Virtualization in Linux

Kirill Kolyshkin <kir@openvz.org>

September 1, 2006

Abstract

Three main virtualization approaches — emulation, paravirtualization, and operating system-level virtualization — are covered, followed by the implementation examples, comparison of the technologies and their applications. OS-level virtualization is described in detail, with examples from OpenVZ. The main kernel components (isolation and virtualization, resource management, checkpointing and live migration) and user-level tools are also explained. Typical usage scenarios of virtualization solutions are presented.

1 Virtualization. Types of virtualization.

In the context of this report, *virtualization* is a system or a method of dividing computer resources into multiple isolated environments. It is possible to distinguish four types of such virtualization: emulation, paravirtualization, operating system-level virtualization, and multiserver (cluster) virtualization. The last approach is out of scope of this report.

Each virtualization type has its pros and cons that condition its appropriate applications.

Emulation makes it possible to run any non-modified operating system which supports the platform being emulated. Implementations in this category range from pure emulators (like *Bochs*) to solutions which let some code to be executed on the CPU natively, in order to increase performance. The main disadvantages of emulation are low performance and low density. Examples: *VMware* products, *QEmu*, *Bochs*, *Parallels*.

Paravirtualization is a technique to run multiple modified OSs on top of a thin layer called a hypervisor, or virtual machine monitor. Paravirtualization has better performance compared to emulation, but the disadvantage is that the "guest" OS needs to be modified. Examples: *Xen*, *UML*.

Operating system-level virtualization enables multiple isolated execution environments within a single operating system kernel. It has the best possible (i. e. close to native) performance and density, and features dynamic resource management. On the other hand, this technology does not allow to run different kernels from different OSs at the same time. Examples: *FreeBSD Jail*, *Solaris Zones/Containers*, *Linux-VServer*, *OpenVZ* and *Virtuozzo*.

2 Concept of a VE

Virtual Environment (*VE*, also known as *VPS*, *container*, *partition* etc.) is an isolated program execution environment, which (from the point of view of its owner) looks and feels like a separate physical server.

A VE has its own set of processes starting from `init`, file system, users (including `root`), network interfaces with IP addresses, routing tables, firewall rules (`netfilter/iptables`), etc.

Multiple VEs co-exist within a single physical server. Different VEs can run different Linux distributions, but all VEs operate under the same kernel.

3 OpenVZ kernel

The OpenVZ kernel is a modified Linux kernel which adds the following functionality: virtualization and isolation of various subsystems, resource management, and checkpointing.

Virtualization and isolation enables many virtual environments within a single kernel.

Resource management subsystem limits (and in some cases guarantees) resources such as CPU, RAM, and disk space on a per-VE basis.

Checkpointing — a process of "freezing" a VE, saving its complete state to a disk file, with the ability to "unfreeze" that state later.

These components are described below.

3.1 Virtualization and isolation

Each VE has its own set of resources provided by the operating system kernel. Inside the kernel, those resources are either virtualized or isolated. Each VE has its own set of objects, such as the ones described below.

Files – System libraries, applications, virtualized /proc and /sys, virtualized locks, etc.

Users and groups – Each VE has its own root user, as well as other users and groups.

Process tree – A VE sees only its own set of processes, starting from `init`. PIDs are virtualized, so that the `init` PID is 1 as it should be.

Network – Virtual network device, which allows the VE to have its own IP addresses, as well as a set of netfilter (iptables) and routing rules.

Devices – Some devices are virtualized. In addition, if there is a need, any VE can be granted (an exclusive) access to real devices like network interfaces, serial ports, disk partitions, etc.

IPC objects – Shared memory, semaphores, and messages.

3.2 Resource management

Resource management is of paramount importance for operating system-level virtualization solutions, because there is a finite set of resources within a single kernel that are shared among multiple Virtual Environments. All those resources need to be controlled in a way that lets many VEs co-exist on a single system, and not influence each other.

