IT資格道場

LinuC レベル4 4SA02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-Japan公式 LinuC レベル4 システムアーキテクト 4SA02試験(LinuC-4SA Version 1.0) 出題範囲 準拠
2026年7月版

本書のご利用にあたって

- 本書は「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。
- 無料サンプルであっても著作物です。本書の転載・改変しての再配布・二次販売を禁じます。
- 本書を AI モデルの学習データとして利用すること(学習目的の入力・データセット化を含む)を禁じます。
- 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。

不正な配布を見つけた場合は info@shikaku-doujou.com までご連絡ください。

目次

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（本書）	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: 本書は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。

試験の基本情報

- 対象試験: LinuC レベル4 システムアーキテクト **4SA02試験**（出題範囲 LinuC-4SA Version 1.0）
- 認定元: LPI-Japan
- 出題数: 公式の試験概要によれば **1試験あたり約40問**
- 出題形式: **マウスによる選択方式がほとんど**。キーボード入力問題も多少出題される場合があります。**実技や面接はありません**。団体受験用にペーパーテスト（PBT）もあります

- 試験時間: **90分**。ただし試験後のアンケートに5分を要するため、公式は「試験問題を解く時間は実質 **85分**」としています
- 前提条件: 受験自体に実務経験や前提資格の条件はありません。ただし **LinuC レベル4 システムアーキテクトとして認定されるには、「有意なLinuCレベル2の認定」を持ち、かつ 4SA01・4SA02 の両試験に5年以内で合格する**必要があります（受験順は問いません）。レベル2認定を持たずに2試験へ合格した場合は、レベル2認定を取得した時点で認定されます
- 受験方法: ピアソンVUE経由のCBT（テストセンター受験、または自宅・職場からの OnVUE オンライン受験）
- 試験実施言語: 日本語・英語
- 出題比重: 公式は副主題ごとに **「重要度」** の数値を公表しています（パーセンテージの配点比率は公表されていません）。4SA02 の重要度は合計 40 で、これは出題数「約40問」とちょうど並びます。公式は「重要度が高いトピックスほど、試験において多くの問題が出題されます」と説明しています
- 合格ラインは公表されていません。本書は点数・正答率の数値を一切示しません
- 試験日程・受験料など、変わりうる数字は本書には書きません。公式サイトの最新の記載で必ず確認してください

LinuC レベル4 システムアーキテクトは **4SA01試験と4SA02試験の2本立て**です。出題範囲の主題番号でいうと、**4SA.01〜4SA.04 が 4SA01試験**、**4SA.05〜4SA.09 が 4SA02試験**にあたります。本書が扱うのは後者、すなわち次の5主題です。

主題	名称	重要度の合計	本書の章
4SA.05	仮想マシンとコンテナの設計	9	第1〜3章
4SA.06	セキュリティ	9	第4〜6章
4SA.07	監視と分析	8	第7〜9章
4SA.08	継続的開発とテスト・デプロイ	8	第10〜12章
4SA.09	トラブルシューティング	6	第13〜14章

本書の章立て

章は副主題（到達目標）そのままの単位で並べています。章の分量は、公式が公表している**重要度に比例**させてあります。

章	副主題	重要度	扱う中心
第1章	4SA.05.1 仮想マシンの設計と管理	3	オーバーコミット、NUMA、ピニング、Intel VT-x／AMD-V、リングプロテクション、virtio、QCOW2／RAW、virt-p2v／virt-v2v

章	副主題	重要度	扱う中心
第2章	4SA.05.2 コンテナの設計とビルド	3	cgroup／namespace、イメージレイヤ、scratch／distroless、マルチステージビルド、bind mount／tmpfs、docker compose
第3章	4SA.05.3 コンテナオーケストレーション	3	宣言的 API と突き合わせループ、CRI／CNI／CSI、Kubernetes の構成要素、Envoy／Istio
第4章	4SA.06.1 認証認可とアクセス制御	3	OAuth／OpenID Connect／SAML、TOTP、FIDO、Samba と Active Directory、DAC／MAC／RBAC／ABAC、SELinux／AppArmor／seccomp
第5章	4SA.06.2 セキュリティの予防措置	3	DoS／DDoS、インジェクション、GVM／ZAP／Clair／Katello／Vuls／OpenSCAP／Trivy、WAF／DMZ／UTM、LUKS／dm-crypt／TPM

