Linux-VServers e segurança por contexto

Silvio Rhatto

21 de agosto de 2007

O Linux-VServer é um remendo para o kernel que introduz o isolamento de processos do sistema por contextos, permitindo inclusive que se escolha os conjuntos capacidades POSIX permitidas em cada um deles.

Com esse patch, é possível criar uma abstração de servidores virtuais que não implica em consumo adicional relevante de processamento e memória e não envolve o uso de máquinas virtuais, como é o caso do Xen, do User Mode Linuxe do QEMU.

Ao contrário desses, o Linux-VServer cria um esquema de isolamento onde cada servidor enxerga apenas parte da realidade do que se passa num servidor, mesmo que todos eles compartilhem os mesmos kernel e hardware.

1 Sobre este documento

Apesar de largamente utilizado e ter um código estável, o projeto Linux-VServer tem uma documentação muito pequena e desencontrada, sem falar da total ausência de referências em português.

Por isso, este texto pretende constituir um modelo conceitual de funcionamento de um sistema opreativo padrão POSIX e posteriormente introduzir as funcionalidades do VServer no Linux Kernel, mantendo toda a discussão no nível do usuário, sem mencionar as implementações desses conceitos, tanto em baixo quando em alto nível. Menções ao espaço do kernel e espaço do usuário serão evitadas para manter a simplicidade.

2 Introdução

O Linux-VServer é um patch no Linux Kernel que extende algumas propriedades de um processo, adicionando funcionalidades que permitem isolar aplicações do resto do sistema e criar uma camada de virtualização, os chamados servidores virtuais. Para entender melhor o assunto, façamos antes uma revisão no esquema de segurança de um sistema do tipo UNIX.

A visão tradicionado de um sistema POSIX pode ser resumida em alguns conceitos:

1. Num sistema POSIX, qualquer coisa é um processo ou um arquivo.

2. Tanto os arquivos quanto os processos estão organizados em estruturas em forma de árvore.

3. O sistema é multi-tarefa e multi-usuário.

Um arquivo é uma porção de dados armazenados nalgum local do sistema de arquivos, que é a árvore hierárquica dos objetos do tipo arquivo. Já um processo é uma rotina em andamento no sistema, um programa rodando. Processos também pertencem a uma árvore hierárquica, a árvore de processos. Um arquivo pode dar origrem a um processo e os resultados de um processo podem se tornar aquivos.

3 Árvore do sistema de arquivos

Como organização em árvore, o sistema de arquivo se origina num nó principal – chamado de raíz do sistema de arquivos – que se ramifica em outros nós – que são as chamadas pastas ou diretórios. Tal ramificação pode em teoria se extender indefinidamente e em qualquer pasta ou até na raíz do sistema é possível armazenar arquivos.

A definição de pasta poderia conflitar com os princípios POSIX anteriormente citados se não incluirmos aqui a observação de que uma pasta nada mais é do que um arquivo cujo conteúdo são informações sobre o que dentro dela existe.

4 Árvore de processos

Já a árvore de processos tem como raíz um processo inicial que na maioria dos sistemas POSIX é conhecido como "init". O init tem a função de chamar as primeiras aplicações do sistema quando este é iniciado. Mais do que isso, todos os outros processos são "filhos"do init e portando sobre ele há um poder total sobre o funcionamento dos seus filhos.

5 Validade destes conceitos

Os sistemas operacionais modernos tendem a não manter uma consistência absoluta com esses conceitos. Esses novos sistemas estão extendendo o padrão POSIX ao criarem objetos que não são nem arquivos nem processos, conhecidos como "interfaces"e que resolvem o problema de disponibilizar um único recurso simultaneamente a vários processos, como é o caso dos dispositivos de rede e som, por exemplo.

6 Segurança do sistema

Por multi-tarefa e multi-usuários entende-se que a árvore de processos é composta por programas em execução "simultânea-- em certo sentido apenas – e

que tanto arquivos como processos estão etiquetados como pertecencentes a "usuários"e "grupos", que nada mais são do que números – posteriormente associados a nomes através dos arquivos */etc/passwd* e */etc/group* – que permitem restringir o acesso de processos a arquivos e outros processos ou a chamadas ao sistema operacional. Os números de usuário são conhecidos como *uid* e os de grupo como *gid*.

O usuário cujo *uid* é zero é conhecido como superusuário ou *root* (raíz, apesar do esquema de usuários não formar uma árvore). O superusuário tem sobre os processos e arquivos um poder absoluto e o grupo correspondente ao superusuário tem gid também zero.

Fora esse usuário, normalmente nenhum outro tem plenos poderes sobre processos e arquivos quaisquer – a não ser que assim se queira e o sistema seja configurado para permitir outros superusuários. Esses outros usuários sem plenos poderes são chamados de usuários comuns.

Um processo ou arquivo possui dois campos de posse/propriedade/pertinência, um para o usuário e outro para o grupo. No caso de um arquivo, esses dois campos não ficam armazenados dentro do arquivo mas sim na árvore do sistema de arquivos.

Um processo que tente acessar um arquivo ou controlar as propriedades de execução tem seus campos *uid* e *gid* comparados com o *gid* e *uid* do arquivo ou processo requisitado e a partir da checagem das permissões do objeto a ação solicitade é deferida ou negada.

É nesse procedimento que se baseia a segurança básica de um sistema POSIX. Levando em conta falhas dos programas e até implementações errôneas desse esquema no sistema operacional, não se pode descartar a hipótese de algum processo obter privilégios sobre arquivos, processos e até chamadas do sistema operacional que não deveria em condições normais.

Existem diversa abordagens contra esse tipo de imprevisto e uma delas é a auditoria do código executado, porém esse tipo de trabalho nem sempre é possível em servidores de produção e a alternativa é executar os processos pouco confiáveis em jaulas.

7 Jaulas

Uma jaula é uma porção do sistema isolada onde processos podem ser executados sem que tenham acesso a outras partes do sistema. Se um processo corrompido compromete os arquivos presentes numa jaula, por exemplo, ele não poderá fazer o mesmo com os arquivos residentes no resto do sistema.

Com o uso de jaulas, os processos inseguros podem ser isolados e a insegurança do sistema residirá em apenas algumas de suas áreas.

Nos sistemas POSIX, as jaulas são criadas com o programa *chroot*, que utiliza a chamada *chroot()* ao sistema operacional. Essa chamada possibilita mudar a raíz do sistema de arquivos para o processo a que ela foi chamada. O processo sob *chroot* enxerga uma outra pasta arbitrária como raíz do sistema de arquivos e não a raíz verdadeira.

Rodar um processo em *chroot* é a maneira usual de se manter um serviço numa jaula, que pode ser qualquer pasta do sistema que possua todos os arquivos necessários pelo programa.