The OpenVZ resource management subsystem consists of three components:

1. **Two-level disk quota** – The OpenVZ server administrator can set up per-VE disk quotas in terms of disk space and number of inodes. This is the first level of disk quota.

   The second level of disk quota lets the VE administrator (VE `root`) use standard UNIX quota tools to set up per-user and per-group disk quotas.

2. **"Fair" CPU scheduler** – The OpenVZ CPU scheduler is also two-level. On the first level it decides which VE to give the time slice to, taking into account the VE's CPU priority and limit settings. On the second level, the standard Linux scheduler decides which process in the VE to give the time slice to, using standard process priorities.

3. **User Beancounters** – This is a set of per-VE counters, limits, and guarantees. There is a set of about 20 parameters which are carefully chosen to cover all the aspects of VE operation, so no single VE can abuse any resource which is limited for the whole computer and thus cause harm to other VEs.

   The resources accounted and controlled are mainly memory and various in-kernel objects such as IPC shared memory segments, network buffers etc.

3.3 Checkpointing and live migration

Checkpointing allows the "live" migration of a VE to another physical server. The VE is "frozen" and its complete state is saved to a disk file. This file can then be transferred to another machine and the VE can be "unfrozen" (restored) there. The whole process takes a few seconds, and from the client's point of view it looks not like a downtime, but rather a delay in processing, since the established network connections are also migrated.

4 OpenVZ utilities

4.1 vzctl

OpenVZ comes with a `vzctl` utility, which implements a high-level command-line interface to manage Virtual Environments. For example, to create and start a new VE it takes just two commands — `vzctl create` and `vzctl start`.

`vzctl set` command is used to change various VE parameters. Note that all the resources (for example, VE virtual memory size) can be changed during runtime. This is usually impossible with other virtualization technologies, like emulation or paravirtualization.

4.2 Templates and vzpkg

Templates are existing images used to create a new VE. A *template* is a set of packages, and *a template cache* is an archive (tarball) of a `chroot`ed environment with those packages installed. During the `vzctl create` stage, this tarball is unpacked. Using a template cache technique, a new VE can be created in seconds, thus enabling fast deployment scenarios.

Vzpkg tools is a set of tools to facilitate template cache creation. It currently supports `rpm` and `yum`-based repositories. For example, to create a template of Fedora Core 5 distribution, one needs to specify a set of (yum) repositories which have FC5 packages, and a set of packages to be installed. In addition, pre- and post-install scripts can be employed to further optimize/modify a template cache. All the above data (repositories, lists of packages, scripts, GPG keys, etc.) form *template metadata*.

With template metadata, a template cache can be created automatically by the `vzpkgcache` utility. It will download and install the listed packages into a temporary VE, and pack the result as a template cache. Template caches for non-RPM distributions can be created as well, although this is more of a manual process.

5 Usage scenarios

The following usage scenarios are common for all virtualization technologies. However, a unique feature of OS-level virtualization like OpenVZ is that one does not have to pay a big virtualization overhead, which makes those scenarios more appealing.

Server consolidation allows an organization to decrease the number of physical servers, by moving their applications into virtual environments; the number of operating system environments remains the same. This leads to savings in hardware costs, rack space, electricity, and management efforts.

Security can be improved drastically by putting each network service (like web server, mail server, etc.) into a separate isolated virtual environment. In the case of a security hole in one of the applications, all the others are not affected. The ability to dynamically manage resources and perform a live migration of applications is an added bonus.

Hosting – Apparently, OS-level virtualization is the only type of virtualization which service providers can afford and use to sell cheap VEs to their customers. Note that each VE has full root access, meaning the VE owner can re-install anything, and even use such things as IP tables (firewall rules).

Software development and testing – Usually developers and testers need access to a handful of Linux distributions, and they need to re-install those from scratch very often. Using virtualization, developers can operate multiple distributions using a single server, without any need to re-boot, with native performance, and a new virtual environment can be created in just a minute. Cloning a VE is also very simple.

Education – Each student can have a VE and can play with different Linux distributions. A new VE can be (re)created in a minute.