

IT 資格 道 場

LPIC-3 305試験 学習用テキスト

予備知識ゼロから合格ラインへ —— 読んで理解する1冊目

無料サンプル(第1章まで)

LPI公式 LPIC-3 Exam 305 Objectives Version 3.0 (305-300) 準拠 (LPI 公式ページ・
2026-10-13 取得)
2026年10月版

ご利用にあたって

- このテキストは「IT資格道場」(www.shikaku-doujou.com)の購入者限定テキストのうち、学習用テキストの第1章までを収録した**無料サンプル**です。どなたでもご覧いただけます。
- 無料サンプルも著作物です。転載・改変しての再配布・二次販売はできません。
- AI モデルの学習データとしてのご利用(学習目的の入力・データセット化を含む)はできません。
- 製品版(学習用テキスト・頻出度別ノート・暗記用ノートの3点)は、ご購入者を識別できる透かし入りのPDFとして提供されます。
- 本PDFはスマートフォン・タブレットでの閲覧に合わせたA5版面・文字サイズで組版しています。

目次

この教材群の使い方 (3点セット)

第1章 Full Virtualization [351.1/351.2/351.3/351.4/351.5]

仮想化の概念と理論 [351.1]

Xen [351.2]

QEMU [351.3]

libvirt による仮想マシンの管理 [351.4]

仮想マシンのディスクイメージの管理 [351.5]

サンプルはここまでです

この教材群の使い方(3点セット)

種類	役割
① 学習用テキスト	これだけで問題が解けるようになる本編
② 頻出度別ノート	テキストを読んだら次はこれ。頻出順に絞った問題と解説で「解ける」に橋渡し。試験直前の最終チェックにも
③ 暗記用ノート	覚えるべき要点だけを一問一答に凝縮。空き時間の反復と用語の再確認用

使い方: ① 学習用を読む → ② 頻出度別で解き方を掴む → ③ 問題演習で腕試し → ④ 頻出度別に戻って再確認(試験直前もここ)。暗記用は空き時間や用語の再確認に。

このテキストの位置づけ: ①の学習用テキストです。まずはここを通読し、これだけで問題が解けるようになることを目標にしてください。

このテキストは、LPIC-3 Virtualization and Containerization (Exam 305) の出題範囲をまとめた本編です。並べ方は公式の出題範囲 (Version 3.0) の Topic 単位。**このテキストの知識だけで、収録問題が解けるようになることを完成の条件に置きます。**

- 対象試験: LPIC-3 Virtualization and Containerization (Exam 305-300・出題範囲 Version 3.0)
- 試験形式: 60問・90分の選択式と穴埋め。合格点は 500 (200～800 のスケール)
- 構成: 全3章。第1章=Topic 351 (Full Virtualization)、第2章=Topic 352 (Container Virtualization)、第3章=Topic 353 (VM Deployment and

Provisioning) に対応

- 各章は、公式の細目 (351.1 など) ごとの節に分かれる。コマンド名や設定ファイル名は、そのまま打てるように本文中に示す

章	Topic	扱う中心
第1章	351 Full Virtualization	仮想化の概念、Xen、QEMU (KVM)、libvirt、ディスクイメージの管理
第2章	352 Container Virtualization	名前空間・cgroup、LXC と LXD、Docker、オーケストレーション
第3章	353 VM Deployment and Provisioning	クラウド管理ツール、Packer、cloud-init、Vagrant

第1章 Full Virtualization

[351.1/351.2/351.3/351.4/351.5]

この章は、Xen、QEMU (KVM を含む)、libvirt と、ディスクイメージのコマンド群を扱います。

仮想化の概念と理論 [351.1]

用語と利点・欠点 (virtualization terminology, pros and cons)

ハイパーバイザーは、物理マシンの CPU・メモリ・デバイスを分け合って、複数の仮想マシン (VM) に見せるソフトウェアです。仮想マシンモニタ (VMM) はほぼ同じ意味の語で、LPI の細目には両方が並びます。仮想化を動かす物理機が**ホスト**、VM の中で動く OS が**ゲスト OS**です。サーバーを VM に集める作業は**統合 (コンソリデーション)**と呼び、台数・電力・設置面積が減ります (統合の効果)。

区分	内容
利点	1台に集約して資源の遊びを使い切れる
利点	VM ごとに障害と設定を隔離できる
利点	スナップショットやクローンで、状態を戻し、増やせる
利点	ホスト間の移行で、保守のときも止めずに済む
欠点	ハイパーバイザーが挟まるオーバーヘッド
欠点	ホストが落ちると、載っている VM が全部止まる 単一障害点

区分	内容
欠点	VM 同士による CPU・ディスク・ネットワークの奪い合い
欠点	作りやすさから VM が乱立するスプロール

場面で考えます。10台の物理サーバーを2台のホストに統合した構成です。集約するほど、1台の故障で止まる VM が増えます。対策は、共有ストレージとライブマイグレーションの組み合わせ、または複数ホストでの冗長化です。ホストの故障後に VM が自動で再起動するだけの構成では、再起動が終わるまで止まります(停止時間は残る)。

ハイパーバイザーとVMMの種類(Hypervisors and Virtual Machine Monitors, Emulation and Simulation)

ハイパーバイザーの種類(動く場所による分類):

種類	動く場所	例
ハイパーバイザー型 (Type 1・ベアメタル型)	ハードウェアの上で直接動く	Xen、KVM (Linux カーネル自体が役割を持つ)、VMware ESXi、Hyper-V
ホスト型 (Type 2)	通常の OS の上のアプリとして動く	VirtualBox、VMware Workstation、加速機能を使わない QEMU

仮想化の方式 (ゲストの扱いによる分類):

方式	ゲスト OS	仕組み
完全仮想化 (HVM、Hardware Virtual Machine)	無改造のまま動く	CPU の仮想化支援機能を使う。BIOS・ディスク・NIC は QEMU などのデバイスモデルがエミュレートする
準仮想化 (PV、Paravirtualization)	ハイパーバイザー対応のカーネルが要る	ゲストが仮想化を知っていて、ハイパーコールで直接頼む。エミュレートが要らず速い
PVH (Xen)	PVH 対応のカーネルが要る	両者の中間。デバイスモデルが無い軽量な HVM に近い

HVM のゲストに PV ドライバ (virtio など) を入れると、ディスクとネットワークだけが準仮想化されます。PV のゲストカーネルは古い技術でも、PV ドライバは現役です。

エミュレーションは、ある機械の動作を別の機械の上で再現して、元のソフトウェアをそのまま動かします。QEMU が x86 の上で ARM のゲストを動かす場合が、これにあたります (CPU 命令を1つずつ変換する TCG を使う)。**シミュレーション**は、対象の内部の振る舞いまでモデル化して再現し、性能や挙動を観察・分析する技術です。ゲストを実用速度で動かすのが目的ならエミュレーション、設計の検証が目的ならシミュレーションと区別します。

KVM は QEMU の代わりではありません。QEMU の CPU 実行部分を、ハードウェアで高速化する層です。Type 2 に見える QEMU も、KVM を有効にすれば、実質的にハイパーバイザー型の性能が出ます。

CPU の仮想化支援と CPU フラグ (CPU flags、/proc/cpuinfo)

x86 の仮想化支援は、2社で名前が違う:

項目	Intel	AMD
仮想化支援 (/proc/cpuinfo のフラグ)	VT-x (<code>vmx</code>)	AMD-V (<code>svm</code>)
2段のページ変換	EPT (<code>ept</code>)	NPT (<code>npt</code>)。製品名は RVI
KVM のモジュール	<code>kvm_intel</code>	<code>kvm_amd</code>

確かめるコマンド:

```
grep -E -o 'vmx|svm' /proc/cpuinfo | sort -u
lscpu | grep -i virtualization
ls -l /dev/kvm
```

`lscpu` で目印になる行:

lscpu の行	意味
<code>Virtualization: VT-x / AMD-V</code>	ホストの CPU に仮想化支援がある (Intel / AMD)
<code>Hypervisor vendor: KVM</code>	自分が仮想マシンの中において、その下のハイパーバイザーが KVM
<code>Virtualization type: full</code>	完全仮想化のゲストとして動いている

フラグがあっても、BIOS/UEFI で仮想化支援が無効なら KVM は使えません。 `/dev/kvm` が存在すれば、KVM が使える状態です[^c1-2]。ゲストの中の `/proc/cpuinfo` には、 `hypervisor` フラグが付きます。自分が仮想マシン上にいる印で、 `vmx` や `svm` の代わりにはなりません[^c1-1]。

SMP は、1台の VM に複数の仮想 CPU (vCPU) を持たせる構成です。物理コア数より多い vCPU を割り当てる**オーバーコミット**もできます (競合で遅くなる)。**NUMA** は、メモリがノードごとに分かれ、自分のノードのメモリが近くて速い構成です。VM の vCPU とメモリを同じノードに寄せると速くなります (ノードの構成は `numactl --hardware` で見る)。

移行 (Migration: P2V、V2V、ホスト間のマイグレーション)

P2V (Physical to Virtual) は、物理マシンの中身を仮想マシンに変える作業です。**V2V** (Virtual to Virtual) は、別のハイパーバイザー向けの VM を変換する作業を指します。道具は libguestfs の virt-p2v と virt-v2v です (後の節で詳しく扱う)。

ホスト間のマイグレーションは、VM を別のホストへ移す作業です。止めて移す**オフライン(コールド)**と、動かしたまま移す**ライブ**があります。ライブマイグレーションの基本はプレコピー方式です。メモリを繰り返し転送し、更新される分が十分小さくなった時点で一瞬だけ止めて切り替えます。この停止時間が**ダウンタイム**です (ミリ秒単位の短さ)。

ライブマイグレーションが成立する条件:

条件	内容
CPU	移行先で CPU の機能が互換であること。世代が違うホスト同士なら、ゲストに見せる CPU モデルを共通の下位に揃える
ネットワーク	移行先でも、同じネットワークにつながるこ と
ディスク	共有ストレージ (NFS・iSCSI・Ceph など) にあること。共有でなければ、ディスクも一緒に転送する移行を選ぶ

