



Virtualization Best Practices

SUSE Linux Enterprise Server 12 SP2

Publication Date: March 31, 2017

Contents

- 1 Virtualization Scenarios 2
- 2 Before You Apply Modifications 2
- 3 Recommendations 3
- 4 VM Host Server Configuration and Resource Allocation 3
- 5 VM Guest Images 24
- 6 VM Guest Configuration 36
- 7 VM Guest-Specific Configurations and Settings 42
- 8 Hypervisors Compared to Containers 46
- 9 Xen: Moving from PV to FV 49
- 10 External References 50

1 Virtualization Scenarios

Virtualization offers a lot of capabilities to your environment. It can be used in multiple scenarios. To get more details about [Virtualization Capabilities \(https://www.suse.com/documentation/sles-12/book_virt/data/sec_virtualization_introduction_capabilities.html\)](https://www.suse.com/documentation/sles-12/book_virt/data/sec_virtualization_introduction_capabilities.html) and [Virtualization Benefits \(https://www.suse.com/documentation/sles-12/book_virt/data/sec_virtualization_introduction_benefits.html\)](https://www.suse.com/documentation/sles-12/book_virt/data/sec_virtualization_introduction_benefits.html), refer to the [Virtualization Guide \(https://www.suse.com/documentation/sles-12/book_virt/data/book_virt.html\)](https://www.suse.com/documentation/sles-12/book_virt/data/book_virt.html).

This best practice guide will provide advice for making the right choice in your environment. It will recommend or discourage the usage of options depending on your workload. Fixing configuration issues and performing tuning tasks will increase the performance of VM Guest's near to bare metal.

2 Before You Apply Modifications

2.1 Back Up First

Changing the configuration of the VM Guest or the VM Host Server can lead to data loss or an unstable state. It is really important that you do backups of files, data, images etc. before making any changes. Without backups you cannot restore the original state after a data loss or a misconfiguration. Do not perform tests or experiments on production systems.

2.2 Test Your Workloads

The efficiency of a virtualization environment depends on many factors. This guide provides a reference for helping to make good choices when configuring virtualization in a production environment. Nothing is *carved in stone*. Hardware, workloads, resource capacity, etc. should all be considered when planning, testing, and deploying your virtualization infra-structure. Testing your virtualized workloads is vital to a successful virtualization implementation.

3 Recommendations

3.1 Prefer the libvirt Framework

SUSE strongly recommends using the libvirt framework to configure, manage, and operate VM Host Servers, containers and VM Guest. It offers a single interface (GUI and shell) for all supported virtualization technologies and therefore is easier to use than the hypervisor-specific tools.

Using libvirt and hypervisor-specific tools at the same time is not recommended, because changes done with the hypervisor-specific tools may not be recognized by the libvirt tool set. See https://www.suse.com/documentation/sles-12/book_virt/data/cha_libvirt_overview.html for more information on libvirt.

3.2 qemu-system-i386 Compared to qemu-system-x86_64

Similar to real 64-bit PC hardware, qemu-system-x86_64 supports VM Guests running a 32-bit or a 64-bit operating system. Because qemu-system-x86_64 usually also provides better performance for 32-bit guests, SUSE generally recommends using qemu-system-x86_64. Scenarios where qemu-system-i386 is known to perform better are not supported by SUSE.

4 VM Host Server Configuration and Resource Allocation

Allocation of resources for VM Guests is a crucial point when administrating virtual machines. When assigning resources to VM Guests, be aware that overcommitting resources may affect the performance of the VM Host Server and the VM Guests. If all VM Guests request all their resources simultaneously, the host needs to be able to provide all of them. If not, the host's performance will be negatively affected and this will in turn also have negative effects of the VM Guest's performance.

4.1 Memory

Linux manages memory in units called pages. On most systems the default page size is 4 KB. Linux and the CPU need to know which pages belong to which process. That information is stored in a page table. If a lot of processes are running, it takes more time to find where the memory is mapped, due to the time required to search the page table. To speed up the search, the TLB (Translation Lookaside Buffer) was invented. But on a system with a lot of memory, the TLB is not enough. To avoid any fallback to normal page table (resulting in a cache miss, which is time consuming), huge pages can be used. Using huge pages will reduce TLB overhead and TLB misses (pagewalk). A host with 32 GB ($32 \times 1024 \times 1024 = 33,554,432$ KB) of memory and a 4 KB page size has a TLB with: $33,554,432 / 4 = 8,388,608$ entries. Using a 2 MB (2048 KB) page size, the TLB only has $33,554,432 / 2048 = 16,384$ entries, considerably reducing TLB misses.

4.1.1 Configuring the VM Host Server and the VM Guest to use Huge Pages

Current CPU architectures support larger pages than 4 KB: huge pages. To determine the size of huge pages available on your system (could be 2 MB or 1 GB), check the `flags` line in the output of `/proc/cpuinfo` for occurrences of `pse` and/or `pdpe1gb`.

TABLE 1: DETERMINE THE AVAILABLE HUGE PAGES SIZE

| CPU flag | Huge pages size available |
|--------------|---------------------------|
| Empty string | No huge pages available |
| pse | 2 MB |
| pdpe1gb | 1 GB |

Using huge pages improves performance of VM Guests and reduces host memory consumption. By default the system uses THP. To make huge pages available on your system, activate it at boot time with `hugepages=1`, and—optionally—add the huge pages size with, for example, `hugepagesz=2MB`.



Note: 1 GB huge pages

1 GB pages can only be allocated at boot time and cannot be freed afterward.

To allocate and use the huge page table (HugeTlbPage) you need to mount `hugetlbfs` with correct permissions.



Note: Restrictions of Huge Pages

Even if huge pages provide the best performance, they do come with some drawbacks. You lose features such as Memory ballooning (see [Section 6.1.3, “virtio balloon”](#)), KSM (see [Section 4.1.4, “KSM and Page Sharing”](#)), and huge pages cannot be swapped.

PROCEDURE 2: CONFIGURING THE USE OF HUGE PAGES

1. Mount `hugetlbfs` to `/dev/hugepages`:

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

2. To reserve memory for huge pages use the `sysctl` command. If your system has a huge page size of 2 MB (2048 KB), and you want to reserve 1 GB (1,048,576 KB) for your VM Guest, you need $1,048,576/2048 = 512$ pages in the pool:

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

The value is written to `/proc/sys/vm/nr_hugepages` and represents the current number of *persistent* huge pages in the kernel's huge page pool. *Persistent* huge pages will be returned to the huge page pool when freed by a task.

3. Add the `memoryBacking` element in the VM Guest configuration file (by running `virsh edit CONFIGURATION`).

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

4. Start your VM Guest and check on the host whether it uses hugepages:

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

- ① Size of the pool of huge pages

- ② Number of huge pages in the pool that are not yet allocated
- ③ Number of huge pages for which a commitment to allocate from the pool has been made, but no allocation has yet been made
- ④ Number of huge pages in the pool above the value in `/proc/sys/vm/nr_hugepages`. The maximum number of surplus huge pages is controlled by `/proc/sys/vm/nr_overcommit_hugepages`

4.1.2 Transparent Huge Pages

Transparent huge pages (THP) provide a way to dynamically allocate huge pages with the **khugepaged** kernel thread, rather than manually managing their allocation and use. Workloads with contiguous memory access patterns can benefit greatly from THP. A 1000 fold decrease in page faults can be observed when running synthetic workloads with contiguous memory access patterns. Conversely, workloads with sparse memory access patterns (like databases) may perform poorly with THP. In such cases it may be preferable to disable THP by adding the kernel parameter `transparent_hugepage=never`, rebuild your grub2 configuration, and reboot. Verify if THP is disabled with:

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

If disabled, the value `never` is shown in square brackets like in the example above.



Note: Xen

THP is not available under Xen

4.1.3 Xen-specific Memory Notes

4.1.3.1 Managing Domain-0 Memory

When using the Xen hypervisor, by default a small percentage of system memory is reserved for the hypervisor, with all remaining memory automatically allocated to Domain-0. When virtual machines are created, memory is ballooned out of Domain-0 to provide memory for the virtual machine. This process is called "autoballooning".