7.1 Criando uma jaula

A forma mais simples de criar uma jaula é construir uma estrutura de pastas contendo os aplicativos e pastas que você quiser rodar. No exemplo a seguir, criaremos uma jaula que possuirá apenas os programas *bash* e *ls*.

```
mkdir -p /tmp/jaula/{bin,lib}
cp -a /lib/* /tmp/jaula/lib/
cp -a /bin/{bash,ls} /tmp/jaula/bin/
```

Para entrar na jaula, basta o comando:

```
chroot /tmp/jaula /bin/bash
```

Uma vez nela, você pode dar o comando *ls* e navegar pela estrutura de pastas e nada mais do que isso. Sem nenhum programa dentro da jaula que possa fazer uma chamada *chroot()*, você não conseguirá mais sair dela, a não ser que termine o processo do *bash* que você iniciou:

```
exit
```

7.2 Traçando dependências

Aqui cabe uma pequena observação. Note que todos os arquivos da pasta */lib* do sistema foram copiadas para a jaula, já que o bash e o ls dependem de uma série de bibliotecas armazenadas nesse local. Mas a cópia de todas as bibliotecas dessa pasta significa que possivelmente sua jaula conterá bibliotecas que não são usadas por nenhum desses dois aplicativos.

Para criar uma jaula que contenha apenas as bibliotecas que serão usadas pelos seus habitantes, use o comando *ldd(1)* para obter uma lista de todas as bibliotecas ligadas ao seu aplicativo:

```
ldd /usr/bin/bzip2
        libbz2.so.1 => /lib/libbz2.so.1 (0x4002e000)
        libc.so.6 => /lib/libc.so.6 (0x4003d000)
        /lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
```

No exemplo acima, vemos que o *bzip2* depende das bibliotecas */lib/libbz2.so.1*, */lib/libc.so.6* e */lib/ld-linux.so.2*. Portanto, somente elas precisam ser copiadas para a jaula caso queiramos adicionar o *bzip2* na brincadeira.

7.3 Segurança do chroot

O chroot sozinho não é a melhor solução para se construir uma jaula, já que um programa dentro dela que chame a função *chroot()* pode facilmente escapar da prisão. Para implementação segura de jaulas em ambientes GNU/Linux, uma boa alternativa é o uso de VServers.

8 Vservers

A abordagem do Linux-VServer – aqui abreviado simplesmente para VServer, Vserver ou vserver – é semelhante ao Zones do Solaris e consiste num patch no kernel e alguns aplicativos, efetuando as seguintes alterações no sistema:

- introdução de "contextos"de processos através da chamada *chcontext()*
- isolamento de rede nos processos através da chama *chbind()*
- uso de atributos extendidos nos sistemas de arquivos para criar uma barreira na chamada *chroot()*
- implementação de diversas capacidades POSIX para os contextos

Além dos campos tradicionais dos processos (uid, gid, pid, nice, etc), esse patch introduz o campo contexto (context). Processos enxergam apenas processos com o mesmo contexto – com algumas exceções a essa regra no caso do processo estar rodando sob o usuário root.

O vserver também possui uma melhor implementação do *chroot()* através de um esquema de *barreira*, utilizando atributos extendidos do sistema de arquivos para sinalizar que nenhuma chamada *chroot* poderá mudar a raíz do sistema para pastas de nível superior a uma dada marcação.

A terceira alteração no kernel é a chamada *chbind()*, que restringe um processo a escutar requisições de rede a um dado IP. Isso permite que um dado processo (pense num *daemon*) não escute por conexões no IP *127.0.0.1* mas apenas num IP específico, caso contrário o programa aceitaria conexões endereçadas a qualquer IP da máquina local.

Essas três chamadas de sistema são utilizadas pelo pacote *util-vserver* – um conjunto de aplicativos no espaço do usuário – para criar uma abstração de servidores virtuais.[1]

[1] Na verdade, apesar de inúmeras vezes nos referirmos a três chamadas ao sistema, as versões mais recentes do Linux VServer utilizam apenas uma chamada, que pode realizar separadamente as funções atribuídas ao *chcontext()* e ao *chbind()*. A chamada *chroot()* é a própria implementação padrão do kernel. A escolha deste texto por separar a descrição das funcionalidades do vserver em três chamadas é apenas didática e reflete seu código original. Consulte sempre o código fonte, que é o local mais confiável para saber de onde as coisas vem, já que nesse projeto tudo muda rapidamente e a documentação nem sempre é atualizada.

9 Instalando e construindo vservers

A instalação do Linux-Vserver envolve os seguintes passos:

1. aplicar o patch no kernel, recompilá-lo e instalá-lo
2. reiniciar a máquina
3. instalar o pacote *util-vserver*
4. configurar o sistema para suporte a vservers
5. criar e executar vservers

Os detalhes específicos para cada distribuição não são cobertos por este texto e serão encaminhados para os seguintes documentos:

- Slackware: Criando Vservers em Slackware
- Gentoo: Gentoo Linux-VServer Howto
- Debian: Linux-Vservers no Debian Grimoire
- Ubuntu: VServer no Ubuntu Wiki

10 Usando vservers

Os vservers tem a aparência de uma máquina virtual, com a diferença que não existe uma virtualização nesse nível. Todos os vservers utilizam o próprio kernel da máquina, ou seja, os processos de um vserver nada mais são do que processos normais da máquina, mas com um valor de *contexto* diferente.

A configuração dos vservers possui inúmeras funcionalidades e aqui faremos apenas um tour geral, sem nos preocuparmos muito nos detalhes.

10.1 Iniciando uma jaula

Para inicializar um vserver existente é utilizado o comando

```
vserver jaula start
```

Esse comando gera uma série de procedimentos através das chamadas *chcontext()*, *chbind()* e *chroot()* para criar um novo contexto no sistema tendo seu próprio endereço IP e sua própria porção isolada no sistema de arquivos, correspondente ao local de armazenamento dos arquivos da jaula, que geralmente ficam dentro da pasta */vservers*.

Depois de criar esse ambiente isolado, o comando vserver

- simula a existência de um processo *init* para executar os scripts de inicialização da jaula *ou, dependendo da configuração,*

- roda o */sbin/init* dentro da jaula – usando para isso seu próprio *inittab*

e a partir daí o vserver passa a se comportar como se fosse uma máquina isolada.

10.2 Isolamento de contexto

Para entrar num vserver inicializado, utiliza-se o comando:

```
vserver jaula enter
```

As seguintes mensagens indicarão que o contexto em que o terminal está operando mudou – quer dizer, foi iniciado o processo de uma shell no contexto do vserver *jaula* –, assim como o IP em que todos os processos nele gerados escutarão:

```
ipv4root is now 10.0.1.1
New security context is 49160
```

Uma vez dentro do vserver, estaremos operando sob um novo contexto, cujo número – ou *xid* –, nesse caso, é *49160*. Execute um *ps aux* para ver o que acontece. Uma saída de exemplo pode ser esta:

```
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root      1271  0.0  0.0   1664   628 ?        Ss   Apr12   0:00 /usr/sbin/syslogd
root      1312  0.0  0.0   1716   588 ?        S    Apr12   0:00 /usr/sbin/crond -l10
root     25851  5.4  0.1   2388  1396 pts/3    S    13:35   0:00 /bin/bash -login
root     25904  1.0  0.0   2512   924 pts/3    R+   13:35   0:00 ps aux
```