仮想マシンの機能と影響 (snapshotting、pausing、cloning、resource limits)

機能	内容	影響
スナップショット	ある時点のディスク(と任意でメモリ)の状態を残す	差分が増えて容量を使い、連鎖が長いと遅くなる。バックアップの代わりにはならない
一時停止 (pause)	実行を止めるが、メモリは残す	メモリを確保したままで、CPU 時間は使わない
保存 (save/suspend)	メモリの内容をファイルに書いて停止する	再開すると元の状態に戻る。ホストのメモリは解放される
クローン	VM を複製して新しい VM を作る	ホスト名・MAC・SSH ホスト鍵・machine-id が重複するので、初期化が要る
リソース制限	CPU・メモリ・ディスク I/O・ネットワークの上限や重みを決める	1台の暴走を抑える。上限が低すぎると遅くなる

一時停止ではメモリが物理に残り、保存では状態がディスクのファイルに移ります (紛らわしい違い)。クローンには、イメージ全体を複製する**フルクローン**と、差分だけ持つ**リンククローン**があります。リンククローンはバックアップファイルを使うので、速く小さく作れる代わりに、元のイメージを消すと壊れます。元のイメージは、消してはいけません。

管理基盤など知っておく名前 (oVirt、Proxmox、systemd-machined、VirtualBox、Open vSwitch)

名前	内容
oVirt	KVM の仮想化を Web で集中管理する基盤。管理サーバー (engine) が、各ホストの VDSM (ホスト側のエージェント) を通して操作する[^c1-6]
Proxmox VE	KVM の仮想マシンと LXC のコンテナを、1 つの Web 画面で管理する Debian ベースのディストリビューション[^c1-5]
systemd-machined	ホスト上で動く仮想マシンとコンテナを登録・追跡する systemd のサービス。 machinectl list で一覧を見る。サブコマンドは status、login、shell、start、poweroff、terminate。 poweroff は穏やかな停止、terminate は即座に止める[^c1-3]
VirtualBox	Oracle のホスト型 (Type 2) 仮想化ソフト。コマンドラインは VBoxManage [^c1-7]
Open vSwitch	多層の仮想スイッチ。VLAN (802.1Q)、ボンディング、OpenFlow、GRE/VXLAN などのトンネルを扱い、KVM などのホストで、Linux のブリッジの代わりに使える。管理は ovs-vsctl [^c1-4]

ovs-vsctl show は、Bridge の下に Port、Port の下に Interface を入れ子で示します。1 つの Port の下に Interface が 2 つあればボンド、Interface に type: vxlan があれば VXLAN のトンネルです。

```
$ ovs-vsctl show

Bridge br0
    Port br0
        Interface br0
            type: internal
    Port bond0
        Interface eth2
        Interface eth1
    Port vx0
        Interface vx0
            type: vxlan
            options: {remote_ip="192.0.2.20"}
```

Xen [351.2]

アーキテクチャ (Domain0、DomainU、PV-DomU、HVM-DomU、networking、storage)

Xen のハイパーバイザーは、ハードウェアの上で直接動き、CPU・メモリ・タイマー・割り込みを受け持ちます。ブートローダーの次に最初に動くプログラムで、その上に**ドメイン** (VM のこと) が載ります[^c1-8]。

Domain0 (Dom0) は、特権を持つ最初のドメインです。物理デバイスのドライバを持ち、すべてのドメインの I/O を取り次ぎ、管理ツール (xl) と XenStore も動かします。Dom0 には Xen 対応のカーネルが要ります。**DomainU (DomU)** は、特権のない一般のゲストドメインを指します。DomU の型 (3種類):

型	xl.cfg の type	特徴
PV-DomU (準仮想化)	pv	CPU の仮想化支援が要らない。Xen 対応のゲストカーネルが要る。x86 の既定値
HVM-DomU (完全仮想化)	hvm	Intel VT-x か AMD-V が要る。BIOS・ディスク・NIC を QEMU のデバイスモデルがエミュレートする。無改造の OS (Windows など) が動く
PVH	pvh	HVM に近い軽量版。デバイスモデルが無く、PV ドライバで I/O をする。PVH 対応のカーネルが要る。Arm の既定値

`builder` は古いキーで、`type` に置き換えられました (`hvm` が完全仮想化、`generic` が PV) [^c1-9]。

ネットワークとストレージは、**フロントエンドとバックエンド**の組で動きます (下の表)。DomU のフロントエンドが要求をバックエンドへ渡し、Dom0 が物理デバイスに届けます。

側	ネットワーク	ストレージ
DomU のフロントエンド	netfront	blkfront
Dom0 のバックエンド	netback	blkback

Dom0 から見ると、DomU の仮想 NIC は `vifDOMID.DEVID` という名前で現れます。HVM のエミュレート用の tap は、`vifDOMID.DEVID-emu` という名前です。この仮想 NIC を Dom0 の**ブリッジ** (既定の名前は `xenbr0`) につなぐ形が標準です。ブリッジ自体は xl が作らないので、ホスト OS のネットワーク設定で用

意します[^c1-10]。ストレージは、ファイルのイメージ、LVM の論理ボリューム、iSCSI などを DomU に渡せます。

設定ファイル (/etc/xen/、xl.cfg、xl.conf)

ドメインの設定は、/etc/xen/ の下に置く xl.cfg 形式のファイルに書きます (キー=値 を改行か ; で区切る)。

```
name = "guest1"
type = "hvm"
memory = 2048
maxmem = 4096
vcpus = 2
maxvcpus = 4
disk = [ 'format=raw, vdev=xvda, access=rw, target=/dev/vg/guest1' ]
vif = [ 'mac=00:16:3e:aa:bb:cc, bridge=xenbr0' ]
boot = "cd"
on_poweroff = "destroy"
on_reboot = "restart"
on_crash = "destroy"
```

キー	意味
name	ドメイン名。必須で、同じホストの中で重複できない
memory、maxmem	起動時のメモリと上限(MB)。maxmem は memory 以上。両者が違くと「事前にバルーンした」状態で起動し、バルーンドライバが要る

キー	意味
<code>vcpus</code> 、 <code>maxvcpus</code>	起動時にオンラインの vCPU 数と、増やせる上限
<code>uuid</code>	ドメインの UUID。省略すると自動で付く
<code>boot</code>	HVM の起動順。 <code>c</code> (ハードディスク)、 <code>d</code> (CD-ROM)、 <code>n</code> (ネットワーク) を並べる。 <code>dc</code> なら CD-ROM を先に試す。既定は <code>cd</code>
<code>on_poweroff</code> 、 <code>on_reboot</code> 、 <code>on_crash</code>	停止・再起動・クラッシュ時の動作。値は <code>destroy</code> 、 <code>restart</code> 、 <code>rename-restart</code> 、 <code>preserve</code> (残して調べる)、 <code>coredump-destroy</code> 、 <code>coredump-restart</code> 。既定は順に <code>destroy</code> 、 <code>restart</code> 、 <code>destroy</code>

`disk` と `vif` の中身は、キーと値を組にした文字列を並べる:

項目	書き方
<code>disk</code> の <code>format</code>	既定は <code>raw</code> 。 <code>qcow2</code> 、 <code>vhd</code> 、 <code>qed</code> なども指定できる
<code>disk</code> の <code>vdev</code>	ゲストから見える名前 (<code>xvda</code> 、 <code>hda</code> 、 <code>sda</code>)。必須
<code>disk</code> の <code>access</code>	<code>rw</code> か <code>ro</code> 。CD-ROM (<code>devtype=cdrom</code> かフラグ <code>cdrom</code>) の既定は <code>ro</code>
<code>disk</code> の <code>target</code>	ホスト側のパス。位置で書く短縮形 <code>'/dev/vg/guest,raw,xvda,rw'</code> も同じ意味
<code>vif</code> の <code>mac</code>	省略すると Xen 用の <code>00:16:3e</code> で始まる値が自動で付く

項目	書き方
vif の bridge	既定は xenbr0
vif の model	HVM のエミュレート NIC の種類。既定は rtl8139
vif の script	既定は /etc/xen/scripts/vif-bridge

PV のゲストをホスト側のカーネルで直接起動する場合に使うキー[^c1-11]:

キー	内容
kernel	カーネルのパス
ramdisk	initrd のパス
root	root= に渡す値
extra	カーネルへの追加引数
cmdline	書くと、root と extra は無視される

ゲスト内の GRUB を使うなら、kernel の代わりに bootloader="pygrub" を指定します。

/etc/xen/xl.conf は、ドメインごとではなく、ホスト全体の xl の設定です[^c1-12]。

項目	内容
autoballoon	on、off、auto。新しいドメインのために、xl が Dom0 のメモリを自動で減らすかを決める。既定の auto は、Xen の起動引数に dom0_mem があれば自動調整を止める

項目	内容
<code>vif.default.bridge</code>	既定のブリッジ。既定値は <code>xenbr0</code>
<code>vif.default.script</code>	既定のフックスクリプト。既定値は <code>/etc/xen/scripts/vif-bridge</code>
<code>lockfile</code>	ドメイン作成などを直列にするロックファイル。既定値は <code>/var/lock/xl</code>
<code>output_format</code>	機械向け出力の形式。 <code>json</code> (既定) か <code>sxp</code>

DomUの自動起動に関わる設定[^c1-13]:

