Using a the Xen Hypervisor to Supercharge OS Deployment

Mike D. Day
International Business Machines
ncmike@us.ibm.com

Ryan Harper
International Business Machines
ryanh@us.ibm.com

Michael Hohnbaum
International Business Machines
hohnbaum@us.ibm.com

Anthony Liguori
International Business Machines
aliguori@us.ibm.com

Andrew Theurer
International Business Machines
habanero@us.ibm.com

Abstract

Hypervisor technology presents some promising opportunities for optimizing Linux deployment. By isolating a server's unique properties into a set of patches to initialization scripts and other selected files, deployment of a new server will be demonstrated to occur in a few seconds by creating a new Xen domain, re-using an existing file system image and applying patches to it during domain initialization. To capture changes to a server's configuration that occur while it is running, the paper discusses the potential of copy-on-write file systems to hold changes to selected files. By separating the initialization and file data that make a linux server instance unique, that data can be stored and retrieved in a number of ways. The paper demonstrates how to store and retrieve different initialization patches over the network and integrate these capabilities into the Xen tools. Potential uses for the techniques demonstrated in the paper include capacity on demand, and new methods of provisioning servers and workstations.

1 Introduction

Virtual machine technology is rapidly becoming ubiquitous for commodity processors. Commercial product has established a foothold in this space. Open Source products are emerging and maturing at a rapid pace. This paper demonstrates how the use of virtualization technology can improve deployment and maintenance of Linux servers.

The virtualization technology used for this paper is Xen, an open source hypervisor developed at the University of Cambridge[1]. Xen supports para-virtualized guests, that is operating systems are modified to run in domains on top of Xen.

[1] http://www.cl.cam.ac.uk/Research/SRG/netos/xen/

All I/O devices are owned by one or more privileged domains. Typically the first domain to be created (called domain 0), but other domains may have control over one or more I/O device. The privileged domain runs a kernel that is configured with regular device drivers. The privileged domain initializes and services I/O hardware.

Block, network, and USB devices are virtualized by the privileged domain. Backend device drivers run in the privilege domain to provide a bridge between the physical device and the user domains. Front end virtual device drivers execute in user domains and appear to Linux as a regular device driver.

While Xen includes management and control tools (`xend` and others), an alternate toolset, vmtools[2], is used for the work discussed in this paper. vmtools is a re-implementation in "C" of the Xen toolset, which is implemented in python. vmtools provides the capabilities needed to configure domains.

`vmtools` consists of a daemon, `xenctld`; a set of command line tools, `vm-*`; and `vmm`—a script that provides a more user-friendly frontend to the vmtools. vmtools provides commands for creating a domain, assigning resources to the domain, starting and stopping a domain, querying information about domains. The tools are modular, provide ease of use within scripts, and are easy to modify and extend.

vmtools are used to demonstrate the flexibility of the Xen architecture by showing it can be controlled by multiple toolsets, and also as a vehicle for extending the Xen configuration syntax[3].

[2] `http://www.cs.utexas.edu/users/aliguori/vm-tools-0.0.9a.tar.gz`

[3] `http://www.cl.cam.ac.uk/Research/SRG/netos/xen/readmes/user/user.html`

2 DEPLOYMENT OVERVIEW

Deployment is the provisioning of a new operating system and associated configuration for a unique instance of a computer system. Throughout this paper a unique instance of an operating system is referred to as a system image. Traditionally, each computer system has one system image deployed on it. With virtualization technology, each computer system may have one to many system images deployed, each executing within its own virtual machine environment.

Related to deployment is maintenance. After a system image is established, it must be maintained. Software components must be updated (for example, replaced with new versions) to address security problems, provide new funtionality, or correct problems with existing software. Sometimes this involves replacing one component, a subset of the overall software components, or a complete replacement of all operating system software. Similarly, application software and middleware needs to be maintained.

Data centers have numerous computer systems, and numerous system images. To keep things manageable, most datacenters strive to keep system images as common as possible. Thus, it is common practice to choose one specific version of an operating system and deploy that on all (or a large percentage of) the system images.

