

The Pixie Project

Massive Ubuntu Linux desktop deployment at lower cost

2010-06-04



Mathematical Institute, Leiden University, The Netherlands



Marten Vijn, mvn@math.leidenuniv.nl,

Martijn Feleus, feleus@math.leidenuniv.nl



Contents

1	Summary	4
1.1	Abstract	4
1.2	Audience	4
1.3	About the authors	4
2	For the decision makers	5
2.1	About the Mathematical Institute	5
2.2	The problem	5
2.2.1	Changed user behavior	5
2.2.2	Performance	5
2.2.3	Hardware support	5
2.3	The solutions	5
2.3.1	Summarization of the issues	5
2.3.2	Demands	6
2.3.3	Hardware options	6
2.3.4	Hardware selection	6
2.3.5	Software solution	6
2.3.6	Summary of the solution	7
2.4	Financial consequences	8
2.4.1	Hardware	8
2.4.2	Administration	8
2.4.3	Licenses	8
2.4.4	Development	8
2.4.5	Server setup	8
2.4.6	Deployment	8
2.4.7	Example: deploying 100 desktops	8
2.5	Comparison of a local installation, terminal server, Pixie setup	10
2.6	User satisfaction	10
3	For the administrators	11
3.1	Mathematical Institute	11
3.1.1	Administrators	11
3.1.2	Desktops	11
3.1.3	Hardware overview	11
3.2	Thin clients and Gentoo terminal servers	11
3.2.1	Used techniques	11
3.3	Issues with terminal servers	12
3.3.1	64-bit	12
3.4	Solution (design)	12
3.4.1	Scaling	13
3.4.2	Maintainability	13
3.4.3	Performance	13
3.5	Stacking the techniques	14
3.5.1	Preboot eXecutable Environment (PXE)	14
3.5.2	Casper over NFS	14
3.5.3	Homedirs	14
3.5.4	LDAP and Kerberos	14
3.6	(Re)Building an image	15

3.6.1	The base install	15
3.6.2	The kernel and initrd	15
3.6.3	Preseed package configuration	16
3.6.4	Adding packages	16
3.6.5	Adding localized files	16
3.6.6	Casper	16
3.6.7	Build squashfs image	16
3.6.8	Build results	16
3.6.9	Staging to production	16
3.7	Results	17
3.7.1	Scalability	17
3.7.2	User satisfaction	17
3.8	Failsafe and redundancy	17
3.9	Known issues	17
4	Future Plans	17
4.1	Ubuntu 10.4	17
4.2	USB drive	17
4.3	Saving unionfs	17
4.4	Http(s) boot	18
5	Acknowledgements	18
6	Licenses	18
7	Appendix	19
7.1	Code examples	19
7.1.1	Preseed	19
7.1.2	Adding packages	19
7.1.3	Adding files from subversion to the tree	19
7.1.4	Casper	19
7.1.5	Build squashfs image	20

1 Summary

1.1 Abstract

For over 10 years, the Mathematical Institute (MI)¹, Leiden University has been a Linux-oriented organization, using various flavours of Linux on both desktops and servers. The setup used thin clients and terminal servers. In 2009 the end users reported that the thin client and terminal server setup was becoming increasingly slow and unresponsive. This document details the steps taken to significantly improve performance and reliability whilst sharply reducing the total cost of ownership.

The main goals were tasked with finding new solutions that improve desktop performance for end users and decrease the amount of work per desktop for administrators. The selected solution was found in combining known technologies like networked booting, a read-only system disk image with live-cd techniques and network attached storage. The Pixie Project derived its name from PXEboot², a system that enables a desktop to boot over the network instead of starting from a hard disk. In early 2010 this setup was tested and deployed. A user satisfaction review showed increased satisfaction after the deployment. The Pixie Project solution was found to be highly scalable, easy to maintain and highly customizable. The complete approach has been documented on the Ubuntu Wiki³.

1.2 Audience

This document is mainly intended for decision makers (Chapt. 2, non-technical) and system administrators (Chapt. 3, technical), but may be interesting for a wider audience as well.

1.3 About the authors

- ⊕ Marten Vijn has been an opensource evangelist for more than a decade. He has a Bachelor degree in health care as physical and occupational therapist. He has been a Unix, Linux and network professional for several years now. His favorite OS flavors are FreeBSD and Ubuntu.
- ⊕ Martijn Feleus studied astronomy at the Leiden Observatory, where he was introduced to Unix and Linux. He works as Unix, Linux and network administrator at the MI. Martijn's favorite OS flavors are Gentoo en Ubuntu.

¹<http://www.math.leidenuniv.nl>

²The Preboot eXecutable Environment offers a networkcard as bootable device to the BIOS

³<https://help.ubuntu.com/community/Desktop/PXE>

2 For the decision makers

2.1 About the Mathematical Institute

The Mathematical Institute has about 450 users and 130 workplaces. Main deployment (before this project) had four terminal servers and a thin client setup⁴. The end-users mainly use Linux and are familiar with a Linux desktop. The Mathematical Institute employs three Unix system administrators, who together fill two fulltime jobs. They also man a helpdesk which is located close to the staff.