項目	内容
<code>/etc/xen/auto/</code>	設定ファイルへのシンボリックリンクを置くと、そのドメインがホスト起動時に自動で起動する
<code>xendomains</code>	自動起動を実行するサービス。有効にしておく
<code>XENDOMAINS_AUTO</code>	<code>/etc/default/xendomains</code> の項目。自動起動のディレクトリを決める
<code>XENDOMAINS_SAVE</code> 、 <code>XENDOMAINS_RESTORE</code>	ホスト停止時の保存先 (既定 <code>/var/lib/xen/save</code>) と、起動時に復元するか

```
ln -s /etc/xen/guest1.cfg /etc/xen/auto/guest1.cfg
systemctl enable xendomains
```

xl によるドメインの管理 (management of nodes and domains)

```
xl create /etc/xen/guest1.cfg      # ドメインを作って起動（設定ファイルは絶対パス）
xl create -c /etc/xen/guest1.cfg  # 起動してすぐコンソールにつなぐ
xl list                           # ドメインの一覧
xl console guest1                 # コンソール（Ctrl+] で抜ける）
xl pause guest1 ; xl unpause guest1 # 一時停止と再開
xl shutdown guest1               # ゲストOSに終了を頼む
xl reboot guest1
xl destroy guest1                 # 電源を抜くのと同じ強制停止
```

`xl create` は、ドメインを起動した時点で戻ります。OS の起動完了は待ちません。`-p` は作成後に一時停止のままにし、`-F` はドメインが終わるまで前面で動かします。設定は上書きもできます。後ろへ `キー=値` を並べます (例: `xl create /etc/xen/guest1.cfg 'memory=1024'`)。 `xl shutdown` は、HVM では PV ドライバが無いと効きません。効かない時は、`-F` で ACPI の電源イベントを送るか、`xl destroy` を使います[^c1-14]。 `destroy` は、ゲスト OS に終了処理の機会を与えません。

```
xl save guest1 /var/lib/xen/save/guest1.chk          # 状態をファイルへ書いて停止
xl restore /var/lib/xen/save/guest1.chk              # ファイルから戻して再開
xl migrate guest1 host2                              # 稼働中のまま別ホストへ（既定は ssh 経由）
xl vcpu-list guest1 ; xl vcpu-set guest1 4           # vCPU の確認と変更
xl vcpu-pin guest1 0 2,3                             # vCPU 0 を物理CPU 2,3 に固定
xl mem-set guest1 2048m ; xl mem-max guest1 4096m    # メモリの目標と上限
xl block-attach guest1 'format=raw,vdev=xvdb,target=/dev/vg/data'
xl block-list guest1 ; xl block-detach guest1 xvdb
xl cd-insert guest1 hdc /iso/os.iso ; xl cd-eject guest1 hdc
```

`xl save` は、既定でドメインを止めます（`-c` で動かしたまま、`-p` で一時停止にできる）。`xl mem-set` と `xl mem-max` の数値には、`2048m` (MiB)・`2g` (GiB)・`512k` (KiB) のように単位を必ず付けます。単位を書かない時の扱いは、`man` ページ (KiB) と `xl help` や実装 (MiB) で食い違うためです。`mem-set` はゲストのバルーンドライバに目標を伝える操作なので、ドライバが無いなどの理由でゲストが目標に届かないことがあります。`mem-max` は上限を決めるだけで、ゲストのメモリを増やしません。`mem-set` の値は `maxmem` を、`vcpu-set` の値は `maxvcpus` を超えられません。`cd-insert` と `cd-eject` が使えるのは、HVM のドメインだけです。PV のドメインでは動きません。`block-attach` で渡せるのは、PV のブロックデバイスだけです。`vdev=hdc` のようにエミュレート用の名前で指定しても、ゲストからは PV の見え方になります[^c1-15]。

稼働中のドメインの資源を増やす場面:

`vcpus=2`、`maxvcpus=2` で起動したドメインに、vCPU を4つまで増やしたい場合、`xl vcpu-set guest1 4` は `maxvcpus` を超えるのでエラーになります。設定ファイルを `maxvcpus=4` に直し、ドメインを作り直します。メモリも同じで、`maxmem=2048` のドメインに稼働中のまま 3 GB を与えるには、メモリのホットプラグに対応したゲストが要ります。通常は `maxmem` を上げて再作成します。`maxmem=4096` で起動していれば、再起動なしに増やせます (`xl mem-set guest1 3072m` 。バルーンドライバが要る)。

監視とトラブルシューティング(troubleshooting、xentop)

`xl list` の出力:

Name	ID	Mem	VCPUs	State	Time(s)
Domain-0	0	750	4	r-----	11794.3
win	1	1019	1	r-----	0.3
linux	2	2048	2	--p---	5624.2

`Mem` は MB、`VCPUs` は vCPU 数です。`Time(s)` は CPU 時間です。ただし、仮想 I/O の処理時間は Dom0 に計上されるので、実際の負荷とは一致しません。

`xl list` の `State` 列は6文字で、該当する位置に文字、他は `-` が入ります (`r-----` は実行中)。一時停止中のドメインも、メモリは解放されません。

文字	状態	意味
<code>r</code>	running	いま CPU 上で動いている
<code>b</code>	blocked	I/O 待ちか、やることが無くて眠っている

文字	状態	意味
p	paused	一時停止 (<code>xl pause</code>)。メモリは使うが、スケジュールされない
s	shutdown	ゲストが終了を呼んだが、まだ片付いていない
c	crashed	クラッシュした。 <code>on_crash=preserve</code> のように残す設定で見える
d	dying	終了処理の途中で、まだ完全には消えていない

`xl dmesg` は、Linux の `dmesg` ではなく、Xen ハイパーバイザー自身のメッセージバッファを読みます。起動時のエラーや警告が出ます (`-c` で消去)。 `xl info` は、ホストの CPU 数、メモリ、空きメモリ、Xen のバージョン、起動引数を出します。 `xl uptime` は、各ドメインの稼働時間です。

`xl top` (`xentop`) は、ドメインごとの CPU、メモリ、ネットワーク、ブロックの使用量を実時間で表示します[^c1-16]。

オプション	内容
<code>-b</code>	バッチ出力 (標準出力へ)。スクリプト向け
<code>-i N</code>	N 回表示して終了
<code>-d 秒</code>	更新間隔。既定は3秒
<code>-n</code> 、 <code>-x</code> 、 <code>-v</code>	ネットワーク、ブロックデバイス、vCPU の情報を足す

ログは `/var/log/xen/` の下にあります。`xl`-ドメイン名`.log` が `xl` の動き、`qemu-dm-`ドメイン名`.log` が HVM のデバイスモデルの出力です[^c1-17]。ドメインが起動しない時は、`xl create -c` で画面を見ながら、まずこの2つを開きます。

知っておく名前 (XAPI、XenStore、Xen Boot Parameters、xm)

名前	内容
XAPI	XenServer と XCP-ng が使う管理ツールスタック。複数ホストをまたいで管理できる API で、 <code>xl</code> とは別の系統[^c1-18]
XenStore	ドメイン間で設定と状態を共有する、階層構造の小さなデータベース。 <code>xenstored</code> が動かし、 <code>xenstore-ls</code> で全体を、 <code>xenstore-read</code> で1項目を読む。大きなデータの転送には使わない[^c1-19]
Xen のブートパラメータ	ハイパーバイザー自身の起動引数で、GRUB の Xen の行に書く。Dom0 のメモリは <code>dom0_mem=2048M,max:2048M</code> のように決める。書く場所は GRUB の Xen の起動行 (multiboot の行。GRUB 1 では kernel の行) で、GRUB2 では <code>/etc/default/grub</code> の <code>GRUB_CMDLINE_XEN_DEFAULT</code> に書き、 <code>update-grub</code> を実行する
xm	xend という古い管理デーモンと組だった古い管理ツール。現在の <code>xl</code> が置き換えた。設定ファイルの書式も異なる

`dom0_mem` で固定したら、`xl.conf` の `autoballoon` は `off` か `auto` にします。`on` のままだと、新しいドメインを作るたびに、`xl` が Dom0 のメモリを減らします[^c1-20]。

QEMU [351.3]

アーキテクチャと KVM (architecture、networking、storage)

QEMU は、ゲスト1台につき1つのユーザー空間プロセスとして動きます。vCPU はそのプロセスのスレッドで、ディスク・NIC・VGA などのデバイスは QEMU が受け持ちます。CPU の実行方式は2つ:

方式	内容
TCG	CPU 命令をソフトウェアで変換するエミュレーション。ホストと違う CPU (ARM など) のゲストも動かせるが、遅い
KVM	Linux カーネルのモジュール。CPU とメモリの仮想化を VT-x や AMD-V に任せるので、ホストと同じ CPU のゲストが実用速度で動く

`-accel kvm` を付けないと、QEMU は TCG のままです[^c1-21]。TCG だけでは、KVM ほどの速度は出ません。

要素	内容
<code>kvm</code>	KVM の共通モジュール
<code>kvm_intel</code> / <code>kvm_amd</code>	Intel / AMD の仮想化支援を使う CPU 別のモジュール (kvm-intel、kvm-amd とも書かれる)。どちらか片方だけが読み込まれる
<code>/dev/kvm</code>	KVM の入り口になるデバイスファイル。QEMU がここを開く。通常はグループ <code>kvm</code> が書き込める

要素	内容
<code>qemu-system-x86_64</code>	x86-64 のゲストを動かす QEMU 本体。アーキテクチャごとに <code>qemu-system-arm</code> などがある

```
lsmmod | grep kvm           # kvm と kvm_intel (か kvm_amd) が出ればよい
sudo modprobe kvm_intel    # 読み込まれていない時 (AMD なら kvm_amd)
ls -l /dev/kvm
```

古い配布物では、`qemu` や `qemu-kvm` という名前のコマンドも出てきます。
`qemu-kvm` は、KVM を既定で有効にしたラッパーでした。

コマンドラインからの起動 (Start QEMU instances from the command line)

```
qemu-system-x86_64 -accel kvm -name vm1 -m 2G -smp 2 \
  -drive file=vm1.qcow2,if=virtio,format=qcow2 \
  -cdrom install.iso -boot order=dc \
  -nic user,hostfwd=tcp::2222-:22 \
  -monitor stdio
```

オプション	意味
<code>-m 2G</code>	ゲストのメモリ。単位なしは MiB。 <code>M</code> と <code>G</code> が使える。省略すると 128 MiB[^c1-22]
<code>-smp 4</code>	vCPU 数。 <code>-smp cpus=4,sockets=1,cores=2,threads=2</code> のように構成も指定できる