Autoballooning has several limitations:

- Reduced performance while dom0 is ballooning down to free memory for the new domain.
- Memory freed by ballooning is not confined to a specific NUMA node. This can result in performance problems in the new domain because of using a non-optimal NUMA configuration.
- Failure to start large domains due to delays while ballooning large amounts of memory from dom0.

For these reasons, it is strongly recommended to disable autoballooning and give Domain-0 a predefined amount of memory.

Autoballooning is controlled by the toolstack used to manage your Xen installation. For the xl/libxl toolstack, autoballooning is controlled by the `autoballoon` setting in `/etc/xen/xl.conf`. For the libvirt+libxl toolstack, autoballooning is controlled by the `autoballoon` setting in `/etc/libvirt/libxl.conf`.

The amount of memory initially allocated to Domain-0 is controlled by the Xen hypervisor `dom0_mem` parameter. For example, to set the memory of Domain-0 to 8GB, add `dom0_mem=8G` to the Xen hypervisor parameters.

- To set `dom0_mem` on SLE 11 products, modify `/boot/grub/menu.lst`, adding `dom0_mem=XX` to the Xen hypervisor (`xen.gz`) parameters. The change will be applied at next reboot.
- To set `dom0_mem` on SLE 12 products, modify `/etc/default/grub`, adding `dom0_mem=XX` to `GRUB_CMDLINE_XEN_DEFAULT`. See [Section 7.5, “Change Kernel Parameters at boot time”](#) for more information.

Autoballooning is enabled by default since it is extremely difficult to determine a predefined amount of memory required by Domain-0. Memory needed by Domain-0 is heavily dependent on the number of hosted virtual machines and their configuration. Users must ensure Domain-0 has sufficient memory resources to accommodate virtual machine workloads.

4.1.3.2 `xenstore` in `tmpfs`

When using Xen, it is recommended to place the `xenstore` database on `tmpfs`. `xenstore` is used as a control plane by the `xm/xend` and `xl/libxl` toolstacks and the front-end and back-end drivers servicing domain I/O devices. The load on `xenstore` increases linearly as the number of running

domains increase. If you anticipate hosting many VM Guest on a Xen host, move the xenstore database onto tmpfs to improve overall performance of the control plane. Mount the `/var/lib/xenstored` directory on tmpfs:

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

4.1.4 KSM and Page Sharing

Kernel Samepage Merging is a kernel feature that allows for lesser memory consumption on the VM Host Server by sharing data VM Guests have in common. The KSM daemon `ksmd` periodically scans user memory looking for pages of identical content which can be replaced by a single write-protected page. To enable KSM run:

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

One advantage of using KSM from a VM Guest's perspective is that all guest memory is backed by host anonymous memory, so you can share *pagecache*, *tmpfs* or any kind of memory allocated in the guest.

KSM is controlled by `sysfs`. You can check KSM's values in `/sys/kernel/mm/ksm/`:

- `pages_shared`: The number of shared pages that are being used (read-only).
- `pages_sharing`: The number of sites sharing the pages (read-only).
- `pages_unshared`: The number of pages that are unique and repeatedly checked for merging (read-only).
- `pages_volatile`: The number of pages that are changing too fast to be considered for merging (read-only).
- `full_scans`: The number of times all mergeable areas have been scanned (read-only).
- `sleep_millisecs`: The number of milliseconds `ksmd` should sleep before the next scan. A low value will overuse the CPU, consuming CPU time that could be used for other tasks. A value greater than `1000` is recommended.
- `pages_to_scan`: The number of present pages to scan before `ksmd` goes to sleep. A high value will overuse the CPU. It is recommended to start with a value of `1000`, and then adjust as necessary based on the KSM results observed while testing your deployment.
- `merge_across_nodes`: by default the system merges pages across NUMA nodes. Set this option to `0` to disable this behavior.



Note: Use Cases

KSM is a good technique to over-commit host memory when running multiple instances of the same application or VM Guest. When applications and VM Guest are heterogeneous and do not share any common data, it is preferable to disable KSM. In a mixed heterogeneous and homogeneous environment, KSM can be enabled on the host but disabled on a per VM Guest basis. Use **virsh edit** to disable page sharing of a VM Guest by adding the following to the guest's XML configuration:

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



Warning: Avoid Out-Of-Memory Conditions

KSM can free up some memory on the host system, but the administrator should also reserve enough swap to avoid out-of-memory conditions in the event shareable memory decreases. A decrease in the amount of shareable memory results in an increase in the use of physical memory.



Warning: Memory Access Latencies

By default, KSM will merge common pages across NUMA nodes. This may degrade VM Guest performance if the merged, common page is now located on a distant NUMA node, relative to the node running the VM Guest vCPUs. If increased memory access latencies are noticed in the VM Guest, disable cross-node merging with the **merge_across_nodes** sysfs control:

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

4.1.5 VM Guest: Memory Hotplug

To optimize the usage of your host memory, it may be useful to hotplug more memory for a running VM Guest when required. To support memory hotplugging, you must first configure the `<maxMemory>` tag in the VM Guest's configuration file:

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

- ❶ Runtime maximum memory allocation of the guest.
- ❷ Number of slots available for adding memory to the guest
- ❸ Valid units are:
 - "KB" for kilobytes (1,000 bytes)
 - "k" or "KiB" for kibibytes (1,024 bytes)
 - "MB" for megabytes (1,000,000 bytes)
 - "M" or "MiB" for mebibytes (1,048,576 bytes)
 - "GB" for gigabytes (1,000,000,000 bytes)
 - "G" or "GiB" for gibibytes (1,073,741,824 bytes)
 - "TB" for terabytes (1,000,000,000,000 bytes)
 - "T" or "TiB" for tebibytes (1,099,511,627,776 bytes)
- ❹ Maximum allocation of memory for the guest at boot time
- ❺ Actual allocation of memory for the guest

To hotplug memory devices into the slots, create a file `mem-dev.xml` like the following:

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

And attach it with the following command:

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

For memory device hotplug, the guest must have at least 1 NUMA cell defined (see [Section 4.6.3.1, “VM Guest Virtual NUMA Topology”](#)).

4.2 Swap

Swap is usually used by the system to store underused physical memory (low usage, or not accessed for a long time). To prevent the system running out of memory, setting up a minimum swap is highly recommended.

4.2.1 swappiness

The `swappiness` setting controls your system's swap behavior. It defines how memory pages are swapped to disk. A high value of *swappiness* results in a system that swaps more often. Available values range from `0` to `100`. A value of `100` tells the system to find inactive pages and put them in swap. A value of `0` disables swapping.

To change the value and do some testing on a live system, change the value of `/proc/sys/vm/swappiness` on the fly and check the memory usage afterward:

```
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
```

To permanently set a swappiness value, add a line in `/etc/sysctl.conf`, for example:

```
vm.swappiness = 35
```

You can also control the swap by using the `swap_hard_limit` element in the XML configuration of your VM Guest. It is highly recommended to do some testing before setting this parameter and using it in a production environment, because the host can terminate the domain if the value is too low.

```
<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>
```

- ❶ This element provides memory tunable parameters for the domain. If this is omitted, it defaults to the defaults provided by the operating system.
- ❷ Maximum memory the guest can use. To avoid any problems on the VM Guest it is strongly recommended to do not use this parameter.
- ❸ The memory limit to enforce during memory contention.
- ❹ The maximum memory plus swap the VM Guest can use.

4.3 I/O

4.3.1 I/O Scheduler

The default I/O scheduler is Completely Fair Queuing (CFQ). The main aim of the CFQ scheduler is to provide a fair allocation of the disk I/O bandwidth for all processes that request an I/O operation. You can have different I/O schedulers for different devices.

To get better performance in host and VM Guest it is recommended to use noop in the VM Guest (disable the I/O scheduler) and the deadline scheduler for a virtualization host.

PROCEDURE 3: CHECKING AND CHANGING THE I/O SCHEDULER AT RUNTIME

1. To check your current I/O scheduler for your disk (replace sdX by the disk you want to check), run:

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

The value in square brackets is the one currently selected (cfq in the example above).