2.2 The problem

2.2.1 Changed user behavior

Some 10 years ago, people at the MI would read their email in pine, mutt or a graphical client, produce scientific documents in tetex/latex and browse the web, which was far simpler than it is today. Over the past decade the usage has shifted to using browsers as a front-end to the world outside, reading email, watching movies and lectures through them. Our users need to be able to use Flash and want to read PDF files from within their browser. The browser needs to perform well, and to do so, it requires a lot of power. This can be a strain on the terminal servers, which share their resources among their logged-in users. At the same time, high performance desktops and notebooks are getting more and more available for the consumer market at low cost. Because of this, the user expectations in performance and usability have changed a lot. This meant that our thin client setup was getting absolated. This is clearly confirmed by a user satisfaction review⁵, where performance issues were the main complaint.

2.2.2 Performance

Although the thin client and terminal server setup runs on relatively fast hardware⁶ the changes in demand have made the overall performance sluggish. In a terminal server setup all the user processes are on the server. The graphical output of these processes is pushed to the thin clients and is displayed on the screen. Bottlenecks include the storage environment and the protocols used to access the terminal servers (XDMCP and RDP), in combination with thin client hardware. Also, some programs (browsers, Flash, Acrobat Reader) use a lot of resources on the terminal servers which can push the terminal servers to a high load. This was 'unworkably slow', users reported.

2.2.3 Hardware support

User expectations have changed here too. With the terminal server solution is accessing media like dvd/cdrom is not available. There was no support of optical media, and access to a usb drive was only (reliably) possible using a command line tool. There was no working solution for sound. These days users expect such things to just work.

2.3 The solutions

2.3.1 Summarization of the issues

Our current setup is getting obsoleted by user expectations. It is getting slow and new solutions are needed soon. No large budget is available in time or money.

⁴In our case 120 thin clients open a terminal session to one of the four terminal servers. A terminal session displays to desktop on a 'thin client' and all the user programs run on the terminal server.

⁵https://projects.math.leidenuniv.nl/trac/howto/wiki/user_satisfaction

⁶Intel Xeon Quadcore with 8Gb Ram

2.3.2 Demands

The solution needs to be:

- ⊕ user friendly
- ⊕ scalable
- ⊕ low cost (hardware and administration time)
- ⊕ perform well

2.3.3 Hardware options

The current options are:

- ⊕ faster thin clients with terminal server
- ⊕ pc-style desktops
- ⊕ nettops
- ⊕ notebooks

2.3.4 Hardware selection

We tested some faster thin clients. These thin clients still did not perform well enough and gave reduced access to hardware, so this isn't a good solution for us in the long term. We used to use desktops before we had thin clients. PC-style desktop gives good access to hardware, but costs a lot of time to deploy and maintain and has a shorter life time than thin clients. Notebooks cost even more time and money than PC's, perform less, are more vulnerable to physical damage and can get lost or stolen more easily as well. Nettops, which are low-power PC's, are low cost and perform well enough for our purposes, since no heavy calculations need to be done on the machines. Therefore, we chose to try out a good nettop PC as a platform.

2.3.5 Software solution

A user friendly linux desktop

The Mathematical Institute has been using Linux-based desktops for over 10 years, merely migrating from one Linux operating system to another as the need arose. We chose Ubuntu because of its well-deserved reputation as "an operating system for human beings". One of the results is that Ubuntu is very easy to install on desktops and notebooks, even for non-technical people. Ubuntu has a special edition for netbooks: Ubuntu Netboot Remix. Having the same desktop at all places serves user convenience and makes it easy to support from one helpdesk. The MI offers scripts which end users can use to build the same desktop at home as they use at the MI. Users can connect to the MI network over the internet and have access to files and special mathematical software.

Scalability and hardware support

Installing an operating system on a computer and running your desktop on it means you should have easy access to all the hardware your computer has, like DVD drive, USB ports and so on. This requires more administrative work though. We decided to go for a network-based operating system that needed no local installation and would be maintained centrally and therefore requires little effort to maintain for 100 desktops or more. Having no local installation on the desktops means a fast and easy deployment is

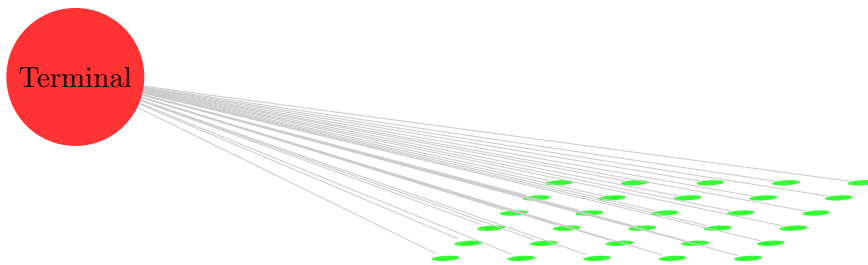
possible. Over 100 desktops can be served from one operational server; more is possible since we haven't seen the limit yet. The hardware for this server is today's commodity hardware⁷. The estimated load may peak at 15%, but the average load is closer to 1%. Serving so many desktops makes this setup very scalable. To serve another 100 desktops a second server would suffice, however since we don't know where the limit actually is, even that may not be necessary. It may be the wise thing to do though; having a backup system in place is always good, and one extra server really doesn't add to the cost in any significant way.