オプション	意味
<code>-hda file</code>	ファイルを IDE の1台目のディスクにする (<code>-hdb</code> ~ <code>-hdd</code> が2~4台目)
<code>-cdrom file</code>	ファイルを CD-ROM にする。x86 では IDE の2番目のバスの master を使うので、 <code>-hdc</code> とは同時に使えない
<code>-drive file=...,if=virtio,format=qcow2</code>	ディスクの詳細な指定。 <code>if</code> がバスの種類、 <code>format</code> がイメージの形式、 <code>media=cdrom</code> で CD-ROM、 <code>index</code> で番号
<code>-boot order=dc</code>	起動順。 <code>a</code> 、 <code>b</code> (フロッピー)、 <code>c</code> (ハードディスク)、 <code>d</code> (CD-ROM)、 <code>n</code> ~ <code>p</code> (ネットワーク)。既定はハードディスク
<code>-boot once=d</code>	初回の起動だけ CD-ROM を使う。リブート後は通常の順に戻る
<code>-accel kvm</code>	アクセラレータの選択。古い書き方は <code>-enable-kvm</code> と <code>-machine accel=kvm</code>
<code>-name vm1</code>	ゲスト名。ウィンドウのタイトルやプロセス名に使う
<code>-readconfig file</code>	設定をファイルから読み込む
<code>-monitor stdio</code>	モニタを標準入出力に出す。 <code>-nographic</code> では、シリアルコンソールとモニタが標準入出力に多重化されるのが既定で、 <code>Ctrl+a c</code> で切り替える
<code>-usb -device usb-tablet</code>	絶対座標のポインタ。VNC でマウスがずれる時に付ける

`-cdrom file` は、`-drive file=file,index=2,media=cdrom` と同じ意味です。`-hda file` は、`-drive file=file,index=0,media=disk` に近い書き方です。

ホストのカーネルを直接起動すると、ゲストの `/boot` を気にせず試せます:

```
qemu-system-x86_64 -accel kvm -m 1G -nographic \  
-kernel bzImage -initrd initrd.img \  
-append "root=/dev/vda1 console=ttyS0" \  
-drive file=root.img,if=virtio,format=raw
```

`-kernel` がカーネル、`-initrd` が初期 RAM ディスク、`-append` がカーネルの引数です。

QEMU のネットワーク(networking)

何も指定しないと、QEMU は NIC を1つと、**ユーザーモードネットワーク**(`-nic user`) を付けます[^c1-23]。root 権限が要らず、ゲストから外へは出られますが、外からゲストへは入れません。

項目	値
ゲストの IP (DHCP で渡る)	10.0.2.15
ゲートウェイ (ホスト側)	10.0.2.2
DNS	10.0.2.3

外からゲストの SSH に入るには、`hostfwd=tcp::2222-:22` を付けます (ホストの 2222 をゲストの 22 へ)。転送の無いゲストには、外から接続できません。NIC を無くすには `-nic none` を使います。

NIC の書き方は、3通り:

書き方	内容
<code>-nic</code>	NIC と接続先を1つで書く省略形
<code>-netdev</code> と <code>-device</code>	接続先(バックエンド)とゲスト側の NIC を別々に書き、 <code>id</code> と <code>netdev</code> で結ぶ
<code>-net nic -net user</code>	古い書き方。今も通る

```
-nic user,model=e1000
-netdev user,id=n0 -device virtio-net-pci,netdev=n0
-netdev tap,id=n1,ifname=tap0,script=no,downscript=no -device virtio-
net-pci,netdev=n1
-netdev bridge,id=n2,br=br0 -device virtio-net-pci,netdev=n2
```

ゲストを物理 LAN につなぐ標準の方法は、**TAP デバイスとブリッジ**です。

部品	内容
TAP	ホストの仮想 NIC。QEMU がゲストの NIC とつなぐ
ブリッジ	TAP と物理 NIC を、同じ L2 のネットワークにまとめる

作る手順:

```
ip tuntap add dev tap0 mode tap      # 古い道具: tuncctl -t tap0
ip link add br0 type bridge          # 古い道具: brctl addbr br0
ip link set tap0 master br0          # 古い道具: brctl addif br0 tap0
ip link set tap0 up ; ip link set br0 up
```

`tap` バックエンドは、起動時に `/etc/qemu-ifup`、終了時に `/etc/qemu-ifdown` を実行します。`script=no` で実行を止め、手で用意した TAP を使います(最新の QEMU の文書は `script=no` を非推奨とし、`script=` と空にする書き方を勧めている)。`bridge` バックエンドは `qemu-bridge-helper` を使います。非特権ユーザーでも TAP を作り、ブリッジ(既定は `br0`)に付けられます。TAP が動くには、`/dev/net/tun` が必要です[^c1-24]。QEMU がまだ開いていない TAP は `NO-CARRIER` のままです(ゲストの起動後にリンクが上がる)。

QEMU モニタ(QEMU monitor、snapshots)

モニタは、動いているゲストを操作する対話画面です。`-monitor stdio` で標準入出力に出します。SDL・GTK の画面では、`Ctrl+Alt+2` でモニタ、`Ctrl+Alt+1` でゲスト画面です[^c1-25]。

コマンド	内容
<code>info status</code>	<code>running</code> か <code>paused</code> か
<code>info block</code>	ブロックデバイス(ディスクと CD-ROM)の一覧
<code>info network</code>	ネットワークデバイス
<code>info cpus</code>	vCPU の一覧
<code>info kvm</code>	KVM が有効かどうか
<code>info snapshots</code>	スナップショットの一覧
<code>savevm 名前</code>	現在の状態(ディスク・メモリ・デバイス)をスナップショットとして保存
<code>loadvm 名前</code>	スナップショットに戻す
<code>delvm 名前</code>	スナップショットを削除

コマンド	内容
<code>change ide1-cd0 file.iso</code>	CD-ROM のメディアを差し替える
<code>eject [-f] ide1-cd0</code>	CD-ROM を取り出す。ロックされていれば <code>-f</code> で強制
<code>stop</code> / <code>cont</code>	一時停止 / 再開
<code>system_powerdown</code>	電源ボタンを押した扱い (ACPI)
<code>quit</code>	QEMU を終了する。ゲストの電源を即座に 切るのと同じ

`info block` と `info snapshots` の出力例:

```
(qemu) info block
virtio0 (#block149): /tmp/vm/d.qcow2 (qcow2)
    Attached to:      /machine/peripheral-anon/device[0]/virtio-backend
    Cache mode:      writethrough
ide1-cd0: [not inserted]
    Removable device: not locked, tray closed
(qemu) info snapshots
List of snapshots present on all disks:
ID      TAG          VM_SIZE      DATE          VM_CLOCK
ICOUNT
--      snap1        1.63 MiB 2026-10-13 19:26:03 0000:00:03.931
--
```

`ide1-cd0: [not inserted]` は、CD-ROM が空の状態です。`savevm` が書き込む
先は、ゲストに接続されている **qcow2 のディスクイメージの内部**です。ディス

クが raw だけのゲストでは、`does not support snapshots` というエラーになります。raw には、スナップショットを保存できません。内部スナップショットは `qemu-img snapshot -l` でも一覧できます。メモリの内容も含むので、`loadvm` で戻すと保存時点の実行状態から再開します。CD-ROM の入れ替えで使うデバイス名は、`info block` に出ます (`ide1-cd0` など)。

ゲストエージェントと VirtIO (QEMU Guest Agent、VirtIO device drivers)

VirtIO は、仮想化専用の準仮想化デバイスの規格です。エミュレートする実機のデバイス (IDE や e1000) より速くなります (準仮想化の効果)。

デバイス	役割
<code>virtio-blk</code>	ディスク
<code>virtio-net</code>	NIC
<code>virtio-balloon</code>	メモリのバルーン
<code>virtio-serial</code>	ホストとゲストの間の通信路

Linux のカーネルは、標準で VirtIO ドライバを持ちます。Windows には、`virtio-win` のドライバを別に入れます。Windows の標準のインストーラーは、VirtIO のディスクを認識しません。原因は、このドライバの不足です。

QEMU ゲストエージェント (`qemu-ga`) は、ゲストの中で動くデーモンです。ホストから実行できる操作:

操作	例
情報の取得	ゲストのホスト名、ネットワーク、ファイルシステム

操作	例
時刻	ゲストの時刻の設定
ファイルシステム	凍結と解凍
電源	シャットダウン
アカウント	パスワードの設定

ホストとの通信は、既定で virtio-serial の `/dev/virtio-ports/org.qemu.guest_agent.0` です[^c1-26]。QEMU 側は、次の3つを組み合わせで用意します:

```
-chardev socket,path=/tmp/qga.sock,server=on,wait=off,id=qga0 \  
-device virtio-serial \  
-device virtserialport,chardev=qga0,name=org.qemu.guest_agent.0
```

ゲストで `qemu-guest-agent` を入れて動かすと、ホストの UNIX ソケット経由でゲストを操作できます。スナップショットを取る前にファイルシステムを凍結する用途が代表的です。

トラブルシューティングと主な設定値 (troubleshooting、configuration parameters)

症状	原因と対処
<code>Could not access KVM kernel module</code> や <code>/dev/kvm</code> が無い	BIOS/UEFI で仮想化支援が無効か、 <code>kvm_intel</code> / <code>kvm_amd</code> が読み込まれていない。 <code>/dev/kvm</code> があっても開けない時は、実行ユーザーがグループ <code>kvm</code> に入っていない

