

SUSE Linux Enterprise Server 15 SP2

虚拟化最佳实践

SUSE Linux Enterprise Server 15 SP2

出版日期: December 12, 2024

目录

- 1 虚拟化场景 2
- 2 实施修改之前 2
- 3 建议 3
- 4 VM 主机服务器配置和资源分配 3
- 5 VM Guest 映像 24
- 6 VM Guest 配置 37
- 7 特定于 VM Guest 的配置和设置 43
- 8 超级管理程序与容器的对比 47
- 9 Xen: 将半虚拟 (PV) Guest 转换为全虚拟 (FV/HVM) Guest 50
- 10 外部参考信息 54

1 虚拟化场景

虚拟化能为您的环境提供许多功能。该技术可以用于多种场景。如需更多相关细节，请参考《虚拟化指南》（尤其是以下章节）：

- 《虚拟化指南》，第 1 章 “虚拟化技术”，第 1.2 节 “虚拟化功能”
- 《虚拟化指南》，第 1 章 “虚拟化技术”，第 1.3 节 “虚拟化的优点”

本最佳实践指南提供有助于在您的环境中做出正确选择的建议。指南将根据您的工作负载建议或不建议使用某些选项。修复配置问题和执行微调任务会将 VM Guest 的性能提高到近乎裸机的水平。

2 实施修改之前

2.1 先行备份

更改 VM Guest 或 VM 主机服务器的配置可能会导致数据丢失或状态不稳定。在做出任何更改之前，请务必先对文件、数据、图像等进行备份。如不备份，您将无法在数据丢失或配置错误后恢复原始状态。请勿在生产系统上执行测试或试验。

2.2 测试您的工作负载

虚拟化环境的效率取决于众多因素。本指南将提供一些参考，帮助您在生产环境中配置虚拟化时做出正确选择。做任何事情都无法一劳永逸。在规划、测试和部署虚拟化基础结构时，应将硬件、工作负载、资源容量等所有因素都考虑在内。测试您的虚拟化工作负载对成功实施虚拟化至关重要。

3 建议

3.1 建议使用 libvirt 框架

SUSE 强烈建议使用 libvirt 框架来配置、管理和操作 VM 主机服务器、容器和 VM Guest。该框架提供了一个适用于所有支持的虚拟化技术的统一界面（GUI 和外壳），因此比特定于超级管理程序的工具更容易使用。

我们不建议同时使用 libvirt 和特定于超级管理程序的工具，因为 libvirt 工具集可能无法识别使用特定于超级管理程序的工具进行的更改。有关 libvirt 的详细信息，请参见《虚拟化指南》，第 8 章“启动和停止 libvirtd”。

3.2 qemu-system-i386 与 qemu-system-x86_64 的对比

qemu-system-x86_64 与真正的 64 位 PC 硬件类似，支持运行 32 位或 64 位操作系统的 VM Guest。由于 qemu-system-x86_64 通常也可用于 32 位 Guest 提供更佳性能，因此 SUSE 一般建议对 KVM 上的 32 位和 64 位 VM Guest 都使用 qemu-system-x86_64。已知 qemu-system-i386 可提供更佳性能的方案不受 SUSE 支持。

Xen 也使用 qemu 软件包中的二进制文件，但更倾向于使用 qemu-system-i386，qemu-system-i386 也可用于 32 位和 64 位 Xen VM Guest。为保持与上游 Xen 社区的兼容性，SUSE 建议对 Xen VM Guest 使用 qemu-system-i386。

4 VM 主机服务器配置和资源分配

为 VM Guest 分配资源是管理虚拟机时至关重要的一点。在向 VM Guest 分配资源时，需注意过量使用资源可能会影响 VM 主机服务器和 VM Guest 的性能。如果所有 VM Guest 同时请求它们的所有资源，主机需要能够提供所有这些资源。如果不能，主机的性能将受到负面影响，进而也会对 VM Guest 的性能产生负面影响。

4.1 内存

Linux 以称为页的单元来管理内存。在大多数系统上，默认页大小为 4 KB。Linux 和 CPU 需要知道哪些页属于哪些进程。该信息储存在页表中。如果有大量进程在运行，则需要花更多时间来查找内存映射的位置，因为搜索页表需要时间。为加快搜索速度，发明了 TLB（转换后援缓冲区）技术。但在具有大量内存的系统上，TLB 也无法完全满足需求。为避免回退到普通页表（会导致发生缓存不命中情况，这将耗费大量时间），可以使用大页。使用大页将减少 TLB 开销和找不到 TLB 的情况 (pagewalk)。对于内存为 32 GB ($32 \times 1024 \times 1024 = 33,554,432$ KB)、页大小为 4 KB 的主机，TLB 具有 $33,554,432 / 4 = 8,388,608$ 个项。使用 2 MB (2048 KB) 的页大小时，TLB 只有 $33,554,432 / 2048 = 16,384$ 个项，可大大减少找不到 TLB 的情况。

4.1.1 将 VM 主机服务器和 VM Guest 配置为使用大页

当前 CPU 体系结构支持大于 4 KB 的页，即大页。要确定系统上可用大页的大小（可以是 2 MB 或 1 GB），请查看 `/proc/cpuinfo` 输出中的 `flags` 行中是否包含 `pse` 和/或 `pdpe1gb`。

表 1：确定可用的大页大小

CPU 标志	可用的大页大小
空字符串	没有可用的大页
pse	2 MB
pdpe1gb	1 GB

使用大页可以提升 VM Guest 的性能并减少主机内存的消耗。

默认情况下，系统使用 THP。要使系统上可以使用大页，请在引导时使用 `hugepages=1` 来激活大页，您也可以选择使用 `hugepagesz=2MB`（举例而言）来添加大页大小。



注意：1 GB 大页

1 GB 的页只能在引导时分配，之后将无法释放。

要分配和使用大页表 (HugeTlbPage)，您需要使用正确的权限装入 `hugetlbfs`。



注意：大页的相关限制

大页虽然能够提供最佳性能，但也存在一些缺点。使用大页时，内存气球（请参见第 6.1.3 节“virtio balloon”）、KSM（请参见第 4.1.4 节“KSM 和页共享”）等功能将不可用，并且大页无法交换。

过程 2：配置大页的使用

1. 将 `hugetlbfs` 装入 `/dev/hugepages`：

```
tux > sudo mount -t hugetlbfs hugetlbfs /dev/hugepages
```

2. 要为大页保留内存，请使用 `sysctl` 命令。如果系统的大页大小为 2 MB (2048 KB)，而您想要为 VM Guest 保留 1 GB (1,048,576 KB)，那么池中需要有 $1,048,576/2048=512$ 个页：

```
tux > sudo sysctl vm.nr_hugepages=512
```

该值将写入 `/proc/sys/vm/nr_hugepages`，表示内核的大页池中当前持续存在的大页数量。被任务释放后，持续存在的大页将返回到大页池。

3. 通过运行 `virsh edit CONFIGURATION` 将 `memoryBacking` 元素添加到 VM Guest 配置文件中。

```
<memoryBacking>
  <hugepages/>
</memoryBacking>
```

4. 启动 VM Guest，检查主机是否使用了大页：

```
tux > cat /proc/meminfo | grep HugePages_
HugePages_Total: ①      512
HugePages_Free: ②       92
HugePages_Rsvd: ③       0
HugePages_Surp: ④       0
```

- ① 大页池的大小
- ② 池中尚未分配的大页数量

- ③ 已承诺从池中分配但尚未进行分配的大页数量
- ④ 池中大于 `/proc/sys/vm/nr_hugepages` 中的值的大页数量。最大剩余大页数由 `/proc/sys/vm/nr_overcommit_hugepages` 控制

4.1.2 透明大页

利用透明大页 (THP)，您可以使用 **khugepaged** 内核线程动态分配大页，无需再手动管理大页的分配和使用。THP 对于采用连续内存访问模式的工作负载很有用。在运行采用连续内存访问模式的合成工作负载时，页错误可减少 1000 倍。反之，对于采用稀疏内存访问模式的工作负载（例如数据库），THP 可能表现较差。在这类情况下，比较好的做法可能是通过添加内核参数 `transparent_hugepage=never` 来禁用 THP，然后重建 grub2 配置并重引导。使用以下命令校验是否已禁用 THP：

```
tux > cat /sys/kernel/mm/transparent_hugepage/enabled
always madvise [never]
```

如已禁用，方括号中将会显示值 `never`，如上例中所示。



注意：Xen

THP 在 Xen 下不可用。

4.1.3 特定于 Xen 的内存说明

4.1.3.1 管理 Domain-0 内存

在以前的 SUSE Linux Enterprise Server 版本中，Xen 主机的默认内存分配模式是将所有主机物理内存都分配给 Dom0，并启用自动气球式调节功能。当有其他域启动后，内存会自动从 Dom0 进行气球式调节。此行为总是容易出错，因此强烈建议将其禁用。从 SUSE Linux Enterprise Server 15 SP1 开始，默认已禁用自动气球式调节，而是为 Dom0 分配 10% 的主机物理内存外加 1GB。例如，在物理内存大小为 32GB 的主机上，将为 Dom0 分配 4.2GB 内存。

我们仍支持并建议在 `/etc/default/grub` 中使用 `dom0_mem` Xen 命令行选项（请参见第 7.5 节“在引导时更改内核参数”了解详细信息）。您可以通过将 `dom0_mem` 设置为主机物理内存大小，并在 `/etc/xen/xl.conf` 中启用 `autoballoon` 选项，来恢复旧行为。

4.1.3.2 tmpfs 中的 xenstore

使用 Xen 时，我们建议将 xenstore 数据库放到 `tmpfs` 上。`xm/xend` 和 `xl/libxl` 工具栈以及为域 I/O 设备提供服务的前端和后端驱动程序将 xenstore 作为控制平面。xenstore 上的负载随着运行的域数量的增加而线性增加。如果您预计要在 Xen 主机上托管许多 VM Guest，可以将 xenstore 数据库转移到 `tmpfs` 上，以提高控制平面的总体性能。在 `tmpfs` 上装入 `/var/lib/xenstored` 目录：

```
tux > sudo mount -t tmpfs tmpfs /var/lib/xenstored/
```

4.1.4 KSM 和页共享

内核同业合并 (Kernel Samepage Merging) 是一项内核功能，允许通过共享 VM Guest 共有的数据来减少 VM 主机服务器上消耗的内存。KSM 守护程序 `ksmd` 会定期扫描用户内存，查找内容相同、可由单个写保护页替换的页。要启用 KSM，请运行以下命令：

```
tux > sudo echo 1 > /sys/kernel/mm/ksm/run
```

从 VM Guest 的角度而言，使用 KSM 的优点之一是所有 Guest 内存都受主机匿名内存支持。您可以共享 `pagecache`、`tmpfs` 或 Guest 中分配的任何类型的内存。

KSM 由 `sysfs` 控制。您可以在 `/sys/kernel/mm/ksm/` 中查看 KSM 的值：

- `pages_shared`：正在使用的共享页数量（只读）。
- `pages_sharing`：正在共享页的站点数量（只读）。
- `pages_unshared`：为进行合并而反复检查的唯一页数量（只读）。
- `pages_volatile`：更改太快而认为无法合并的页数量（只读）。
- `full_scans`：已扫描所有可合并区域的次数（只读）。