Low cost (administration time)

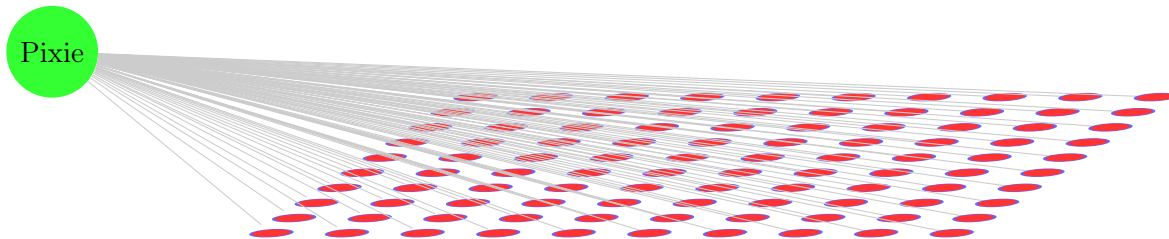
One thing the thin client setup was very good at was ease of administration. Almost nothing needed to be done on the thin client side beyond setting one up for the first time. All other administration was done on the terminal servers. Maintaining four servers is far more efficient than maintaining 100 desktops. So the centralized model of administration needs to be preserved as much as possible; after all, there is no additional budget for extra administrators.

Performance

The main strategy is to move high cost processes to places where other users are not involved. In order to increase performance, browsing the web, watching Flash movies and reading pdf-files need to happen on the local desktop, not on a centralized server in a data center. This necessitates running more than just a bare minimum operating system at the desktop, which is something thin clients are not designed to do.



A terminal server setup has the load on the terminal server.



The Pixie server setup has the load on the clients.

2.3.6 Summary of the solution

Combining the chosen hardware with Ubuntu and the techniques detailed above gave us a promising solution. The techniques themselves were found to perform well, even though the desktops boot from the network: one could hardly tell the difference from a local installation. The results, both in cost and user satisfaction, are given below.

⁷Intel Xeon Quad Core, 2Gb Ram

2.4 Financial consequences

2.4.1 Hardware

Common PC hardware drops in price every month. By choosing commodity hardware instead of thin clients we get more performance for our investment and lower costs for maintenance.

2.4.2 Administration

Having no local installation on the desktops saves lots of time on installing, maintenance and recovering user data. The highest reduction of costs is expected from the low maintenance approach. Once the machine is set up to start from its network card, all the work is done. No per-machine maintenance is needed. Updates are done once, in one image used by all Pixies. After rebooting a Pixie, using the new image, the update is done. Also, a rollback of software is easily performed by booting from a previous image.

2.4.3 Licenses

The Ubuntu promise is clear: “Ubuntu will always be free of charge, along with its regular enterprise releases and security updates”⁸. Not only is it free of cost, it is also available on a longer term basis, with a 5 year period of updates and maintenance. Some third party software may ask end users to agree with their license though.

2.4.4 Development

A quick calculation shows a research and development period of one full time month for a senior system administrator. In practice, it was done in three months, since there were also other tasks to do. We developed all scripts on one development machine.

2.4.5 Server setup

A development server is installed with Ubuntu Linux and offers some basic services. To improve security and to have the ability to do staging, it is recommend to build the images on a development or build server. From the development server we move successful builds to the production server. The development server can also serve as failover for the production server and ensure a higher availability.

2.4.6 Deployment

Preparing a deployment and the actual deployment costs less time than would be the case with local installation of the operating system on all computers. Most of the time is spent on building a correct image; once that is done, deployment can be very quick and simple. How much time is involved depends on the skill of the administrators, the complexity of the environment and the amount of desktops to be deployed.

2.4.7 Example: deploying 100 desktops

Depending on your setup, knowing the content of this document, a deployment (including fine tuning the installation image) would probably take approximately one or two (full time) admin months. This includes unpacking and installing a few test desktops and servers. Once installed, a period of fine tuning commences; expect to have to address a few issues daily and fix them. The image would be rebuilt on a daily basis (or more frequently, depending on need). Once the testing is done, deploying the rest of

⁸and more on <http://www.ubuntu.com/>

the desktops can be done quickly by, for instance, helpdesk employees. After this phase, you would enter 'maintenance mode', spending about 10 minutes a day for monitoring, reading logs, helping users, and making builds with critical security fixes. This estimate depends a lot on your environment of course, so your mileage may vary.