De cara vemos que não há o processo *init* e nem os processos que rodam no espaço do kernel. Vemos um *syslogd* com *pid* 1271 e um *crond* com *pid* 1312. Comparemos isso com a saída do mesmo comando obtida ao sairmos do vserver, isto é, ao habitarmos o contexto principal da máquina:

Para encerrar a shell atual e consequentemente volta para a shell que está no contexto principal, use o comando *exit*. De volta ao contexto principal, um trecho de saída possível para o *ps aux* é

```
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0    680    72 ?        S    Apr12   0:00 init [3]
root         2  0.0  0.0      0     0 ?        S    Apr12   0:00 [migration/0]
root         3  0.0  0.0      0     0 ?        SN   Apr12   0:00 [ksoftirqd/0]
[snip]
root        85  0.0  0.0   1520   372 ?        Ss   Apr12   0:00 /usr/sbin/syslogd
root        88  0.0  0.0   1480   248 ?        Ss   Apr12   0:00 /usr/sbin/klogd -c 3 -x
root       158  0.0  0.0   3400   568 ?        Ss   Apr12   0:01 /usr/sbin/sshd
root       167  0.0  0.0   1680   376 ?        S    Apr12   0:00 /usr/sbin/crond -l10
[snip]
```

Aqui vemos o init, os processos do *kernelspace*, o *syslogd* e o *crond*, mas estes dois últimos tem *pid's* respectivamente 85 e 167, já que eles são processos diferentes daqueles que estão rodando dentro da jaula. Além disso, o *ps aux* não foi capaz de enxergar os processos da jaula. Isso pode ser visto de uma outra forma com o exame da estrutura do */proc* do contexto principal e do */proc* da jaula. Esta é a base do isolamento de contextos.

10.3 Resistência da jaula

Na seção anterior, conseguimos entrar e sair da jaula porque:

1. É permitido a um processo do contexto principal, rodando como usuário root, criar um processo num novo contexto.

2. Em geral os vservers estão configurados para permitir o recebimento de processos originados em outros contextos.

Se o estivéssmos no sistema como usuário comum ou então num contexto que não fosse o principal, o sistema operacional não permitiria que executássemos um comando num outro contexto.

10.4 Barreira do chroot

Conforme dito anteriormente, *chroot()* pode ser invocada mesmo dentro de uma jaula para mudar o referencial de raíz de sistema num processo. Para evitar que um processo escape para fora de sua jaula, o vserver trabalha com *barreiras*, que são sinalizadores especiais mantidos na pasta principal das jaulas.

Esses sinalizadores são atributos extendidos do sistema de arquivos e as chamadas de chroot vindas de dentro de um vserver são checadas contra eles: se a chamada requisitar a mudança de raíz de um processo para um nível igual ou acima de onde houver um sinalizador, a mudança de raíz não é efetuada e o comando roda utilizando a raíz atual.

Os atributos de um sistema de arquivos são registros que guardam informações a respeito dos seus arquivos armazenados. Tamanho, permissões, usuário e grupo de um arquivo são informações armazenadas no sistema de arquivos através de atributos, ou seja, estão no nível do *inode*. Além desses atributos clássicos, os sistemas de arquivos modernos suportam um maior número dessas *etiquetas*, que podem ser utilizadas para criar esquemas mais complexos de permissões de arquivos.

A barreira dos vservers para o patch da série 2.x em diante é construída através do comando

```
setattr --barrier /vservers
```

O comando *setattr* vem no pacote util-vserver. Se o patch utilizado for da série 1.x, a barreira é feita com os comandos

```
chmod 000 /vservers
chattr +t /vservers
```

Uma vez estabelecidos, esses atributos permanecem no sistema de arquivos, o que implica que esses comandos precisam ser dados apenas quando a pasta usada para armazenar os vservers é criada ou movida.

10.5 Isolamento de rede

Rodar uma aplicação em *chroot* sem vserver produz apenas um isolamento no sistema de arquivos e, se o */proc* não estiver montado na jaula, um isolamento simples de processos (mas pode fazer com que programas não funcionem direito). Nesse caso de *chroot* simples, não é possível isolar processos a único IP: ao invés disso, os processos da jaula conseguirão escutar o tráfego de rede da maneira usual, isto é, aplicativos rodando como superusuário conseguem ouvir em qualquer porta e processos rodando como usuário comum poderão escutar em portas maiores que 1024 – em todas as interfaces de rede da máquina.

Alguns programas, como o Apache e o OpenSSH possuem parâmetros de configuração que possibilitam a especificação dos IPs da máquina em que o *daemon* irá escutar. Mas nem todos os programas permitem esse tipo de sintonia e além disso uma jaula deve assumir que os programas estão isolados num único IP independentemente da sua configuração.

O *chbind* trava um processo e todos os seus filhos num único endereço IP, mesmo que ele esteja relacionado a uma interface onde estão designados mais endereços. Esse uso de diferentes IPs asociados a uma mesma interface é feito pela implementação de rede do kernel, sem nenhuma modificação pela suite VServer. Por exemplo, no caso do nosso vserver *jaula*, o IP a ele designado é o 10.0.1.1, que é um alias para a interface *eth0*. Vejamos a saída do *ìfconfig*:

```
eth0      Link encap:Ethernet  HWaddr 00:00:00:00:00:00
          inet addr:IP-DO-SERVIDOR  Bcast:BCAST-DO-SERVIDOR  Mask:MASCARA-DO-SERVIDOR

eth0:jaul Link encap:Ethernet  HWaddr 00:00:00:00:00:00
          inet addr:10.0.1.1  Bcast:BCAST-DO-SERVIDOR  Mask:MASCARA-DO-SERVIDOR

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
```

Além das tradicionais interfaces *eth0* e *lo*, temos a *eth0:jaul* cujo broadcast é o mesmo da *eth0*. Efetivamente, *eth0* e *eth0:jaul* representam o mesmo dispositivo de rede, mas cada uma possui um IP distinto.

O vserver jaula está "travado"no IP da *eth0:jaul* e por isso seus processos – independentemente de rodarem como superusuário ou não – conseguem escutar apenas o tráfego destinado ao endereço 10.0.1.1.

11 Detalhes de funcionamento

Até aqui fizemos ou pequeno *tour* pelas funcionalidades básicas dos vservers. É preciso agora dar um pequeno detalhamento em cada conceito antes de nos determos na configuração dos vservers para uso real.

11.1 Capacidades POSIX

Numa das primeiras seções deste texto, dissemos que o usuário cujo *uid* é zero tem poderes absolutos sobre o sistema. Esses poderes nada mais são do que um conjunto de capacidades atribuídas aos processos que rodam como *uid* zero e que permitem a execução de uma série de operações privilegiadas. Essas capacidades são conhecidas como *Capacidades POSIX*.

Em outras palavras, as *Capacidades POSIX* são um conjunto de potencialidades disponíveis para um dado processo desfrutar dos privilégios de superusuário. Dentre elas, estão as capacidades de alterar a data do sistema, mudar usuário e grupo de arquivos, executar *chroot()*, criar dispositivos e reiniciar o sistema. Essas são tarefas que em geral podem ser executadas apenas por processos rodando como usuário root. Mas, se retirarmos uma capacidade de um processo executar determinada tarefa, ele não conseguirá realizá-la mesmo se estiver rodando como root.