- `sleep_millisecs`：ksmd 在下次扫描前应休眠的毫秒数。值过低会过度使用 CPU，导致消耗可以用于其他任务的 CPU 时间。我们建议将该值设为 `1000` 以上。
- `pages_to_scan`：在 ksmd 进入睡眠状态之前要扫描的当前页数。值过高会过度使用 CPU。我们建议先将该值设为 `1000`，然后根据在测试部署时观察到的 KSM 结果进行必要的调整。
- `merge_across_nodes`：默认情况下，系统会跨 NUMA 节点合并页。将此选项设为 `0` 会禁用此行为。



注意：使用案例

KSM 技术非常适合在运行同一个应用程序或 VM Guest 的多个实例时过量使用主机内存。当应用程序和 VM Guest 类型不一并且不共享任何共用数据时，最好禁用 KSM。在异类和同类混杂的环境中，可以在主机上启用 KSM，但要对每个 VM Guest 禁用 KSM。可以使用 `virsh edit` 在 VM Guest 的 XML 配置中添加以下内容来禁用 Guest 页共享：

```
<memoryBacking>
  <nosharepages/>
</memoryBacking>
```



警告：避免出现内存不足的情况

尽管 KSM 可以释放主机系统上的一些内存，但是管理员应当保留足够的交换来避免可共享内存减少时内存不足的情况。如果可共享内存的量减少，就会增加物理内存的使用量。



警告：内存访问延迟

默认情况下，KSM 会跨 NUMA 节点合并共用页。如果合并后的共用页现在位于较远的 NUMA 节点上（相对于运行 VM Guest vCPU 的节点），这样可能会降低 VM Guest 的性能。如果在 VM Guest 中发现内存访问延迟增加，请使用 `merge_across_nodes sysfs` 控制禁用跨节点合并：

```
tux > sudo echo 0 > /sys/kernel/mm/ksm/merge_across_nodes
```


4.1.5 VM Guest: 内存热插拔

为了优化主机内存的使用，在需要时为运行中 VM Guest 热插入更多内存可能会很有用。要支持内存热插拔，必须先在 VM Guest 的配置文件中配置 `<maxMemory>` 标记：

```
<maxMemory ❶ slots='16' ❷ unit='KiB'>20971520 ❸</maxMemory>
  <memory ❹ unit='KiB'>1048576</memory>
<currentMemory ❺ unit='KiB'>1048576</currentMemory>
```

- ❶ 为 Guest 分配的运行时最大内存。
- ❷ 可用于向 Guest 添加内存的槽数
- ❸ 有效单位包括：
 - “KB”，代表千字节（1,000 个字节）
 - “k” 或 “KiB”，代表千位二进制字节（1,024 个字节）
 - “MB”，代表兆字节（1,000,000 个字节）
 - “M” 或 “MiB”，代表兆位二进制字节（1,048,576 个字节）
 - “GB”，代表千兆字节（1,000,000,000 个字节）
 - “G” 或 “GiB”，代表千兆位二进制字节（1,073,741,824 个字节）
 - “TB”，代表万亿字节（1,000,000,000,000 个字节）
 - “T” 或 “TiB”，代表万亿位二进制字节（1,099,511,627,776 个字节）
- ❹ 在引导时为 Guest 分配的最大内存
- ❺ 实际为 Guest 分配的内存

要将内存设备热插入到槽中，请创建如下所示的 `mem-dev.xml` 文件：

```
<memory model='dimm'>
  <target>
    <size unit='KiB'>524287</size>
    <node>0</node>
  </target>
</memory>
```

然后使用以下命令附加该文件：

```
tux > virsh attach-device vm-name mem-dev.xml
```

要使用内存设备热插拔，必须至少为 Guest 定义 1 个 NUMA 单元（请参见第 4.6.3.1 节“VM Guest 虚拟 NUMA 拓扑”）。

4.2 交换

系统通常使用交换来储存未充分使用的物理内存（使用率较低或长时间未访问）。为防止系统内存不足，强烈建议设置最小交换。

4.2.1 swappiness

swappiness 设置控制系统的交换行为。它定义如何将内存页交换到磁盘。swappiness 的值较高会导致系统频繁交换。可用的值的范围从 0 到 100。值为 100 时，系统会查找非活动页并将这些页置于交换模式。值为 0 时会禁用交换。

要在在线系统上执行某些测试，可以即时更改 /proc/sys/vm/swappiness 的值，然后检查内存使用情况：

```
tux > sudo echo 35 > /proc/sys/vm/swappiness
```

```
tux > free -h
```

total	used	free	shared	buffers	cached	
Mem:	24616680	4991492	19625188	167056	144340	2152408
-/+ buffers/cache:		2694744	21921936			
Swap:	6171644	0	6171644			

要永久设置 swappiness 值，请在 /etc/sysctl.conf 中添加如下所示的行：

```
vm.swappiness = 35
```

您还可以通过在 VM Guest 的 XML 配置中使用 swap_hard_limit 元素来控制交换。设置此参数并将其用于生产环境中之前，请先进行一些测试，因为如果该值过低，主机可能会终止域。

```
<memtune> ①  
  <hard_limit unit='G'>1</hard_limit> ②
```

```
<soft_limit unit='M'>128</soft_limit> ③  
<swap_hard_limit unit='G'>2</swap_hard_limit> ④  
</memtune>
```

- ① 此元素为域提供内存可调参数。如果忽略，则默认设为操作系统提供的默认值。
- ② Guest 可以使用的最大内存。为避免 VM Guest 上出现任何问题，强烈建议不使用此参数。
- ③ 在内存争用期间强制执行的内存限制。
- ④ VM Guest 可以使用的最大内存加交换。

4.3 I/O

4.3.1 I/O 调度程序

默认的 I/O 调度程序为完全公平排队（Completely Fair Queuing，CFQ）。CFQ 调度程序的主要目标是为请求 I/O 操作的所有进程公平分配磁盘 I/O 带宽。您可以为不同的设备使用不同的 I/O 调度程序。

为了在主机和 VM Guest 上获得更好的性能，可以在 VM Guest 中使用 noop（禁用 I/O 调度程序），而为虚拟化主机使用 deadline 调度程序。

过程 3：在运行时查看和更改 I/O 调度程序

1. 要查看磁盘的当前 I/O 调度程序，请运行以下命令（将 sdX 替换为要查看的磁盘）：

```
tux > cat /sys/block/sdX/queue/scheduler  
noop deadline [cfq]
```

方括号中的值为当前所选的调度程序（在上例中为 cfq）。

2. 您可以使用以下命令在运行时更改调度程序：

```
tux > sudo echo deadline > /sys/block/sdX/queue/scheduler
```

要为系统的所有磁盘永久设置 I/O 调度程序，可以使用内核参数 elevator。分别为 VM 主机服务器和 VM Guest 使用 elevator=deadline 和 elevator=noop 值。有关进一步说明，请参见第 7.5 节“在引导时更改内核参数”。

如果您需要为每个磁盘指定不同的 I/O 调度程序，请创建内容类似于下例的 `/usr/lib/tmpfiles.d/I0_ioscheduler.conf` 文件。它为 `/dev/sda` 定义了 `deadline` 调度程序，为 `/dev/sdb` 定义了 `noop` 调度程序。此功能仅在 SLE 12 和 15 上可用。

```
w /sys/block/sda/queue/scheduler - - - - deadline
w /sys/block/sdb/queue/scheduler - - - - noop
```

4.3.2 异步 I/O

许多虚拟磁盘后端在实施中都使用 Linux 异步 I/O (aio)。默认情况下，aio 环境的最大数量设置为 65536，但如果运行的 VM Guest 达数百个，而它们的虚拟磁盘的输入输出由 Linux 异步 I/O 处理，aio 环境数量可能会超过该上限。在 VM 主机服务器上运行大量 VM Guest 时，请考虑增加 `/proc/sys/fs/aio-max-nr`。

过程 4：在运行时查看和更改 AIO-MAX-NR

1. 要查看当前的 aio-max-nr 设置，请运行以下命令：

```
tux > cat /proc/sys/fs/aio-max-nr
65536
```

2. 您可以使用以下命令在运行时更改 aio-max-nr：

```
tux > sudo echo 131072 > /proc/sys/fs/aio-max-nr
```

要永久设置 aio-max-nr，请向本地 `sysctl` 文件添加一项。例如，向 `/etc/sysctl.d/99-sysctl.conf` 追加以下内容：

```
fs.aio-max-nr = 1048576
```

4.3.3 I/O 虚拟化

SUSE 产品支持各种 I/O 虚拟化技术。下表列出了每种技术的优缺点。有关 I/O 虚拟化的详细信息，请参考《SUSE Linux Enterprise Server 15 SP2 虚拟化指南》中的《虚拟化指南》，第 1 章“虚拟化技术”，第 1.5 节“I/O 虚拟化”一章。

表 2：I/O 虚拟化解决方案

技术	优点	缺点
设备分配（直通）	Guest 直接访问设备	多个 Guest 之间不共享
	性能较高	实时迁移较为复杂
		每个 Guest 的 PCI 设备限制为 8
		服务器上的插槽数有限
全虚拟化 (IDE、SATA、SCSI、e1000)	VM Guest 兼容性	性能较差
	便于实时迁移	模拟操作
半虚拟化 (virtio-blk、virtio-net、virtio-scsi)	性能较好	经修改的 Guest (PV 驱动程序)
	便于实时迁移	
	主机与 VM Guest 高效通信	

4.4 储存和文件系统

VM Guest 的储存空间可以是块设备（例如，物理磁盘上的分区），也可以是文件系统上的映像文件：

表 3：块设备与磁盘映像的对比

技术	优点	缺点
块设备	<ul style="list-style-type: none"> 性能更好 使用标准工具进行管理/磁盘修改 可从主机访问（利与弊） 	<ul style="list-style-type: none"> 设备管理
映像文件	<ul style="list-style-type: none"> 更易于系统管理 可轻松移动、克隆、扩展、备份域 提供用于操作映像的综合工具包 (guestfs) 通过稀疏文件减少开销 通过完全分配来获得最佳性能 	<ul style="list-style-type: none"> 性能比块设备低

有关映像格式和维护映像的详细信息，请参见第 5 节“VM Guest 映像”。

如果您的映像储存在 NFS 共享中，您应检查一些服务器和客户端参数来改进对 VM Guest 映像的访问。

4.4.1 NFS 读取/写入（客户端）

`rszize` 和 `wszize` 选项指定客户端与服务器相互之间来回传递的数据区块的大小。您应确保 NFS 读取/写入大小足够大（尤其是对于较大的 I/O）。请将 `/etc/fstab` 中的 `rszize` 和 `wszize` 参数的值增至 16 KB。这将确保在有任何挂起实例时，所有操作都可以被冻结。

```
nfs_server:/exported/vm_images ① /mnt/images ② nfs ③ rw ④,hard ⑤,sync ⑥,
rszize=8192 ⑦,wszize=8192 ⑧ 0 0
```

① NFS 服务器的主机名和导出路径名。

- ② 将 NFS 导出的共享装入到的位置。
- ③ 这是 `nfs` 安装点。
- ④ 此安装点可以读取/写入模式访问。
- ⑤ 确定 NFS 请求超时后 NFS 客户端的恢复行为。`hard` 是避免数据损坏的最佳选项。
- ⑥ 任何将数据写入该安装点上的文件的系统调用都会导致在系统调用将控制权交回给用户空间之前将数据刷新到服务器。
- ⑦ NFS 客户端从 NFS 服务器上的文件读取数据时，在每个网络 READ 请求中可以接收的最大字节数。
- ⑧ NFS 客户端向 NFS 服务器上的文件写入数据时，在每个网络 WRITE 请求中可以发送的最大字节数。

