacted using the server_map and service_map. Those maps allow single modules to reside on different machines.
6. Once the email gets to the mailStore module, depending on the clustered model being used and on user configurations, it will either:
 - forward the email to the mailStore service of the server where all other user emails are kept
 - store the email directly on the shared storage