As capacidades POSIX são informações associadas a um processo. Cada processo tem três conjuntos de capacidades: as *herdadas*, as *permitidas* e as *efetivas*. Quando um processo tenta executar uma operação considerada privilegiada, o sistema operacional checa se ele possui no seu conjunto de capacidades *efetivas* a potencialidade de executar tal operação, ao invés de simplesmente checar se o *uid* do processo é zero.

O conjunto das capacidades *permitidas* contém todas as capacidades que um processo pode desfrutar. Processos podem ter capacidades *permitidas* mas que não são *efetivas*. Isso foi idealizado para permitir que processos desabilitem temporariamente suas capacidades: processos podem adicionar e remover no conjunto das *efetivas* todas as capacidades que estiverem em seu conjunto de *permitidas*.

Já as capacidades *herdadas* de um processo são aquelas que serão propagadas para os processos que forem por ele executados, isto é, as capacidades *permitidas* dos seus processos filhos.[2]

Capacidades POSIX não devem ser confundidas com qualquer tipo de operação realizada num sistema do tipo UNIX ou com o conceito de *capacidade* estudado em sistemas operacionais: capacidades POSIX referem-se à permissão de se realizar uma operação considerada privilegiada pelo sistema operacional, como por exemplo alterar a data do sistema. Permissões de arquivos são capacidades no sentido de restringir o acesso a um recurso para um dado usuário ou grupo, mas não são capacidades POSIX. Daqui em diante nos referiremos a *capacidades* como sinônimo para *Capacidades POSIX*.

[2] Detalhes na Linux Capabilties FAQ.

Um vserver tem um conjunto de capacidades permitidas restrito que isola mais a instância virtual ao imperdir que dentro dela nem o superusuário consiga executar tarefas privilegiadas. Por padrão, um vserver não tem muitas capacidades, o que em geral não impede o funcionamento de nenhum programa e, caso seja necessário, é possível fornecer capacidades a um vserver.

A restrição do uso de capacidades dentro de um vserver é uma medida de segurança e leva em conta que a maioria das aplicações de um servidor não precisa lidar com operações privilegiadas e logo é desnecessário permiti-las, já que alguma vulnerabilidade pode levar a um usuário malicioso que de algum modo ganhou status de superusuário a explorar essas capacidades.

A lista de capacidades do Linux se encontra em */usr/include/linux/capability.h*. O Linux-Vserver ainda introduz uma série de capacidades úteis para fazer uma sintonia fina no servidor virtual.

A atribuição das capacidades de um vserver é feita mudando seus parâmetros de configuração e o administrador ou a administradora de sistema raramente precisará atribuir capacidades aos vservers.

11.2 Flags

Nos vservers, ajustes em funcionalidades que não são fornecidas através de capacidades são atribuídas à *flags*, que controlam, por exemplo:

- Se processos de outros contextos podem criar processos no contexto de cada vserver
- Se o escalonador do sistema operacional dará a mesma prioridade a todos os processos de um vserver
- Aplica limites ao vserver (memória, processamento, disco, etc)

Assim como as capacidades, cada vserver pode ter sua própria configuração de flags.

11.3 Chcontext

O uso de contextos não é restrito apenas a servidores virtuais. O pacote *util-vserver* contém o aplicativo *chcontext* que permite executarmos um processo num outro contexto de segurança. O comando

```
chcontext ps aux
```

executará o comando *ps aux* num novo contexto, que conterá apenas esse processo. Para usar um contexto existente, use

```
chcontext --ctx context-id ps aux
```

onde *context-id* é o número do contexto e se ele não existir será criado com esse número. Dois contextos são reservados:

- O contexto 0 é o principal, onde rodam as aplicações sistema
- O contexto 1 é especial e tem acesso à informação dos processos de todos os vservers

O contexto 0 – também chamado de *contexto principal*, *servidor principal*, *contexto administrativo* ou simplesmente de *hospedeiro* (*host*) – é o espaço usual do sistema, que se comporta como se o patch do vserver não tivesse sido aplicado ao kernel.

No contexto 0, assim como nos outros, é possível enxergar apenas os processos dele próprio. Já o contexto 1 – que aqui chamarei de *contexto supervisor* – podemos enxergar todos os processos rodando na máquina, incluindo os dos vservers:

```
chcontext --ctx 1 ps aux
```

O comando *vps*, do *util-vserver*, pode ser usado como abreviação para o comando acima:

```
vps aux # mostra todos os processos na maquina
```

Se você experimentar rodar o *chcontext* dentro de um vserver para tentar acessar outros contextos, receberá uma mensagem de *operação não permitida*.

11.4 Segurança do proc

O *proc* é o sistema de arquivos que faz a interface entre o espaço do usuário e as estruturas de dados do kernel. No proc ficam tipicamente listadas informações como processos e estado da rede e podem ser utilizado para controlar parâmetros do kernel. É importante evitar que muitas dessas informações estejam indisponíveis dentro dos vservers e por padrão algumas dessas entradas (como as informações de processos de outros contextos) já estarão escondidas.

Mas a configuração padrão não esconde tudo, deixando isso para ser realizado pelo administrador ou a um script. Essa "segurança"do proc é feita com o comando *setattr*, anteriormente usado para criar a barreira dos vservers. Mudar a visibilidade das entradas no */proc* nada mais é do que manipular os atributos extendidos dessas entradas, por isso o uso do *setattr*.

Antes de escondermos uma entrada, vejamos como estão seus atributos extendidos. O comando do *util-vserver* que faz isso é o *showattr*:

```
showattr /proc/cpuinfo
```

cujo resultado é

```
Awh-ui- /proc/cpuinfo
```

A primeira coluna do resultado indica, letra por letra:

1. Sinalizador de visibilidade no contexto principal (admin)

2. Sinalizador de visibilidade no contexto 1 (supervisor)
3. Sinalizador de ativação de invisibilidade
4. Sinalizador de barreira (existente apenas em pastas)
5. Sinalizador de unificação
6. Sinalizador de imutabilidade

Para esconder uma entrada do */proc*, apenas os três campos são utilizados. A seguinte lógica é utilizada para saber quando um arquivo está visível:

```
  admin   | supervisor | ativação | estado do arquivo
-------------------------------------------------------------------------
 qualquer |  qualquer  |    h     | visível em todos os contextos
    A     |     w      |    H     | visível apenas no contexto 0
    a     |     W      |    H     | visível apenas no contexto 1
    A     |     W      |    H     | visível apenas nos contextos 0 e 1
    a     |     w      |    H     | invisível em todos os contextos
```

Quando o campo de invisibilidade estiver minúsculo (desativado), não importa qual for valor dos campos de visibilidade dos contextos principal e supervisor: o arquivo estará visível em todos os contextos, como é o caso do */proc/cpuinfo*. Para escondê-lo nos vservers, utilizamos o comando

```
setattr --hide /proc/cpuinfo
```

Um *showattr* posterior nele indicaria

```
AwH-ui- /proc/cpuinfo
```

Se quisermos desligar o sinalizador de invisibilidade, basta adicionarmos um antes do nome da opção:

```
setattr --~hide /proc/cpuinfo
```

Para ligar e desligar os sinalizadores de visibilidade dos contextos 0 e 1 existem as opções *admin* e *watch*. Por exemplo, para escondermos o */proc/interrupts* do contexto 1 e de todos os vservers, usa-se o comando

```
setattr --hide --~admin --watch /proc/interrupts
```

O pacote *util-vserver* ainda contém o script de inicialização *vprocunhide*, que esconde dos vservers algumas entradas do */proc*.