症状	原因と対処
ユーザーモードで外からゲストに入れない	<code>hostfwd</code> が無い。ゲストから外へ出られれば、ユーザーモードの設定自体は正常
TAP を使うゲストが外と通信できない	TAP がブリッジに付いているか (<code>ip link show master br0</code>)、 <code>/dev/net/tun</code> があるか、TAP とブリッジが <code>up</code> かを調べる
<code>Failed to get "write" lock</code>	同じイメージを別の QEMU プロセスが書き込みで開いている。 <code>qemu-img</code> で同じイメージを開いても同じ警告が出る
ディスクが見えない	<code>if=virtio</code> を使うのに、ゲストに VirtIO ドライバが無い。 <code>format=</code> を書かないと形式を推測するので、明示する
起動しない	<code>-boot order</code> の指定とメディアの有無を確認める。 <code>-cdrom</code> と <code>-hdc</code> を同時に指定すると、同じ場所を取り合う

主な設定値と、目的別の引数:

目的	引数
ホストの CPU の機能をゲストに見せる (汎用なら <code>-cpu qemu64</code>)	<code>-cpu host</code>
ISO からインストールし、次回から通常の順で起動する	<code>-cdrom x.iso -boot once=d</code>
ネットワークから起動する	<code>-boot order=n</code>
画面を出さず、シリアルとモニタを標準入出力に出す	<code>-nographic</code>
VNC で画面を出す (<code>:1</code> は TCP 5901)	<code>-vnc :1</code>

目的	引数
ディスクへの書き込みを一時ファイルに逃がし、終了で捨てる	<code>-snapshot</code>
バックグラウンドで動かす	<code>-daemonize</code>
マシン型を選ぶ	<code>-machine</code>

libvirt による仮想マシンの管理 [351.4]

アーキテクチャ (architecture of libvirt、Xen と QEMU との関わり、dnsmasq と radvd)

libvirt は、KVM/QEMU、Xen、LXC などの仮想化を、同じ API と同じコマンドで扱う管理層です。階層は3つ:

階層	内容
管理ツール	<code>virsh</code> 、 <code>virt-manager</code> 、 <code>virt-install</code> 、 <code>virt-viewer</code> 、 <code>virt-clone</code>
デーモンとドライバ	常駐するデーモンと、その中の ドライバ 。ドライバは仮想化の種類ごとにあり、QEMU/KVM は <code>qemu</code> 、Xen は <code>libxl</code> を使う <code>xen</code>
ハイパーバイザー	QEMU や Xen

ドメイン(libvirt での VM の呼び名)の定義は、すべて **XML** で持ちます。デーモンには一体型と分割型(モジュラー)があります。一体型の **libvirtd** は、全ドライバを1つのプロセスで動かします。設定は `/etc/libvirt/libvirtd.conf` です[^c1-27]。分割型は、ドライバごとに別のデーモンを持ちます(下の表):

デーモン	受け持ち
<code>virtqemud</code>	QEMU/KVM のドメイン
<code>virtxend</code>	Xen のドメイン
<code>virtnetworkd</code>	仮想ネットワーク
<code>virtstoraged</code>	ストレージプール
<code>virtnodedevd</code> 、 <code>virtnwfilterd</code> 、 <code>virtsecret</code> d、 <code>virtinterfaced</code>	ノードのデバイス、ネットワークフィルタ、秘密情報、ホストのインターフェース
<code>virtproxyd</code>	リモートからの接続の受け口。一体型 <code>libvirtd</code> の UNIX ソケットとの互換も担う

設定とファイルの場所:

パス	内容
<code>/etc/libvirt/libvirtd.conf</code>	libvirtd の設定
<code>/etc/libvirt/qemu.conf</code>	QEMU ドライバの設定
<code>/etc/libvirt/libxl.conf</code>	Xen ドライバの設定
<code>/etc/libvirt/qemu/ドメイン名.xml</code>	QEMU ドライバのドメイン定義 XML
<code>/etc/libvirt/qemu/networks/</code>	仮想ネットワークの定義
<code>/etc/libvirt/qemu/autostart/</code>	自動起動の設定。定義へのシンボリックリンクができる

QEMU は、libvirt がコマンドラインを組み立てて起動します (`ps` で、XML から生成された長い引数が見える)。デーモンを再起動しても、動作中のゲスト

は止まりません。Xen の場合、libvirt は `xl` コマンドを呼ばず、`libxl` という Xen のライブラリを直接使います。

libvirt が連携するサービス:

サービス	役割
dnsmasq	仮想ネットワーク(<code>default</code> など)を起動すると、ネットワークごとに起動して、DHCP と DNS をゲストに提供する。IPv6 のルーター広告 (RA) も担う
radvd	古い libvirt が、IPv6 の RA のために起動していた。今の libvirt は dnsmasq 2.64 以降の RA 機能を使い、radvd を起動しない [^c1-28]

接続とノードの管理 (connections and nodes)

接続先は **URI** で指定します。`virsh -c URI` で指定するか、環境変数 `LIBVIRT_DEFAULT_URI` に入れます。

URI	接続先
<code>qemu:///system</code>	ローカルの root のデーモン(システムモード)。VM をシステム全体で動かす標準
<code>qemu:///session</code>	一般ユーザーごとのデーモン(セッションモード)。そのユーザーの VM だけ
<code>qemu+ssh://user@host/system</code>	SSH 経由でリモートの QEMU/KVM に接続
<code>qemu+tls://host/system</code>	TLS 経由のリモート接続
<code>xen:///system</code>	ローカルの Xen。リモートは <code>xen+ssh://user@host/system</code>

URI	接続先
test:///default	動作確認用の模擬ハイパーバイザー

リモート接続の TCP ポートは、TLS が 16514、暗号化なしが 16509 です。16509 は、暗号化できる認証 (GSSAPI/Kerberos) を使う時以外は避けます [^c1-29]。qemu:///session では、システム全体の VM は見えません。

```
virsh -c qemu+ssh://root@host2/system list --all
virsh uri ; virsh hostname
virsh nodeinfo      # ホストの CPU 数・ソケット・メモリ・NUMA
virsh capabilities  # ホストの能力 (XML)
virsh dominfo vm1   # ドメインの情報
```

コマンド	出力
virsh nodeinfo	ホストの CPU のモデル・数・周波数、ソケット・コア・スレッド、NUMA セル、メモリ
virsh dominfo ドメイン	Id、UUID、状態、CPU 数、最大メモリ、使用メモリ、永続か、自動起動か
virsh domstate ドメイン	状態だけ (running 、 paused 、 shut off)
virsh domblklist 、 virsh domiflist	ドメインのディスクの一覧、インターフェースの一覧
virsh capabilities	ホストの能力 (XML)

ドメインの作成と管理 (QEMU and Xen domains、virsh subcommands)

ドメインの種類:

種類	動き
永続	XML が保存され、止めても残る
一時(トランジェント)	止めると消える

コマンド	内容
<code>virsh define file.xml</code>	XML から永続ドメインを登録する。起動はしない
<code>virsh create file.xml</code>	XML から一時ドメインを作って、すぐ起動する。止めると消える
<code>virsh start vm1</code>	登録済みで停止中のドメインを起動する (<code>-console</code> で起動後に接続、 <code>--paused</code> で一時停止のまま起動)
<code>virsh shutdown vm1</code>	ゲスト OS に終了を頼む
<code>virsh reboot vm1</code>	ゲスト OS に再起動を頼む
<code>virsh destroy vm1</code>	即座に強制停止する。電源を抜くのと同じで、定義やディスクは残る
<code>virsh undefine vm1</code>	定義を削除する。ディスクは残る (<code>--remove-all-storage</code> で一緒に削除)
<code>virsh autostart vm1</code>	ホスト起動時に自動で起動する。 <code>--disable</code> で解除
<code>virsh list</code> / <code>virsh list --all</code>	稼働中のみ / 停止中も含める

コマンド	内容
<code>virsh console vm1</code>	シリアルコンソールにつなぐ。 <code>[Ctrl+]</code> で抜ける
<code>virsh suspend vm1</code> / <code>virsh resume vm1</code>	一時停止 / 再開(メモリは残る)
<code>virsh save vm1 file</code> / <code>virsh restore file</code>	状態をファイルに保存して停止 / 復元
<code>virsh managedsave vm1</code>	libvirt が管理する場所に状態を保存して停止。次の <code>start</code> で自動的に復元
<code>virsh dump vm1 file</code>	メモリのコアダンプを取る。既定ではダンプの間だけ一時停止し、終わると再開する。 <code>--live</code> なら止めずに取る
<code>virsh change-media vm1 hdc file.iso</code>	CD-ROM のメディアを差し替える(<code>--eject</code> で取り出し)
<code>virsh qemu-monitor-command vm1 --hmp 'info status'</code>	QEMU モニタのコマンドを libvirt 経由で送る。 <code>--hmp</code> は人向けのモニタコマンドの形式

`virsh list --all` と `virsh dominfo` の出力例:

```
$ virsh list --all
```

Id	Name	State
----	------	-------

-	vm1	shut off
---	-----	----------

```
$ virsh dominfo vm1
```

Id: -

Name: vm1

UUID: 373f88b4-3d7e-4590-bbdd-616efd6ce15b

OS Type: hvm

State: shut off

CPU(s): 2

Max memory: 2097152 KiB

Used memory: 2097152 KiB

Persistent: yes

Autostart: disable

Managed save: no

停止中のドメインの `Id` は `-` です。`Max memory` は定義の上限 (`<memory>`)、`Used memory` は現在の割り当て (`<currentMemory>`) です。単位は KiB です。`Persistent` が永続、`Autostart` が自動起動の有無です。

`destroy` は、ドメインを消す操作ではなく、強制停止です (名前から取り違えやすい)。定義もディスクも消しません。定義を消すのは `undefine` の役目です。付けるオプション:

オプション	必要な場合
<code>--remove-all-storage</code>	ディスクも一緒に消したい(付けなければ、ディスクは残る)
<code>--managed-save</code>	保存状態 (managedsave) があるドメイン。 付けないとエラー
<code>--snapshots-metadata</code>	スナップショットのメタデータがあるドメイン。 付けないとエラー