章	副主題	重要度	扱う中心
第6章	4SA.06.3 セキュリティインシデントの検出	3	Linux 監査フレームワーク、AIDE／Tripwire、chkrootkit／rkhunter、OSSEC、Snort、tcpdump／Wireshark、SIEM
第7章	4SA.07.1 ログ・メトリクス・トレースの取得・収集	3	プッシュ型／プル型、IPMI／SNMP、Zabbix、Prometheus、Fluentd、OpenTelemetry／Jaeger
第8章	4SA.07.2 監視と対処	2	アラートの発生条件設計、通知とアクション、監視体制の改善
第9章	4SA.07.3 収集したデータの保全と分析	3	トレースビュー／サービスマップ、Grafana、長期トレンド分析、ログの保全と改ざん防止
第10章	4SA.08.1 テストの設計と効率化	3	単体〜全体のテスト観点、差分テスト・回帰テスト、Apache JMeter／Chaos Toolkit、InSpec／serverspec／Ansible、Selenium／Cypress

章	副主題	重要度	扱う中心
第11章	4SA.08.2 機能変更時の設計	2	後方互換の判定、API・スキーマ変更、バックポート管理
第12章	4SA.08.3 継続的な統合とデプロイ	3	カナリア／シャドー、インプレース／ローリング／ブルーグリーン、OpenTofu／Ansible／cloud-init、Jenkins／GitLab CI/CD
第13章	4SA.09.1 障害時の基本手順	2	検知と評価、根本原因の特定、暫定対応から恒久対応まで
第14章	4SA.09.2 ケーススタディ	4	起動・データ・ネットワーク・冗長構成・各種サービスの障害を「症状→切り分け→原因→対処」で

各章は次の3点セットで構成しています。

1. **本文** …… 「そう決まっている」ではなく「なぜその設計になるのか」まで書いています。4SA02 は暗記型の設問が少なく、**要件を読んで手段を選ばせる**形が中心なので、仕組みごと理解しておくほうが取り違えが起きにくくなります
2. **混同しやすい対の概念** …… この試験でいちばん多いひっかけは「似た役割の2つを入れ替える」型です。入れ替えられやすい組を表にしてあります
3. **章の試験ポイント** …… 実際に壊されやすい箇所と、その見分け方を並べています

4SA02 の設問はどう問われるか

4SA02 は **構築・運用・改善の試験**です。到達目標の文末を見ると「〜を理解している」だけでなく、「**設計できる**」「**選定する**」「**比較し、適切な方式を選択する**」「**計画を立案する**」が並んでいます。つまり、知っているかではなく **「どれを選ぶか・なぜそちらか」** が問われます。

特徴	内容
出題形式	マウスによる選択方式が中心。キーボード入力（コマンド名・ツール名などの綴り）も一部出ることがあります
設問の立て方	状況（構成・制約・許容できる停止時間・すでに起きている症状）を先に示し、**「最も適切なものはどれか」**を選ばせる形が大半
誤り肢の作り	「その技術では実現できない」ではなく、**「別の技術ならできるが、これではできない」**という近接技術の性質を持ち込む形が多い
ケーススタディ	主題 4SA.09 は障害の**類型**が細かく列挙されています。症状の文面から類型を当てられるかが勝負になります

壊され方（ひっかけ）は7つに整理できる

本文中の「試験ポイント」では、この番号で型を示します。

型	仕掛け
① 近接概念ペアの入れ替え	対になる2つの中身を丸ごと入れ替える（プッシュ型／プル型、bind mount／tmpfs、AIDE／chkrootkit、カナリア／シャドー、virt-p2v／virt-v2v など。**最頻**）