11.5 Namespaces

As novas versões do Linux-VServer incluem o isolamento via *namespaces*, que neste caso faz com que diferentes vservers tenham uma visão diferente do sistema de arquivos. Isso é feito principalmente mudando o conteúdo do */proc/mounts* de cada vserver para conter apenas as partições que foram montadas para ele. Como essa funcionalidade já foge um pouco do escopo deste texto, deixaremos sua descrição para o documento Namespaces.

11.6 Estimativas de uso

A pasta */proc/virtual* contém estimativas de uso e consumo de recursos de cada contexto. Cada pasta do */proc/virtual* tem o nome do contexto a que se refere e dentro dela temos os seguintes arquivos:

- info: informações básicas do contexto, como o seu nome, seu *xid* e o *pid* do seu init
- cacct: contém contagens de mensagens trocadas entre processos em diferentes protocolos
- cvirt: contém número de forks, número de threads, etc
- limit: contém colunas que listam os limites: atual (1), máximo (2), limite (3) e total (4)
- sched: valores do escalonador do sistema associados ao contexto
- status: lista capacidades, flags, etc

Detalhes das estimativas no texto HowTo Read ProcFS.

11.7 Unificação

A unificação dos vservers consiste no compartilhamento de arquivos do sistema – comandos e bibliotecas – entre as jaulas sem que ocorra uma perda de segurança.

O atributo extendido conhecido como *imutabilidade* é utilizado para impedir que um arquivo seja apagado do sistema: mesmo um *rm -f* não funciona num arquivo imutável, sendo necessário remover o atributo de imutabilidade e depois apagá-lo. Para ter controle sobre a imutabilidade de um arquivo, um processo precisa ter a capacidade POSIX correspondente no seu conjunto das permitidas. Se os arquivos de um vserver estão marcados como imutáveis e o vserver não tem essa capacidade, nenhum processo de dentro do vserver poderá apagá-los.

A imutabilidade sozinha é uma medida de segurança e que eventualmente pode trazer alguns problemas na hora de atualizar pacotes no vserver. Quando usada com *hardlinks*, a imutabilidade ajuda a unificar vservers: arquivos comuns a diversos vservers passam a ser *hardlinks* para uma única cópia que fica marcada como imutável para impedir que uma alteração num vserver seja feita e consequentemente afete todos os outros.

Diferentemente dos links simbólicos – que nada mais são do que referências a outros arquivos –, os hardlinks podem ser entendidos como nomes diferentes para um mesmo arquivo. Enquanto um link simbólico aponta para o caminho de outro arquivo, o hardlink aponta diretamente para o *inode* do arquivo, isto é, como se apontasse diretamente para o seu conteúdo. Dessa forma, dois arquivos idênticos ocupam muito mais espaço do que apenas uma cópia mais um hardlink para ela.

A unificação economiza espaço em disco: cinco vservers não-unificados que consomem 500MB poderiam consumir apenas 100MB numa unificação: apenas os arquivos específicos – como sites, pastas de usuários e configurações próprias – permaneceriam separados, algo também muito importante para a construção de backups. A economia de memória também é substancial pois menos arquivos precisarão ser carregados. É também eficiente quando os vservers utilizam a mesma versão e distribuição de GNU/Linux, pois assim conterão muitos arquivos idênticos.

A unificação ainda não é largamente utilizada, sendo um esquema novo e por enquanto não recomendável para servidores de produção. As novas versões do pacote *util-vserver* contém os aplicativos *vunify* e *vhashify* para criar vservers unificados a partir de uma jaula de referência. Detalhes estão na página alpha util-vserver.

11.8 Limites

Os vservers ainda são passíveis das seguintes limitações:

- Quotas de disco por contexto
- Número máximo de processos
- Limite de memória
- Limites do escalonador
- Limite de tráfego de rede

Todas essas limitações – à exceção do controle de banda, que é feito com as ferramentas usuais do GNU/Linux – são impostas através dos arquivos de configuração de cada vserver.

11.9 Virtualização parcial

O pacote *util-vserver* ainda vem com o aplicativo *chbind*, que assim como o *chcontext* permite o isolamento parcial de uma aplicação sem utilizar um vserver. O *chbind* isola um programa num endereço IP. Esses programas podem ser utilizados separadamente para criar ambientes semi-virtualizados, onde a criação de um vserver é desnecessária.

12 Configuração de vservers

Agora que o funcionamento do Linux VServer foi em certa medida desvendado, podemo passar tranquilamente para a configuração e uso cotidiano do sistema. As próximas seções descrevem a configuração de um vserver e sugere algumas sugestões simples de uso com algumas aplicações populares, como um servidor web e um banco de dados operando em jaulas separadas.

12.1 Formatos de configuração

Em suas primeiras versões, o Linux VServer baseava toda a configuração de um vserver num arquivo único de configuração. A partir das versões 1.9.x do vserver, há um novo formato de configuração baseada em pastas, que é muito mais flexível e recomendada.

12.1.1 Formato antigo

O formato antigo usa um único arquivo de configuração. No caso do vserver de nome *jaula*, a configuração no formato antigo é o arquivo */etc/vservers/jaula.conf*, cujo conteúdo é mais ou menos este:

```
if [ "" = "" ] ; then
PROFILE=prod
fi
# Select the IP number assigned to the virtual server
# This IP must be one IP of the server, either an interface
# or an IP alias
# A vserver may have more than one IP. Separate them with spaces.
# do not forget double quotes.
# Some examples:
# IPROOT="1.2.3.4 2.3.4.5"
# IPROOT="eth0:1.2.3.4 eth1:2.3.4.5"
# If the device is not specified, IPROOTDEV is used
case $PROFILE in
prod)
#IPROOT=10.0.0.1
IPROOT="eth0:192.168.0.1"
# The netmask and broadcast are computed by default from IPROOTDEV
#IPROOTMASK=
#IPROOTBCAST=
# You can define on which device the IP alias will be done
# The IP alias will be set when the server is started and unset
# when the server is stopped
#IPROOTDEV=eth0
# You can set a different host name for the vserver
# If empty, the host name of the main server is used
S_HOSTNAME=skel
```

```
;;
backup)
IPROOT=1.2.3.4
#IPROOTMASK=
#IPROOTBCAST=
#IPROOTDEV=eth0
S_HOSTNAME=
;;
esac
# Uncomment the onboot line if you want to enable this
# virtual server at boot time
#ONBOOT=yes
# You can set a different NIS domain for the vserver
# If empty, the current on is kept
# Set it to "none" to have no NIS domain set
S_DOMAINNAME=
# You can set the priority level (nice) of all process in the vserver
# Even root won't be able to raise it
S_NICE=
# You can set various flags for the new security context
# lock: Prevent the vserver from setting new security context
# sched: Merge scheduler priority of all processes in the vserver
#        so that it acts a like a single one.
# nproc: Limit the number of processes in the vserver according to ulimit
#        (instead of a per user limit, this becomes a per vserver limit)
# private: No other process can join this security context. Even root
# Do not forget the quotes around the flags
S_FLAGS="lock nproc"
# You can set various ulimit flags and they will be inherited by the
# vserver. You enter here various command line argument of ulimit
# ULIMIT="-HS -u 200"
# The example above, combined with the nproc S_FLAGS will limit the
# vserver to a maximum of 200 processes
ULIMIT="-HS -u 1000"
# You can set various capabilities. By default, the vserver are run
# with a limited set, so you can let root run in a vserver and not
# worry about it. He can't take over the machine. In some cases
# you can to give a little more capabilities (such as CAP_NET_RAW)
# S_CAPS="CAP_NET_RAW"
S_CAPS="CAP_SETGID"
# Select an unused context (this is optional)
# The default is to allocate a free context on the fly
# In general you don't need to force a context
#S_CONTEXT=
```