新しいドメインは、XML を書く代わりに **virt-install** で作れます。

```
virt-install --name vm1 --memory 2048 --vcpus 2 \  
  --disk size=20,format=qcow2 --cdrom /iso/debian.iso \  
  --os-variant debian12 --network network=default --graphics none
```

引数	意味
<code>--name</code> 、 <code>--memory</code> 、 <code>--vcpus</code>	名前、メモリ (MiB)、vCPU 数
<code>--disk</code>	ディスク。 <code>size=</code> は GiB、 <code>format=</code> は形式
<code>--cdrom</code>	インストール用の ISO
<code>--location</code>	インストールツリー (URL かディレクトリ)。 <code>--extra-args</code> でカーネル引数を渡せる
<code>--os-variant</code>	OS を指定すると、VirtIO などの既定が選ばれる
<code>--network</code> 、 <code>--graphics</code>	ネットワークの接続先、画面の出し方 (<code>none</code> でテキスト)

引数	意味
<code>--import</code>	既存のディスクイメージを使い、インストールを省く
<code>--pxe</code> 、 <code>--boot</code>	PXE でのインストール、起動方法の指定
<code>--connect</code>	接続先の URI。 <code>xen:///</code> なら Xen のゲストを作る

`--noautoconsole` を付けると、ゲストのコンソールへ自動で接続しません (`virt-install` はすぐ戻る) [^c1-38]。

ツール	内容
<code>virt-manager</code>	libvirt のデスクトップ GUI。ドメインの作成・操作・コンソール表示ができる
<code>virt-viewer</code>	稼働中のドメインのグラフィカルコンソール (VNC か SPICE) を開く軽量なビューア。 <code>virt-viewer -c qemu:///system vm1</code> のように使う
<code>virt-clone</code>	既存のドメインを複製する

```
virt-clone --original vm1 --name vm2 --auto-clone
```

`virt-clone` の元のドメインは、停止している必要があります。`--auto-clone` を付けると、新しい名前とディスクのパスを自動で決めます。MAC アドレスや UUID など、衝突してはいけないものは新しく振ります。ただし、ゲストの内側 (ホスト名、固定 IP、SSH ホスト鍵、パスワード) には触れません。そこは `virt-sysprep` の役割です [^c1-37]。

スナップショット(snapshots)

```
virsh snapshot-create-as vm1 snap1 "アップグレード前"
virsh snapshot-list vm1
virsh snapshot-current vm1 --name
virsh snapshot-revert vm1 snap1
virsh snapshot-delete vm1 snap1
```

コマンド	内容
snapshot-create-as	引数で作る。--disk-only でディスクだけ、--atomic で「全部成功するか、何も変えずに失敗するか」のどちらかにする(対応しないハイパーバイザーもある)
snapshot-create	XML ファイルで作る
snapshot-revert	戻す。スナップショット以後の変更を捨てる破壊的な操作。--running や --paused で、戻した後の状態を指定できる
snapshot-delete	データごと消す。--metadata ならメタデータだけ消す

内部スナップショットは、ディスクが qcow2 であることが前提です。

資源の管理と分析(resource consumption of domains)

コマンド	内容
virsh setmem vm1 1G	ゲストのメモリを変える。単位なしは KiB 。--live は稼働中、--config は次回起動、--current は現在の状態に合わせる

コマンド	内容
<code>virsh setmaxmem vm1 4G</code>	メモリの上限を変える
<code>virsh setvcpus vm1 4</code>	vCPU 数を変える。 <code>--maximum</code> で上限、 <code>--config</code> と <code>--live</code> で対象を選ぶ
<code>virsh vcpuinfo vm1</code>	vCPU ごとの状態、載っている物理 CPU、親和性 (y が許可)
<code>virsh vcpupin vm1 0 0-3,^2</code>	vCPU 0 を物理 CPU の 0~3 (2 を除く) に固定する。r で全部に戻す
<code>virsh domstats vm1</code>	ドメインの統計を一括で出す (CPU、バールン、ブロック、ネットワーク)
<code>virsh memtune</code> 、 <code>virsh schedinfo</code>	メモリの制限値、CPU スケジューラのパラメータ

`setmem` はメモリバールンで変えるので、`<memory>` (`setmaxmem`) の範囲内で動きます。稼働中のドメインに `--live` だけで加えた変更は、次の起動で元に戻るため、永続化には `--config` も付けます。Xen で稼働中に調整できるのは、準仮想化か、PV のバールンドライバを持つドメインだけです^[^c1-30]。

停止せずに vCPU を2から4へ増やし、次の起動後も4にするには、`virsh setvcpus vm1 4 --live --config` を使います。ただし、定義の `<vcpu>` の上限が2のままだと、4にはできません。先に `virsh setvcpus vm1 4 --maximum --config` で上限を上げ、ドメインを起動し直します。上限は、起動時にしか確定しないためです。`setmem` は、定義された上限を超えられません。

ストレージプールとボリューム(storage pools and volumes)

ストレージプールは、ディスクの置き場を libvirt が管理する単位で、その中の個々のディスクが**ボリューム**です。プールの種類:

種類	内容
dir	ディレクトリ
fs	フォーマット済みのブロックデバイス
netfs	NFS
logical	LVM のボリュームグループ
disk	物理ディスク
iscsi、rbd、zfs	iSCSI、Ceph の RBD、ZFS

```
virsh pool-define-as mypool dir --target /var/lib/mypool
virsh pool-build mypool
virsh pool-start mypool
virsh pool-autostart mypool
virsh pool-list --all
virsh vol-create-as mypool disk1.qcow2 10G --format qcow2
virsh vol-list mypool --details
```

pool-list と vol-list --details の出力例:

```
$ virsh pool-list --all
```

Name	State	Autostart
------	-------	-----------

mypool	active	yes
--------	--------	-----

p1	inactive	no
----	----------	----

```
$ virsh vol-list mypool --details
```

Name	Path	Type	Capacity
------	------	------	----------

Allocation

disk1.qcow2	/var/lib/mypool/disk1.qcow2	file	10.00 GiB	196.00
-------------	-----------------------------	------	-----------	--------

KiB

`Capacity` は仮想サイズ、`Allocation` は実際の使用量です。順序は、定義（`pool-define-as`）、作成（`pool-build`。 `dir` ならディレクトリを作る）、起動（`pool-start`）、自動起動（`pool-autostart`）の4段階です。`vol-create-as` は、プール名、ボリューム名、容量の順に指定します。ボリュームは `vol-delete` で、プールは `pool-destroy` で停止、`pool-undefine` で定義を消します。`pool-destroy` は、データを消さずに停止するだけです[^{^c1-31}]。

仮想ネットワーク(virtual networks)

仮想ネットワークは、ホスト内の仮想スイッチ(Linux のブリッジ) と、その接続方式の組で、XML で定義します。

```
<network>
  <name>default</name>
  <forward mode='nat' />
  <bridge name='virbr0' stp='on' delay='0' />
  <ip address='192.168.122.1' netmask='255.255.255.0'>
    <dhcp>
      <range start='192.168.122.2' end='192.168.122.254' />
    </dhcp>
  </ip>
</network>
```

forward mode	動き
nat	ゲストがホストの IP に変換されて外へ出る。外からゲストへは入れない。 既定の default ネットワーク はこれで、ブリッジは <code>virbr0</code> 、範囲は <code>192.168.122.0/24</code>
route	NAT なしで経路を通す。外側のルーターに、このネットワークへの経路が要る
open	<code>route</code> と同じだが、 <code>libvirt</code> がファイアウォールの規則を入れない
(forward なし)	隔離ネットワーク。ゲスト同士とホストだけが通信できる
bridge	ホストに既にあるブリッジをそのまま使う。DHCP は <code>libvirt</code> の範囲外

```
virsh net-define mynet.xml
virsh net-start mynet
virsh net-autostart mynet
virsh net-list --all
virsh net-edit mynet          # XML を編集
virsh net-dumpxml mynet
virsh net-destroy mynet      # 停止（定義は残る）
virsh net-undefine mynet
```

仮想ネットワークの状態は `net-list --all` で見ます。

```
$ virsh net-list --all

Name          State      Autostart    Persistent
-----
default       inactive  no           yes
```

`State` が `inactive` なら `net-start`、`Autostart` が `no` なら `net-autostart` が必要です。

ドメインのインターフェースは、2通りでつなげる[^c1-32]:

書き方	内容
<code><interface type='network'><source network='default'/></code>	仮想ネットワークへ接続する
<code><interface type='bridge'><source bridge='br0'/></code>	ホストのブリッジに直接つなぐ。物理 LAN と同じセグメントに出したい時は、こちら

ネットワークフィルタ(nwfilter)は、ゲストのインターフェースごとにパケットを制限する仕組みです。扱うコマンド:

コマンド	内容
<code>virsh nwfilter-list</code>	一覧
<code>virsh nwfilter-define</code>	登録
<code>virsh nwfilter-edit</code>	編集
<code>virsh nwfilter-dumpxml</code>	XML の表示
<code>virsh nwfilter-undefine</code>	削除

ドメインの `<interface>` に `<filterref filter='clean-traffic'/>` を書くと、`clean-traffic` のフィルタが付きます。MAC と IP のなりすましを防ぐフィルタです[^c1-33]。

ノード間の移行 (Migrate domains between nodes)

```
virsh migrate --live vm1 qemu+ssh://host2/system
virsh migrate --live --persistent --undefinesource vm1
qemu+ssh://host2/system
virsh migrate --live --copy-storage-all vm1 qemu+ssh://host2/system
```

オプション	内容
<code>--live</code>	止めずに移す
<code>--persistent</code>	移行先で永続にする
<code>--undefinesource</code>	移行元の定義を消す

オプション	内容
<code>--copy-storage-all</code>	共有でないディスクを、全部コピーする
<code>--copy-storage-inc</code>	共有でないディスクを、差分でコピーする。 同じ元イメージを両方に置く
<code>--p2p</code>	ホスト同士が直接やり取りする
<code>--tunnelled</code>	libvirt の経路に載せて暗号化する