4.4.2 NFS 线程（服务器）

NFS 服务器应当有足够多的 NFS 线程来处理多线程工作负载。使用 `nfsstat` 工具可获取服务器上的一些 RPC 统计信息：

```
tux > sudo nfsstat -rc
Client rpc stats:
calls      retrans    authrefrsh
6401066    198          0          0
```

如果 `retrans` 等于 0，则表示一切正常。否则，客户端需要重新传输，此时需增大 `/etc/sysconfig/nfs` 中的 `USE_KERNEL_NFSD_NUMBER` 变量并进行相应调整，直到 `retrans` 等于 0。

4.5 CPU

主机 CPU “组件”在指派时将被“转换”为 VM Guest 中的虚拟 CPU。这些组件可能是：

- CPU 处理器：它描述主 CPU 单元，通常有多个核心并且可能支持超线程。
- CPU 核心：一个主 CPU 单元可以提供多个核心，核心的接近性能够加快计算过程并降低能源成本。
- CPU 超线程：此实现用于改进计算的并行化，但其效率不如专用核心。

4.5.1 指派 CPU

所有 VM Guest 的虚拟 CPU 的累积数量超过主机 CPU 的数量时，会发生 CPU 过量使用。当没有过量使用并且每个虚拟 CPU 均匹配 VM 主机服务器上的一个硬件处理器或核心时，可能会达到最佳性能。事实上，VM Guest 在一台过量使用的主机上运行会发生延迟增加，每个 VM Guest 的吞吐量也可能受到负面影响。因此，您应尽量避免过量使用 CPU。

要决定是否允许过量使用 CPU，需要对整体工作负载有较好的先验了解。例如，如果您知道所有 VM Guest 虚拟 CPU 的负载都不会超过 50%，那么您就可以假设将主机过量使用 2 倍时（这意味着在一台具有 64 个 CPU 的主机上，总共有 128 个虚拟 CPU）可以正常工作。另一方面，如果您知道 VM Guest 的所有虚拟 CPU 在大多数情况下都会尝试以 100% 的负载运行，那么即使虚拟 CPU 比主机的 CPU 多一个也属于不适当的配置。

过量使用达到一定的程度时，即虚拟 CPU 的累积数量超过 VM 主机服务器物理核心数量的 8 倍时，很可能会导致系统出现故障和不稳定，因此应当避免出现这一情况。

除非您确切知道一个 VM Guest 需要多少个虚拟 CPU，否则应当从一个开始。比较好的经验法则 是将 VM 中的 CPU 工作负载控制在大约 70%（有关监视工具的信息，请参见《系统分析和微调指南》，第 2 章“系统监视实用程序”，第 2.3 节“流程”）。如果您在 VM Guest 中分配的处理器超过所需，这将会对主机和 Guest 的性能产生负面影响。循环效率将会降低，因为未使用的 vCPU 仍然会导致计时器中断。如果您主要在 VM Guest 上运行单线程应用程序，那么单个虚拟 CPU 是最好的选择。

要是 一个 VM Guest 具有的虚拟 CPU 比 VM 主机服务器具有的 CPU 还多，则一律属于不适当的配置。

4.5.2 VM Guest CPU 配置

本节介绍如何为 VM Guest 选择和配置 CPU 类型。您还将了解如何将虚拟 CPU 固定到主机系统上的物理 CPU。有关虚拟 CPU 配置和微调参数的详细信息，请参考 <https://libvirt.org/formatdomain.html#elementsCPU> 上的 libvirt 文档。

4.5.2.1 虚拟 CPU 型号和功能

可以为每个 VM Guest 单独指定 CPU 型号和拓扑。配置选项包括选择特定的 CPU 模型和排除某些 CPU 功能等。预定义的 CPU 型号列于 `/usr/share/libvirt/cpu_map.xml` 中。采用与主机类似的 CPU 型号和拓扑通常能提供最佳性能。可以通过运行 `virsh capabilities` 来显示主机系统 CPU 型号和拓扑。

请注意，如果更改了默认虚拟 CPU 配置，在将 VM Guest 迁移到硬件不同的主机时，需要关闭 VM Guest。有关 VM Guest 迁移的详细信息，请参见《虚拟化指南》，第 10 章“基本 VM Guest 管理”，第 10.7 节“迁移 VM Guest”。

要为 VM Guest 指定某个特定的 CPU 型号，请向 VM Guest 配置文件添加相应的项。下面的示例配置了具有固定 TSC 功能的 Broadwell CPU：

```
<cpu mode='custom' match='exact'>
  <model>Broadwell</model>
  <feature name='invttsc' />
</cpu>
```

对于与主机物理 CPU 最类似的虚拟 CPU，可以使用 `<cpu mode='host-passthrough'>`。请注意，`host-passthrough` CPU 型号可能与主机物理 CPU 并不完全相似，因为 KVM 默认会屏蔽任何不可迁移的功能。例如，`invttsc` 就不包含在虚拟 CPU 功能集中。虽然 libvirt 允许任意传递 KVM 命令行参数，但它并不直接支持更改默认 KVM 行为。仍以 `invttsc` 为例，您可以在 VM Guest 配置文件中使以下命令行传递来实现主机 CPU（包括 `invttsc`）的传递：

```
<domain type='kvm' xmlns:qemu='http://libvirt.org/schemas/domain/qemu/1.0'>
  <qemu:commandline>
    <qemu:arg value='-cpu' />
    <qemu:arg value='host,migratable=off,+invttsc' />
  </qemu:commandline>
  ...
</domain>
```



注意：host-passthrough 模式

由于 `host-passthrough` 会向虚拟 CPU 公开物理 CPU 细节，因此无法迁移到不同的硬件。有关更多信息，请参见第 4.5.2.3 节“虚拟 CPU 迁移注意事项”。

4.5.2.2 虚拟 CPU 固定

虚拟 CPU 固定用于将虚拟 CPU 线程限制到一组物理 CPU。`vcupin` 元素指定虚拟 CPU 可以使用的物理主机 CPU。如果未设置此元素且未指定 `vcpu` 元素的 `cpuset` 属性，则虚拟 CPU 可以自由使用任何物理 CPU。

虚拟 CPU 固定可提高物理 CPU 缓存命中率，对 CPU 密集型工作负载用处很大。要将虚拟 CPU 固定到特定的物理 CPU，请运行以下命令：

```
tux > virsh vcpupin DOMAIN_ID --vcpu vCPU_NUMBER
VCPU: CPU Affinity
-----
0: 0-7
root # virsh vcpupin SLE15 --vcpu 0 0 --config
```

最后一个命令会在 XML 配置中生成下面一项：

```
<cpupin>
  <vcupin vcpu='0' cpuset='0' />
</cpupin>
```



注意：NUMA 节点上的虚拟 CPU 固定

要将 VM Guest 的 CPU 和内存限制为一个 NUMA 节点，可以在 NUMA 系统上使用虚拟 CPU 固定和内存分配策略。有关 NUMA 微调的详细信息，请参见第 4.6 节“NUMA 微调”。



警告：虚拟 CPU 固定和实时迁移

虽然 `vcupin` 可以提高性能，但它可能会使实时迁移复杂化。有关虚拟 CPU 迁移注意事项的详细信息，请参见第 4.5.2.3 节“虚拟 CPU 迁移注意事项”。

4.5.2.3 虚拟 CPU 迁移注意事项

选择包含所有最新功能的虚拟 CPU 型号可以提高 VM Guest 工作负载的性能，但通常会牺牲可迁移性。除非群集中的所有主机都包含最新的 CPU 功能，否则如果目标主机缺少新功能，迁移便可能会失败。如果比起最新的 CPU 功能，更为重要的是虚拟 CPU 的可迁移性，那么应当使

用标准化的 CPU 型号和功能集。**virsh cpu-baseline** 命令可帮助定义可以跨所有主机迁移的标准化虚拟 CPU。下面的命令（在迁移群集中的每台主机上运行时）会在 `all-hosts-caps.xml` 中说明所有主机功能的集合。

```
tux > sudo virsh capabilities >> all-hosts-cpu-caps.xml
```

您可以使用 **virsh cpu-baseline** 根据 `all-hosts-caps.xml` 中收集的每个主机的功能创建跨所有主机兼容的虚拟 CPU 定义。

```
tux > sudo virsh cpu-baseline all-hosts-caps.xml
```

生成的 CPU 定义可用作 VM Guest 配置文件中的 `cpu` 元素。

在逻辑层面上而言，虚拟 CPU 固定是一种硬件直通形式。固定会将物理资源与虚拟资源进行配对，也可能会给迁移带来问题。例如，如果请求的物理资源在目标主机上不可用，或者源主机与目标主机的 NUMA 拓扑不同，迁移将会失败。有关实时迁移的更多建议，请参见《虚拟化指南》，第 10 章 “基本 VM Guest 管理”，第 10.7.1 节 “迁移要求”。

4.6 NUMA 微调

NUMA 是 Non Uniform Memory Access（非一致性内存访问）的首字母缩写。NUMA 系统具有多个物理 CPU，每个 CPU 都附有本地内存。每个 CPU 还可以访问其他 CPU 的内存（称为“远程内存访问”），但速度比访问本地内存要慢得多。如果调节不当，NUMA 系统会对 VM Guest 性能造成负面影响。本节介绍了在 NUMA 主机上部署 VM Guest 时应当考虑的控制，不过最终微调取决于工作负载。在配置和部署 VM 时，始终要考虑主机拓扑。

SUSE Linux Enterprise Server 包含一个 NUMA 自动平衡器，该平衡器通过将内存放置在与处理它的 CPU 相同的 NUMA 节点上来尽力减少远程内存访问。此外，标准工具（例如 **cgset**）和虚拟化工具（例如 libvirt）提供了将 VM Guest 资源限制为物理资源的机制。

numactl 用于检查主机 NUMA 功能：

```
tux > sudo numactl --hardware
available: 4 nodes (0-3)
node 0 cpus: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 72 73 74 75 76 77 78
79 80 81 82 83 84 85 86 87 88 89
node 0 size: 31975 MB
```

```

node 0 free: 31120 MB
node 1 cpus: 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 90 91 92 93
94 95 96 97 98 99 100 101 102 103 104 105 106 107
node 1 size: 32316 MB
node 1 free: 31673 MB
node 2 cpus: 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 108 109 110
111 112 113 114 115 116 117 118 119 120 121 122 123 124 125
node 2 size: 32316 MB
node 2 free: 31726 MB
node 3 cpus: 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 126 127 128
129 130 131 132 133 134 135 136 137 138 139 140 141 142 143
node 3 size: 32314 MB
node 3 free: 31387 MB
node distances:
node   0   1   2   3
0:  10  21  21  21
1:  21  10  21  21
2:  21  21  10  21
3:  21  21  21  10

```

该 **numactl** 输出显示这是一个具有 4 个节点或单元的 NUMA 系统，每个节点或单元包含 36 个 CPU 和大约 32G 内存。 **virsh capabilities** 也可用于检查系统 NUMA 功能和 CPU 拓扑。

4.6.1 NUMA 平衡

在 NUMA 计算机上，如果 CPU 访问远程内存，将会造成性能损失。NUMA 自动平衡功能会扫描任务的地址空间并取消页映射。这样做可以检测页是否正确放置，或者是否要将数据迁移到运行任务的本地内存节点。在定义的时间间隔内（使用 **numa_balancing_scan_delay_ms** 配置），该任务会扫描其地址空间中的下次扫描大小页数（使用 **numa_balancing_scan_size_mb** 配置）。在到达地址空间的末尾时，扫描器会从头重新开始。