As principais variáveis do arquivo de configuração e seus valores correspondem a:

- *IPROOT*: IP do servidor virtual, é um alias para o seu dispositivo *eth0*, *eth1*, etc.
- *S_HOSTNAME*: nome do host do vserver.
- *ONBOOT*: indica se o vserver será iniciado ou não via initscript.
- *S_NICE*: qual valor de nice padrão terão todos os processos do vserver.
- *S_FLAGS*: controla algumas opções do contexto:
  - sched: joga todos os processos do vserver para uma única prioridade no escalonador.
  - nproc: limita o numero de processos do vserver de acordo com ULIMIT.
  - private: nenhum outro processo pode entrar nesse contexto, nem mesmo um com uid 0.
- *ULIMIT*: limita o número de processos do vserver.
- *S_CAPS*: lista das capacidades POSIX permitidas.

12.1.2 Novo formato

O novo formato de configuração se baseia em pastas dentro de */etc/vservers*:

- */etc/vservers/.defaults*: contém as opções padrão para vservers.
- */etc/vservers/.distributions*: contém informações específicas de distribuições de GNU/Linux, para o caso de você construir vservers utilizando os scripts do *util-vserver*.
- */etc/vservers/nome-do-vserver*: pasta de configuração do vserver *nome-do-vserver*.

No novo formato, a configuração do nosso vserver *jaula* fica armazenada na pasta */etc/vservers/jaula*. As seções principais dessa pasta são:

- *context*: contém o *xid* do vserver.
- *flags*: contém as flags de controle.
- *fstab*: o fstab usado pelo vserver.
- *name*: nome da jaula.
- *nice*: valor de nice (prioridade de execução de processo) do vserver.

- *apps/*: configuração de aplicações como o *init* do sistema.
- *interfaces/*: configurações de rede.
- *dlimits/*: limites de uso de disco.
- *rlimits/*: limites de recurso.
- *ulimits/*: limites do nível do usuário.
- *scripts/*: scripts de inicialização do vserver.
- *uts/*: atribuições à estrutura de nomes do sistema

Todos os parâmetros de configuração disponíveis para essas seções são descritos na Famosa Página Florida.

Os scripts de inicialização deste caso não devem ser confundidos com os *initscripts* do vserver: os primeiros possibilitam a execução de comandos no contexto principal, enquanto estes últimos serão executados no contexto do vserver. Da mesma forma, a pasta *apps/* não é usada para guardar as configurações dos programas utilizados dentro do vserver, mas sim para configurar seus aplicativos auxiliares, como por exemplo o método de inicialização a ser utilizado ou as pastas que serão unificadas pelo *vunify*.

12.2 Capacidades

As capacidades possíveis de serem utilizadas dentro de um vserver incluem algumas das capacidades POSIX do kernel Linux mais algumas especificamente implementadas pelo projeto Linux VServer. A lista das capacidades implementadas pelo VServer – chamadas de *capacidades de contexto* – se encontra no arquivo lib/ccaps-v13.c e aqui destacaremos apenas algumas:

- *SET_UTSNAME*: mudanças na estrutura de nome do sistema
- *SET_RLIMIT*: permite mudar limites de uso dos recursos do sistema
- *RAW_ICMP*: permite o uso de sockets ICMP (ping, por exemplo)
- *SYSLOG*: permite registrar mensagens do sistema via syslog
- *QUOTA_CTL*: permite estabelecimento de quotas dentro do vserver

Já as capacidades implementadas no próprio kernel do Linux são listadas no arquivo lib/bcaps-v13.c do código fonte do Linux Vserver. Destacaremos algumas, dividindo entre as que estão habilitadas por padrão num vserver e aquelas que precisam ser habilitadas.

12.2.1 Capacidades do sistema habilitadas por padrão

- *CAP_CHOWN*: permite a mudança de usuário e grupo nos arquivos
- *CAP_KILL*: ignora a restrição de que o processo que está sendo morto precisa ter o mesmo *uid* do processo que está tentando matá-lo
- *CAP_SETGID*: permite o uso de setgid
- *CAP_SETUID*: permite o uso de setuid[3].
- *CAP_NET_BIND_SERVICE*: permite utilizar portas menores que 1024
- *CAP_SYS_CHROOT*: permite o uso de *chroot()*

12.2.2 Capacidades do sistema desabilitadas por padrão

As capacidades desabilitadas por padrão permitem modificações do sistema que comprometem muito a sua segurança. O uso delas em vservers não é recomendado.

- *SETPCAP*: permite transferir capacidades listadas no grupo das *permitidas* para os outros grupos
- *LINUX_IMMUTABLE*: permite modificar o atributo de imutabilidade de um arquivo
- *NET_BROADCAST*: permite envio de pacotes de rede em broadcast
- *NET_ADMIN*: permite a configuração da interface de rede (modo promíscuo, por exemplo)
- *NET_RAW*: uso de pacotes no estado bruto
- *SYS_ADMIN*: permite a execução de uma gama de tarefas administrativas
- *SYS_NICE*: permite modificar prioridades de execução de processos
- *SYS_RESOURCE*: permite modificar limites e quotas, dentre outras coisas
- *SYS_TIME*: permite modificar a data do sistema
- *MKNOD*: permite a criação de arquivos de dispositivo

12.2.3 Atribuição de capacidades

As capacidades devem ser atribuídas nos vservers através dos seus arquivos de configuração. O formato antigo atribui as capacidades através do seu nome extenso (da forma como estão listadas acima), enquanto que o novo formato utiliza o nome abreviado (da forma como estão definidos nos arquivos *.c* acima referenciados). Por exemplo, o nome abrevidado para *SET_UTSNAME* é *utsname*.

[3]Há uma boa explicação sobre o funcionamento dos IDs de processo no artigo Gerenciando privilégios.