2. You can change the scheduler at runtime with the following command:

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

To permanently set an I/O scheduler for all disks of a system, use the Kernel parameter elevator. the respective values are elevator=deadline for the VM Host Server and elevator=noop for VM Guests. See [Section 7.5, “Change Kernel Parameters at boot time”](#) for further instructions.

If you need to specify different I/O schedulers for each disk create the file `/usr/lib/tmpfiles.d/IO_ioscheduler.conf` with a content similar to the following example, which defines the `deadline` scheduler for `/dev/sda` and the `noop` scheduler for `/dev/sdb`. This feature is available on SLE 12 only.

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

4.3.2 I/O Virtualization

SUSE products support various I/O Virtualization technologies. The following table lists advantages and disadvantages of each technology. For more information about I/O in virtualization refer to the [I/O in Virtualization \(https://www.suse.com/documentation/sles-12/book_virt/data/sec_vt_io.html\)](https://www.suse.com/documentation/sles-12/book_virt/data/sec_vt_io.html) chapter in the *SUSE Linux Enterprise Server 12 SP2 Virtualization Guide*.

TABLE 2: I/O VIRTUALIZATION SOLUTIONS

| Technology | Advantage | Disadvantage |
|---|---------------------------------------|-------------------------------------|
| Device Assignment (pass-through) | Device accessed directly by the guest | No sharing among multiple guests |
| | High performance | Live migration is complex |
| | | PCI device limit is 8 per guest |
| | | Limited number of slots on a server |
| Full virtualization (IDE, SATA, SCSI, e1000) | VM Guest compatibility | Bad performance |
| | Easy for live migration | Emulated operation |
| Para-virtualization (virtio-blk, virtio-net, virtio-scsi) | Good performance | Modified guest (PV drivers) |
| | Easy for live migration | |

| Technology | Advantage | Disadvantage |
|------------|--|--------------|
| | Efficient host communication with VM Guest | |

4.4 Storage and File System

Storage space for VM Guests can either be a block device (for example a partition on a physical disk), or an image file on the file system:

TABLE 3: BLOCK DEVICES COMPARED TO DISK IMAGES

| Technology | Advantages | Disadvantages |
|---------------|---|--|
| Block devices | <ul style="list-style-type: none"> • Better performance • Use standard tools for administration/disk modification • Accessible from host (pro and con) | <ul style="list-style-type: none"> • Live migration is complex • Impossible to increase capacity |

| Technology | Advantages | Disadvantages |
|-------------|--|--|
| Image files | <ul style="list-style-type: none"> • Easier system management • Easily move, clone, expand, back up domains • Comprehensive toolkit (guestfs) for image manipulation • Reduce overhead through sparse files • Fully allocate for best performance | <ul style="list-style-type: none"> • Lower performance than block devices |

For detailed information about image formats and maintaining images refer to [Section 5, “VM Guest Images”](#).

If your image is stored on an NFS share, you should check some server and client parameters to improve access to the VM Guest image.

4.4.1 NFS Read/Write (Client)

Options `rsize` and `wsize` specify the size of the chunks of data that the client and server pass back and forth to each other. You should ensure NFS read/write sizes are sufficiently large, especially for large I/O. Change the `rsize` and `wsize` parameter in your `/etc/fstab` by increasing the value to 16 KB, which will ensure that all operations can be frozen if there is any instance of hanging.

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

- ① NFS server's host name and export path name
- ② Where to mount the NFS exported share
- ③ This is an `nfs` mount point

- ④ This mount point will be accessible in read/write
- ⑤ Determines the recovery behavior of the NFS client after an NFS request times out. hard is the best option to avoid data corruption
- ⑥ Any system call that writes data to files on that mount point causes that data to be flushed to the server before the system call returns control to user space
- ⑦ Maximum number of bytes in each network READ request that the NFS client can receive when reading data from a file on an NFS server
- ⑧ Maximum number of bytes per network WRITE request that the NFS client can send when writing data to a file on an NFS server.

4.4.2 NFS Threads (Server)

Your NFS server should have enough NFS threads to handle multi-threaded workloads. Use the **nfsstat** tool to get some RPC statistics on your server:

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

If the retrans is equal to 0, everything is fine. Otherwise, the client needs to retransmit, so increase the USE_KERNEL_NFSD_NUMBER variable in /etc/sysconfig/nfs, and adjust accordingly until retrans is equal to 0.

4.5 CPUs

Host CPU “components” will be “translated” to virtual CPUs in a VM Guest when being assigned. These components can either be:

- *CPU processor*: this describes the main CPU unit, which usually has multiple cores and may support Hyper-Threading.
- *CPU core*: a main CPU unit can provide more than one core, and the proximity of cores speeds up the computation process and reduces energy costs.
- *CPU Hyper-Threading*: this implementation is used to improve parallelization of computations, but this is not as efficient as a dedicated core.

4.5.1 Assigning CPUs

You should avoid overcommitting CPUs. Unless you know exactly how many virtual CPUs are required for a VM Guest, you should start with a single virtual CPU per VM Guest. Each virtual CPU should match one hardware processor or core on the VM Host Server.

You should target a CPU workload of approximately 70% inside your VM (see https://www.suse.com/documentation/sles-12/book_sle_tuning/data/sec_util_processes.html for information on monitoring tools). If you allocate more processors than needed in the VM Guest, this will negatively affect the performance of host and guest: cycle efficiency will be degraded, the unused vCPU will consume timer interrupts and will idle loop. In case you primarily run single threaded applications on a VM Guest, a single virtual CPU is the best choice.

4.5.2 VM Guest CPU Configuration

This section describes how to choose and configure a CPU type for a VM Guest. You will also learn how to pin virtual CPUs to physical CPUs on the host system. For more information about virtual CPU configuration and tuning parameters refer to the libvirt documentation at <https://libvirt.org/formatdomain.html#elementsCPU>.

4.5.2.1 Virtual CPU Models and Features

The CPU model and topology can be specified individually for each VM Guest. Configuration options range from selecting specific CPU models to excluding certain CPU features. Predefined CPU models are listed in the `/usr/share/libvirt/cpu_map.xml`. Multiple sockets with a single core and a single thread generally provide the best performance. The list of available CPUs and topologies depends on the host system and can be displayed by running **`virsh capabilities`**.

Note that changing the default virtual CPU configuration will require a VM Guest shutdown when migrating it to a host with a different hardware. More information on VM Guest migration is available at https://www.suse.com/documentation/sles-12/book_virt/data/sec_libvirt_admin_migrate.html.

To specify a particular CPU model for a VM Guest, add a respective entry to the VM Guest configuration file. The following example configures a Broadwell CPU with the invariant TSC feature:

```
<cpu mode='custom' match='exact'>
```

```
<model>Broadwell</model>
<feature name='invtsr' />
</cpu>
```



Note: The host-passthrough Mode

With `<cpu mode='host-passthrough'>` the virtual CPU will be identical to the host CPU. Using this method makes it impossible to reproduce the VM Guest on different hardware.

4.5.2.2 Virtual CPU Pinning

Virtual CPU pinning is used to constrain vCPU threads to a set of physical CPUs. The `vcupin` element specifies to which physical host CPU a virtual CPU will be pinned to. If this element is not set and the attribute `cpuset` of the element `vcpu` is not specified, the virtual CPU is pinned to all the physical CPUs by default.

CPU intensive workloads can benefit from virtual CPU pinning by increasing the physical CPU cache hit ratio. To pin a vCPU to a specific CPU run the following commands:

```
tux > virsh vcpupin DOMAIN_ID --vcpu vCPU_NUMBER
VCPU: CPU Affinity
-----
0: 0-7
root # virsh vcpupin SLE12 --vcpu 0 0 --config
```

The last command generates the following entry in the XML configuration:

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



Note: Virtual CPU pinning on NUMA Nodes

On a NUMA system, virtual CPU pinning and memory allocation policies can be used to ensure a VM Guest's CPUs and its memory are confined to a NUMA node. See [Section 4.6, "NUMA Tuning"](#) for more information related to NUMA tuning.