Example deploying 100 desktops

Hardware	Type	Price/unit	Quantity	Total
Server	1U, 4 cores, 2Gb ram	900 E	2	1800 E
Desktop	ASRock Ion (with nvidia graphics card)	240 E	100	24000 E
Screen	22" Iiyama with speakers	180 E	100	18000 E
Keyboard/mouse	generic	25 E	100	2500 E
Software	Almost all GPL-based			0 E
				46300 E

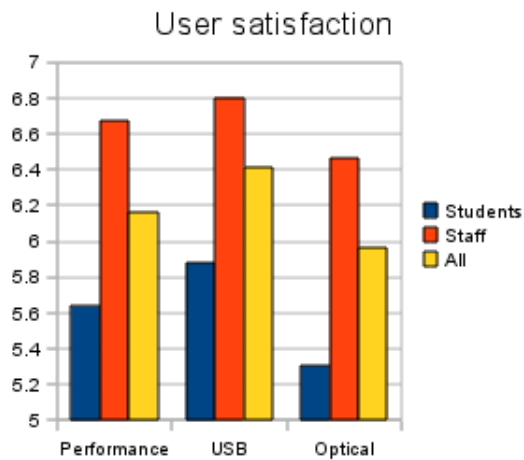
2.5 Comparison of a local installation, terminal server, Pixie setup

	local installation	terminal server	Pixie setup
Scaling	*	**	***
Server hardware (costs)	***	*	**
Client hardware (costs)	*	**	***
Performance	***	*	**
Hardware support	***	**	***
Administration time	*	***	***
User satisfaction	***	**	***
Builtin software rollback	*	*	***
Total	16	14	22

See the footnotes on Hardware support⁹ and Administration Time¹⁰. All items are weighted equally. To be realistic you should assign weights for the items appropriate for your environment.

2.6 User satisfaction

The Pixies, as we call them, were rolled out for our scientific staff users, and their previous one year old thin clients were installed in the student computer rooms, complete with bigger 22" monitors (so both received an upgrade, but the students continued to use the thin client setup). A questionnaire was distributed and filled out among 70 users. Students (blue) and staff (red) each represent 50% of the survey population. The staff indicated a higher level of satisfaction about their new desktops. There are still performance issues, but these are caused by the relatively low performance of our storage environment.



⁹Like: optical drives, graphical acceleration, usb-drive, sound

¹⁰The higher the score the less time it takes.

3 For the administrators

3.1 Mathematical Institute

3.1.1 Administrators

The MI employs three network- and system administrators. Despite individual specialties all of them have the same basic task: to keep everything running. The users of the systems are of course the students and researchers, Ph.D. students, postdocs and (assistant and associate) professors, together with a small support staff. There is a long history of using \TeX and open source platforms like Linux. Currently, four main platforms are deployed; Gentoo, Ubuntu, FreeBSD and Windows. User desktops run on Gentoo, Ubuntu or Windows (via remote desktop, no local installation).

3.1.2 Desktops

As desktops, Gnome, KDE, Xfce4 and Fluxbox are provided. Also, for compatibility reasons and to serve the few Windows users, there is one Windows terminal server. Mathematicians have long used Unix-like systems to work on, and using open source software is a natural extension, given Linux and the various BSD variants. Also, some mathematics software is available in the open source world, or developed in similar ways.

3.1.3 Hardware overview

- ⊕ 40 thin clients
- ⊕ 87 ASRock Ion computers (the Pixies)
- ⊕ 6 PCs
- ⊕ 13 printers
- ⊕ 30 servers
- ⊕ 2 core switches, 8 access layer switches and 15 wlan access points

3.2 Thin clients and Gentoo terminal servers

3.2.1 Used techniques

XDMCP We have three Gentoo based terminal servers. They offer XDMCP to thin clients, allowing users to have a desktop running on any of these servers. We also tested FreeNX but even then the thin client performance is still quite a bit below par.

Thin client administration Our thin clients are IGEL(tm) thin clients. They are administered by a thin client manager running on Java, which uses an external PostgreSQL database. As thin clients go, they're very nice (ours are Linux-based), but with modern websites such as Youtube and Gmail (some of the graphically intense themes; the default theme works fine) they do not perform to our users' expectations. This problem surfaced when we got bigger screens (22") for the thin clients, meaning that their graphics capabilities were being stretched to the limit. Having to update such a big screen at the rate (full-screen) video requires is simply not something these thin clients were designed to do. Of course, we looked for better thin clients, but none that we tested could deliver the required performance, even if the hardware itself looked capable enough. Evidently, the limitation was the fact that the images had to be transmitted to the thin clients over the network, then processed by the thin client and displayed.

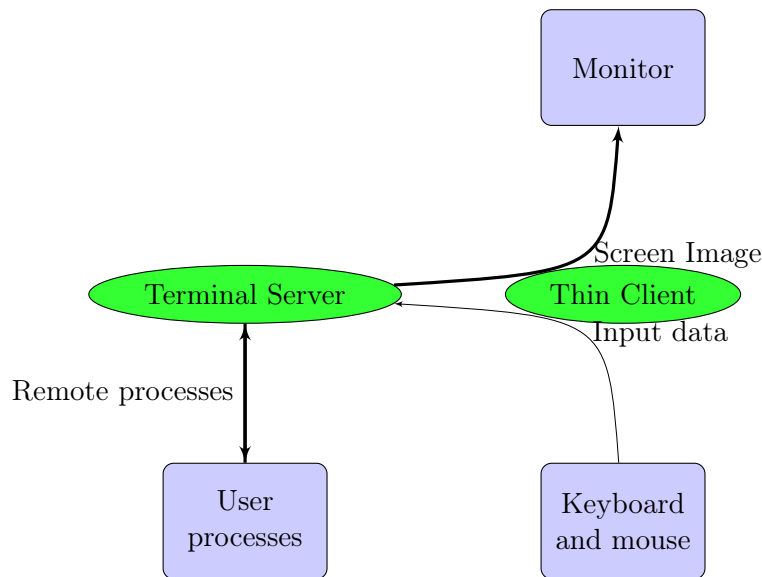


Diagram of a terminal server with thinclient, all user processes are on the terminal server

