Avoiding the One-Off Monster: Reusable System Engineering Techniques

Joe Greco
sol.net Network Services

<jgreco@ns.sol.net>

Abstract

Many mid-sized service providers maintain a number of UNIX-based server platforms on which to run various services. Most of these are created as 'one-off' engineering projects, where an engineer sits down and designs the system in a somewhat arbitrary fashion. While "fast" and "easy" in the short term, such engineering practices may be somewhat short-sighted, and raise future support and upgrade issues.

Concepts presented will be oriented towards reducing unnecessary choices and creating a uniform design paradigm with an emphasis on reproducibility, and will cover hardware selection, hardware use, OS configuration, application installation, security, maintenance, and upgrades.

Most of these techniques are applicable even in an environment of heterogeneous services.

Introduction

At one point or another, most system administrators have been faced with the Unknown Server That Just Broke. Typically, a former employee who is no longer with the company, who used an unfamiliar operating system and ambiguous design goals, set up this server. The hardware may be unusual, the applications a mystery, and the action needed to correct the problem a complete unknown.

Having worked with UNIX based servers for many years, I have frequently run into this situation. Perhaps the worst of all my experiences involved a customer's sickly Squid web-caching cluster, which had gained a reputation with the people supporting it as being a real headache. It was soon obvious why: during its life cycle, different engineers had each put together servers for the cluster. The servers had different types of CPU, different quantities of memory, different hard drive configurations, different SCSI controllers, different network cards, different operating systems, different layouts, different versions of Squid, and in fact shared very few things in common.

Squid is a particularly tunable application, and one that needs to be configured to make best use of available resources. With so much variation in the cluster, and none of the machines truly tuned to the optimum, performance problems were inevitable.

As a small UNIX consulting shop that operates a large number of machines, sol.net had developed a number of system building tools over the years, and had rules-of-thumb for specifying hardware for specific types of server tasks. By generalizing this framework to support a more formal server building procedure for the customer, a new standard was devised that offered a large number of benefits.

The one-off problem is not a problem with a general solution. The focus of this paper is the decision making process that led to one particular solution to this problem. The decisions outlined in this paper may or may not be appropriate in other environments, but the concepts may be helpful in designing similar engineering techniques for other sites.

Other solutions can and do exist, and may work just fine. One common approach is the “Golden Master” approach, where one central repository maintains a filesystem image that is distributed to each server. This suffers from a number of drawbacks: it requires that a large central repository be maintained, that there be sufficient bandwidth to it to do an install, that there be an image for the task at hand, that no crackers break into the repository and as a result install backdoors on all your servers, that the actual servers not have their files set with a “schg” read-only flag if you want to be able to do updates, that the issue of actually setting up an application to begin with must still be handled, etc. Cor Bosman just gave an excellent talk prior to this one on that very subject. However, there are benefits to having a small and portable set of scripts: a floppy and a standard FreeBSD CD are all that is needed in order to install a FreeBSD server using the techniques described here. There is a certain amount of risk with the level of dependence that a Golden Master approach requires.

Hardware Selection

Selecting appropriate hardware for a server is always a tedious task. The numerous questions of “SCSI vs. IDE”, how many drives and of what size, how much memory, what speed CPU, and all sorts of other questions typically cause servers to be built in a one-off fashion. This has negative repercussions, including the inability to economically maintain spare “ready-to-go” standby servers, maintaining a large inventory of spare parts, and significant lead time when ordering new equipment.

As it turns out, a wide variety of server needs can be filled with a smaller selection of standardized hardware. This may not completely eliminate the need for one-off servers, but can result in a significant reduction.

Studying the existing requirements of the customer, it became readily apparent that there were three primary classes of servers: small servers, requiring little in terms of disk, memory, and CPU; midrange servers, requiring good drives, more memory, and a fast CPU; and high-end servers, requiring even more memory and dual fast CPU’s.

In all cases, the servers were standardized to be SCSI, with removable drive modules, a dual Fast Ethernet card, and a floppy. The hardware was selected based on known compatibility with a wide range of operating systems.

By standardizing at this level, numerous design decisions were completely eliminated and compatibility was assured. A number of other benefits became apparent over time, as well: since all the hardware platforms were compatible with each other, upgrading from a slow machine to a faster machine could be trivialized into a chassis swap, unplugging the drives from the old machine and inserting them into a new one; and replacing a failed server hardware platform was simply a matter of having one of each class of server sitting in a staging area, boxed and ready to go.

Due to the nature of the PC hardware market, it was impossible to keep the machines 100% identical, since price fluctuations and vendor availability tended to cause different CPU’s to be selected for each class over time, but this never turned out to be a significant issue. As the earliest mid-range servers aged a year or two, they typically became low-end servers, for example.

OS Selection and Configuration

If hardware selection is troublesome, selecting and configuring an operating system can be even more so. Every engineer has a preferred operating system, and even within a single OS, configuration choices abound. This makes it very difficult to log on to a server designed by another engineer and get meaningful work done quickly.

An important realization is that an operating system is not the end product: an operating system exists in order to run an application platform of some sort. It may be mail, web, news, or something else entirely, but as long as the operating system is capable of running the necessary applications, it should not matter whether Solaris, FreeBSD, Linux, or another operating system is chosen. An engineer’s preference for a given OS should be of relatively little importance, as long as a standardized operating system is sufficient to handle the requirements of all applications.