较高的扫描率会导致较高的系统开销，因为必须捕获页错误，并需要迁移数据。但扫描率越高，当工作负载模式更改时，任务的内存迁移到本地节点的速度就越快。这样可最大限度减少远程内存访问对性能造成的影响。以下 **sysctl** 指令控制扫描延迟的阈值和扫描的页数：

```
tux > sudo sysctl -a | grep numa_balancing
```

```
kernel.numa_balancing = 1 ❶  
kernel.numa_balancing_scan_delay_ms = 1000 ❷  
kernel.numa_balancing_scan_period_max_ms = 60000 ❸  
kernel.numa_balancing_scan_period_min_ms = 1000 ❹  
kernel.numa_balancing_scan_size_mb = 256 ❺
```

- ❶ 启用/禁用基于页错误的 NUMA 自动平衡
- ❷ 任务初始派生时用于任务的起始扫描延迟
- ❸ 扫描任务的虚拟内存的最大时间（以毫秒为单位）
- ❹ 扫描任务的虚拟内存的最小时间（以毫秒为单位）
- ❺ 某次给定的扫描需要扫描的页大小（以兆字节为单位）

有关更多信息，请参见《系统分析和微调指南》，第 10 章“自动平衡非一致性内存访问 (NUMA)”。

NUMA 自动平衡的主要目标是在相同节点的内存上重新安排任务（以便 CPU 跟随内存），或者将内存的页复制到相同的节点上（以便内存跟随 CPU）。



警告：任务放置

由于不同的任务之间可能会共享内存，因此并没有可定义运行任务的最佳位置的通用规则。为了获得最佳性能，建议将共享同一节点上的内存的任务分成一组。使用 `# cat /proc/vmstat | grep numa_` 可查看 NUMA 统计信息。

4.6.2 使用 Cpuset 控制器控制内存分配

cgroup cpuset 控制器可用于将进程使用的内存限制在一个 NUMA 节点。可用的 cpuset 内存策略模式有以下三种：

- interleave：此内存放置策略也称为轮替。对于需要将线程本地数据放到相应节点上的作业，此策略可提供显著的改进。当交错目标不可用时，它将被转移到另一个节点。
- bind：此策略仅将内存放到一个节点上，这意味着如果内存不足，分配将会失败。
- preferred：此策略应用首选项来向节点分配内存。如果此节点上没有足够的内存空间，则会回退到另一个节点。

您可以使用 **cgset** 工具（来自 `libcgroup-tools` 软件包）更改内存策略模式：

```
tux > sudo cgset -r cpuset.mems=NODE sysdefault/libvirt/qemu/KVM_NAME/emulator
```

要将页迁移到某个节点，请使用 **migratepages** 工具：

```
tux > migratepages PID FROM-NODE TO-NODE
```

要检查是否一切正常，请使用 **cat /proc/PID/status | grep Cpus**。



注意：内核 NUMA/cpuset 内存策略

有关详细信息，请参见内核 NUMA 内存策略 (https://www.kernel.org/doc/Documentation/vm/numa_memory_policy.txt) 和 cpusets 内存策略 (<https://www.kernel.org/doc/Documentation/cgroup-v1/cpusets.txt>)。另请查看 Libvirt NUMA Tuning（Libvirt NUMA 微调）文档 (<https://libvirt.org/formatdomain.html#elementsNUMATuning>)。

4.6.3 VM Guest：NUMA 相关配置

`libvirt` 允许设置虚拟 NUMA 和内存访问策略。`virt-install` 或 `virt-manager` 不支持配置这些设置，需要使用 `virsh edit` 手动编辑 VM Guest 配置文件来完成该操作。

4.6.3.1 VM Guest 虚拟 NUMA 拓扑

创建类似于主机 NUMA 拓扑的 VM Guest 虚拟 NUMA (vNUMA) 通常可以提高传统的大型可扩展工作负载的性能。可以在 XML 配置中使用 `numa` 元素指定 VM Guest vNUMA 拓扑：

```
<cpu>
...
  <numa>
    <cell ① id="0" ② cpus='0-1' ③ memory='512000' unit='KiB' />
    <cell id="1" cpus='2-3' memory='256000' ④
      unit='KiB' ⑤ memAccess='shared' ⑥ />
```

```
</numa>
...
</cpu>
```

- ① 每个 `cell` 元素指定一个 vNUMA 单元或节点
- ② 所有单元都应具有 `id` 属性，用于在其他配置块中引用所需单元。如果没有，将按从 0 开始的升序为单元指派 ID。
- ③ 属于节点一部分的 CPU 或 CPU 范围
- ④ 节点内存
- ⑤ 指定节点内存时所用的单位
- ⑥ 可选属性，可控制将内存映射为 共享 还是 私用。仅对支持大页的内存有效。

要查找 VM Guest 将其页分配到的位置，请使用 `cat /proc/PID/numa_maps` 和 `cat /sys/fs/cgroup/memory/sysdefault/libvirt/qemu/KVM_NAME/memory.numa_stat`。



警告：NUMA 规范

`libvirt` VM Guest NUMA 规范目前仅适用于 QEMU/KVM。

4.6.3.2 使用 `libvirt` 控制内存分配

如果 VM Guest 具有 vNUMA 拓扑（请参见第 4.6.3.1 节“VM Guest 虚拟 NUMA 拓扑”），便可使用 `numatune` 元素将内存固定到主机 NUMA 节点。此方法目前仅适用于 QEMU/KVM Guest。请参见 [重要：非 vNUMA VM Guest](#) 了解如何配置非 vNUMA VM Guest。

```
<numatune>
  <memory mode="strict" ① nodeset="1-4,^3" ② />
  <memnode ③ cellid="0" ④ mode="strict" nodeset="1"/>
  <memnode cellid="2" placement="strict" ⑤ mode="preferred" nodeset="2"/>
</numatune>
```

- ① 可用的策略包括 `interleave`（类似于轮替）、`strict`（默认）或 `preferred`。
- ② 指定 NUMA 节点。

- ③ 指定每个 Guest NUMA 节点的内存分配策略（如果未定义此元素，则会回退并使用 `memory` 元素）。
- ④ 对要应用设置的 Guest NUMA 节点寻址。
- ⑤ 可以使用 `placement` 属性来指示域进程的内存放置模式，值可以是 `auto` 或 `strict`。

！ 重要：非 vNUMA VM Guest

在非 vNUMA VM Guest 上，将内存固定到主机 NUMA 节点的命令如下例所示：

```
<numatune>
  <memory mode="strict" nodeset="0-1"/>
</numatune>
```

本例中从主机节点 `0` 和 `1` 分配内存。如果无法满足这些内存要求，启动 VM Guest 的操作将会失败。`virt-install` 也通过 `--numatune` 选项支持此配置。

🚫 警告：跨 NUMA 节点的内存和 CPU

您应当避免跨 NUMA 节点分配 VM Guest 内存，并防止虚拟 CPU 跨 NUMA 节点浮动。

5 VM Guest 映像

映像是用于储存 VM Guest 的操作系统和数据的虚拟磁盘。可以使用 `qemu-img` 命令来创建、维护和查询映像。有关 `qemu-img` 工具和示例的详细信息，请参见《虚拟化指南》，第 31 章“Guest 安装”，第 31.2.2 节“创建、转换和检查磁盘映像”。

5.1 VM Guest 映像格式

QEMU 可识别源自其他虚拟化技术的某些储存格式。通过识别这些格式，QEMU 可以利用最初用于在这些其他虚拟化技术下运行的数据储存或整个 Guest。有些格式只支持只读模式。要以读取/写入模式使用它们，请使用 `qemu-img` 将它们转换为完全受支持的 QEMU 储存格式。否则，它们在 QEMU Guest 中将只能用作只读数据储存。

使用 `qemu-img info VMGUEST.IMG` 可获取有关现有映像的信息，例如格式、虚拟大小、物理大小、快照（如果有）。



注意：性能

建议将磁盘映像转换为 raw 或 qcow2 格式以获得较好的性能。



警告：无法压缩加密映像

创建映像时，无法在输出文件中使用压缩 (`-c`) 的同时指定加密选项 (`-e`)。

5.1.1 Raw 格式

- 此格式十分简单，并且容易导出到所有其他模拟器/超级管理程序。
- 它提供的性能最佳（I/O 开销最少）。
- 如果您的文件系统支持空洞（例如 Linux 上的 Ext2 或 Ext3 或者 Windows* 上的 NTFS），那么只有写入扇区将保留空间。
- Raw 格式允许将 VM Guest 映像复制到物理设备 (`dd if=VMGUEST.RAW of=/dev/sda`)。
- 这与 VM Guest 看到的字节对字节相同，因此会浪费大量空间。

5.1.2 qcow2 格式

- 使用此格式可缩小映像大小（这在文件系统不支持空洞时很有用）。
- 它具有可选的 AES 加密（现已弃用）。
- 基于 Zlib 的压缩选项。
- 支持多个 VM 快照（内部、外部）。
- 可提升性能和稳定性。

- 支持更改后备文件。
- 支持一致性检查。
- 性能低于 raw 格式。

l2-cache-size

qcow2 可以为随机读取/写入访问提供与 raw 格式相同的性能，但需要合适的缓存大小。默认情况下，缓存大小设置为 1 MB。这将提供最大 8 GB 磁盘大小的良好性能。如果您需要更大的磁盘大小，则需要调整缓存大小。假设磁盘大小为 64 GB ($64 \times 1024 = 65536$)，则缓存大小需为 $65536 / 8192B = 8 \text{ MB}$ (`-drive format=qcow2,l2-cache-size=8M`)。

群集大小

qcow2 格式提供更改群集大小的功能。值必须介于 512 KB 到 2 MB 之间。较小的群集大小可以改善映像文件的大小，而较大的群集大小通常可提供更好的性能。

预分配

具有预分配元数据的映像最初便比较大，但在其需要增长时可以提升性能。

迟缓的引用计数

为了避免元数据 I/O 并提高性能，系统会延迟更新引用计数。这对于 `cache=writethrough` 的情况尤为有用。此选项不会批量进行元数据更新，但如果由于主机崩溃而必须重建引用计数表，则在下次使用 `qemu-img check -r all` 打开时会自动完成重建。请注意，此过程需要一段时间。

5.1.3 qed 格式

qed 是一种改进型 qcow (QEMU 写入时复制) 格式。由于 qcow2 可提供 qed 的所有优点以及其他功能，因此 qed 现已弃用。

5.1.4 VMDK 格式

VMware 3、4 或 6 映像格式，用于与该产品交换映像。

5.2 覆盖磁盘映像

qcow2 和 qed 格式提供了一种创建基本映像（也称为后备文件）和在基本映像之上覆盖映像的方法。后备文件对于还原到已知状态并丢弃覆盖非常有用。如果向映像写入数据，后备映像将保持不变，所有更改都将记录在覆盖映像文件中。除非使用 `commit` 监视命令（或 `qemu-img commit`），否则后备文件永远都不会修改。

要创建覆盖映像，请运行以下命令：

```
root # qemu-img create -o ❶ backing_file=vmguest.raw ❷,backing_fmt=raw ❸\  
-f ❹ qcow2 vmguest.cow ❺
```