Network administration The thin clients have a separate VLAN shared with the terminal servers (needed because of the lack of encryption of XDMCP - using ssh tunnels would create performance issues). The terminal servers are connected to the internet via public IP addresses. For the new Pixies we need no such setup, since they have a locally running operating system and all sensitive information can be transmitted encrypted in some form. They all have public IP addresses.

3.3 Issues with terminal servers

3.3.1 64-bit

Some software from Adobe(tm) (Flash, Acrobat Reader) does not work very well using XDMCP on 64-bit machines. It burdens the terminal servers with runaway acroread and flash processes, causing high load and making the terminal servers unusable at times. This aggravated the unfavourable impression many users had of our thin clients.

3.4 Solution (design)

We wanted to try and see if we could get the advantages of personal computers combined with the advantages thin clients offer. Personal computers offer performance and the ability to customize, but have the downside of a local installation to care for and suffer hardware failures because of their moving parts. Thin clients, on the other hand, are more robust and require little to no administration once configured, but cannot deliver the performance our users expect these days. We wanted something that would scale well, perform well and require little maintenance.

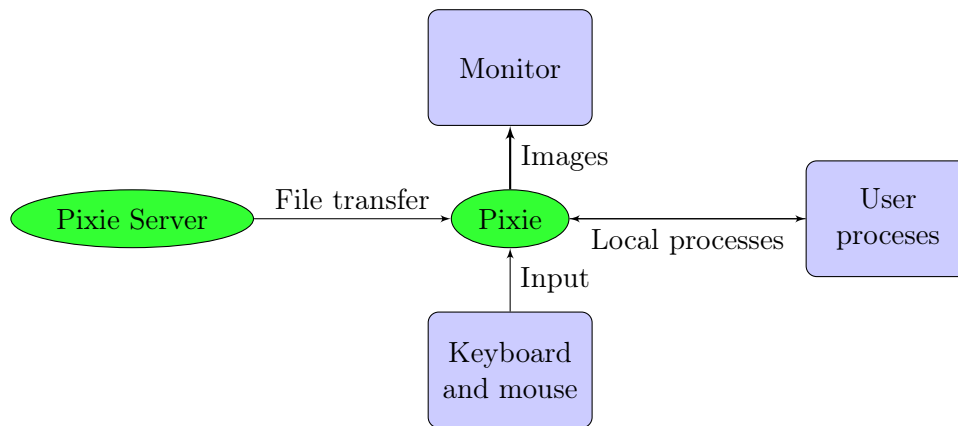


Diagram Pixie server with one client.

3.4.1 Scaling

Our scaling requirements meant that we wanted the following traits:

- ⊕ No local installation, therefore no local administration besides from BIOS settings
- ⊕ Network centered, administration is done centrally
- ⊕ Remote access to a Pixie is possible via ssh for debugging, local support and temporarily installing packages
- ⊕ Read-only image accessed from the network improves NFS performance and reduces storage needs on NFS-server
- ⊕ Easy to replace desktop hardware.

3.4.2 Maintainability

For maintainability, the following were deemed necessary:

- ⊕ Able to be built with commonly available tools
- ⊕ Relatively fast to build a new image (about 2-3 hours)
- ⊕ All components are rebuildable (packages, kernel, initrd, squashfs)
- ⊕ Binary package distribution
- ⊕ Easy to customize
- ⊕ User-friendly Linux environment for users and administrators
- ⊕ User-friendly way to update to a new image
- ⊕ It's possible to install packages locally to test things (lost after reboot)

3.4.3 Performance

Of course, desktop performance must be better than a thin client. Particularly playing video (from Youtube for instance) should be perfectly doable. Any runaway processes would only affect the local user (unlike with a terminal server where everyone is bothered by such a process). The need for local computing power is limited here at the MI; your environment may differ, prompting other choices to be made.

3.5 Stacking the techniques

3.5.1 Preboot eXecutable Environment (PXE)

PXE, often pronounced 'pixie' (hence the name of the project and the name given to the machines), offers nice features for those administering networks. You can tell the hardware to boot from the network. This is configured in the system BIOS. By enabling PXE/BOOTP in the BIOS the network card is presented as bootable device, an initial loader is loaded in the network card from a ROM on the network card or on more recent hardware from the BIOS itself. This binary broadcasts for a DHCP server. From the DHCP server it receives an IP address, subnet, router, dns, filename and a next server. On the next server it fetches a PXE binary over tftp using the flag 'filename'. This binary is executed like the equivalent of an MBR¹¹. It gets a menu or loader configuration via tftp using pxelinux. There are more implementations of PXE, however pxelinux offers nice menus and other facilities like grub does. From here a kernel and initrd can be chosen and loaded. One of the boot options is to tell the kernel to use an NFS root. So it is crucial that the kernel has builtin support for NFS and the network card present in the system.