型	仕掛け
② 層の取り違え	別の層（CPU／ストレージ／ネットワーク／アプリケーション）の仕組みを持ち込んで「これで解決する」と書く
③「量」と「配置」のすり替え	資源の**量**を変える手段と、同じ量を**どこに置くか**を変える手段を入れ替える（オーバーコミット ⇔ NUMA・ピニング）
④ 前提条件の付け替え	成立条件を別の機能の条件に差し替える（ブルーグリーンの前提にスキーマ後方互換不要を持ち込む、FIDO の前提に共有シークレットを持ち込む など）
⑤ 手順の順序反転	正しい要素を並べたうえで**順番だけ逆**にする（データベース更新とアプリケーション更新の順序、切り戻し手順、切り分けの順序）
⑥ 効かない対処	症状は消えるが原因は残る対処を「解決」として書く（閾値を上げる、再起動する、タイムアウトを延ばす、アラートを止める）
⑦ 存在しない機能・オプションの捏造	もっともらしいが実在しない指定を書く（存在しないサブコマンド、実在しない設定項目、ツールにない機能）

⑦は特に注意してください。この試験の誤り肢は「日本語として自然で、やりたいことにも合っている」ように書かれます。**実在するかどうか**を思い出せるかが分かれ目になります。

読み進める前に

本書は **LinuC レベル2 相当の Linux 運用の実務知識**を持っている読者を想定しています。パッケージ管理・systemd・ネットワーク設定・ログの見方・シェ

ルの基本操作は前提として、そのうえに「複数台・複数層のシステムをどう設計し、どう壊れないようにし、壊れたらどう直すか」を積み上げます。

コマンドや設定例は、出題範囲が明示しているものを中心に、本文の理解を助ける範囲で示します。4SA02 は実技試験ではないので、**打てること**より**なぜそれを打つのか**を優先して読んでください。

第1章 仮想マシンの設計と管理（4SA.05.1）[5.1]

到達目標: 仮想化の基本技術を理解し、性能検討やトラブルシューティングができる。仮想マシン及び仮想ディスクの移行など、仮想環境の管理ができる。

この章は「仮想マシンが遅いとき、どこを疑い、何を動かすか」を扱います。4SA02 のこの範囲は **原理を知っているか**ではなく、**原理から性能問題の原因を言い当てられるか**を問います。

この章がカバーする出題範囲の観点 [5.1]

- 仮想マシンの動作や性能に関わる要素（オーバーコミット、NUMA）
- リソース割り当ての手法のメリット・デメリット・制約を理解し、**有効利用**する（ピニング、スケジューリング、レイテンシ設定、優先順位設定）
- 仮想化支援技術と仮想デバイス —— Intel VT-x／AMD-V、x86 の OS レベルのプロセス**保護機能**（リングプロテクション）、**virtio の動作方法及び利点欠点**
- 仮想ディスクの主要な形式（QCOW2、RAW）の特徴
- 既存資産の移行 —— **物理環境**から仮想環境へ（virt-p2v）、**仮想環境間**の移行（virt-v2v）、仮想ディスクのサイズ変更

1-1 仮想マシンの性能を決める4つの要素 [5.1]

物理サーバーと違い、仮想マシンには「ホストとの境界を越える」という固有のコストがあります。性能を検討するときは、次の4か所のどこで詰まっているかを先に切り分けます。

要素	何が起きるか	詰まったときの症状
CPU	ゲストの特権命令やデバイス操作でホスト側へ制御が移る（VM Exit）。回数が多いほど遅い	ゲスト内は暇に見えるのにスループットが出ない。ホストの CPU 使用率だけが高い
メモリ	ゲストの仮想アドレスをホストの物理アドレスへ二段階で変換する。CPU 側のネストページテーブル機能（Intel の EPT、AMD の NPT）が肩代わりする	メモリ確保の多い処理でだけ遅い。ホストがスワップし始めると全 VM が同時に劣化する
ストレージ I/O	完全エミュレーションなら1回の I/O ごとにホストへ抜ける。virtio なら束ねて渡せる	iowait が伸びる。ゲストのディスク帯域がホストの実力に遠く届かない
ネットワーク	同上。加えてホスト側のブリッジ・仮想スイッチの処理が乗る	パケット毎秒（pps）が上がらない。帯域は出るのに小さいパケットで詰まる