- ❶ 使用 `-o ?` 可获取可用选项的概述。
- ❷ 后备文件名。
- ❸ 指定后备文件的文件格式。
- ❹ 指定 VM Guest 的映像格式。
- ❺ VM Guest 的映像名称，它只会记录与后备文件的差异。



警告：后备映像路径

您不应更改后备映像的路径，否则将需要对其进行调整。路径储存在覆盖映像文件中。要更新路径，应当创建一个从原始路径到新路径的符号链接，然后使用 `qemu-img rebase` 选项。

```
root # ln -sf /var/lib/images/vmguest.raw /var/lib/images/SLE15/  
vmguest.raw  
root # qemu-img rebase ❶ -u ❷ -b ❸ /var/lib/images/vmguest.raw /var/lib/  
images/SLE15/vmguest.cow ❹
```

`rebase` 子命令告知 `qemu-img` 更改后备文件映像。 `-u` 选项会激活不安全模式（请参见下面的注释）。使用 `-b` 来指定要使用的后备映像，映像路径是命令的最后一个参数。

`rebase` 有两种不同的运行模式：

- 安全：这是默认模式，执行真正的变基操作。安全模式是一种较为耗时的操作。
- 不安全：不安全模式 (`-u`) 只更改后备文件名和文件名的格式，而不检查文件的内容。如要重命名或移动后备文件，则应使用此模式。

常见的用法是使用后备文件来启动新的 Guest。假设我们有一个可以使用的 `sle15_base.img` VM Guest（未经任何修改的新安装）。这将是我们的后备文件。现在需要在更新后的系统和具有不同内核的系统上测试一个新软件包。我们可以使用 `sle15_base.img`，通过创建指向此后备文件 (`sle15_base.img`) 的 `qcow2` 覆盖文件来实例化新的 SUSE Linux Enterprise VM Guest。

在本例中，我们将为更新后的系统使用 `sle15_updated.qcow2`，为具有不同内核的系统使用 `sle15_kernel.qcow2`。

要创建这两个精简预配的系统，需使用带 `-b` 选项的 **qemu-img** 命令行：

```
root # qemu-img create -b /var/lib/libvirt/sle15_base.img -f qcow2 \
/var/lib/libvirt/sle15_updated.qcow2
Formatting 'sle15_updated.qcow2', fmt=qcow2 size=17179869184
backing_file='sle15_base.img' encryption=off cluster_size=65536
lazy_refcounts=off nocow=off
root # qemu-img create -b /var/lib/libvirt/sle15_base.img -f qcow2 \
/var/lib/libvirt/sle15_kernel.qcow2
Formatting 'sle15_kernel.qcow2', fmt=qcow2 size=17179869184
backing_file='vmguest-sle15_base.img' encryption=off cluster_size=65536
lazy_refcounts=off nocow=off
```

映像现在便可供使用，您可以在不变动初始 `sle15_base.img` 后备文件的情况下执行测试。所有更改都将储存在新的覆盖映像中。此外，您还可以使用这些新映像作为后备文件来创建新的覆盖。

```
root # qemu-img create -b sle15_kernel.qcow2 -f qcow2 sle15_kernel_TEST.qcow2
```

将 `--backing-chain` 选项与 **qemu-img info** 结合使用时，该命令将会递归返回关于整个后备链的所有信息：

```
root # qemu-img info --backing-chain
/var/lib/libvirt/images/sle15_kernel_TEST.qcow2
```

```
image: sle15_kernel_TEST.qcow2
file format: qcow2
virtual size: 16G (17179869184 bytes)
disk size: 196K
cluster_size: 65536
backing file: sle15_kernel.qcow2
Format specific information:
compat: 1.1
lazy refcounts: false
```

```
image: sle15_kernel.qcow2
file format: qcow2
virtual size: 16G (17179869184 bytes)
disk size: 196K
cluster_size: 65536
backing file: SLE15.qcow2
Format specific information:
compat: 1.1
lazy refcounts: false
```

```
image: sle15_base.img
file format: qcow2
virtual size: 16G (17179869184 bytes)
disk size: 16G
cluster_size: 65536
Format specific information:
compat: 1.1
lazy refcounts: true
```

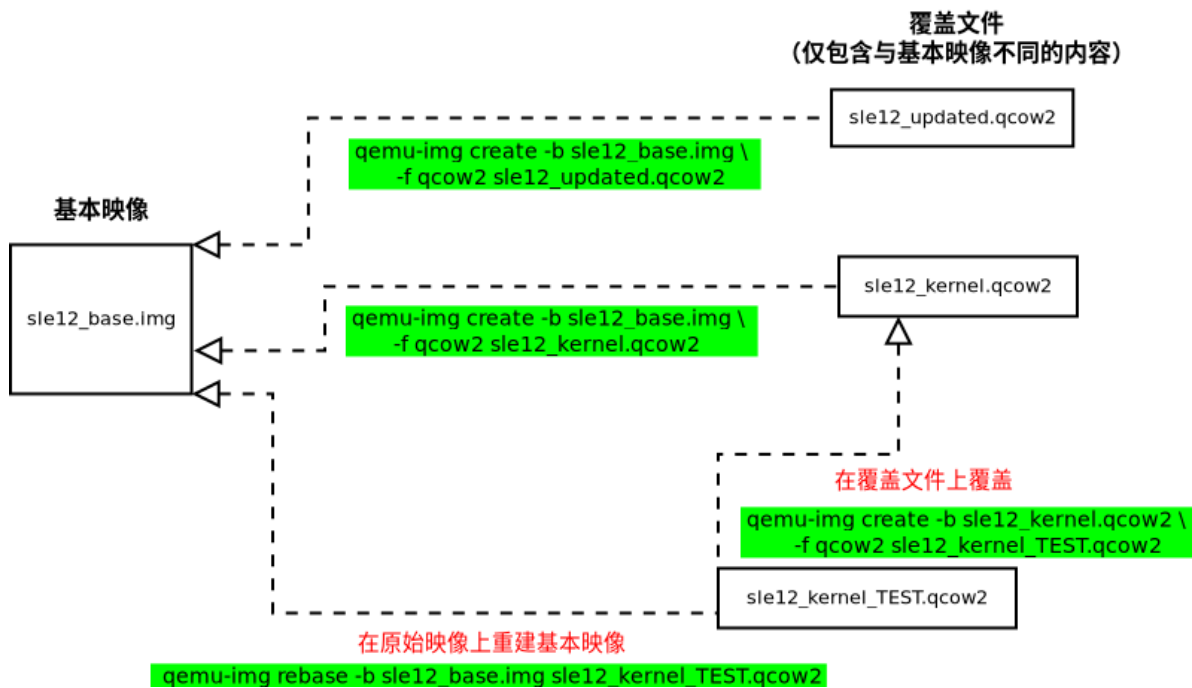


图 1：理解映像覆盖

5.3 打开 VM Guest 映像

要访问映像的文件系统，请使用 `guestfs-tools`。如果系统上没有安装此工具，您可以使用其他 Linux 工具装入映像。应避免访问不可信或未知的 VM Guest 的映像系统，因为这可能会导致安全问题（有关详细信息，请阅读 D. Berrangé 发布的帖子 (<https://www.berrange.com/posts/2013/02/20/a-reminder-why-you-should-never-mount-guest-disk-images-on-the-host-os/>)）。

5.3.1 打开 Raw 映像

过程 5：装入 RAW 映像

1. 要能够装入映像，请找到一台空闲的循环设备。下面的命令会显示第一台未使用的循环设备（在本例中为 `/dev/loop1`）。

```
root # losetup -f
/dev/loop1
```

2. 将映像（在本例中为 `SLE15.raw`）与循环设备相关联：

```
root # losetup /dev/loop1 SLE15.raw
```

3. 通过获取有关循环设备的详细信息来检查映像是否已成功与循环设备关联：

```
root # losetup -l
NAME          SIZE LIMIT  OFFSET  AUTO CLEAR  RO  BACK-FILE
/dev/loop1    0          0        0          0   /var/lib/libvirt/images/
SLE15.raw
```

4. 使用 `kpartx` 查看映像的分区：

```
root # kpartx -a ① -v ② /dev/loop1
add map loop1p1 (254:1): 0 29358080 linear /dev/loop1 2048
```

- ① 添加分区设备映射。
- ② 详细模式。

5. 现在装入映像分区（在下面的示例中，装入到 `/mnt/sle15mount`）：

```
root # mkdir /mnt/sle15mount
root # mount /dev/mapper/loop1p1 /mnt/sle15mount
```



注意：包含 LVM 的 Raw 映像

如果 raw 映像包含 LVM 卷组，您应当使用 LVM 工具装入分区。有关详细信息，请参见第 5.3.3 节“打开包含 LVM 的映像”。

过程 6：卸载 RAW 映像

1. 卸载映像的所有已装入分区，例如：

```
root # umount /mnt/sle15mount
```

2. 使用 `kpartx` 删除分区设备映射：

```
root # kpartx -d /dev/loop1
```

3. 使用 **losetup** 解除设备关联：

```
root # losetup -d /dev/loop1
```

5.3.2 打开 qcow2 映像

过程 7：装入 QCOW2 映像

1. 首先，您需要装载 **nbd**（网络块设备）模块。下面的示例装载了该模块并支持 16 个块设备 (**max_part=16**)。使用 **dmesg** 检查操作是否成功：

```
root # modprobe nbd max_part=16
root # dmesg | grep nbd
[89155.142425] nbd: registered device at major 43
```

2. 使用 **qemu-nbd** 命令将 VM Guest 映像（例如 **SLE15.qcow2**）与 NBD 设备（在下面的示例中为 **/dev/nbd0**）相连。请务必使用空闲的 NBD 设备：

```
root # qemu-nbd -c ❶ /dev/nbd0 ❷ SLE15.qcow2 ❸
```

- ❶ 将 **SLE15.qcow2** 与本地 NBD 设备 **/dev/nbd0** 相连
- ❷ 要使用的 NBD 设备
- ❸ 要使用的 VM Guest 映像



提示：检查是否有空闲的 NBD 设备

要检查 NBD 设备是否空闲，请运行以下命令：

```
root # lsof /dev/nbd0
COMMAND  PID USER  FD  TYPE DEVICE SIZE/OFF  NODE NAME
qemu-nbd 15149 root   10u  BLK   43,0      0t0 47347 /dev/nbd0
```

如果命令生成如上所示的输出，则表示设备处于忙碌状态（非空闲）。还可以通过查看是否存在 **/sys/devices/virtual/block/nbd0/pid** 文件来确认设备是否空闲。

3. 使用 **partprobe** 通知操作系统有关分区表的更改：

```
root # partprobe /dev/nbd0 -s
/dev/nbd0: msdos partitions 1 2
root # dmesg | grep nbd0 | tail -1
[89699.082206] nbd0: p1 p2
```

4. 在上面的示例中，**SLE15.qcow2** 包含 **/dev/nbd0p1** 和 **/dev/nbd0p2** 这两个分区。 在装入这些分区之前，使用 **vgscan** 检查它们是否属于 LVM 卷：

```
root # vgscan -v
Wiping cache of LVM-capable devices
Wiping internal VG cache
Reading all physical volumes. This may take a while...
Using volume group(s) on command line.
No volume groups found.
```

5. 如果没有找到 LVM 卷，可以使用 **mount** 装入分区：

```
root # mkdir /mnt/nbd0p2
# mount /dev/nbd0p1 /mnt/nbd0p2
```