Warning: Virtual CPU pinning and Live Migration

Even though `vcupin` can improve performance, keep in mind that live migration of a pinned VM Guest can be problematic. If, for example, the requested physical resources are not be available on the destination host, or the source and destination hosts have different NUMA topologies, the migration will fail. For more recommendations about Live Migration see [Virtualization Live Migration Requirements \(https://www.suse.com/documentation/sles-12/book_virt/data/sec_libvirt_admin_migrate.html#libvirt_admin_live_migration_requirements\)](https://www.suse.com/documentation/sles-12/book_virt/data/sec_libvirt_admin_migrate.html#libvirt_admin_live_migration_requirements).

4.6 NUMA Tuning

NUMA is an acronym for Non Uniform Memory Access. A NUMA system has multiple physical CPUs, each with local memory attached. Each CPU can also access other CPUs' memory, known as “remote memory access”, but it is much slower than accessing local memory. NUMA systems can negatively impact VM Guest performance if not tuned properly. Although ultimately tuning is workload dependent, this section describes controls that should be considered when deploying VM Guests on NUMA hosts. Always consider your host topology when configuring and deploying VMs.

SUSE Linux Enterprise Server contains a NUMA auto-balancer that strives to reduce remote memory access by placing memory on the same NUMA node as the CPU processing it. In addition, standard tools such as `cgset` and virtualization tools such as libvirt provide mechanisms to constrain VM Guest resources to physical resources.

`numactl` is used to check for host NUMA capabilities:

```
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

```

The **numactl** output shows this is a NUMA system with 4 nodes or cells, each containing 36 CPUs and approximately 32G memory. **virsh capabilities** can also be used to examine the systems NUMA capabilities and CPU topology.

4.6.1 NUMA Balancing

On NUMA machines, there is a performance penalty if remote memory is accessed by a CPU. Automatic NUMA balancing scans a task's address space and unmaps pages to detect whether pages are properly placed or whether the data should be migrated to a memory node local to where the task is running. In defined intervals (configured with numa_balancing_scan_delay_ms), the task scans the next scan size number of pages (configured with numa_balancing_scan_size_mb) in its address space. When the end of the address space is reached the scanner restarts from the beginning.

Higher scan rates cause higher system overhead as page faults must be trapped and data needs to be migrated. However, the higher the scan rate, the more quickly a task's memory is migrated to a local node when the workload pattern changes. This minimizes the performance impact due to remote memory accesses. These **sysctl** directives control the thresholds for scan delays and the number of pages scanned:

```

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 Enables/disables automatic page fault-based NUMA balancing
- ❷ Starting scan delay used for a task when it initially forks
- ❸ Maximum time in milliseconds to scan a task's virtual memory

- ④ Minimum time in milliseconds to scan a task's virtual memory
- ⑤ Size in megabytes worth of pages to be scanned for a given scan

For more information see https://www.suse.com/documentation/sles-12/book_sle_tuning/data/cha_tuning_numactl.html.

The main goal of automatic NUMA balancing is either to reschedule tasks on the same node's memory, so the CPU follows the memory, or to copy the memory's pages to the same node, so the memory follows the CPU.



Warning: Task Placement

There are no rules to define the best place to run a task, because tasks could share memory with other tasks. For best performance, it is recommended to group tasks sharing memory on the same node. Check NUMA statistics with `# cat /proc/vmstat | grep numa_`.

4.6.2 Memory Allocation control with the Cpuset Controller

The cgroups cpuset controller can be used to confine memory used by a process to a NUMA node. There are three cpuset memory policy modes available:

- interleave: This is a memory placement policy which is also known as round-robin. This policy can provide substantial improvements for jobs that need to place thread local data on the corresponding node. When the interleave destination is not available, it will be moved to another node.
- bind: This will place memory only on one node, which means in case of insufficient memory, the allocation will fail.
- preferred: This policy will apply a preference to allocate memory to a node, but if there is not enough space for memory on this node, it will fall back to another node.

You can change the memory policy mode with the `cgset` tool from the `libcgroup-tools` package:

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

To migrate pages to a node, use the `migratepages` tool:

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

To check everything is fine. use: `cat /proc/PID/status | grep Cpus`.



Note: Kernel NUMA/cpuset memory policy

For more information see [Kernel NUMA memory policy \(https://www.kernel.org/doc/Documentation/vm/numa_memory_policy.txt\)](https://www.kernel.org/doc/Documentation/vm/numa_memory_policy.txt) and [cpusets memory policy \(https://www.kernel.org/doc/Documentation/cgroup-v1/cpusets.txt\)](https://www.kernel.org/doc/Documentation/cgroup-v1/cpusets.txt). Check also the [Libvirt NUMA Tuning documentation \(https://libvirt.org/formatdomain.html#elementsNUMATuning\)](https://libvirt.org/formatdomain.html#elementsNUMATuning).

4.6.3 VM Guest: NUMA Related Configuration

`libvirt` allows to set up virtual NUMA and memory access policies. Configuring these settings is not supported by `virt-install` or `virt-manager` and needs to be done manually by editing the VM Guest configuration file with `virsh edit`.

4.6.3.1 VM Guest Virtual NUMA Topology

Creating a VM Guest virtual NUMA (vNUMA) policy that resembles the host NUMA topology can often increase performance of traditional large, scale-up workloads. VM Guest vNUMA topology can be specified using the `numa` element in the XML configuration:

```
<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 Each `cell` element specifies a vNUMA cell or node
- ② All cells should have an `id` attribute, allowing to reference the cell in other configuration blocks. Otherwise cells are assigned ids in ascending order starting from 0.
- ③ The CPU or range of CPUs that are part of the node
- ④ The node memory
- ⑤ Units in which node memory is specified

- ⑥ Optional attribute which can control whether the memory is to be mapped as shared or private. This is valid only for hugepages-backed memory.

To find where the VM Guest has allocated its pages, use: `cat /proc/PID/numa_maps` and `cat /sys/fs/cgroup/memory/sysdefault/libvirt/qemu/KVM_NAME/memory.numa_stat`.



Warning: NUMA specification

The libvirt VM Guest NUMA specification is currently only available for QEMU/KVM.