押さえどころ: 仮想マシンの性能問題の多くは「ホストへ抜ける回数」に還元されます。**回数を減らす手段（virtio、パススルー）**と、**抜けた先の待ち時間を減らす手段（ピニング、NUMA 整合）**は別物です。設問はこの2つをよく入れ替えます。

1-2 オーバーコミット ―― 「足し算が合わない」を意図的に作る [5.1]

オーバーコミットとは、ホストが持つ物理リソースの総量を超えて、仮想マシンにリソースを割り当てることです。全 VM が同時に全力で使うことはない、という前提に賭けて集約率を上げます。

	CPU オーバーコミット	メモリ オーバーコミット
何を超えるか	全 VM の vCPU 数の合計 > ホストの物理 CPU（論理コア）数	全 VM に割り当てたメモリ量の合計 > ホストの物理メモリ量
成立する理由	vCPU は「実行可能なときだけ」物理 CPU を使う。アイドルの vCPU は場所を取らない	割り当てただけでは物理ページは確保されない（実際に触った分だけ確保される）
破綻したときの症状	**CPU steal time** が伸びる。ゲスト内から見ると「実行したいのに順番が回ってこない」	ホストがスワップを始める。**同居している無関係な VM まで一斉に遅くなる**
支える仕組み	ホストのスケジューラ	メモリバルーニング（virtio-balloon）、KSM による同一ページの共有、ホストのスワップ

メモリのオーバーコミットのほうが危険です。 CPU が足りないときは各 VM が「遅くなる」だけで済みますが、メモリが足りないときはホストがスワップに落ち、ディスク I/O を共有している全 VM が巻き添えになります。しかも劣化は非線形で、ある閾値を越えた瞬間に一気に悪化します。

設計上の判断軸: レイテンシ要件のある VM（データベース、リアルタイム処理）が同居するホストではオーバーコミットしない、あるいはその VM のメモリだけ確保を固定する（ロックする）。バッチ処理や開発環境のように「遅くても終わればよい」VM だけを集約対象にします。

1-3 NUMA ——「量」ではなく「配置」の問題 [5.1]

NUMA（Non-Uniform Memory Access）は、CPU ソケットごとにメモリコントローラとメモリが紐づいている構成です。自分のソケットに直結したメモリ（ローカル）へのアクセスは速く、他のソケットに繋がったメモリ（リモート）へのアクセスは相互接続を経由するぶん遅くなります。

仮想マシンで問題になるのは、**vCPU が動いているノードと、その VM のメモリが置かれたノードがずれる**ケースです。メモリの「量」は足りているのに、アクセスのたびにノードをまたいで遅くなります。

状況	起きること	対処
VM のメモリが複数ノードに分散した	アクセスの一部が常にリモートになる	メモリの確保先ノードを固定する（libvirt の `<numatune>`、`numactl --membind`）
VM が1ノードの物理メモリ量を超える大きさ	どうやってもノードをまたぐ	ゲストにも NUMA トポロジを見せる（**vNUMA**）。ゲスト OS 自身が「ローカル／リモート」を意識して配置できるようになる
ホストのスケジューラが vCPU を別ノードへ移した	移動した瞬間から全アクセスがリモートになる	vCPU をピニングしてノードから出さない

確認には `numactl --hardware`（ノード構成と距離行列）と `numastat`（ノードごとのローカル／リモートヒット数）を使います。カーネル側には自動再配置の機能（`kernel.numa_balancing`）もありますが、これは**移動させて直す**仕組みなので、移動中は一時的に遅くなります。レイテンシを厳しく求める VM では自動任せにせず、明示的に固定するのが定石です。

ひっかけ注意:「メモリが足りないので NUMA を調整する」は誤りです。NUMA は**量の問題ではなく配置の問題**を扱います。量を変えるのはオーバーコミット・バルーニング・ホットプラグの側です。

1-4 ピニング(pinning)とスケジューリング [5.1]

ピニング(CPU ピン留め)は、VM の vCPU を特定の物理 CPU(論理コア)に固定することです。libvirt では `<cputune>` の `<vcpupin>`、実行中の変更は `virsh vcpupin` で行います。