2.1 Deployment Tasks

Deploying a new system image involves:

- Configuring the physical (or virtual) machine, such as processor count, physical memory, I/O devices

- Installing the operating system software, such as kernel configuration (smp vs up, highmem, and so on), device drivers, shells, tools, documentation, and so on.
- Configuring the operating system (such as hostname, network parameters, security, and so on).
- Creating user accounts
- Installing application software
- Configuring application environment

2.2 Current Deployment Methods

There are different ways to deploy multiple copies of the same system. These include manual deployment, use of a higher-level installation tool for example kickstart, and installation customization then cloning.

2.2.1 Manual

The most basic mechanism is to do a manual install from the same installation media to each system image. This method is time consuming and can be error prone (as the system administrator must execute a series of steps and with repetition is inclined to miss a step or make a subtle variation in the process that can have unforseen consequences).

2.2.2 Kickstart

Kickstart[4] is a tool provided by Red Hat that enables repeating a system install with identical parameters. In effect, all questions that are normally asked by the system installer are answered in advanced and saved in a configuration file. Thus, identical system images may be installed on multiple machines with reasonable automation.

2.2.3 YaST Auto Installer

AutoYaST[5] functions according to the same principal as Kickstart. Configuration and deployment of the platform is driven by a configuration file, and the process can be repeated (with configuration changes) for multiple deployments.

2.2.4 Clone/Customize

Another install method is to clone an installed system and customize the resulting system image. In many cases a clone operation, which consists of copying the contents of the original installed root file system, is quicker than going through the complete install process. After the clone operation, system image specific customization is then performed. For example, setting hostname.

3 IMPROVEMENTS AVAILABLE THROUGH VIRTUALIZATION

Virtualization technology provides opportunities to improve deployment mechanisms. Improved aspects of deployment include:

- normalization of hardware configuration
- dynamic control over hardware configuration

[4] `http://www.redhat.com/docs/manuals/linux/RHL-9-Manual/custom-guide/part-install-info.html`.

[5] `http://yast.suse.com/autoinstall/ref.html`.

- omission of hardware discovery and probing
- use of virtual block devices (VBD)
- file system reuse
- virtual networking (VLAN)

3.1 Dynamic Control Of Hardware Configuration

Without virtualization, changing the number of CPUs available, the amount of physical memory, or the types and quantity of devices requires modifying the physical platform. Typically this requires shutting down the system, modifying the hardware resources, then restarting the system. (It may also involve rebuilding the kernel.)

Using virtualization, resources can be modified through software control. This makes it possible to take disparate hardware, and still create normalized virtual machine configurations, without having to physically reconfigure the machine. Further, it provides the capability of redefining virtual machines with more resources available to address capacity issues.

For example, Xen allows you to add and remove processors, network, and block devices from and to user domains by editing a configuration file and running a command-line utility. No kernel configuration is necessary, and you don't need to shut down the physical computer. This operation can be repeated as often as necessary.

In addition to the advantages in deploying and maintaining Linux systems, dynamic hardware configuration makes more advanced workload management applications easier to implement.

3.2 Virtual Block Devices

Xen privileged domains virtualize block devices by exporting virtual block devices (VBD) to domU's. Any block device accessible by Linux can be exported as a VBD. As part of the process of setting up a VBD, the system administrator specifies the device path the VBD should appear to the domU as. For example `/dev/sda1` or `/dev/hda5`. Disk partitions may be exported this way, or a VBD may be backed by a file in dom0's file system.

Virtual block devices provide two benefits to deployment and maintenance of Linux servers. First, they provide hardware normalization as described above. (Every domain can have an identical fstab, for example). Secondly, VBDs make the reuse of file systems with Xen domains exceedingly simple, even for read/write file systems.

3.3 Virtual Networking

Xen privileged domains virtualize network devices in a manner similar to VDBs. The privileged domain kernel initializes network interfaces and starts networking services just as a normal kernel does. In addition, Xen privileged domains implement a virtual LAN and use the Xen network back end (netback) driver to export virtual network interfaces to user domains.