4.6.3.2 Memory Allocation Control with libvirt

If the VM Guest has a vNUMA topology (see [Section 4.6.3.1, “VM Guest Virtual NUMA Topology”](#)), memory can be pinned to host NUMA nodes using the numatune element. This method is currently only available for QEMU/KVM guests. See [Important: Non-vNUMA VM Guest](#) for how to configure non-vNUMA VM Guests.

```
<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 Policies available are: interleave (round-robin like), strict (default) or preferred.
- ② Specify the NUMA nodes.
- ③ Specify memory allocation policies for each guest NUMA node (if this element is not defined then this will fall back and use the memory element).
- ④ Addresses the guest NUMA node for which the settings are applied.
- ⑤ The placement attribute can be used to indicate the memory placement mode for a domain process, the value can be auto or strict.



Important: Non-vNUMA VM Guest

On a non-vNUMA VM Guest, pinning memory to host NUMA nodes is done like in the following example:

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

In this example memory is allocated from the host nodes 0 and 1. In case these memory requirements cannot be fulfilled, starting the VM Guest will fail. **virt-install** also supports this configuration with the **--numatune** option.



Warning: Memory and CPU across NUMA nodes

You should avoid allocating VM Guest memory across NUMA nodes, and prevent virtual CPUs from floating across NUMA nodes.

5 VM Guest Images

Images are virtual disks used to store the operating system and data of VM Guests. They can be created, maintained and queried with the **qemu-img** command. Refer to [SLE12 qemu-img documentation \(https://www.suse.com/documentation/sles-12/singlehtml/book_virt/book_virt.html#cha.qemu.guest_inst.qemu-img.create\)](https://www.suse.com/documentation/sles-12/singlehtml/book_virt/book_virt.html#cha.qemu.guest_inst.qemu-img.create) for more information on the **qemu-img** tool and examples.

5.1 VM Guest Image Formats

Certain storage formats which QEMU recognizes have their origins in other virtualization technologies. By recognizing these formats, QEMU can leverage either data stores or entire guests that were originally targeted to run under these other virtualization technologies. Some formats are supported only in read-only mode, enabling either direct use of that read-only data store in a QEMU guest or conversion to a fully supported QEMU storage format (using **qemu-img**) which could then be used in read/write mode. See SUSE Linux Enterprise [Release Notes \(https://www.suse.com/releasenotes/x86_64/SUSE-SLES/12/#fate-317891\)](https://www.suse.com/releasenotes/x86_64/SUSE-SLES/12/#fate-317891) to get the list of supported formats.

Use **qemu-img info vmguest.img** to get information about an existing image, such as: the format, the virtual size, the physical size, snapshots if available.



Note: Performance

It is recommended to convert the disk images to either raw or qcow2 to achieve good performance.



Warning: Encrypted Images Cannot be Compressed

When you create an image, you cannot use compression (-c) in the output file together with the encryption option (-e).

5.1.1 Raw Format

- This format is simple and easily exportable to all other emulators/hypervisors.
- It provides best performance (least I/O overhead).
- If your file system supports holes (for example in Ext2 or Ext3 on Linux or NTFS on Windows*), then only the written sectors will reserve space.
- The raw format allows to copy a VM Guest image to a physical device (`dd if=vmguest.raw of=/dev/sda`).
- It is byte-for-byte the same as what the VM Guest sees, so this wastes a lot of space.

5.1.2 qcow2 Format

- Use this to have smaller images (useful if your file system does not supports holes, for example on Windows*).
- It has optional AES encryption.
- Zlib-based compression option.
- Support of multiple VM snapshots (internal, external).
- Improved performance and stability.
- Supports changing the backing file.
- Supports consistency checks.
- Less performance than raw format.

l2-cache-size

qcow2 can provide the same performance for random read/write access as raw format, but it needs a well-sized cache size. By default cache size is set to 1 MB. This will give good performance up to a disk size of 8 GB. If you need a bigger disk size, you need to adjust the cache size. For a disk size of 64 GB ($64 \times 1024 = 65536$), you need $65536 / 8192B = 8$ MB of cache (`-drive format=qcow2,l2-cache-size=8M`).

Cluster Size

The qcow2 format offers the capability to change the cluster size. The value must be between 512 KB and 2 MB. Smaller cluster sizes can improve the image file size whereas larger cluster sizes generally provide better performance.

Preallocation

An image with preallocated metadata is initially larger but can improve performance when the image needs to grow.

Lazy Refcounts

Reference count updates are postponed with the goal of avoiding metadata I/O and improving performance. This is particularly beneficial with `cache=writethrough`, which does not batch metadata updates, but in case of host crash, the reference count tables must be rebuilt, this is done automatically at the next open with `qemu-img check -r all`, but this takes some time.

5.1.3 qed format

qed is the next-generation qcow (QEMU Copy On Write). Its characteristics include:

- Strong data integrity because of simple design.
- Retains sparseness over non-sparse channels (for example HTTP).
- Supports changing the backing file.
- Supports consistency checks.
- Fully asynchronous I/O path.
- Does not support internal snapshots.
- Relies on the host file system and cannot be stored on a logical volume directly.

5.1.4 VMDK format

VMware 3, 4, or 6 image format, for exchanging images with that product.

5.2 Overlay Disk Images

The qcow2 and qed formats provide a way to create a base image (also called backing file) and overlay images on top of the base image. A backing file is useful to be able to revert to a known state and discard the overlay. If you write to the image, the backing image will be untouched and all changes will be recorded in the overlay image file. The backing file will never be modified unless you use the `commit` monitor command (or `qemu-img commit`).

To create an overlay image:

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

- ❶ Use `-o ?` for an overview of available options.
- ❷ The backing file name.
- ❸ Specify the file format for the backing file.
- ❹ Specify the image format for the VM Guest.
- ❺ Image name of the VM Guest, it will only record the differences from the backing file.



Warning: Backing Image Path

You should not change the path to the backing image, otherwise you will need to adjust it. The path is stored in the overlay image file. If you want to update the path, you should make a symbolic link from the original path to the new path and then use the `qemu-img rebase` option.

```
root # ln -sf /var/lib/images/vmquest.raw /var/lib/images/SLE12/vmquest.raw  
root # qemu-img rebase ❶ -u ❷ -b ❸ /var/lib/images/vmquest.raw /var/lib/images/  
SLE12/vmquest.cow ❹
```

The `rebase` subcommand tells `qemu-img` to change the backing file image. The `-u` option activates the unsafe mode (see note below). The backing image to be used is specified with `-b` and the image path is the last argument of the command.

There are two different modes in which rebase can operate:

- *Safe*: This is the default mode and performs a real rebase operation. The safe mode is a time-consuming operation.
- *Unsafe*: The unsafe mode (-u) only changes the backing files name and the format of the file name without making any checks on the files contents. You should use this mode to rename or moving a backing file.

A common use is to initiate a new guest with the backing file. Let's assume we have a sle12_base.img VM Guest ready to be used (fresh installation without any modification). This will be our backing file. Now you need to test a new package, on an updated system and on a system with a different kernel. We can use sle12_base.img to instantiate the new SUSE Linux Enterprise VM Guest by creating a qcow2 overlay file pointing to this backing file (sle12_base.img).

In our example we will use sle12_updated.qcow2 for the updated system, and sle12_kernel.qcow2 for the system with a different kernel.

To create the two thin provisioned systems use the **qemu-img** command line with the -b option:

```
root # qemu-img create -b /var/lib/libvirt/sle12_base.img -f qcow2 \
/var/lib/libvirt/sle12_updated.qcow2
Formatting 'sle12_updated.qcow2', fmt=qcow2 size=17179869184
backing_file='sle12_base.img' encryption=off cluster_size=65536
lazy_refcounts=off nocow=off
root # qemu-img create -b /var/lib/libvirt/sle12_base.img -f qcow2 \
/var/lib/libvirt/sle12_kernel.qcow2
Formatting 'sle12_kernel.qcow2', fmt=qcow2 size=17179869184
backing_file='vmguest-sle12_base.img' encryption=off cluster_size=65536
lazy_refcounts=off nocow=off
```

The images are now usable, and you can do your test without touching the initial sle12_base.img backing file, all changes will be stored in the new overlay images. Additionally, you can also use these new images as a backing file, and create a new overlay.

```
root # qemu-img create -b sle12_kernel.qcow2 -f qcow2 sle12_kernel_TEST.qcow2
```

When using **qemu-img info** with the option --backing-chain, it will return all information about the entire backing chain recursively:

```
root # qemu-img info --backing-chain
```

```
/var/lib/libvirt/images/sle12_kernel_TEST.qcow2
image: sle12_kernel_TEST.qcow2
file format: qcow2
virtual size: 16G (17179869184 bytes)
disk size: 196K
cluster_size: 65536
backing file: sle12_kernel.qcow2
Format specific information:
compat: 1.1
lazy refcounts: false

image: sle12_kernel.qcow2
file format: qcow2
virtual size: 16G (17179869184 bytes)
disk size: 196K
cluster_size: 65536
backing file: SLE12.qcow2
Format specific information:
compat: 1.1
lazy refcounts: false