12.3 Flags

As flags implementadas para um vserver são listadas no arquivo lib/cflags-v13.c e uma descrição se encontra na página Caps and Flags. Dentre elas destacam-se:

- INFO_LOCK (lock): proíbe o uso de *chcontext()* dentro do vserver
- INFO_NPROC (nproc): aplica limites de processos no vserver
- INFO_PRIVATE (private): não permite que processos de outros contextos usem o *chcontext()* para criar um processo no contexto do vserver em questão
- INFO_INIT (fakeinit): mostra um processo init falso dentro do vserver
- INFO_ULIMIT (ulimit): aplica limites ao contexto
- INFO_NAMESPACE (namespace): atribui o uso de um *namespace* no vserver

Tanto no formato antigo quanto no novo formato esses sinalizadores devem ser atribuidos ao vserver usando o nome abreviado (*lock* ao invés de *INFO_LOCK*, por exemplo).

13 Comandos diversos

Como referência, segue a descrição e a sintaxe básica de alguns dos programas o pacote *util-vserver* usados para execução de tarefas administrativas. Todos eles possuem página manual onde o uso é detalhado e foram feitas para serem utilizadas no contexto principal e não dentro dos vservers.

13.1 vserver

O aplicativo *vserver* é o programa principal para lidar com vservers. Ele serve para iniciar, reiniciar, parar e até construir um vserver usando algumas distribuições suportadas. Sua sintaxe simplificada é:

```
vserver [nome do vserver] [opcoes]
```

Dentre as opções, temos:

- *build*: constrói um vserver usando um dos sistemas de pacote disponíveis (slackware não é suportado)
- *enter*: entra num vserver
- *exec*: executa um comando num vserver como usuário root
- *suexec*: executa um comando num vserver com um *uid* arbitrário

- *service*: controla um serviço dentro de um vserver (start/stop/restart etc.)
- *start*: inicia um vserver
- *stop*: termina a execução de um vserver
- *running*: informa, através do seu código de saída, se um vserver está em funcionamento
- *status*: informa se o vserver está em funcionamento e imprime algumas de suas informações de estado

13.2 vps

Como dito anteriormente, *vps* é executa um *ps* no contexto 1, servidor para exibir informações de processos de todos os contextos. Sua sintáxe é análoga ao *ps*.

13.3 vserver-stat

O comando *vserver-stat* exibe estimativas de funcionamento de todos os contextos da máquina.

13.4 vtop

O *vtop*, de modo semelhante ao *vps*, executa um *top* no contexto 1, permitindo aferições periódicas da execução de todos os processos do sistema.

14 Uso de vservers em sistemas de produção

Os aplicativos que usualmente são rodados em plataforma GNU/Linux podem ser na maioria das vezes facilmente configurados para rodarem dentro de servidores virtuais. A maioria deles não depende de Capacidades POSIX e nem de leituras especiais no */proc*. Aqui faremos um pequeno apanhado de configurações úteis para um servidor do tipo LAMP (GNU/Linux, Apache, Mysql, Perl/PHP/Python) com servidor de email.

14.1 Compartilhando um único IP real

Muitos servidores possuem mais de um IP real para que possam atribuir diferentes serviços e aplicações na mesma porta: o roteamento multiplexa a conexão que chega entre os diferentes vservers. Assim, podemos ter dois apaches em vservers diferentes, cada um escutando um IP e possuindo seus próprios *VirtualHosts*. Para isso, poderíamos utilizar o *iptables* para reservar um IP real para cada vserver e estabelecer o roteamento, ou então simplesmente configurar cada vserver para utilizar seu respectivo IP real.

Mas esse caso de servidores possuírem muitos IPs reais é raro, principalmente numa internet que utiliza IPv4. Em geral cada máquina possui um único IP e este precisa ser compartilhado por todos os vservers. Para um servidor que utiliza apenas uma instância de cada tipo de aplicação – um único apache, ou um único *sshd* –, basta rotearmos conexões entrantes na porta que cada protocolo utiliza para o vserver em que o serviço se encontra. A próxima seção explica como rotear conexões entrantes para um vserver.

Quando quisermos utilizar mais de uma instância de um mesmo serviço possuindo apenas um único IP real, por exemplo, temos duas alternativas:

- Utilizar um proxy para o serviço
- Utilizar portas fora do padrão e distribuir as várias instâncias entre elas

A primeira solução funciona muito bem com servidores web, principalmente por não poluir as URLs (http://servidor.net:porta1 e http://servidor.net:porta2): o proxy web separa as requisições pelo cabeçalho *http*, o que permite atribuir domínios diferentes para instâncias em vservers diferentes. Usando um servidor de email bem configurável é possível também construir um proxy para reenviar emails para instâncias de MTAs em diferentes vservers. Para web, o Apache (veja exemplo a seguir) é mais do que suficiente, e proxies como o Delegate servem para *http* e para outros protocolos.

Já a segunda solução funciona bem para os outros protolocos que não estão tão rigidamente ligados a uma porta padrão, como *subversion*, *icecast*, *silc*, *ssh*, etc). As portas e seus usos são especificados pela IANA.

14.2 Roteamento de rede

O roteamento de rede e o controle da banda para os vservers é feito com a infra-estrutura já existente no GNU/Linux (*iptables*, *iproute2*, etc). Aqui consideramos o caso de vservers que tanto podem acessar máquinas remotas quanto receber conexões entrantes da internet.

A primeira medida é estabelecer a tradução de endereços via NAT se o vserver não possuir IP real. Isso é feito usualmente e depende do sistema de firewall que você está utilizando. O mais simples é um iptables direto. Considerando que a faixa de IPs para os seus vservers é a *192.168.0.X*, use

```
iptables -A POSTROUTING -t nat -s 192.168.0.0/24 -d 0/0 -j SNAT --to IP-DO-SERVIDOR
```

Onde *IP-DO-SERVIDOR* é o endereço real da sua interface. Em seguida, temos que rotear as conexões entrantes para as respectivas aplicações enjauladas. O comando

```
iptables --protocol tcp -t nat -A PREROUTING -i eth0 -s 0/0 -d IP-DO-SERVIDOR \
         --dport PORTA -j DNAT --to-destination DESTINO:PORTA2
```

serve para direcionar uma conexão chegando no *IP-DO-SERVIDOR* na porta *PORTA* para o vserver de IP *DESTINO* na porta *PORTA2*. Nos casos em que o servidor possui mais de um IP real e um deles for atribuído ao vserver, este tipo de roteamento não é necessário.

14.3 Apache e proxies de http

Para a configuração das instâncias do Apache no contexto principal e nos vservers, consideraremos dois casos:

1. Um IP real para cada vserver e um IP real para o contexto principal.
2. Um único IP real para toda a máquina (modo proxy).

Consideraremos apenas o caso do Apache 1.3.