User domains import virtualized network interfaces as "devices," usually `eth0...ethN`. The virtualized eth0, for example, is really a stub that uses Xen inter-domain communication channels to communicate with the netback driver running in a privileged domain. Finally, the Xen privileged domain bridges virtualized network interfaces to the physical network using standard Linux bridge tools.

The most common practice is to use private IP addresses for all the virtual network interfaces

and then bridge them to a physical network interface that is forwarded using Network Address Translation (NAT) to the "real world."

A significant benefit of this method for deployment and maintenance of servers is that every server can have identical network configurations. For example, every user domain can have the same number of network interfaces and can use the same IP configuration for each interface. Each server can use the bridging and NAT forwarding services of the privileged domain to hide their private addresses. Note that bridging without NAT is also a common practice, and allows user domains to host externally visible network interfaces.

3.4 File System Reuse

Xen's Virtual Machine technology can export file systems and file images to virtual machines as devices. Sharing file systems among Linux platforms is a time-honored technique for deploying Linux servers, and virtual machine technology simplifies the sharing of file systems.

File System reuse is an especially helpful technique for deploying and maintaining Linux systems. The vast majority of the time spent deploying a new Linux system is spent creating and populating the file systems.

Re-using read-only file systems is exceedingly simple in Xen. All you have to do is export the file system as a device to Xen. For example, the line `disk = ['file:/var/images/xen_usr,sda1,r']` causes the file system image `/var/images/xen_usr` to be exported to the user domain as `/dev/sda1`. (All configuration commands are relative to the privilged domain's view of the world.) Because this is a read-only file system you don't need to do anything special to synchronize access among domians.

In addition to file system images, the Xen domain configuration syntax allows you to export both physical devices and network file systems as devices into the new domain. A future version of Xen will the exporting of a VFS directory tree to a Xen domain as a device.

Read/write file systems are not as easy to share among domains because write access must be synchronized among domains. There are at least three ways to do this:

- Use a storage server that provides external, sychronized shared storage. There is a range of systems that have this capability.
- Use a copy-on-write file system. One such file system is unionfs.[6]
- "Fork" an existing file system by duplicating it for each new domain. This is a simple and expedient (if not efficient) way to re-use read-write file systems.

The Logical Volume Manager (LVM)[7] has an interesting snapshot capability that was designed primarily to support hot backups of file systems, but which could evolve into a copy-on-write file system appropriate for use with Xen.

One problem with re-use of read-write file systems is that they usually contain configuration files that are specific to an individual Linux system. For example, `/etc` on a Linux system contains most of the uniqueness of a system. If you are going to re-use an `/etc` file system, you need an automated way to "fork" and modify it. Fortunately the typical system does not

[6] `http://www.fsl.cs.sunysb.edu/project-unionfs.html`

[7] `http://www.tldp.org/HOWTO/LVM-HOWTO/index.html`.

need a vast number of changes in `/etc` and as a result it is possible to automate the "forking" process. Later this paper discusses some tools we have developed to automate the creation, modification, and exporting of file systems under Xen.

4 EXPLOITING XEN TO DEPLOY SERVERS

A variation of the clone and modify approach is proposed to deploy Linux on Xen virtual machines. In addition, an extended configuration syntax and Xen deployment tool is proposed to integrate the deployment process with Xen domain creation. This approach uses Xen to improve on the existing clone methods in the following ways:

- Xen allows exporting of VBDs to domains, where they appear as virtual devices, such as SCSI or IDE drives. This is an improvement over cloning a file system image to a bare-metal server.
- Xen allows the exporting of NFS volumes as virtual devices. This privides a file system with some of the same advantages as VBDs.
- Xen allows provides control over the "hardware" environment of each new server. By exporting specific devices to the new domain, it is not necessary to accomodate all the possible hardware configurations when deploying a new server. For example, all domains within an organization may appear to have only SCSI block devices, despite variation in the underlying physical hardware.