`--persistent` と `--undefinesource` を付けないと、移行先では一時ドメインになり、移行元には定義が残ります。コピー移行では、移行先の同じパスに、同じ形式と大きさのイメージを先に用意します (libvirt の移行の文書は、移行先のパスが libvirt のストレージプールの中なら自動で作られる、とも書く) [^c1-34]。共有ストレージが無い2台のホストで、`vm1` を停止せずに移し、移行先に永続で残すには、同じ大きさの空のイメージを先に作ります。続けて次のコマンドを実行します:

```
virsh migrate --live --persistent --copy-storage-all vm1
qemu+ssh://host2/system
```

移行元の定義まで消すなら、`--undefinesource` を足します。

XML 設定 (libvirt XML configuration files)

```
virsh dumpxml vm1 > vm1.xml    # 現在の定義を書き出す (--inactive で次回起動時の定義)
virsh edit vm1                 # エディタで直接編集。保存時に検証される
virsh define vm1.xml           # 編集した XML を登録
```

ドメイン XML の骨格:

SAMPLE

```
<domain type='kvm'>
  <name>vm1</name>
  <memory unit='GiB'>2</memory>
  <currentMemory unit='GiB'>2</currentMemory>
  <vcpu placement='static'>2</vcpu>
  <os>
    <type arch='x86_64' machine='pc'>hvm</type>
    <boot dev='hd' />
  </os>
  <devices>
    <emulator>/usr/bin/qemu-system-x86_64</emulator>
    <disk type='file' device='disk'>
      <driver name='qemu' type='qcow2' />
      <source file='/var/lib/libvirt/images/vm1.qcow2' />
      <target dev='vda' bus='virtio' />
    </disk>
    <interface type='network'>
      <source network='default' />
      <model type='virtio' />
    </interface>
    <graphics type='vnc' port='-1' autoport='yes' />
    <console type='pty' />
  </devices>
</domain>
```

要素	意味
<code><domain type='kvm'></code>	ハイパーバイザーの種類 (<code>kvm</code> 、 <code>qemu</code> 、 <code>xen</code> 、 <code>lxc</code> など)
<code><memory></code> 、 <code><currentMemory></code>	上限と現在のメモリ。 <code>unit</code> を省くと <code>KiB[^c1-35]</code>
<code><vcpu></code>	vCPU 数
<code><os><type></code>	<code>hvm</code> は完全仮想化。 <code>arch</code> と <code>machine</code> も指定できる
<code><boot dev='hd' /></code>	起動デバイス (<code>hd</code> 、 <code>cdrom</code> 、 <code>network</code>)
<code><devices></code> の <code><emulator></code>	QEMU の実行ファイル
<code><disk></code>	<code><driver></code> が形式、 <code><source></code> がファイル、 <code><target></code> がゲストからの名前とバス
<code><interface></code> 、 <code><graphics></code> 、 <code><video></code> 、 <code><console></code>	NIC、画面、ビデオ、コンソール
<code><on_poweroff></code> 、 <code><on_reboot></code> 、 <code><on_crash></code>	停止・再起動・クラッシュ時の動作

virtlogd と virtlockd

デーモン	役割
virtlogd	QEMU のコンソール出力などのログを受ける。QEMU にパイプを渡し、ログファイルへの書き込みと、サイズ制限・ローテーションを受け持つ。libvirt を再起動してもログを失わない。ログは <code>/var/log/libvirt/qemu/ドメイン名.log</code>

デーモン	役割
virtlockd	ディスクイメージなどにロックをかける。同じイメージを2つのドメインが同時に書き込む事故を防ぐ。ロックは QEMU プロセスがディスクを開いている間、保持される。使うには <code>qemu.conf</code> で <code>lock_manager = "lockd"</code> を指定する

どちらも、動作中の QEMU がある間は停止してはいけません。ソフトウェアを更新する時は、`SIGUSR1` を送って、自分自身を再実行させます[^c1-36]。

仮想マシンのディスクイメージの管理 [351.5]

ディスクイメージの形式と OVF (raw、qcow2、VMDK、Open Virtualization Format)

形式	特徴
<code>raw</code>	ディスクの内容をそのまま並べたファイル。機能は無いが、変換が要らず速い。 スパースファイル にできる
<code>qcow2</code>	QEMU の標準形式。使った分だけ領域を確保する。内部スナップショット、圧縮、暗号化、 バックアップファイル (差分イメージ)を持つ
<code>vmdk</code>	VMware の形式
<code>vdi</code>	VirtualBox の形式
<code>vhd (vpc)、vhdx</code>	Hyper-V の形式。 <code>vhdx</code> のほうが新しい

スパースファイルは、中身が全部 0 の区間にディスク領域を割り当てないファイルです。見かけの大きさ（仮想サイズ）が 1 GiB でも、実際の使用量は 0 に近くなります（スパースの効果）。大きさの見方：

道具	表示
<code>qemu-img info</code>	<code>virtual size</code> （ゲストから見える大きさ）と <code>disk size</code> （ホストでの実使用量）
<code>ls -l</code>	見かけの大きさ
<code>du</code>	実際の使用量

```
$ qemu-img info overlay.qcow2
image: overlay.qcow2
file format: qcow2
virtual size: 20 GiB (21474836480 bytes)
disk size: 196 KiB
cluster_size: 65536
backing file: /tmp/vm/base.qcow2
backing file format: qcow2
```

`backing file` の行があれば、差分イメージです。`truncate -s 1G file` で作った raw のファイルは、`disk size` が 0 のスパースです。qcow2 は、書き込まれた分だけ大きくなります。

OVF (Open Virtualization Format) は、仮想マシンを異なる仮想化製品の間で持ち運ぶパッケージの標準です。

拡張子	内容	個数
.ovf	記述ファイル (XML)。仮想ハードウェアとディスクの参照を書く	1
.mf	マニフェスト。各ファイルの SHA ダイジェストを持ち、改変を検査する	0～1
.cert	マニフェストへの署名の証明書	0～1
ディスクイメージ(多くは .vmdk)	仮想ディスク	0以上

これらを tar で1つにまとめたファイルが .ova です[^c1-39]。

qemu-img によるイメージの管理 (Manage virtual machine disk images using qemu-img)

```
qemu-img create -f qcow2 vm.qcow2 20G # 新しいイメージ
qemu-img create -f qcow2 -b base.qcow2 -F qcow2 overlay.qcow2 # バッ
キングファイル付きの差分
qemu-img info vm.qcow2 # 形式・仮想サイズ・
実サイズ
qemu-img info --backing-chain overlay.qcow2 # バックアップファイル
の連鎖も出す
qemu-img convert -f raw -O qcow2 in.img out.qcow2 # 形式の変換 (-c で
圧縮、-p で進捗)
qemu-img resize vm.qcow2 +10G # 10G 大きくする
qemu-img resize --shrink vm.qcow2 5G # 縮める
qemu-img check vm.qcow2 # 整合性の検査 (-r
leaks / -r all で修復)
qemu-img snapshot -l vm.qcow2 # 内部スナップショットの一覧
qemu-img snapshot -c snap1 vm.qcow2 # 作成
qemu-img snapshot -a snap1 vm.qcow2 # 適用 (戻す)
qemu-img snapshot -d snap1 vm.qcow2 # 削除
```

項目	内容
convert の -f と -O	入力の形式と出力の形式 (-O は大文字)。結果は新しいファイルで、入力の上書きされない

項目	内容
<code>convert -c</code>	圧縮。圧縮に対応した形式(qcow2 など)の出力でだけ使える
<code>resize</code> の <code>+10G</code> と <code>10G</code>	増分と、絶対値
<code>resize --shrink</code>	縮める操作に必要。データの切り捨てを伴うので、付けないとエラーになる
<code>resize</code> の対象	イメージの大きさだけ。ゲストの中のパーティションとファイルシステムは変わらず、中を広げるのはゲスト内の作業か <code>virt-resize</code>
<code>snapshot</code>	qcow2 の内部スナップショット。QEMU モニタの <code>savevm</code> で作ったものも、同じ一覧に出る[^c1-40]

サブコマンド	内容
<code>commit</code>	差分イメージの内容を、バックアップファイルへ書き戻す
<code>rebase</code>	バックアップファイルを付け替える
<code>map</code>	イメージの割り当て状況を示す
<code>measure</code>	変換後に必要なファイルの大きさを見積もる
<code>compare</code>	2つのイメージの内容が同じかを調べる

libguestfs: guestfish と guestmount (libguestfish)

libguestfs は、ゲストを起動せずに、ディスクイメージの中を直接読み書きするライブラリです。内部で小さな Linux の環境 (アプライアンス) を起動してイメージを開くので、ホストの `mount` 権限や `losetup` は要りません。同梱のコマ

ンドは、すべて**ゲストを停止してから**使います（動作中のゲストのイメージに書き込むと、中身が壊れる[^c1-41]。動作中のゲストを調べるなら、読み取り専用の `--ro` にする）。

イメージの指定は、どのコマンドでも共通:

指定	内容
<code>-a</code> ファイル	ディスクイメージ。複数ディスクなら <code>-a</code> を繰り返す
<code>-d</code> ドメイン名	libvirt のドメイン
<code>-i</code>	OS を調べて、ファイルシステムを適切な場所に自動でマウントする
<code>--ro</code>	読み取り専用

guestfish は、イメージの中を操作する対話シェルです。

```
guestfish --ro -a disk.img

><fs> run                                # アプライアンスを起動（最初に必要）
><fs> list-filessystems                   # /dev/sda1: ext4 のように出る
><fs> mount-ro /dev/sda1 /                # マウント
><fs> ls /etc
><fs> cat /etc/fstab
><fs> exit

guestfish -a disk.img -i                 # OS を調べて自動でマウント
guestfish --ro -d dom1 -i cat /etc/hostname
```