14.3.1 Apache com múltiplos IPs reais na máquina

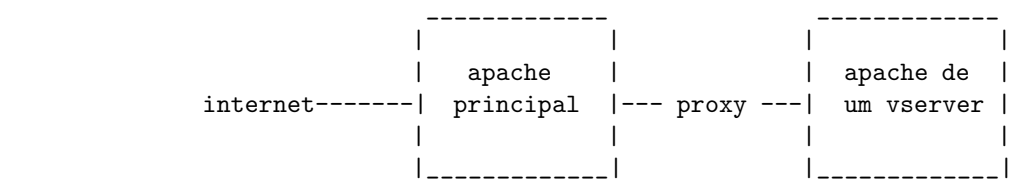
Para Apaches operando simultaneamente em vários vservers e/ou no contexto principal, tudo o que precisa ser feito em cada um deles é:

1. Atribuir o IP de cada vserver no respectivo *httpd.conf* através da diretiva Listen
2. Utilizar esse mesmo IP nas diretivas NameVirtualHost e VirtualHost

Se você não associar os IPs aos respectivos Apaches, um deles irá responder por todas as requisições entrante na porta 80 da máquina.

14.3.2 Apache com um único IP real

Utilizar vários Apaches com um único IP real requer uma configuração mais intrincada e segue o seguinte esquema:



Nessa configuração, haverá uma instância de apache – que pode estar tanto dentro de um vserver quanto no contexo principal – recebendo todas as conexões entrantes na porta 80 e as repassará para as outras instâncias de apache que estarão nos vservers. Por isso, teremos dois tipos de configuração:

1. Configuração do apache principal como proxy
2. Configuração dos apaches que rodam atrás do proxy

A configuração do apache principal envolve a diretiva *Listen* associada ao IP real do servidor e um bloco de VirtualHost para cada domínio utilizado, juntamente com a respectiva entrada no */etc/hosts*. A entrada na configuração do apache principal é, para cada domínio:

```
<VirtualHost *>
    ServerName site.projeto.org
    ProxyPass / http://site.projeto.org/
    ProxyPassReverse / http://site.projeto.org/
</VirtualHost>
```

As diretivas ProxyPass e ProxyPassReverse criam um proxy reverso que passará todas as requisições a http://site.projeto.org ao IP associado a esse domínio. Para criar essa associação, é preciso que exista a seguinte entrada no */etc/hosts* do contexto desse apache principal – já que o apache fará a busca de DNS primeiro nesse arquivo:

```
192.168.0.1 site.projeto.org
```

Nesse caso, estamos considerando que o IP do vserver cujo apache hospeda o *site.projeto.org* é o *192.168.0.1*. Se você usa o *Bind* ou outro *daemon* de DNS, efetue o procedimento correspondente.

No apache do vserver, você deverá configurar a diretiva *NameVirtualHost* para usar esse IP:

```
NameVirtualHost 192.168.0.1
```

E a entrada para esse sítio no apache do vserver teria o seguinte formato:

```
<VirtualHost 192.168.0.1>
    ServerName site.projeto.org
    DocumentRoot /var/www/
</VirtualHost>
```

14.4 Servidor de email

Todos os vservers e o contexto principal podem utilizar um único MTA enclausurado num vserver para enviar mensagens de email. Existem uma série de gateways de email, como por exemplo o ssmtp, que podem ser utilizados para repassar todas as mensagens de email de um vserver para outro através de uma conexão SMTP entre eles. Geralmente eles funcionam como wrappers para o comando *sendmail*, o que torna seu uso totalmente transparente para o sistema.

14.5 Banco de dados

Da mesma forma como no caso do servidor de email, apenas uma instalação do sistema de banco de dados é necessária para servidor às aplicações de todos os vservers. O *mysql* por exemplo funciona muito bem enjaulado dentro de um vserver. Os aplicativos de outros vservers ou do contexto principal podem utilizar tranquilamente esse serviço, mas ao invés de utilizarem *localhost* como endereço para a conexão *mysql*, devem usar o endereço IP do vserver onde ele se encontra.

14.6 SSH

Você pode utilizar vários servidores de OpenSSH para cada um deles dar acesso a um vserver. Apenas o *sshd* do contexto principal deve estar configurado para ouvir conexões apenas no IP real do servidor, o que pode ser feito através da seguinte diretiva no $sshd_config$:

```
ListenAddress IP-DO-SERVIDOR
```

Existem outras alternativas de configuração para múltiplos OpenSSH, descritas no howto SSH Login.

14.7 Reiniciando um vserver de dentro

Em algumas ocasiões pode ser interessante fazer com que seja possível reiniciar um vserver de dentro dele. Para isso foi criado o daemon rebootmgr*, que deve rodar no contexto principal. Dentro dos vservers é preciso copiar o aplicativo* vreboot *que acompanha o pacote* util-vserver*. Basta executar o* vreboot *dentro do vserver para que ele seja reiniciado.*

15 Possíveis aplicações

Para finalizar este texto, vejamos uma lista de possíveis aplicações e vantagens de se utilizar vservers num sistema de produção:

- *Toolchains automáticos para construção de distros*
- *Sandbox para ensino de administração de servidores*
- *Construção de pacotes para não sujar o sistema principal*
- *Organização de servidores compartilhados por grupos e projetos diferentes*
- *Replicabilidade de sistemas: basta copiar a pasta do vserver e as configurações*
- *Isolamento de serviços e aplicações possivelmente inseguras*

Isolar os serviços de um servidor talvez seja aplicação mais imediata, mas a organização que a separação de cada tipo de aplicação ou conjunto de sites no seu próprio sistema independente ajuda muito na administração de servidores. É possível até fornecer a um dado grupo ou projeto privilégios totais de uso num vserver, sem que a segurança do resto do sistema seja comprometida.

É possível até construir vservers com uma distribuição inteiramente compilada contra a uClibc (gentoo embedded, por exemplo), para que o sistema tenha o mínimo de sobrecarga de recursos, diminuindo por exemplo o uso de memória.

16 Onde encontrar ajuda

- *Página oficial: http://linux-vserver.org/*
- *Página do projeto: http://www.13thfloor.at/vserver/project/*
- *Lista de discussão: http://list.linux-vserver.org/mailman/listinfo/vserver*

17 Projetos relacionados

O Linux Vserver não é a única possibilidade para criar uma estrutura de virtualização e isolar aplicações em contêiners. Na verdade, existem dois grupos de soluções para esse tipo de organização de um sistema, cada um deles com diversas soluções.

17.1 Máquinas virtuais: virtualizadores e para-virtualizadores

Os virtualizadores simulam uma plataforma computacional e sobre ela rodam um sistema operacional. Os virtualizadores mais utilizados para criar servidores virtuais são:

- *QEMU*
- *User Mode Linux*

Os para-virtualizadores não simulam uma plataforma computacional mas precisam de modificações no sistema operacional para rodarem um kernel no espaço do usuário. O mais utilizado para-virtualizador é o Xen Virtual Machine Monitor.

17.2 Virtualização no nível do sistema operacional

Uma outra classe de virtualizadores é construída sem simular uma plataforma computacional e utilizando o mesmo kernel do sistema para todos os servidores virtuais. O Linux Vserver faz parte deste grupo de soluções, e além dele existem as seguintes implementações:

- *OpenVZ*
- *FreeVPS*
- *FreeBSD Jails*
- *Solaris Zones*

18 Licença e Contato

Este texto foi escrito por Silvio Rhatto, que pode ser contatado através do email rhatto em riseup.net.