4.1 Deploying Application Stacks

The flexibility to export specific file systems to the new partitions means that it is much easier to deploy new servers for specific applications. For example, a file system image can be prepared with a complete DBMS stack. When a new data base server is needed, a Xen domain can be created using the DBMS file system images. In this case, Xen can export the DBMS image to the new domain. The new domain can and mount the image read-only as `/opt/dbms/`. Exporting of pre-built file systems as virtual devices to Xen domains simplifies the deployment of application-specific servers.

4.2 Xen Deployment Methodology

The general methodology used is to create a handful of "canned" file systems that can be mixed-and-matched to create new Xen domains by exporting them as VDBs or NFS mounts. For example, `/usr` and `/bin` as standard read-only file systems; `/etc` as a read/write file system that needs to be preprocessed; `/var/` and `/home` as read-write file systems that need COW or snapshot capability; Variations of `/opt` for specific application stacks, and so on.

Extending vmtools to support integrated deployment and domain creation requires some new configuration properties for domains, as well as some shell scripts to perform preprocessing on the images to customize them (when necessary) for each domain.

The "Xen Domain Container" is comprised of the following:

- An overall configuration file for the new domain. This is an extended version of

the existing domain configuration file used by the `vmm` command. The extensions include information about the domain's VDB or NFS file systems and how they should be processed by vmtools prior to domain creation. The extended-syntax configuration file is called a "container file."

- File System Images. Each image consists of a file system stored in a compressed cpio archive (just as initrd). In addition, each file system image has metadata in the container file for the file system and processing instructions for vmtools. The metadata and processing instructions describe characteristics of the file system including where it should be mounted by the new domain, whether it should be read-only or read-write, and how it needs to be customized for each new domain.

  For example, a file system that is to be mounted by the new domain as `/etc` needs to be customized for each new domain. The `/etc` file system includes the sysinit data and configuration files, plus user and group accounts, file system mounting, hostnames, terminal configuration, etc.

- Init hooks. Each file system can include shell scripts that will be driven by a configuration file, also in that file system. The idea is to have vmtools preprocess the file system, then mount it on a device (or export it using NFS). During domain startup, the initrd/init process looks for a "post processing" shell script and executes the script on the mounted file system. Depending upon the context of the init process, it may remount file systems and execute a pivot-root and restart the init process.

4.3 Composing a Xen Container

An important goal is to make composing and maintaining Xen domain containers as simple as possible. The container file may contain standard Xen domain configuration statements in addition to "container" syntax. Both types of statements (standard Xen configuration and container) may be intermixed throughout the file.

The container syntax refers to file system images using URIs. Each URI may point to a file system image stored locally, as in `file:///var/images/etc.cpio.gz`; or remotely, as in `http://foo.org/images/etc.cpio.gz`. This reference syntax has two important advantages:

- Simplification of file system deployment. Using a URI reference for each file system image allows the administrator to keep a canonical image on a network server. When starting the domain, vmtools will follow the URI and download the file system and perform pre-processing on a local copy. The tools follow this process for each URI reference configured for use by the domain.

- Simplification of file system maintenenace. For read-only file systems that contain applications, such as `/bin`, `/sbin`, and `/usr`, applying updates and patches comprise a large percentage of the administrator's time. The URI reference allows the administrator to patch or update the cononical, network-resident file system image. Domains can be configured to retrieve their file system images every time they start. A more advanced design would provide a way for the domain initialization to check for recent updates to its file system images.

4.3.1 Domain Customization

Domain customization involves applying modifications to Linux configuration files residing within a file system image. After retrieving a file system image, vmtools can mount and modify the image before starting the Xen domain.

The container syntax provides three different methods for modifying files within a file system image:

- File replacement. This mechanism causes vmtools to replace the content of a file with text embedded in the configuration file itself. The conainter syntax for file replacement is shown in Figure 1.

  This simple expression in Figure 1 will cause vmtools to retrieve the file system archive at `http://images.xen.foo.org/etc.cpio.gz`, expand the archive, and replace the file /etc/HOSTNAME with a new file. The new file will contain a single line, "FCDOBBS." If `/etc/HOSTNAME` does not exist, it will be created.