有关如何处理 LVM 卷的信息，请参见第 5.3.3 节“打开包含 LVM 的映像”。

过程 8：卸载 QCOW2 映像

1. 卸载映像的所有已装入分区，例如：

```
root # umount /mnt/nbd0p2
```

2. 断开映像与 **/dev/nbd0** 设备的连接。

```
root # qemu-nbd -d /dev/nbd0
```

5.3.3 打开包含 LVM 的映像

过程 9：装入包含 LVM 的映像

1. 要检查映像是否包含 LVM 组，请使用 **`vgscan -v`**。如果映像包含 LVM 组，命令的输出将如下所示：

```
root # vgscan -v
Wiping cache of LVM-capable devices
Wiping internal VG cache
Reading all physical volumes. This may take a while...
Finding all volume groups
Finding volume group "system"
Found volume group "system" using metadata type lvm2
```

2. 在系统上找到了 **`system`** LVM 卷组。您可以使用 **`vgdisplay VOLUMEGROUPNAME`**（在此处的示例中，**`VOLUMEGROUPNAME`** 为 **`system`**）获取有关此卷的详细信息。应当激活这个卷组以将 LVM 分区作为设备公开，这样系统便能装入它们。使用 **`vgchange`**：

```
root # vgchange -ay -v
Finding all volume groups
Finding volume group "system"
Found volume group "system"
activation/volume_list configuration setting not defined: Checking only
host tags for system/home
Creating system-home
Loading system-home table (254:0)
Resuming system-home (254:0)
Found volume group "system"
activation/volume_list configuration setting not defined: Checking only
host tags for system/root
Creating system-root
Loading system-root table (254:1)
Resuming system-root (254:1)
Found volume group "system"
activation/volume_list configuration setting not defined: Checking only
host tags for system/swap
Creating system-swap
Loading system-swap table (254:2)
```

```
Resuming system-swap (254:2)
Activated 3 logical volumes in volume group system
  3 logical volume(s) in volume group "system" now active
```

3. 卷组中的所有分区都列于 `/dev/mapper` 目录中。现在即可装入它们。

```
root # ls /dev/mapper/system-*
/dev/mapper/system-home /dev/mapper/system-root /dev/mapper/system-swap

root # mkdir /mnt/system-root
root # mount /dev/mapper/system-root /mnt/system-root

root # ls /mnt/system-root/
bin  dev  home  lib64      mnt  proc      root  sbin      srv  tmp  var
boot etc  lib   lost+found opt  read-write run  selinux  sys  usr
```

过程 10：卸载包含 LVM 的映像

1. 使用 `umount` 卸载所有分区

```
root # umount /mnt/system-root
```

2. 使用 `vgchange -an VOLUMEGROUPNAME` 停用 LVM 卷组

```
root # vgchange -an -v system
Using volume group(s) on command line
Finding volume group "system"
Found volume group "system"
Removing system-home (254:0)
Found volume group "system"
Removing system-root (254:1)
Found volume group "system"
Removing system-swap (254:2)
Deactivated 3 logical volumes in volume group system
0 logical volume(s) in volume group "system" now active
```

3. 现在有以下两种选择：

- 如果是 qcow2 映像，按步骤 2 (`qemu-nbd -d /dev/nbd0`) 中所述继续。
- 如果是 raw 映像，按步骤 2 (`kpartx -d /dev/loop1`、`losetup -d /dev/loop1`) 中所述继续。

❗ 重要：检查卸载是否成功

您应使用 `losetup`、`qemu-nbd`、`mount` 或 `vgscan` 等系统命令确认卸载是否成功。如果没有成功卸载，您在使用 VM Guest 时可能会遇到问题，因为它的系统映像会在不同的地方使用。

5.4 文件系统共享

您可以使用 `filesystem` 元素访问 VM Guest 中的主机目录。在下面的示例中，我们将共享 `/data/shared` 目录并将其装入 VM Guest 中。请注意，对于 QEMU/KVM 驱动程序，`accessmode` 参数仅可与 `type='mount'` 搭配使用（`type` 的大多数其他值专用于 LXC 驱动程序）。

```
<filesystem type='mount' ❶ accessmode='mapped' ❷>
  <source dir='/data/shared' ❸>
  <target dir='shared' ❹ />
</filesystem>
```

- ❶ 要装入 VM Guest 的主机目录。
- ❷ 访问模式（安全模式）设置为 `mapped` 将为访问提供超级管理程序的权限。使用 `passthrough` 会以 VM Guest 中用户的权限访问此共享。
- ❸ 要与 VM Guest 共享的路径。
- ❹ 装入命令的路径的名称或标签。

要在 VM Guest 上装入 `shared` 目录，请使用以下命令：在 VM Guest 下，现在需要装入 `target dir='shared'`：

```
root # mkdir /opt/mnt_shared
root # mount shared -t 9p /opt/mnt_shared -o trans=virtio
```

有关详细信息，请参见 [libvirt 文件系统 \(https://libvirt.org/formatdomain.html#elementsFilesystems\)](https://libvirt.org/formatdomain.html#elementsFilesystems) 和 [QEMU 9psetup \(http://wiki.qemu.org/Documentation/9psetup\)](http://wiki.qemu.org/Documentation/9psetup)。

6 VM Guest 配置

6.1 Virtio 驱动程序

为了提高 VM Guest 性能，建议在 VM Guest 中使用半虚拟化驱动程序。此类用于 KVM 的驱动程序的虚拟化标准是 virtio 驱动程序，它专为在虚拟环境中运行而设计。Xen 使用类似的半虚拟化设备驱动程序（例如 Windows* Guest 中的 [VMDP \(https://www.suse.com/products/vmdriverpack/\)](https://www.suse.com/products/vmdriverpack/)）。要想更好地理解这个主题，请参见《虚拟化指南》，第 1 章“虚拟化技术”，第 1.5 节“I/O 虚拟化”。

6.1.1 virtio blk

virtio_blk 是磁盘的 virtio 块设备。要为块设备使用 virtio blk 驱动程序，请指定 bus='virtio' 属性（在 disk 定义中）：

```
<disk type='....' device='disk'>
    ....
    <target dev='vda' bus='virtio' />
</disk>
```

！ 重要：磁盘设备名称

virtio 磁盘设备命名为 /dev/vd[a-z][1-9]。如果从非 virtio 磁盘迁移 Linux Guest，您需要调整 GRUB 配置中的 root= 参数，并重新生成 initrd 文件，否则系统将无法引导。在运行其他操作系统的 VM Guest 上，可能还需要相应地调整或重新安装引导加载程序。

！ 重要：通过 **qemu-system-ARCH** 使用 virtio 磁盘

运行 **qemu-system-ARCH** 时，请使用 `-drive` 选项来向 VM Guest 添加磁盘。请参见《虚拟化指南》，第 31 章 “Guest 安装”，第 31.1 节 “使用 **qemu-system-ARCH** 进行基本安装” 中的示例。 `-hd[abcd]` 选项不适用于 virtio 磁盘。

6.1.2 virtio net

`virtio_net` 是 virtio 网络设备。内核模块应在 Guest 引导时自动在其中装载。您需要启动该服务以使网络可用。

```
<interface type='network'>
  ...
  <model type='virtio' />
</interface>
```

6.1.3 virtio balloon

virtio balloon 用于对 Guest 实施主机内存过量使用。对于 Linux Guest，气球驱动程序在 Guest 内核中运行；而对于 Windows Guest，气球驱动程序包含在 VM DP 软件包中。`virtio_balloon` 是用于从 VM Guest 提供或获取内存的 PV 驱动程序。

- 充气气球：将内存从 Guest 返回到主机内核（对于 KVM）或超级管理程序（对于 Xen）
- 放气气球：Guest 将有更多可用内存

它由 `currentMemory` 和 `memory` 选项控制。

```
<memory unit='KiB'>16777216</memory>
  <currentMemory unit='KiB'>1048576</currentMemory>
  [...]
  <devices>
    <memballoon model='virtio' />
  </devices>
```

您也可以使用 `virsh` 来更改它：

```
tux > virsh setmem DOMAIN_ID MEMORY in KB
```

6.1.4 检查 virtio 是否存在

可以使用以下命令检查 virtio 块 PCI:

```
tux > find /sys/devices/ -name virtio*  
/sys/devices/pci0000:00/0000:00:06.0/virtio0  
/sys/devices/pci0000:00/0000:00:07.0/virtio1  
/sys/devices/pci0000:00/0000:00:08.0/virtio2
```

要查找与 vdX 关联的块设备, 请运行以下命令:

```
tux > find /sys/devices/ -name virtio* -print -exec ls {}/block 2>/dev/null \  
/sys/devices/pci0000:00/0000:00:06.0/virtio0  
/sys/devices/pci0000:00/0000:00:07.0/virtio1  
/sys/devices/pci0000:00/0000:00:08.0/virtio2  
vda
```

要获取有关 virtio 块的详细信息, 请运行以下命令:

```
tux > udevadm info -p /sys/devices/pci0000:00/0000:00:08.0/virtio2  
P: /devices/pci0000:00/0000:00:08.0/virtio2  
E: DEVPATH=/devices/pci0000:00/0000:00:08.0/virtio2  
E: DRIVER=virtio_blk  
E: MODALIAS=virtio:d00000002v00001AF4  
E: SUBSYSTEM=virtio
```

要检查所有正在使用的 virtio 驱动程序, 请运行以下命令:

```
tux > find /sys/devices/ -name virtio* -print -exec ls -l {}/driver 2>/dev/null \  
\  
/sys/devices/pci0000:00/0000:00:06.0/virtio0  
lrwxrwxrwx 1 root root 0 Jun 17 15:48 /sys/devices/pci0000:00/0000:00:06.0/  
virtio0/driver -> ../../../../bus/virtio/drivers/virtio_console  
/sys/devices/pci0000:00/0000:00:07.0/virtio1  
lrwxrwxrwx 1 root root 0 Jun 17 15:47 /sys/devices/pci0000:00/0000:00:07.0/  
virtio1/driver -> ../../../../bus/virtio/drivers/virtio_balloon  
/sys/devices/pci0000:00/0000:00:08.0/virtio2
```

```
lrwxrwxrwx 1 root root 0 Jun 17 14:35 /sys/devices/pci0000:00/0000:00:08.0/
virtio2/driver -> ../../../../bus/virtio/drivers/virtio_blk
```

6.1.5 查找设备驱动程序选项

Virtio 设备和其他驱动程序具有各种选项。要列出所有选项，请使用 `qemu-system-ARCH` 命令的 `help` 参数。

```
tux > qemu-system-x86_64 -device virtio-net,help
virtio-net-pci.ioeventfd=on/off
virtio-net-pci.vectors=uint32
virtio-net-pci.indirect_desc=on/off
virtio-net-pci.event_idx=on/off
virtio-net-pci.any_layout=on/off
.....
```

6.2 Cirrus 视频驱动程序

为了获得 16 位色、高兼容性和更佳性能，建议使用 `cirrus` 视频驱动程序。



注意：libvirt

`libvirt` 会忽略 `vram` 值，因为视频大小已硬编码在 QEMU 中。

```
<video>
  <model type='cirrus' vram='9216' heads='1' />
</video>
```

6.3 更好的熵

Virtio RNG（随机数字生成器）是作为硬件 RNG 设备向 Guest 公开的半虚拟化设备。在主机端，如果没有硬件支持，可以将其连接到多个熵来源之一（包括一台真正的硬件 RNG 设备和主机的 `/dev/random`）。Linux 内核包含 2.6.26 和更高版本的设备的 Guest 驱动程序。