	メリット	デメリット	制約
vCPU ピニング	CPU キャッシュの局所性が保たれる／NUMA ノードをまたがない／レイテンシのばらつきが小さくなる	ホストのスケジューラが空いているコアへ逃がせなくなる。**ピン先が混むと固定していないときより遅くなる**	他のプロセスを同じコアから排除しないと効果が薄い(`isolcpus`、cpuset による隔離を併用する)
エミュレータスレッドのピニング(`<emulatorpin>`)	vCPU の邪魔をしないコアへ I/O 処理を追い出せる	追い出した先が細いと I/O が頭打ちになる	vCPU 用と別のコアを用意できることが前提
ピニングしない	空いているコアが自動で使われ、平均スループットは出やすい	レイテンシがばらつく／ノードをまたぐことがある	—

「ピニングすれば速くなる」は誤りです。ピニングが効くのは、**レイテンシのばらつきを潰したい**場合と、**NUMA 整合を保証したい**場合です。平均スループットだけを求めるなら、スケジューラに任せたほうが良い結果になることが珍しくありません。

レイテンシ設定と優先順位設定 [5.1]

vCPU の実行時間の配分は、cgroup のコントローラを通じて調整できます。libvirt の `<cputune>` に対応する項目が並んでいます。

設定	意味	使いどころ
相対的な重み（`shares`）	競合したときの取り分の比。競合していなければ制限にならない	「混んだときに優先させたい VM」がある場合
上限（`period` と `quota`）	一定時間あたりに使える CPU 時間の**絶対的な上限**。競合がなくても超えられない	暴走した VM がホストを食い潰すのを防ぐ／課金や SLA に合わせて頭を押さえる
リアルタイムスケジューリング（`SCHED_FIFO` ／ `SCHED_RR`）	通常のタスクより常に先に走らせる	レイテンシ要件が絶対の VM。**設定を誤るとホストごと固まる**ため慎重に

重み（shares）と上限（quota）の入れ替えは定番のひっかけです。「他の VM が動いていなくても 2 コア分を超えさせたくない」なら上限、「混雑時に優先したい」なら重みです。重みでは上限にならず、上限では優先度になりません。

1-5 仮想化支援技術とリングプロテクション [5.1]

x86 のリングプロテクション [5.1]

x86 アーキテクチャには **リング（特権レベル）** という保護機構があり、Ring 0（最高特権）から Ring 3（最低特権）まで4段階が定義されています。実際の Linux が使うのは2つだけです。

リング	通常の Linux での用途
Ring 0	カーネル。特権命令の実行、ハードウェアへの直接アクセスができる
Ring 1・2	通常は未使用
Ring 3	ユーザー空間のプロセス。特権命令を実行しようとすると例外が発生する

仮想化で困るのはここです。**ゲスト OS のカーネルも「自分は Ring 0 で動いている」つもりで特権命令を出します。**しかし本物の Ring 0 はホストのカーネルが使っているので、ゲストをそのまま Ring 0 に置くことはできません。

Intel VT-x / AMD-V が解いたこと [5.1]

Intel VT-x と **AMD-V** は、リングの内側にもう一段の区別、すなわち **ホスト用のモードとゲスト用のモード**を CPU に追加しました。ゲストは**ゲスト用モードの Ring 0** で動けるため、ゲストカーネルを改変せずそのまま動かせます。特権命令のうちホストが介入すべきものだけが、CPU の働きで自動的にホスト側へ制御を移します（この移り変わりが **VM Exit** と **VM Entry**）。

Linux 側でこれを利用する機能が **KVM** です。CPU ベンダーに応じたカーネルモジュール（Intel なら `kvm_intel`、AMD なら `kvm_amd`）が読み込まれ、`/dev/kvm` という装置ファイルを通じてユーザー空間（QEMU など）から仮想 CPU を作れるようになります。CPU 側の機能が無効だと（BIOS／UEFI の設定で切られているなど）、このモジュールは読み込めません。**「仮想マシンが作れない・極端に遅い」ときの最初の確認先**がここです。

なお、VT-x／AMD-V が無い時代には、ゲストカーネルを Ring 1 へ落とす方式や、特権命令を実行時に書き換える方式、ゲストカーネル自身を改変してホストへ直接依頼させる方式（準仮想化）が使われていました。**現在の設問で「ゲ**