  There are additional options in the file replacement syntax to create a patch file by comparing the original and modified file systems, and to "fork" the archive by creating a new copy (with the modifications)

  vmtools makes some simple attempts to be efficient. It will only retrieve and expand file system image once per invocation. Thereafter, it will use a locally expanded copy. The creator of the container file can order expressions so that the file system is forked only after it has been completely processed.

  The remaining methods for modifying files follow the same patterns as the replacement method.

- File copy. This mechanism causes vmtools to retrieve a file and copy the retrieved file over an existing file.

- File system patching. This mechanism retrieves a patch and then applies the patch to the file system.

4.3.2 Steps to Compose a Xen "Container"

Composing a Xen container, then, involves:

- Preparing file system images. This step only needs to be performed initially, after which you can use the same file system images repeatedly to deploy further Linux domains. The tools discussed in this paper provide commands that automate file system preparation. (Remember, a file system image is simply a compressed cpio archive).

- Creating the container file. The container file defines the new domain, including the location of the kernel, the amount of memory, the number of virtual processors, virtual block devices, virtual ethernets, and so on. The proposed container expressions prepare, retrieve, and process file system images for use by the new domain.

All information describing the domain is present in the container file: resources, devices, kernel, and references to file systems. Further, the container file includes processing instructions for each file system, with the ability to retrieve updated file systems whenever the domain is started. This collection of information is referred to as a "domain container" because it is self-contained and portable from one xen platform to another.

At the present time one container file must be created for each domain. However, because

CONTAINER SYNTAX FOR FILE REPLACEMENT

```
[replace /etc/HOSTNAME
 archive http://foo.org/images/etc.cpio.gz

 FCDOBBS

] [end]
```

Figure 1: Container Syntax for File Replacement. This simple example shows the `/etc/HOSTNAME` file being replaced with one text line containing "FCDOBBS."

most of the configuration syntax (including the extensions we propose) is boilerplate, there are improvements which will allow reuse of a container template to control the deployment and maintenance of multiple domains.

To complete the deployment, you must process the domain container using `vm-container`, as shown in Figure 2. This example is assumed to be running as a user process in the Xen Domain0 virtual machine. Domain0 is always the first domain to run on a Xen platform, and it is created implicitly by Xen at boot time.

The command in Figure 2 parses the container file `my-domain` and processes all the extended-syntax expressions within that file. It also produces the standard Xen configuration file `my-config`. Output is logged to `/var/log/domain`, and `/var/images` is used as the working directory for processing file system images.

At this point all that's left is to start the domain using `vmm create my-config`.

4.4 Xen Container Syntax

The Xen container syntax is a superset of "standard" Xen configuration syntax. Using the standard Xen syntax you can define the domain boot kernel and boot parameters, the amount of memory to allocate for the domain, which network and disk devices to virtualize, and more. The expressions discussed below are *in addition* to the standard Xen syntax and both types of expressions may be mingled in the same container file.

The Xen container syntax will expand as further experience using it to deploy Linux systems is gained. The syntax is presently complete enough to manage the creation, deployment, and maintenance of Xen domains, including the composition and reuse of file system images.

The Xen container syntax is explained below using examples. In actual use, the container file will have a number of container expressions. The `vm-container` parser only makes one pass through the container file and it processes each expression in the order it is declared within the file. Dependent expressions, such as a `populate` expression which refers to an archive instantiated by a `create` expression, must be in the correct order.

4.5 Creating a File System Image

A file system image for a Xen container can be created from any existing file system. For example, the expression

PROCESSING THE DOMAIN CONTAINER

```
vm-container --container my-domain \
--stripped my-config --log /var/log/domain \
--dir /var/images
```

Figure 2: Processing the Domain Container

```
[create
 /etc/
ftp://foo.org/images/etc.cpio.gz
][end]
```

will create a compressed `cpio` archive out of the contents of the local `/etc/` directory tree. It will then store that archive using `ftp` to the URI `ftp://foo.org/images/etc.cpio.gz`

4.6 Creating a Sparse File System

Loopback devices are especially convenient to use with Xen. The `image` expression will cause `vm-container` to create a sparse file system image, formatted as an `ext3` volume.

```
[image /var/images/fido-etc
 50MB
fido_etc] [end]
```

This example will cause `vm-container` to create a sparse 50 MB file system image at `/var/images/fido-etc`. The file system will be formatted and labelled as `fido-etc`.