3.5.2 Casper over NFS

Casper is a set of files which are added to the initrd. It is used on livecd and readonly images like cdrom, dvd and sometimes usbdrives. It provides functions such as autologin, mounting squash filesystem and a union filesystem¹². It mounts unionfs over the readonly squashfs¹³ image. Unionfs is a writable filesystem. Actually, casper does a lot more, but we removed most of its functions. We use the squashfs and unionfs filesystems. With this functionality we can:

- ⊕ serve one read-only image, avoiding expensive file locking for write permissions
- ⊕ serve an image from one file which fits in memory instead of multiple files from disk
- ⊕ use the unionfs to allow the machines to update state (like /var/run etc) and even install software for testing purposes

3.5.3 Homedirs

Since we are a Unix-centered shop, we have used NFS to offer home directories for a long time. The PXE-booted machines have to be able to mount the NFS shares we use for our home directories and other storage needs (like a scratch dir, a software repository and a place to put websites in). An unfortunate downside to NFS is that security is an issue here. We tried using NFSv4 with Kerberos (which we have also used for a long time), but that meant that some people could not log in to a different account using SSH keys any more, as they would not get the home directory for that account¹⁴. Not using Kerberos here saves us some administration elsewhere, so the dark cloud of lessened security has a silver lining at least.

3.5.4 LDAP and Kerberos

For making accounts known to systems on our network, we use LDAP; for authenticating those accounts we use Kerberos. This means that our machines have to be able to access those servers in question; therefore we have to install config files specifying where those servers can be found.

¹¹The MBR is the Master Boot Record which is stored on the first 512 Kb of a harddisk.

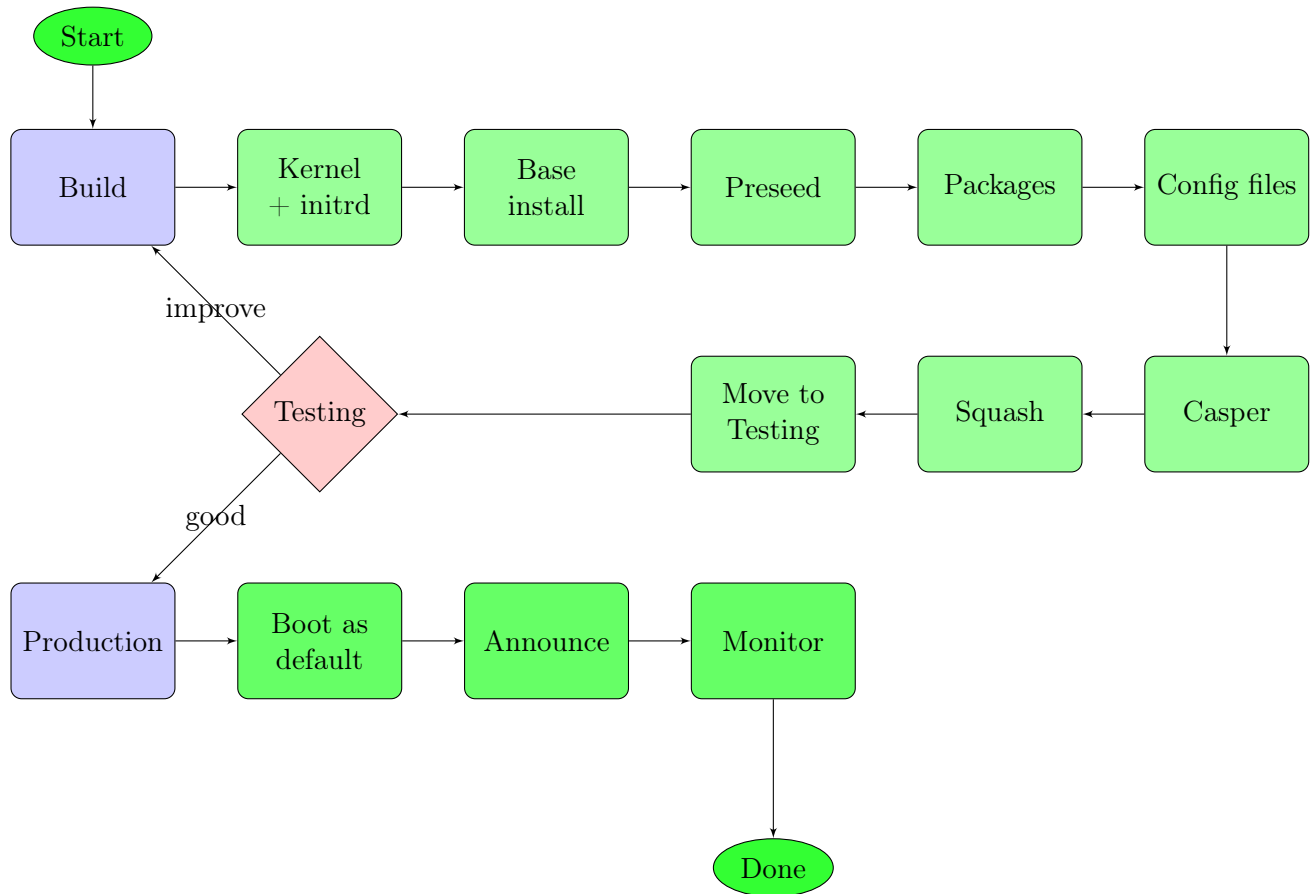
¹²<http://en.wikipedia.org/wiki/UnionFS>