系统熵收集自各种非确定性硬件事件，主要用于加密应用程序。虚拟随机数字生成器设备（半虚拟化设备）允许主机将熵传递到 VM Guest 操作系统。这将在 VM Guest 中产生更好的熵。要使用 Virtio RNG，请在 **virt-manager** 中或直接在 VM Guest 的 XML 配置中添加 **RNG** 设备：

```
<devices>
  <rng model='virtio'>
    <backend model='random'>/dev/random</backend>
  </rng>
</devices>
```

主机现在应该在使用 **/dev/random**：

```
tux > lsof /dev/random
qemu-syst 4926 qemu      6r   CHR    1,8      0t0 8199 /dev/random
```

在 VM Guest 上，可使用以下命令检查熵的来源：

```
tux > cat /sys/devices/virtual/misc/hw_random/rng_available
```

可使用以下命令检查当前用于熵的设备：

```
tux > cat /sys/devices/virtual/misc/hw_random/rng_current
virtio_rng.0
```

您应在 VM Guest 上安装 **rng-tools** 软件包，启用该服务，然后将其启动。在 SUSE Linux Enterprise Server 15 下，执行以下命令：



```
root # zypper in rng-tools
root # systemctl enable rng-tools
root # systemctl start rng-tools
```

6.4 禁用未使用的工具和设备

对每台主机应仅使用一种虚拟化技术。例如，不要在同一台计算机上使用 KVM 和容器。否则可能会出现可用资源减少、安全风险增加，以及软件更新队列变长的问题。即使仔细配置了分配给每种技术的资源量，主机也可能会出现整体可用性降低和性能下降的情况。

应尽量减少主机上可用的软件和服务的数量。大多数默认安装的操作系统都没有针对 VM 的使用进行优化。因此请安装真正需要的组件，并去除 VM Guest 中的所有其他组件。

Windows* Guest:

- 禁用屏幕保护程序
- 去除所有图像效果
- 禁用硬盘索引（如不需要）
- 检查已启动的服务列表并禁用不需要的服务
- 检查并去除所有不需要的设备
- 禁用系统更新（如不需要），或对其进行配置以避免重引导或关闭主机时出现任何延迟
- 检查防火墙规则
- 适当地安排备份和防病毒更新
- 安装 VMDP (<https://www.suse.com/products/vmdriverpack/>)  半虚拟化驱动程序以获得最佳性能
- 查看操作系统建议，例如[优化 Windows 以提高性能](http://windows.microsoft.com/en-us/windows/optimize-windows-better-performance#optimize-windows-better-performance=windows-7) (<http://windows.microsoft.com/en-us/windows/optimize-windows-better-performance#optimize-windows-better-performance=windows-7>)  网页上的建议。

Linux Guest:

- 去除或不要启动 X Window 系统（如不需要）
- 检查已启动的服务列表并禁用不需要的服务
- 查看有关可实现更佳性能的内核参数的操作系统建议
- 仅安装真正需要的软件
- 优化可预测任务（系统更新、硬盘检查等）的安排

6.5 更新 Guest 计算机类型

QEMU 计算机类型定义与迁移和会话管理尤为相关的体系结构细节。随着对 QEMU 的更改或改进，新的计算机类型将会增加。出于兼容性原因，旧计算机类型仍然受支持，但是为了利用改进的功能，我们建议在升级时一律迁移到最新的计算机类型。

为 Linux Guest 更改 Guest 的计算机类型基本上是透明的。对于 Windows* Guest，我们建议截取 Guest 的快照或备份 Guest，以防出现 Windows* 在其检测到的更改的计算机类型上运行时有问题，随后用户决定还原到创建 Guest 时使用的原始计算机类型的情况。



注意：更改计算机类型

相关文档，请参见《虚拟化指南》，第 15 章 “使用 **virsh** 配置虚拟机”，第 15.2 节 “更改计算机类型”。

7 特定于 VM Guest 的配置和设置

7.1 ACPI 测试

能否更改 VM Guest 的状态在很大程度上取决于操作系统。在生产环境中使用 VM Guest 之前，务必要测试此功能。例如，大多数 Linux 操作系统默认会禁用此功能，因此需要启用此操作（通常通过 PolKit）。

必须在 Guest 中启用 ACPI 才能正常关机。要检查是否已启用 ACPI，请运行以下命令：

```
tux > virsh dumpxml VMNAME | grep acpi
```

如果没有列显任何内容，则说明计算机上没有启用 ACPI。使用 **virsh edit** 在 XML 的 `<domain>` 下添加以下内容：

```
<features>
  <acpi/>
</features>
```

如果 ACPI 是在 Windows Server* Guest 安装期间启用的，那么仅在 VM Guest 配置中开启它并不够。有关更多信息，请参见 <https://support.microsoft.com/en-us/kb/309283>。

无论 VM Guest 的配置如何，始终都可以从 Guest 操作系统内部实现正常关机。

7.2 键盘布局

尽管您可以使用 **qemu-system-ARCH** 命令来指定键盘布局，但仍建议在 `libvirt` XML 文件中配置键盘布局。要在使用 `vnc` 连接到远程 VM Guest 时更改键盘布局，应编辑 VM Guest XML 配置文件。例如，要添加 `en-us` 键映射，请在 `<devices>` 部分添加：

```
<graphics type='vnc' port='-1' autoport='yes' keymap='en-us' />
```

检查 `vncdisplay` 配置并连接到 VM Guest：

```
tux > virsh vncdisplay sles15 127.0.0.1:0
```

7.3 Spice 默认监听 URL

如果主机上除了 `lo` 以外没有为其他网络接口指派 IPv4 地址，`spice` 服务器监听的默认地址将不起作用。将会出现如下错误：

```
tux > virsh start sles15
error: Failed to start domain sles15
error: internal error: process exited while connecting to monitor:
((null):26929): Spice-Warning **: reds.c:2330:reds_init_socket:
getaddrinfo(127.0.0.1,5900): Address family for hostname not supported
2015-08-12T11:21:14.221634Z qemu-system-x86_64: failed to initialize spice
server
```

要修复此问题，可以使用本地 IPv6 地址 `:::1` 更改 `/etc/libvirt/qemu.conf` 中的默认 `spice_listen` 值。您也可以为每个 VM Guest 更改 `spice` 服务器监听地址，使用 **virsh edit** 向 `graphics type='spice'` 元素添加监听 XML 属性：

```
<graphics type='spice' listen=':::1' autoport='yes' />>
```

7.4 XML 到 QEMU 命令行

有时，使用 QEMU 命令行从 XML 文件启动 VM Guest 可能很有用。

```
tux > virsh domxml-to-native ❶ qemu-argv ❷ SLE15.xml ❸
```

- ❶ 将域 XML 格式的 XML 文件转换为本机 Guest 配置
- ❷ 对于 QEMU/KVM 超级管理程序，格式参数需要设为 qemu-argv
- ❸ 要使用的域 XML 文件

```
tux > sudo virsh domxml-to-native qemu-argv /etc/libvirt/qemu/SLE15.xml
LC_ALL=C PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin \
  QEMU_AUDIO_DRV=none /usr/bin/qemu-system-x86_64 -name SLE15 -machine \
  pc-i440fx-2.3,accel=kvm,usb=off -cpu SandyBridge -m 4048 -realtime \
  mlock=off -smp 4,sockets=4,cores=1,threads=1 -uuid \
  8616d00f-5f05-4244-97cc-86aeaed8aea7 \
  -no-user-config -nodefaults -chardev socket,id=charmonitor,path=/var/lib/ \
  libvirt/qemu/SLE15.monitor,server,nowait \
  -mon chardev=charmonitor,id=monitor,mode=control -rtc base=utc,driftfix=slew \
  \
  -global kvm-pit.lost_tick_policy=discard -no-hpet \
  -no-shutdown -global PIIX4_PM.disable_s3=1 -global PIIX4_PM.disable_s4=1 \
  -boot strict=on -device ich9-usb-ehci1,id=usb,bus=pci.0,addr=0x4.0x7 \
  -device ich9-usb- \
  uhci1,masterbus=usb.0,firstport=0,bus=pci.0,multifunction=on,addr=0x4 \
  -device ich9-usb-uhci2,masterbus=usb.0,firstport=2,bus=pci.0,addr=0x4.0x1 \
  -device ich9-usb-uhci3,masterbus=usb.0,firstport=4,bus=pci.0,addr=0x4.0x2 \
  -drive file=/var/lib/libvirt/images/SLE15.qcow2,if=none,id=drive-virtio- \
  disk0,format=qcow2,cache=none \
  -device virtio-blk-pci,scsi=off,bus=pci.0,addr=0x6,drive=drive-virtio- \
  disk0,id=virtio-disk0,bootindex=2 \
  -drive if=none,id=drive-ide0-0-1,readonly=on,format=raw \
  -device ide-cd,bus=ide.0,unit=1,drive=drive-ide0-0-1,id=ide0-0-1 -netdev \
  tap,id=hostnet0 \
  -device virtio-net- \
  pci,netdev=hostnet0,id=net0,mac=52:54:00:28:04:a9,bus=pci.0,addr=0x3,bootindex=1 \
  \
  -chardev pty,id=charserial0 -device isa-serial,chardev=charserial0,id=serial0 \
  \
  -vnc 127.0.0.1:0 -device cirrus-vga,id=video0,bus=pci.0,addr=0x2 \
  -device virtio-balloon-pci,id=balloon0,bus=pci.0,addr=0x5 -msg timestamp=on
```

7.5 在引导时更改内核参数

7.5.1 SUSE Linux Enterprise 11

要在引导时更改 SLE 11 产品的值，需要修改 `/boot/grub/menu.lst` 文件，在其中添加 `OPTION=parameter`。然后重引导系统。

7.5.2 SUSE Linux Enterprise 12 和 15

要在引导时更改 SLE 12 和 15 产品的值，需要修改 `/etc/default/grub` 文件。查找以 `GRUB_CMDLINE_LINUX_DEFAULT` 开头的变量，然后在末尾添加 `OPTION=parameter`（如已存在，则更改为使用正确的值）。

现在需要重新生成 `grub2` 配置：

```
# grub2-mkconfig -o /boot/grub2/grub.cfg
```

然后重引导系统。

7.6 在 XML 配置中添加设备

要基于 XML 文件创建新的 VM Guest，可使用特殊标记 `qemu:commandline` 来指定 QEMU 命令行。例如，要添加 `virtio-balloon-pci`，请在 XML 配置文件的末尾（`</domain>` 标记之前）添加此块：

```
<qemu:commandline>
  <qemu:arg value='-device' />
  <qemu:arg value='virtio-balloon-pci,id=balloon0' />
</qemu:commandline>
```

8 超级管理程序与容器的对比

表 4：超级管理程序与容器的对比

功能	超级管理程序	容器
技术	模拟物理计算环境	使用内核主机
系统层级别	由虚拟化层（超级管理程序）管理	依赖于内核名称空间和 cgroup
级别（层）	硬件级别	软件级别
可用的虚拟化模式	FV 或 PV	无，只有用户空间
安全性	强	警告：安全性非常低
限制	完全隔离	警告：主机内核（操作系统必须与内核版本兼容）
操作系统	任何操作系统	仅限 Linux（必须与“内核”兼容）
系统类型	需要完整操作系统	范围是 Linux 实例
引导时间	启动缓慢（操作系统延迟）	启动非常快
开销	高	非常低
高效	视操作系统而定	效率很高
与主机共享	警告：因相互隔离而较为复杂	共享很容易（主机能够看到一切，容器可以看到自己的对象）
迁移	支持迁移（实时模式）	警告：无法迁移