image: sle12_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
```

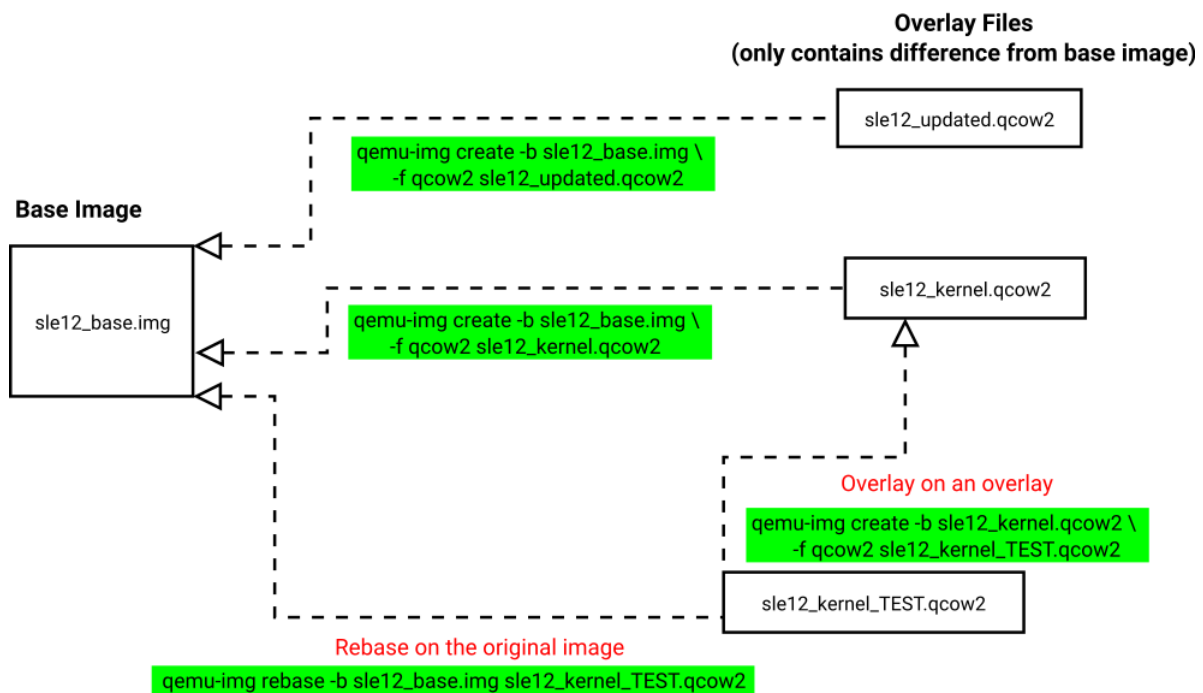


FIGURE 1: UNDERSTANDING IMAGE OVERLAY

5.3 Opening a VM Guest Image

To access the file system of an image, use the `guestfs-tools`. If you do not have this tool installed on your system you can mount an image with other Linux tools, but you should avoid accessing an untrusted or unknown VM Guest's image system because this can lead to security issues (read [D. Berrangé's post \(https://www.berrange.com/posts/2013/02/20/a-reminder-why-you-should-never-mount-guest-disk-images-on-the-host-os/\)](https://www.berrange.com/posts/2013/02/20/a-reminder-why-you-should-never-mount-guest-disk-images-on-the-host-os/) for more information).

5.3.1 Opening a Raw Image

PROCEDURE 4: MOUNTING A RAW IMAGE

1. To be able to mount the image, find a free loop device. The following command display the first unused loop device, `/dev/loop1` in this example.

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

2. Associate an image (SLE12.raw in this example) with the loop device:

```
root # losetup /dev/loop1 SLE12.raw
```

3. Check whether the images has successfully been associated with the loop device by getting detailed information about the loop device:

```
root # losetup -l
NAME          SIZELIMIT OFFSET AUTOCLEAR RO BACK-FILE
/dev/loop1     0         0         0 0    /var/lib/libvirt/images/SLE12.raw
```

4. Check the image's partitions with kpartx:

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

❶ Add partition device mappings.

❷ Verbose mode.

5. Now mount the image partition(s) (to /mnt/sle12mount in the following example):

```
root # mkdir /mnt/sle12mount
root # mount /dev/mapper/loop1p1 /mnt/sle12mount
```



Note: Raw image with LVM

If your raw image contains an LVM volume group you should use LVM tools to mount the partition. Refer to [Section 5.3.3, "Opening Images Containing LVM"](#)

PROCEDURE 5: UNMOUNTING A RAW IMAGE

1. Unmount all mounted partitions of the image, for example:

```
root # umount /mnt/sle12mount
```

2. Delete partition device mappings with kpartx:

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

3. Detach the devices with losetup

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

5.3.2 Opening a qcow2 Image

PROCEDURE 6: MOUNTING A QCOW2 IMAGE

1. First you need to load the `nbdk` (network block devices) module. The following example loads it with support for 16 block devices (`max_part=16`). Check with `dmesg` whether the operation was successful:

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

2. Connect the VM Guest image (for example `SLE12.qcow2`) to an NBD device (`/dev/nbd0` in the following example) with the `qemu-nbd` command. Make sure to use a free NBD device:

```
root # qemu-nbd -c ❶ /dev/nbd0 ❷ SLE12.qcow2 ❸
```

- ❶ Connect `SLE12.qcow2` to the local NBD device `/dev/nbd0`
- ❷ NBD device to use
- ❸ VM Guest image to use



Tip: Checking for a free NBD devices

To check whether an NBD device is free run the following command:

```
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
```

If the command produces an output like in the example above, the device is busy (not free). This can also be confirmed by the presence of the `/sys/devices/virtual/block/nbd0/pid` file.

3. Inform the operating system about partition table changes with `partprobe`:

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


4. In the example above, the `SLE12.qcow2` contains two partitions: `/dev/nbd0p1` and `/dev/nbd0p2`. Before mounting these partitions, use `vgscan` to check whether they belong to an LVM volume:

```
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. If no LVM volume has been found, you can mount the partition with `mount`:

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

Refer to [Section 5.3.3, “Opening Images Containing LVM”](#) for information on how to handle LVM volumes.

PROCEDURE 7: UNMOUNTING A QCOW2 IMAGE

1. Unmount all mounted partitions of the image, for example:

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

2. Disconnect the image from the `/dev/nbd0` device

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

5.3.3 Opening Images Containing LVM

PROCEDURE 8: MOUNTING IMAGES CONTAINING LVM

1. To check images for LVM groups use `vgscan -v`. If an image contains LVM groups, the output of the command looks like the following:

```
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
```

- The system LVM volume group has been found on the system. You can get more information about this volume with **vgdisplay** *VOLUMEGROUPNAME* (in our case *VOLUMEGROUPNAME* is system). You should activate this volume group to expose LVM partitions as devices so the system can mount them. Use **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
```

- All partitions in the volume group will be listed in the /dev/mapper directory. You can simply mount them now.

```
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
```

PROCEDURE 9: UNMOUNTING IMAGES CONTAINING LVM

- Unmount all partitions (with **umount**)

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

2. Deactivate the LVM volume group (with `vgchange -an VOLUMEGROUPNAME`)

```
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. Now you have two choices:

- If case of a qcow2 image, proceed as described in [Step 2](#) (`qemu-nbd -d /dev/nbd0`).
- In case of a raw image, proceeds as described in [Step 2](#) (`kpartx -d /dev/loop1; losetup -d /dev/loop1`).



Important: Check for a Successful Unmount

You should double-check that unmounting succeeded by using a system command like `losetup`, `qemu-nbd`, `mount` or `vgscan`. If this is not the case you may have trouble using the VM Guest because its system image is used in different places.

5.4 File System Sharing

You can access a host directory in the VM Guest using the `filesystem` element. In the following example we will share the `/data/shared` directory and mount it in the VM Guest. Note that the `accessmode` parameter only works with `type='mount'` for the QEMU/KVM drive (most other values for `type` are exclusively used for the LXC driver).

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

- ❶ A host directory to mount VM Guest.
- ❷ Access mode (the security mode) set to mapped will give access with the permissions of the hypervisor. Use passthrough to access this share with the permissions of the user inside the VM Guest.
- ❸ Path to share with the VM Guest.
- ❹ Name or label of the path for the mount command.

To mount the shared directory on the VM Guest, use the following commands: Under the VM Guest now you need to mount the target dir='shared' :

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

See libvirt File System (<https://libvirt.org/formatdomain.html#elementsFilesystems>)[↗] and QEMU 9psetup (<http://wiki.qemu.org/Documentation/9psetup>)[↗] for more information.

6 VM Guest Configuration

6.1 Virtio Driver

To increase VM Guest performance it is recommended to use paravirtualized drivers within the VM Guests. The virtualization standard for such drivers for KVM are the virtio drivers, that are designed for running in a virtual environment. Xen is using similar paravirtualized device drivers (like VMDP (<https://www.suse.com/products/vmdriverpack/>)[↗] in a Windows* guest). For a better understanding of this topic, refer to the I/O Virtualization (https://www.suse.com/documentation/sles-12/book_virt/data/sec_vt_io.html)[↗] section in the official Virtualization Guide.