¹³<http://en.wikipedia.org/wiki/SquashFS>

¹⁴If you know a solution to this, please let us know - we couldn't find any

3.6 (Re)Building an image

This chapter shows the steps to build an image for use in pxebooting. There are code samples in the appendix and there is a howto on the Ubuntu wiki¹⁵.



Flow chart of building a Pixie image.

3.6.1 The base install

This could work with a lot of Linux distro's, here we have chosen Ubuntu Karmic (9.10) to work with. At first we need to create a directory for our new image. Then we use the debootstrap command to set up a basic installation. If you build images as often as we do a local mirror can help a lot. It helps both the Ubuntu project and yourself to maintain high performance on the software archives. Install apt-mirror and apache to create your local mirror.

3.6.2 The kernel and initrd

Building a kernel package and an initrd is not very difficult. As said we like to use existing Ubuntu tool sets. We build packages for the kernel, initrd and kernel headers. Once built, there is no need to repeat this step in every build.

¹⁵<https://help.ubuntu.com/community/Desktop/PXE>

3.6.3 Preseed package configuration

Ubuntu has utilities to set configuration options for packages. In this way it's possible to make an installation non-interactive. This is useful for automatic rebuilds like tinderboxes¹⁶. To preseed package settings before installing packages the commands `debconf-set-selections` and `debconf-get-selections` are used. In this way we feed the environment variables to avoid dialogs during the build, and if it's working we add the correct variables to the environment, else we fix it later by adding localized files.

3.6.4 Adding packages

For adding packages there is a similar procedure, using the commands `dpkg --get-selections` and `dpkg --set-selections`. The first command shows the list of installed packages. The second command writes a package list to the package database. Maintenance of the package list happens on a live Pixie and extracting a new list from there or by chrooting to the build tree and extracting a package list from there.

3.6.5 Adding localized files

We did not manage to set everything with configuration tools. We have a set of files, config files mostly, we need to add manually. We keep those files in a subversion repository¹⁷ to keep track on them.

3.6.6 Casper

As said before, we need to remove most files from casper. We maintain a list of files we want to keep and remove the rest. To do this we first extract the `initrd` to a temporary workspace. After the removal the rest is repacked to a new `initrd` and copied to the PXE tree.

3.6.7 Build squashfs image

When the build tree is complete everything gets compressed into a squash file. This file is the largest, in our setup it is over three gigabytes.

3.6.8 Build results

At the end of the build procedure there are five files:

- ⊕ kernel
- ⊕ initrd
- ⊕ filesystem.manifest
- ⊕ filesystem.manifest-desktop
- ⊕ filesystem.squashfs

3.6.9 Staging to production

The five files are copied to the production server. After testing the changes the PXE-menu is changed to add the new image to the production setup. Users are informed through zenity about the update and are requested to reboot into the new image.

¹⁶A tinderbox is "a detective tool for software development, [http://en.wikipedia.org/wiki/Tinderbox_\(software\)](http://en.wikipedia.org/wiki/Tinderbox_(software))

¹⁷<http://en.wikipedia.org/wiki/Subversion>

3.7 Results

3.7.1 Scalability

While deploying the Pixies our fear was a high load on the NFS server. Not scaling the NFS would be a serious setback for the project. We were already re-thinking the design, in mind adding memory, disks and scenarios with ramdrives and more. But all stayed fine after using 20 Pixies from this server. So that meant it did scale, to some extent at least. We found it stayed fine after 40 Pixies, so we speeded up our deployment. In the end, about 80 pixies are deployed running from one server (and that's not a high-end server at all). This server has an average load of 0.01 with peaks of 0.6. If a NFS server is getting to much load, the NFS clients would notice. Since all clients are performing excellent, we conclude that the server is more than capable of handling the load.

3.7.2 User satisfaction

Within days, we received about six spontaneous compliments while walking in the hallways of the building. With our previous thin client environment, we seldom got compliments. We collected some numbers and held a user satisfaction review¹⁸.

3.8 Failsafe and redundancy

In a production environment we always want the availability of a failback. In our case we use the buildhost as a failback. Our file servers and dhcp servers are already redundantly deployed.

3.9 Known issues

- ⊕ PXE fails sometimes,
- ⊕ Some setting somewhere, (we could not find it yet), corrupts the usage of the left and down arrow key. This can be fixed by defining your keyboard in the user environment. It has probably to do with old configuration files in the home directory.
- ⊕ BIOS resets itself sometimes on our chosen hardware.

4 Future Plans

4.1 Ubuntu 10.4

At the moment of development and writing Ubuntu 9.10 was the current version of Ubuntu. As 10.4 was released some time ago we are planning to update to 10.4 as it is a long term support (LTS) version.