8.1 兼顾两种模式的优点

即使上表似乎表明无法以高度安全的方式运行单个应用程序，但从 SUSE Linux Enterprise Server 12 SP1 开始，**virt-sandbox** 将允许在 KVM Guest 中运行单个应用程序。**virt-sandbox** 使用极简根文件系统引导 Linux 内核中的任何命令。

Guest 根文件系统可以是以只读方式装入的根文件系统，也可以是磁盘映像。下面的步骤演示如何设置使用 qcow2 磁盘映像作为根文件系统的沙箱。

1. 使用 **qemu-img** 创建磁盘映像：

```
root # qemu-img create -f qcow2 rootfs.qcow2 6G
```

2. 格式化磁盘映像：

```
root # modprobe nbd ❶  
root # /usr/bin/qemu-nbd --format qcow2 -n -c /dev/nbd0 $PWD/test-  
base.qcow2 ❷  
root # mkfs.ext3 /dev/nbd0 ❸
```

- ❶ 确保已装载 nbd 模块。该模块默认不会装载，仅用于格式化 qcow 映像。
- ❷ 为 qcow2 映像创建 NBD 设备。此设备随后将像任何其他块设备一样运行。示例中使用的是 `/dev/nbd0`，但您也可以使用任何其他空闲 NBD 设备。
- ❸ 直接格式化磁盘映像。请注意，目前尚未创建分区表，**virt-sandbox** 将映像视为分区，而非磁盘。
可使用的分区格式有限。引导沙箱的 Linux 内核需要内置相应的功能。沙箱启动时也可使用 Ext4 模块。

3. 现在填充新格式化的映像：

```
root # guestmount -a base.qcow2 -m /dev/sda:/ /mnt ❶  
  
root # zypper --root /mnt ar cd:///?devices=/dev/dvd SLES15_DVD  
root # zypper --root /mnt in -t pattern Minimal ❷  
  
root # guestunmount /mnt ❸
```

- ❶ 使用 **guestfs** 工具装入 qcow2 映像。

- 2 使用带 `--root` 参数的 `zypper` 添加 SUSE Linux Enterprise Server 储存库，并在磁盘映像中安装 `Minimal` 模式。任何额外的软件包或配置更改都应在此步骤执行。



注意：使用后备链

要在几个沙箱之间共享根文件系统，请按第 5.2 节“覆盖磁盘映像”中所述使用公用磁盘映像作为后备链来创建 `qcow2` 映像。

- 3 卸载 `qcow2` 映像。

4. 使用 `virt-sandbox` 运行沙箱。这个命令有许多有趣的选项，请阅读其手册页了解所有选项。该命令可以 `root` 或非特权用户的身份运行。

```
root # virt-sandbox -n NAME \  
-m host-image:/$PWD/rootfs.qcow2 \ ①  
-m host-bind:/srv/www=/guests/www \ ②  
-m ram:/tmp=100MiB \  
-m ram:/run=100MiB \ ③  
-N source=default,address=192.168.122.12/24 \ ④  
-- \  
/bin/sh
```

- ① 将创建的磁盘映像装入为根文件系统。请注意，如果没有任何映像装入为 `/`，主机根文件系统会作为 Guest 根文件系统以只读模式装入。
`host-image` 装入并不会专为根文件系统保留，它可以用于装入 Guest 中任何位置的任何磁盘映像。
- ② 使用 `host-bind` 装入，可非常方便地在主机与 Guest 之间共享文件和目录。在本例中，主机目录 `/guests/www` 在沙箱中装入为 `/srv/www`。
- ③ RAM 装入定义沙箱中的 `tmpfs` 装入。
- ④ 网络使用在 `libvirt` 中定义的网络。以非特权用户身份运行时，可以省略源，并会使用 KVM 用户网络功能。使用此选项需要安装 `dhcp-client` 和 `iproute2` 软件包，它们包含在 SUSE Linux Enterprise Server `Minimal` 模式中。

9 Xen：将半虚拟 (PV) Guest 转换为全虚拟 (FV/HVM) Guest

本章介绍如何将 Xen 半虚拟机转换为 Xen 全虚拟机。

过程 11：GUEST 端

要在 FV 模式下启动 Guest，必须在 Guest 中执行以下步骤。

1. 在转换 Guest 之前，安装所有待应用的补丁并重引导 Guest。
2. FV 计算机使用 `-default` 内核。如果尚未安装此内核，请安装 `kernel-default` 软件包（在 PV 模式下运行时）。
3. PV 计算机通常使用 `vda*` 这样的磁盘名称。这些名称必须更改为 FV `hd*` 语法。此更改必须在以下文件中进行：

- `/etc/fstab`
- `/boot/grub/menu.lst`（仅 SLES 11）
- `/boot/grub*/device.map`
- `/etc/sysconfig/bootloader`
- `/etc/default/grub`（仅 SLES 12 和 15）



注意：建议使用 UUID

应当在 `/etc/fstab` 中使用 UUID 或逻辑卷。使用 UUID 能够简化关联的网络储存、多路径和虚拟化的使用。要确定磁盘的 UUID，请使用 `blkid` 命令。

4. 为了避免在使用所需模块重新生成 `initrd` 时出现任何错误，您可以创建一个从 `/dev/hda2` 到 `/dev/xvda2` 等的符号链接，具体做法是使用 `ln`：

```
ln -sf /dev/xvda2 /dev/hda2
ln -sf /dev/xvda1 /dev/hda1
.....
```

5. PV 和 FV 计算机使用不同的磁盘和网络驱动程序模块。必须手动将这些 PV 模块添加到 `initrd`。预期的模块是 `xen-vbd`（用于磁盘）和 `xen-vnif`（用于网络）。这些是全虚拟化 VM Guest 仅有的 PV 驱动程序。所有其他模块（例如 `ata_piix`、`ata_generic` 和 `libata`）应该会自动添加。

提示：将模块添加到 `initrd`

- 在 SLES 11 上，可以将模块添加到 `/etc/sysconfig/kernel` 文件中的 `INITRD_MODULES` 行。例如：

```
INITRD_MODULES="xen-vbd xen-vnif"
```

运行 `mkinitrd` 以构建包含这些模块的新 `initrd`。

- 在 SLES 12 和 15 上，打开或创建 `/etc/dracut.conf.d/10-virt.conf`，并通过添加下列所示的行使用 `force_drivers` 来添加这些模块（注意前导空格）。

```
force_drivers+=" xen-vbd xen-vnif"
```

运行 `dracut -f --kver KERNEL_VERSION-default` 以构建包含所需模块的新 `initrd`（用于内核的默认版本）。

注意：确定您的内核版本

使用 `uname -r` 命令可获取系统上当前使用的版本。

6. 在关闭 Guest 之前，使用 `yast bootloader` 将默认引导参数设置为 `-default` 内核。
7. 在 SUSE Linux Enterprise Server 11 下，如果 Guest 上在运行 X 服务器，您需要调整 `/etc/X11/xorg.conf` 文件来调整 X 驱动程序。搜索 `fbdev` 并将其更改为 `cirrus`。

```
Section "Device"
    Driver      "cirrus"
```

.....
EndSection



注意：SUSE Linux Enterprise Server 12/15 和 Xorg

在 SUSE Linux Enterprise Server 12/15 下，Xorg 将自动调整 X 服务器能够正常运行所需的驱动程序。

8. 关闭 Guest。

过程 12：主机端

下面的步骤介绍必须在主机上执行的操作。

1. 要以 FV 模式启动 Guest，必须修改 VM 的配置以匹配 FV 配置。使用 **virsh edit [DOMAIN]** 可轻松编辑 VM 的配置。建议进行以下更改：

- 确保将 OS 部分中的 **machine**、**type** 和 **loader** 项从 **xenpv** 更改为 **xenfv**。更新后的 OS 部分应如下所示：

```
<os>
    <type arch='x86_64' machine='xenfv'>hvm</type>
    <loader>/usr/lib/xen/boot/hvmloader</loader>
    <boot dev='hd' />
</os>
```

- 在 OS 部分，去除所有特定于 PV Guest 的内容：

- `<bootloader>pygrub</bootloader>`
- `<kernel>/usr/lib/grub2/x86_64-xen/grub.xen</kernel>`
- `<cmdline>xen-fbfront.video=4,1024,768</cmdline>`

- 在 **devices** 部分，采用以下形式添加 **qemu** 模拟器：

```
<emulator>/usr/lib/xen/bin/qemu-system-i386</emulator>
```

- 更新磁盘配置，使目标设备和总线使用 FV 语法。这需要将 xen 磁盘总线替换为 ide，并将 vda 目标设备替换为 hda。更改应如下所示：

```
<target dev='hda' bus='ide' />
```

- 将鼠标和键盘的总线从 xen 更改为 ps2。另外添加一个新的 USB 绘图板设备：

```
<input type='mouse' bus='ps2' />
    <input type='keyboard' bus='ps2' />
<input type='tablet' bus='usb' />
```

- 将控制台目标类型从 xen 更改为 serial：

```
<console type='pty'>
    <target type='serial' port='0' />
</console>
```

- 将视频配置从 xen 更改为 cirrus，其中 VRAM 大小为 8 MB：

```
<video>
    <model type='cirrus' vram='8192' heads='1' primary='yes' />
</video>
```

- 如果需要，向 VM 的功能添加 acpi 和 apic：

```
<features>
    <acpi />
    <apic />
</features>
```

2. 启动 Guest（使用 virsh 或 virt-manager）。如果 Guest 运行的是 kernel-default（通过 uname -a 校验），计算机将以全虚拟模式运行。



注意：guestfs-tools

要编写此过程的脚本，或直接在磁盘映像上工作，可以使用 guestfs-tools 套件（有关详细信息，请参见《虚拟化指南》，第 19 章 “libguestfs”，第 19.3 节 “Guestfs 工具”）。有许多工具可以帮助修改磁盘映像。

10 外部参考信息

- 使用 KSM 增加内存密度 (<https://kernel.org/doc/ols/2009/ols2009-pages-19-28.pdf>) ↗
- linux-kvm.org KSM (<http://www.linux-kvm.org/page/KSM>) ↗
- KSM 的内核文档 (<https://www.kernel.org/doc/Documentation/vm/ksm.txt>) ↗
- ksm - 适用于 linux v4 的动态页共享驱动程序 (<https://lwn.net/Articles/329123/>) ↗
- 内存气球 (<http://www.espenbraastad.no/post/memory-ballooning/>) ↗
- libvirt virtio (<http://wiki.libvirt.org/page/Virtio>) ↗
- CFQ 的内核文档 (<https://www.kernel.org/doc/Documentation/block/cfq-iosched.txt>) ↗
- sysctl 的文档 (<https://www.kernel.org/doc/Documentation/sysctl/kernel.txt>) ↗
- LWN 随机数字 (<https://lwn.net/Articles/525459/>) ↗
- Khoa Huynh 博士，IBM Linux 技术中心 (http://events.linuxfoundation.org/sites/events/files/slides/CloudOpen2013_Khoa_Huynh_v3.pdf) ↗
- 内核参数 (<https://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/plain/Documentation/admin-guide/kernel-parameters.txt>) ↗
- 大页管理 (Mel Gorman) (<http://lwn.net/Articles/374424/>) ↗
- 内核 hugetlbpage (<https://www.kernel.org/doc/Documentation/vm/hugetlbpage.txt>) ↗