6.1.1 virtio blk

virtio_blk is the virtio block device for disk. To use the virtio blk driver for a block device, specify the bus='virtio' attribute in the disk definition:

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

! Important: Disk Device Names

virtio disk devices are named /dev/vd[a-z][1-9]. If you migrate a Linux guest from a non-virtio disk you need to adjust the root= parameter in the GRUB configuration, and regenerate the initrd file, otherwise the system cannot boot. On VM Guests with other operating systems, the boot loader may need to be adjusted or reinstalled accordingly, too.

! Important: Using virtio disks with **qemu-system-ARCH**

When running **qemu-system-ARCH**, use the -drive option to add a disk to the VM Guest. See the [Basic Installation with qemu-system-ARCH \(https://www.suse.com/documentation/sles-12/book_virt/data/cha_qemu_guest_inst_qemu-kvm.html\)](https://www.suse.com/documentation/sles-12/book_virt/data/cha_qemu_guest_inst_qemu-kvm.html) section in the official Virtualization guide for an example. The -hd[abcd] option will not work for virtio disks.

6.1.2 virtio net

virtio_net is the virtio network device. The kernel modules should be loaded automatically in the guest at boot time. You need to start the service to make the network available.

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

6.1.3 virtio balloon

The virtio balloon is used for host memory over-commits for guests. For Linux guests, the balloon driver runs in the guest kernel, whereas for Windows guests, the balloon driver is in the VMDP package. virtio_balloon is a PV driver to give or take memory from a VM Guest.

- *Inflate balloon*: Return memory from guest to host kernel (for KVM) or to hypervisor (for Xen)
- *Deflate balloon*: Guest will have more available memory

It is controlled by the currentMemory and memory options.

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

You can also use **virsh** to change it:

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

6.1.4 Checking virtio Presence

You can check the virtio block PCI with:

```
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
```

To find the block device associated with vdX:

```
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
```

To get more information on the virtio block:

```
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
```

To check all virtio drivers being used:

```
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 Find Device Driver Options

Virtio devices and other drivers have various options. To list all of them, use the help parameter of the **qemu-system-ARCH** command.

```
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 Video Driver

To get 16-bit color, high compatibility and better performance it is recommended to use the cirrus video driver.



Note: libvirt

libvirt ignores the vram value because video size has been hardcoded in QEMU.

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

6.3 Better Entropy

Virtio RNG (random number generator) is a paravirtualized device that is exposed as a hardware RNG device to the guest. On the host side, it can be wired up to one of several sources of entropy, including a real hardware RNG device and the host's /dev/random if hardware support does not exist. The Linux kernel contains the guest driver for the device from version 2.6.26 and higher.

The system entropy is collected from various non-deterministic hardware events and is mainly used by cryptographic applications. The virtual random number generator device (paravirtualized device) allows the host to pass through entropy to VM Guest operating systems. This results in a better entropy in the VM Guest.

To use Virtio RNG, add an RNG device in virt-manager or directly in the VM Guest's XML configuration:

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

The host now should use /dev/random:

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

On the VM Guest, the source of entropy can be checked with:

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

The current device used for entropy can be checked with:

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

You should install the rng-tools package on the VM Guest, enable the service, and start it. Under SLE12 do the following:



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

6.4 Disable Unused Tools and Devices

Per host, use one virtualization technology only. For example, do not use KVM and Containers on the same computer. Otherwise, you may find yourself with a reduced amount of available resources, increased security risk and a longer software update queue. Even when the amount of resources allocated to each of the technologies is configured carefully, the host may suffer from reduced overall availability and degraded performance.

Minimize the amount of software and services available on hosts. Most default installations of operating systems are not optimized for VM usage. Install what you really need and remove all other components in the VM Guest.

Windows* Guest:

- Disable the screen saver
- Remove all graphical effects
- Disable indexing of hard disks if not necessary
- Check the list of started services and disable the ones you do not need
- Check and remove all unneeded devices
- Disable system update if not needed, or configure it to avoid any delay while rebooting or shutting down the host
- Check the Firewall rules
- Schedule backups and anti-virus updates appropriately
- Install the VMDP (<https://www.suse.com/products/vmdriverpack/>)  paravirtualized driver for best performance
- Check the operating system recommendations, such as on the [Microsoft Windows* 7 better performance](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>)  Web page.

Linux Guest:

- Remove or do not start the X Window System if not necessary
- Check the list of started services and disable the ones you do not need
- Check the OS recommendations for kernel parameters that enable better performance
- Only install software that you really need
- Optimize the scheduling of predictable tasks (system updates, hard disk checks, etc.)

6.5 Updating the Guest Machine Type

QEMU machine types define details of the architecture that are particularly relevant for migration and session management. As changes or improvements to QEMU are made, new machine types are added. Old machine types are still supported for compatibility reasons, but to take advantage of improvements, it is recommended to always migrate to the latest machine type when upgrading.

Changing the guest's machine type for a Linux guest will mostly be transparent, whereas for Windows* guests, it is probably a good idea to take a snapshot or backup of the guest in case Windows* has issues with the changes it detects and subsequently the user decides to revert to the original machine type the guest was created with.



Note: Changing the machine type

Refer to the Virtualization guide section [Change Machine Type \(https://www.suse.com/documentation/sles-12/singlehtml/book_virt/book_virt.html#sec.libvirt.config.machine.type.virsh\)](https://www.suse.com/documentation/sles-12/singlehtml/book_virt/book_virt.html#sec.libvirt.config.machine.type.virsh) for documentation.

7 VM Guest-Specific Configurations and Settings

7.1 ACPI Testing

The ability to change a VM Guest's state heavily depends on the operating system. It is very important to test this feature before any use of your VM Guests in production. For example, most Linux operating system disable this capability by default, so this requires you to enable this operation (mostly through Policy Kit).

ACPI must be enabled in the guest for a graceful shutdown to work. To check if ACPI is enabled, run:

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

If nothing is printed, ACPI is not enabled for your machine. Use **virsh edit** to add the following XML under `<domain>`:

```
<features>
```

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

If ACPI was enabled during a Windows XP/Windows Server 2003 guest installation, turning it on in the VM Guest configuration only is not sufficient. For more information, see:

- <https://support.microsoft.com/en-us/kb/314088> ↗
- <https://support.microsoft.com/en-us/kb/309283> ↗

Regardless of the VM Guest's configuration, a graceful shutdown is always possible from within the guest operating system.

7.2 Keyboard Layout

Though it is possible to specify the keyboard layout from a **qemu-system-ARCH** command, it is recommended to configure it in the `libvirt` XML file. To change the keyboard layout while connecting to a remote VM Guest using `vnc`, you should edit the VM Guest XML configuration file. For example, to add an `en-us` keymap, add in the `<devices>` section:

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

Check the `vncdisplay` configuration and connect to your VM Guest:

```
tux > virsh vncdisplay sles12 127.0.0.1:0
```

7.3 Spice default listen URL

If no network interface other than `lo` is assigned an IPv4 address on the host, the default address the spice server will listen on will not work. An error like the following one will occur:

```
tux > virsh start sles12
error: Failed to start domain sles12
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
```

To fix this, you can change the default `spice_listen` value in `/etc/libvirt/qemu.conf` using the local IPv6 address `:::1`. The spice server listening address can also be changed on a per VM Guest basis, use **virsh edit** to add the listen XML attribute to the `graphics type='spice'` element:

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

7.4 XML to QEMU command line

Sometimes it could be useful to get the QEMU command line to launch the VM Guest from the XML file.

```
tux > virsh domxml-to-native ❶ qemu-argv ❷ SLE12.xml ❸
```