サブコマンド	内容
<code>run</code>	アプライアンスを起動する
<code>list-filesystems</code> 、 <code>list-partitions</code>	ファイルシステム、パーティションの一覧
<code>mount</code> 、 <code>mount-ro</code> 、 <code>umount</code>	マウントと解除
<code>ls</code> 、 <code>cat</code> 、 <code>edit</code> 、 <code>write</code> 、 <code>mkdir</code>	ゲスト内のファイル操作
<code>upload</code> 、 <code>download</code>	ホストからイメージへ、イメージからホストへ
<code>copy-in</code> 、 <code>copy-out</code>	ディレクトリごと再帰的にコピー

スクリプトでは、ヒアドキュメントで1行ずつ並べます。

guestmount は、イメージ内のファイルシステムを FUSE でホストにマウントし、通常のファイル操作で扱えるようにします。外す時は **guestunmount** を使う。試験範囲の一覧には `guestmount` と書かれていますが、`libguestfs` の文書と配布物 (Debian 13 で確認) にあるコマンドは `guestunmount` だけです。

```
guestmount -a disk.img -m /dev/sda1 --ro /mnt # -m でデバイスを指定
guestmount -a disk.img -i /mnt                # -i で自動
guestmount -d dom1 -i --ro /mnt              # libvirt のドメインから
guestunmount /mnt
```

FUSE を使うので、`root` でなくてもマウントできます。`guestunmount` は `fusermount` を包んだコマンドで、`umount` の代わりに使います[^c1-42]。

virt-* ツール(virt-cat から virt-sysprep まで)

イメージに書き込む道具はゲストを停止して使います。読むだけの道具 (virt-cat・virt-df など) は動いているゲストにも使えます。-a ファイル と -d ドメイン の指定が共通です。

コマンド	内容
virt-cat -a disk.img /etc/hostname	イメージ内のファイルの中身を表示する
virt-copy-in -a disk.img file /dest	ホストのファイルやディレクトリを、イメージ内へ再帰的にコピーする。最後の引数がイメージ内の宛先ディレクトリ
virt-copy-out -d dom /home/guest .	イメージ内のファイルやディレクトリを、ホストへ再帰的にコピーする。最後の引数がホスト側の宛先
virt-diff -a old.img -A new.img	2つのイメージのファイルの差を出す。1つ目が -a / -d、2つ目が大文字の -A / -D
virt-inspector -a disk.img	OS の種類・版・マウント構成・入っているアプリを XML で出す
virt-filesystems -a disk.img -- long -h --all	ファイルシステム、パーティション、LV、物理ボリューム、ブロックデバイスを一覧する
virt-rescue --ro -a disk.img	壊れて起動しないゲストのために、救助用のシェルを開く。-i で /sysroot に自動マウント
virt-df -a disk.img -h	ゲストの中のファイルシステムの空き容量を出す。-a / -d を省くと、libvirt の全ドメインを調べる
virt-resize --expand /dev/sda2 old.img new.img	パーティションを広げた、新しいイメージを作る

コマンド	内容
<code>virt-sparsify in.img out.img</code>	空き領域を0にして、イメージを薄くする(スパースにする)
<code>virt-sysprep -a disk.img</code>	ひな形にする前に、ゲストの固有情報を消す

`virt-filesystems` と `virt-df` の出力例:

```
$ virt-filesystems -a gf.img --long -h --all
Name          Type          VFS    Label  MBR   Size  Parent
/dev/sda1     filesystem    ext4   -       -     55M   -
/dev/sda1     partition    -      -       83    64M   /dev/sda
/dev/sda      device       -      -       -     64M   -

$ virt-df -a gf.img -h
Filesystem                                Size      Used  Available
Use%
gf.img:/dev/sda1                          55M       16K     50M
1%
```

`virt-cat` や `-i` は、OS が見つかるイメージでしか動きません。OS の無いイメージに使うと、`no operating system was found on this disk` というエラーになります。その場合は、guestfish の `run`、`list-filesystems`、`mount` で手動でマウントします。`virt-rescue` のコンソールに出るのは、救助用アプライアンスのシェルで、ゲストの `/` ではありません。`virt-df` は、ゲストを起動する必要がありません。ディスクを読み取り専用で開くので、動いているゲストにも使えます[^c1-43]。

virt-resize は、その場での拡大ができず、入力とは別に出力のイメージを作ります。動作中のゲストには使いません。手順:

- 1. ゲストを停止する
- 2. `virt-filesystems` で、現在のパーティションを確認する
- 3. 出力用のイメージを大きい容量で作る (raw は `truncate -r old new` と `truncate -s +5G new`、qcow2 は `qemu-img create`)
- 4. `virt-resize` を実行する

`--expand /dev/sda2` で指定したパーティションに、余った領域を割り当てます。LVM なら、`--LV-expand /dev/vg/lv_root` で論理ボリュームも広げます[^c1-44]。 `--resize /dev/sda1=+200M` のように、特定のパーティションの大きさも指定できます。

virt-sparsify は、`virt-sparsify in.img out.img` で入力のコピーを薄くして出力します。出力の形式は入力に合わせ、`--convert qcow2` で形式も変えられます。 `--in-place` なら同じファイルをその場で薄くします。raw の入力からはスパースな raw ができるので、サイズは `ls` ではなく `du` で測ります[^c1-45]。

virt-sysprep は、ひな形を作るときに、クローンごとに変えたい情報を消します。ゲストを直接書き換えるので、中身を残したいときは、事前にコピーかスナップショットを取ります。既定で動く操作は、次のようなものです。

操作	消すもの
<code>ssh-hostkeys</code>	SSH のホスト鍵
<code>bash-history</code>	シェルの履歴
<code>logfiles</code>	ログ

操作	消すもの
<code>machine-id</code>	machine-id
<code>net-hwaddr</code>	MAC アドレスの記録
<code>dhcp-client-state</code>	DHCP クライアントの状態
<code>udev-persistent-net</code>	udev のネットワーク名の固定

`--list-operations` で一覧を見て、`--operations ssh-hostkeys,logfiles` で操作を選びます。Windows のゲストには使えません[^c1-46]。

ひな形の VM から10台を作って配りたい場合は、ひな形のゲストを停止し、`virt-sysprep` で固有情報を消します。さらに `virt-sparsify` で薄くしてから、`qemu-img convert -c -O qcow2` で圧縮すると、配る容量が減ります。複製先でSSHのホスト鍵や `machine-id` が重複するのは、`virt-sysprep` を省いた時の症状です。`virt-clone` は、ゲストの内側を変えません。

`/etc/fstab` の誤記でゲストが起動しない場合は、ゲストを停止し、`guestfish -d dom1 -i` で開いて `edit /etc/fstab` で直せます。シェルで調べながら直すなら `virt-rescue -d dom1 -i` を使います(ゲストのファイルは `/sysroot` の下に見える)。

物理ディスクから仮想ディスクへ・形式間の移行(Copy physical disk content、Migrate disk content between formats)

物理ディスクの中身を仮想ディスクにするには、`qemu-img convert` の入力にブロックデバイスを指定する方法もあります。ドライバや起動設定まで直し、物理マシン全体を KVM の VM にするなら、`virt-p2v` と `virt-v2v` を組み合わせます。

virt-v2v は、VMware、Hyper-V、Xen などの外部のハイパーバイザー上のゲストを KVM 用に変換します。入力と出力の指定:

指定	値の例
入力 <code>-i</code>	<code>disk</code> 、 <code>libvirt</code> 、 <code>libvirtxml</code> 、 <code>ova</code> 、 <code>vmx</code>
出力 <code>-o</code>	<code>local</code> 、 <code>libvirt</code> 、 <code>qemu</code> 、 <code>openstack</code> 、 <code>ovirt</code>

元のゲストは変更しません^[^c1-47]。

```
virt-v2v -i disk disk.img -o local -os /var/tmp      # ディスクイメージ
を交換してディレクトリへ
virt-v2v -ic vpx://vcenter.example.com/Datacenter/esxi vmware_guest -o
local -os /var/tmp
```

virt-p2v は、物理マシンを起動用メディア (CD・ISO・USB・PXE) で立ち上げ、中身を**変換サーバー**へ送る GUI です。変換サーバーは virt-v2v を入れた別のマシンで、物理マシンから SSH で接続します。SSH のリモートポート転送 (`ssh -R`) が必須なので、変換サーバーの `sshd_config` は `AllowTcpForwarding yes` にします。ファイアウォールで開けるのは、変換サーバーの SSH ポート (通常 22) だけです^[^c1-48]。

起動用のディスクイメージや USB キーは、**virt-p2v-make-disk** で作ります。

```
virt-p2v-make-disk -o /dev/sdX                      # USB キーに書く。既存の内容
は消える
virt-p2v-make-disk -o /var/tmp/p2v.img              # ディスクイメージとして作る
```

virt-p2v-make-disk は virt-builder を使う短いシェルスクリプトで、出力先の `-o` が必須です。ここで使う OS は、変換する物理マシンの OS とは無関係です[^c1-49]。

SAMPLE

サンプルはここまでです

続き(第2章～巻末)は、LPIC-3 305 問題集のご購入で、頻出度別ノート・暗記用ノートと合わせて3点すべてダウンロードできます。

SAMPLE

LPIC-3 305試験 学習用テキスト(無料サンプル)

版

2026年10月版(LPI公式 LPIC-3 Exam 305 Objectives Version 3.0 (305-300) 準拠 (LPI 公式ページ・2026-10-13 取得))

発行

IT資格道場(<https://www.shikaku-doujou.com>)

お問い合わせ

info@shikaku-doujou.com

LPI および LPIC は Linux Professional Institute Inc. の商標または登録商標です。Linux は Linus Torvalds 氏の米国およびその他の国における商標または登録商標です。記載されているその他の会社名・製品名・サービス名は、各社の商標または登録商標です。

本教材は公式の出題範囲 (Version 3.0・305-300) の内容を基に作成したもので、Linux Professional Institute との提携・承認関係はありません。

本文・図表・設問はすべてオリジナルの著作物です。無断での転載・複製・再配布・二次販売はできません。お気づきの点は info@shikaku-doujou.com までご連絡ください。