4.7 Populating a File System Image

Any type of volume (LVM, NFS, loopback, or physical device) exported to a Xen domain needs to have a formatted file system and be populated with files. The `populate` expression will make it happen.

```
[populate image
/var/images/fido-etc
/mnt/
ftp://foo.org/images/etc.cpio.gz
][end]
```

The example above will cause `vm-container` to mount the file system `/var/images/fido-etc` to `/mnt` using a loopback device. It will then retrieve the archive `ftp://foo.org/images/etc.cpio.gz`, expand the archive into `/mnt`, `sync`, `umount`, and delete the loop device.

4.8 Replacing and Copying

Figure 1 shows an example of replacing a specific file within a file system. The `replace` expression can also be used to generate a `diff` file that isolates the modifications made to the file system. It can also create an new file system archive based on the modifications.

The `copy` expression is almost identical to `replace`, except that it retrieves whole files using URI references and copies those file into the file system being modified. It also supports patch generation and forking.

4.9 Patching a File System

The `replace` and `copy` expressions can both generate a patch file that isolates modifications

to a file system. Once that patch file is created, you can use it repeatedly to modify file systems during domain initialization.

```
[patch
file://var/ images/fido-etc
ftp://foo.org/ images/fido-etc.patch 1
file:///var/ images/fido-etc-patched
][end]
```

This example will retrieve a patch file from `ftp://foo.org/images/fido-etc.patch1`. It will then expand and patch the file system image at `/var/images/fido-etc`. It will then "fork" the file system by saving a patched archive at `file:///var/images/fido-etc-patched`.

4.10 Forking File Systems

While each of the `replace`, `copy`, and `patch` expressions will "fork" the file system, doing so should only occur after that file system had undergone all the modifications indicated by the container file. The statement that causes the file system to be copied and stored is always optional.

5 Further Work

The notion of using a hypervisor to supercharge OS deployment is valuable and warrants further development effort. In particular, the integration of file system image customization with Xen management and control tools proved very successful. The concept of capturing the unique personality of a domain as a set of changes to file system images was straightforward and familiar, and it worked as expected. A number of files were successfully patched during domain initialization, including the `/etc/passwd`, `/etc/shadow`, and `/etc/groups`. These last three examples show how user accounts and group member can be modified during domain initialization.

Patching user accounts and authorization data during domain initialization is dangerous, especially since our tools retrieved patchfiles over the network. High on the list of further work is generation and verification of cryptographic signatures for all file system images and difference files. It would also be prudent to generate and verify signatures for the extended configuration file.

While modifying file systems during domain initialization from Domain 0's userspace was very reliable, mixed success was achieved when modifying file systems during the kernel init process. Sometimes patches were successful but usually the patches failed or the init process died and was respawned. Continued experimentation with the init process as a vehicle for domain customization is warrented.

5.0.1 LVM

LVM has great potential to augment the approach to domain deployment. In fact, it is already a great tool for use with virtual machines. The LVM snapshot capability, while design for hot backups, works as a COW file system but needs to be evaluated further with this particular use model in mind.

6 Legal Statement

This work represents the views of the authors and does not necessarily represent the view of IBM.

Proceedings of the Linux Symposium

Volume One

July 20nd–23th, 2005
Ottawa, Ontario
Canada

Conference Organizers

Andrew J. Hutton, *Steamballoon, Inc.*
C. Craig Ross, *Linux Symposium*
Stephanie Donovan, *Linux Symposium*

Review Committee

Gerrit Huizenga, *IBM*
Matthew Wilcox, *HP*
Dirk Hohndel, *Intel*
Val Henson, *Sun Microsystems*
Jamal Hadi Salimi, *Znyx*
Matt Domsch, *Dell*
Andrew Hutton, *Steamballoon, Inc.*

Proceedings Formatting Team

John W. Lockhart, *Red Hat, Inc.*