Once this conclusion is reached, it becomes relatively easy to design a set of tools that allows for the rapid installation of an operating system.

One of the benefits, in hindsight, is that engineers building one-off servers typically take an unreasonably large amount of time to load and configure a system. There is time spent to perform standard configuration, tuning the system for the nonstandard hardware platform being used, installation of drivers for that funny network card and SCSI card, installation of favored packaged software, security audits and securing the system, etc. The end result is frequently quite different from system to system – even with the same engineer and the same hardware!

By having a scripted installation system, even a junior engineer can be delegated the task of loading up the OS for a new server, and it will come out the same every time.

Some issues to consider:

Separate disks for the base operating system and an application package allow either portion to be upgraded by simply replacing a disk, without interfering with the other portion. In fact, it becomes easier to upgrade operating systems by replacing the OS disk than it is to upgrade it "in-place".

That decision allows the OS disk layout to be standardized and minimized – no source distributions required, and a 9GB disk becomes more than sufficient to handle an entire OS, plus swap and /var for almost any application.

Since the OS is being used as an application-running platform, there is no need for application-specific files to be stored on the OS drive, aside from very minimal hooks from rc.d and cron.

Host-specific configuration can be scripted separately, such that a cluster of machines may have OS drives dd'ed from an installed image, and a minimalist host configuration can be done per-machine.

Common software packages (for example, *top*, in older FreeBSD releases) can be built in as part of the base OS install process.

A default kernel configuration should also be created and made available, with broad application compatibility and liberally tuned values. More specialized kernel configurations can be specified as part of the application install process, if needed.

Application Installation

To minimize the intermingling of OS and application, the use of a separate directory tree (for example, /www, /mail, /news) allows the application to be placed on a separate filesystem and hard disk. Causing applications to install in an alternate directory is generally not too difficult, although this too can and should be scripted, to avoid any unusual variations in things like configure incantations. At the same time, an application installation script should install minimal appropriate hooks into the operating system in order to allow proper booting and maintenance.

In this area, some experience yields interesting ideas. Many servers contain large numbers of application files, and fsck may take a long period of time. By specifying ,noauto in fstab, the system fsck will only check the base OS files, and the system will continue booting. In particular, networking will be brought up, and then the application rc.d script may fsck the application filesystem(s) and then launch the application itself. The result is that users who are trying to connect to the server receive a "Connection Refused" message much more rapidly, and may have a chance to connect to a different machine in a cluster environment with round-robin DNS.

This method has worked well, and our startup script specifies "fsck –y || fsck –y", because it is more useful to have the machine up and running with some possibly missing application files than it is to have it unreachable via the net.

Security

Many OS installs fail to be secure in numerous ways. Scripting allows for the deployment of a consistently secure base OS, assuming the installation scripts put sufficient effort into the matter.

As administrators, most of us have deployed servers without an onboard firewall, or without auditing SUID/SGID files on the OS platform. It is easy to overlook, and when in a rush, almost always forgotten. By building these into the OS load process, the creation of a more secure application platform is guaranteed.

Because our OS load is standardized, the location of all SUID/SGID files can be easily coded into a script that removes generally useless bits. After all, even on a SMTP server, as long as you don't have shell users, even Sendmail does not need to have a SUID bit on it – it is started as root to begin with. Certainly there is little use for UUCP on most servers.

Additionally, running a server in BSD securelevel = 3 and setting all binaries and important configuration files with the "schg" flag adds an additional layer of difficulty for any would-be cracker. The downside that it makes it more difficult for the administrator to change system files does exist, but experience suggests that if a significant security hole exists, it may be easier to simply reload and distribute a secured version of the OS.

Creating a packet firewall for each server can also be a tedious task, and therefore one many administrators skip. However, the creation of a basic template firewall, using shell variables for interface addresses and ports to allow, is not particularly difficult. Our local version has grown exceedingly detailed, capable of automatically detecting the network configuration of the server, generating appropriate rules, and allowing a trivial amount of configuration data at the top to control the generation of hundreds of rules.

Installing applications in a minimalist chrooted or jailed environment helps to provide the application with additional security as well.

Maintenance

The maintenance of systems built through scripted installs can be accomplished in a number of ways.

Upgrading software is not difficult. Thanks to the standardized "API" between the base OS and the application platform, simply installing a new OS or new application disk is among the easiest ways, requiring very minimal downtime. However, any large number of machines is best managed via serial console, and this allows for remote maintenance through the distribution of changes that can be applied in single-user mode. Sites that do not run with securemode enabled can, of course, simply update running system files.

Repairing defective hardware ceases to be an exercise in trying to locate the bad module in between crashes. A wholesale chassis swap allows the existing OS and application to be transferred to a new computing platform, and the old unit can then undergo a very thorough hardware debugging process.

Hardware upgrades are just as easy. To move up to a faster, larger server is merely a matter of moving the existing OS and application disks to a new chassis. The old chassis becomes immediately available for some new application or use.

Conclusion

Any site maintaining more than a handful of servers may be able to derive some additional efficiency by considering reusable system engineering techniques. A well-planned scheme will allow for easier deployment and faster maintenance, while reducing the long-term engineering resources required.