- ❶ Convert the XML file in domain XML format to the native guest configuration
- ❷ For the QEMU/KVM hypervisor, the format argument needs be `qemu-argv`
- ❸ Domain XML file to use

```
tux > sudo virsh domxml-to-native qemu-argv /etc/libvirt/qemu/SLE12.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 SLE12 -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/ \
  SLE12.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/SLE12.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 Change Kernel Parameters at boot time

7.5.1 SUSE Linux Enterprise 11

To change the value for SLE 11 products at boot time, you need to modify your /boot/grub/menu.lst file, adding the OPTION=parameter, and then reboot your system.

7.5.2 SUSE Linux Enterprise 12

To change the value for SLE 12 products at boot time, you need to modify your /etc/default/grub file. Find the variable starting with GRUB_CMDLINE_LINUX_DEFAULT and add at the end OPTION=parameter (or change it with the correct value if it is already available).

Now you need to regenerate your grub2 configuration:

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

Then reboot your system



7.6 Add Device to an XML Configuration


If you want to create a new VM Guest based on an XML file, and you are not familiar with XML configuration syntax, you can specify the QEMU command line using the special tag qemu:commandline. For example if you want to add a virtio-balloon-pci, add this block at the end of the XML configuration file (before the `</domain>` tag).

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

8 Hypervisors Compared to Containers

TABLE 4: HYPERVISORS COMPARED TO CONTAINERS

| Features | Hypervisors | Containers |
|-------------------------------|--|--|
| Technologies | Emulation of a physical computing environment | Use kernel host |
| System layer level | Managed by a virtualization layer (Hypervisor) | Rely on kernel namespaces and cgroups |
| Level (layer) | Hardware level | Software level |
| Virtualization mode available | FV or PV | None, only user space |
| Security | Strong |  Warning Security is very low |
| Confinement | Full isolation |  Warning Host kernel (OS must be compatible with kernel version) |
| Operating system | Any operating system | Only Linux (must be "kernel" compatible) |
| Type of system | Full OS needed | Scope is an instance of Linux |
| Boot time | Slow to start (OS delay) | Really quick start |
| Overhead | High | Very low |
| Efficiency | Depends on OS | Very efficient |
| Sharing with host |  Warning Complex because of isolation | Sharing is easy (host sees everything; container sees its own objects) |

| Features | Hypervisors | Containers |
|-----------|--------------------------------|--|
| Migration | Supports migration (live mode) |  Warning Not possible |

8.1 Getting the Best of Both Worlds

Even if the above table seems to indicate that running a single application in a highly secure way is not possible, starting with SUSE Linux Enterprise Server 12 SP1 **virt-sandbox** will allow running a single application in a KVM guest. **virt-sandbox** bootstraps any command within a Linux kernel with a minimal root file system.

The guest root file system can either be the root file system mounted read-only or a disk image. The following steps will show how to set up a sandbox with qcow2 disk image as root file system.

1. Create the disk image using **qemu-img**:

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

2. Format the disk image:

```
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 Make sure the nbd module is loaded: it is not loaded by default and will only be used to format the qcow image.
- ❷ Create an NBD device for the qcow2 image. This device will then behave like any other block device. The example uses **/dev/nbd0** but any other free NBD device will work.
- ❸ Format the disk image directly. Note that no partition table has been created: **virt-sandbox** considers the image to be a partition, not a disk.
The partition formats that can be used are limited: the Linux kernel bootstrapping the sandbox needs to have the corresponding features built in. The Ext4 module is also available at the sandbox start-up time.

3. Now populate the newly formatted image:

```
root # guestmount -a base.qcow2 -m /dev/sda:/ /mnt ❶

root # zypper --root /mnt ar cd:///?devices=/dev/dvd SLES12_DVD
root # zypper --root /mnt in -t pattern Minimal ❷

root # guestunmount /mnt ❸
```

- ❶ Mount the qcow2 image using the **guestfs** tools.
- ❷ Use Zypper with the **--root** parameter to add a SUSE Linux Enterprise Server repository and install the **Minimal** pattern in the disk image. Any additional package or configuration change should be performed in this step.



Note: Using backing chains

To share the root file system between several sandboxes, create qcow2 images with a common disk image as backing chain as described in [Section 5.2, “Overlay Disk Images”](#).

- ❸ Unmount the qcow2 image.
4. Run the sandbox, using **virt-sandbox**. This command has many interesting options, read its man page to discover them all. The command can be run as **root** or as an unprivileged user.

```
root # virt-sandbox -n name \  
-m host-image:/$PWD/rootfs.qcow2 \ ❶  
-m host-bind:/srv/www=/guests/www \ ❷  
-m ram:/tmp=100MiB \  
-m ram:/run=100MiB \ ❸  
-N source=default,address=192.168.122.12/24 \ ❹  
-- \  
/bin/sh
```

- ❶ Mount the created disk image as the root file system. Note that without any image being mounted as **/**, the host root file system is read-only mounted as the guest one. The host-image mount is not reserved for the root file system, it can be used to mount any disk image anywhere in the guest.

- 2 The host-bind mount is pretty convenient for sharing files and directories between the host and the guest. In this example the host directory `/guests/www` is mounted as `/srv/www` in the sandbox.
- 3 The RAM mounts are defining `tmpfs` mounts in the sandbox.
- 4 The network uses a network defined in libvirt. When running as an unprivileged user, the source can be omitted, and the KVM user networking feature will be used. Using this option requires the `dhcp-client` and `iproute2` packages, which are part of the SUSE Linux Enterprise Server `Minimal` pattern.

9 Xen: Moving from PV to FV

This chapter explains how to convert a Xen para-virtual machine into a Xen fully virtualized machine.

First you need to change to the `-default` kernel, if not already installed, you need to install it while in PV mode.

In case you are using `vda*` disk naming, you need to change this to `hd*` in `/etc/fstab`, `/boot/grub/menu.lst` and `/boot/grub/device.map`.



Note: Prefer UUIDs

You should use UUIDs or logical volumes within your `/etc/fstab`. Using UUID simplifies attached network storage, multipathing, and virtualization.

Moving from PV to FV will lead to disk driver modules being missing from the `initrd`. The modules expected are `xen-vbd` (and `xen-vnif` for network). These are the PV drivers for a fully virtualized VM Guest. All other modules like `ata_piix`, `ata_generic` and `libata` should be added automatically.



Tip: Adding Modules to the `initrd`

On SLES 11, you can add modules to the `INITRD_MODULES` line in the `/etc/sysconfig/kernel` file, for example:

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

Run `mkinitrd` to build a new `initrd` containing the modules.

On SLES 12, open `/etc/dracut.conf.d/01-dist.conf` and add the modules with `force-drivers` by adding a line as in the example below:

```
force-drivers+="xen-vbd xen-vnif"
```

Run `dracut -f` to build a new initrd containing the modules.

You need to change a few parameters in the XML configuration file to describe your VM Guest:
Set the OS section to something like:

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

In the devices section, you need to add:

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

Replace the `xen` disk bus with `ide`, and the `vda` target device with `hda`.



Note: guestfs-tools

If you want to script this process, or work on disk images directly, you can use the [guestfs-tools](https://www.suse.com/documentation/sles-12/book_virt/data/sec_guestfs_tools.html) (https://www.suse.com/documentation/sles-12/book_virt/data/sec_guestfs_tools.html) suite, as numerous tools exist there to help to modify disk images.

10 External References

- Increasing memory density using KSM (<https://kernel.org/doc/ols/2009/ols2009-pages-19-28.pdf>)
- linux-kvm.org KSM (<http://www.linux-kvm.org/page/KSM>)
- KSM's kernel documentation (<https://www.kernel.org/doc/Documentation/vm/ksm.txt>)
- ksm - dynamic page sharing driver for linux v4 (<https://lwn.net/Articles/329123/>)

- Memory Ballooning (<http://www.espenbraastad.no/post/memory-ballooning/>) ↗
- libvirt virtio (<http://wiki.libvirt.org/page/Virtio>) ↗
- CFQ's kernel documentation (<https://www.kernel.org/doc/Documentation/block/cfq-iosched.txt>) ↗
- Documentation for sysctl (<https://www.kernel.org/doc/Documentation/sysctl/kernel.txt>) ↗
- LWN Random Number (<https://lwn.net/Articles/525459/>) ↗
- KVM / Xen tweaks (http://wiki.mikejung.biz/KVM/_Xen) ↗
- Dr. Khoa Huynh, IBM Linux Technology Center (http://events.linuxfoundation.org/sites/events/files/slides/CloudOpen2013_Khoa_Huynh_v3.pdf) ↗
- Kernel Parameters (<https://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/plain/Documentation/admin-guide/kernel-parameters.txt>) ↗
- Huge pages Administration (Mel Gorman) (<http://lwn.net/Articles/374424/>) ↗
- kernel hugetlbpage (<https://www.kernel.org/doc/Documentation/vm/hugetlbpage.txt>) ↗