4.2 USB drive

Initial work is done on using Ubuntu Netbook Remix and adding our packages to that. The home directories are mountable over sshfs¹⁹. To have our current setup working from a USB-drive an 8 GB drive is required.

4.3 Saving unionfs

One of the features we are thinking of is to store the unionfs layer in the homedirectory of the user. In this way the state of the Pixie can be restored over a reboot, allowing users to install packages which would give our users a more customizable desktop and more flexibility.

¹⁸see: https://projects.math.leidenuniv.nl/trac/howto/wiki/user_satisfaction

¹⁹read more on: https://projects.math.leidenuniv.nl/trac/howto/wiki/netboot_remix

4.4 Http(s) boot

Currently it's possible to boot over the web using http (or iscsi). Work is getting done to make this work over https (secure). As soon as this works we could provide our users with a binary image that boots from a UBS drive or in a virtual machine. Then no confidential file systems need to leave the building and users can work as soon as they have an internet connection. It may be clear that the development is not there yet and may take some time to deliver.

5 Acknowledgements

The authors would like to thank:

- ⊕ Linus Torvald and others for their work on the Linux kernel
- ⊕ Debra and Ian Murdock for starting Debian GNU/Linux and creating a magnificent package manager
- ⊕ Mark Shuttleworth for starting Ubuntu Linux and bringing Linux to the crowd
- ⊕ Xiwen Cheng for his help on tikz to make better charts.
- ⊕ the Mathematical Institute for their trust in us to spend time developing new solutions and allowing us to spend time to write this paper.
- ⊕ our test readers: Maarten Verwijs, Dirk-Willem van Gulik, Finne Boonen, Laura Vrijenhoef and Prof. dr. J.J. Meulman for the comments and corrections on this paper
- ⊕ Haley Davis for publishing the frontpage illustration under a cc-by license

6 Licenses

This document is licensed as CC-By (3.0)²⁰, excepting the university seal which is GNU-FDL²¹, so you are free to:

- ⊕ Share — to copy, distribute and transmit the work
- ⊕ Remix — to adapt the work

The buildscripts²² have MI license²³ which is a two-clause BSD-style license.

²⁰<http://creativecommons.org/licenses/by/3.0/>

²¹http://nl.wikipedia.org/wiki/Bestand:Seal_Leiden_University.jpg

²²<https://wiki.ubuntu.com/MartenVijn?action=AttachFile>

²³https://projects.math.leidenuniv.nl/trac/howto/wiki/mileiden_license

⊕

7 Appendix

7.1 Code examples

7.1.1 Preseed

To preseed environment settings for installing packages we use `debconf-get-selections` and `debconf-set-selections`. To retrieve there setting we use

```
debconf-get-selections | grep foo > preseed.txt
```

To set the environment do:

```
cat preseed.txt | debconf-set-selections
```

7.1.2 Adding packages

Adding packages is a similar procedure. To collect the list of installed packages do:

```
dpkg --get-selections > packages.txt
```

Then later during the build this is done:

```
cat packages.txt | dpkg --set-selections
apt-get update
apt-get -y dselect-upgrade
```

7.1.3 Adding files from subversion to the tree

Check out your svn tree and:

```
cd svn_tree
tar -cvf - --exclude=.svn . | tar -xpf - -C ${DIR}/
```

7.1.4 Casper

As said before, we need to remove most files from casper. We maintain a list of files we want to keep and remove the rest. To do this we first extract the initrd to a temporary workspace. After the removal the rest is repacked to a new initrd

```
mkdir /tmp/casper_tmp
mv ${DIR}/initrd.img ${DIR}/initrd.img_orig
cd /tmp/casper_tmp 7z e -so ${DIR}/initrd.img_orig | cpio -id
for ITEM in `ls /tmp/casper_tmp/scripts/casper-bottom/`
do
    if [ `echo $CASPER_KEEP | grep -c $ITEM` = "1" ]
    then
        echo "Casper keeps $ITEM"
    else
        echo "Casper removes $ITEM"
        rm /tmp/casper_tmp/scripts/casper-bottom/$ITEM
```

```
fi
```

```
done
```

```
find . | cpio -o -H newc | gzip > ${DIR}/initrd.img
```

7.1.5 Build squashfs image

When the new tree is ready to be compressed into a squashfs filesystem:

```
mkdir -p ${PXE_DIR}
```

```
cp ${DIR}/vmlinuz ${PXE_DIR}/vmlinuz
```

```
cp ${DIR}/initrd.img ${PXE_DIR}/initrd.img
```

```
chroot $DIR dpkg-query -W -showformat='${Package} ${Version}\n' > \
```

```
    ${PXE_DIR}/casper/filesystem.manifest
```

```
cp ${PXE_DIR}/casper/filesystem.manifest ${PXE_DIR}/casper/filesystem.manifest-desktop
```

```
for i in $MANIFEST_REMOVE
```

```
do
```

```
    sed -i "${i}/d" ${PXE_DIR}/casper/filesystem.manifest-desktop
```

```
done
```

```
mksquashfs ${DIR} ${PXE_DIR}/casper/filesystem.squashfs
```