ストカーネルの改変が必要」と書かれていたら、それは VT-x／AMD-V を使う構成の説明としては誤りです。

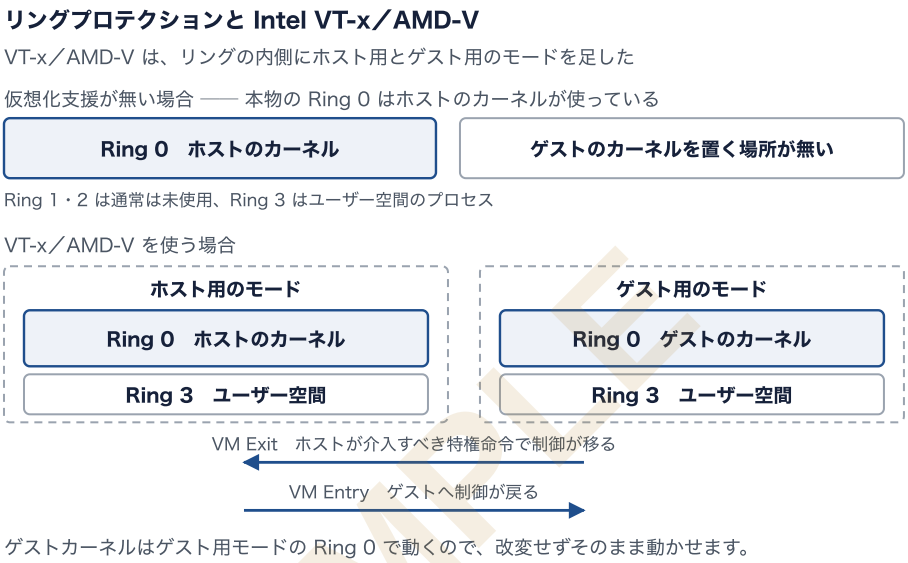

図 1-1　リングプロテクションと Intel VT-x／AMD-V

virtio —— 準仮想化デバイス [5.1]

virtio は、デバイスを「本物そっくりにエミュレートする」のをやめて、**仮想環境であることを前提にした専用の受け渡し方式**に置き換える仕組みです。

- ゲスト側に **フロントエンドドライバ**（`virtio_blk`、`virtio_net` など）、ホスト側に **バックエンド**（QEMU や vhost）が置かれる
- 両者は **virtqueue** と呼ばれるリング構造を共有メモリ上に持ち、要求と完了をここに積む
- **1件ごとにホストへ抜けるのではなく、積んでからまとめて通知する。**これが速さの正体

	完全エミュレーション	virtio	パススルー（デバイス直結）
ゲスト側の要件	標準的なドライバで動く（**改変不要**）	**virtio 対応ドライバが必要**	実デバイス用のドライバが必要
速度	遅い	速い	最速
移行（ライブマイグレーション）	可能	可能	**困難**（物理デバイスに縛られる）
主な欠点	ホストへ抜ける回数が多い	**対応していないゲスト OS では使えない**／ドライバの導入が前提	ホストと共有できない、移行できない

virtio の欠点は「速度」ではなく「前提」です。古い OS や、インストーラの段階で virtio ドライバを持たない環境では、ディスクが見えず起動できません。この場合はインストール時だけ完全エミュレーションのデバイスを使い、ドライバ導入後に virtio へ切り替える、といった手順を取ります。

完全エミュレーション・virtio・パススルー

違いは「ホストへ抜ける回数」と「ゲスト側に何を求めるか」

virtio の欠点は速度ではなく前提です。ゲスト側に対応ドライバが必要になります。
パススルーは最速ですが、物理デバイスに縛られるためライブマイグレーションが困難です。

図 1-2　完全エミュレーション・virtio・パススルー

1-6 仮想ディスクの形式 —— QCOW2 と RAW [5.1]

	QCOW2	RAW
構造	参照テーブル（L1／L2）を経由してデータブロックを指す	ディスクの内容がそのまま並んでいるだけ
実容量	**実際に書いた分だけ増える**（薄いプロビジョニング）	原則として最初から確保。ファイルシステムがスパースファイルに対応していれば薄くも持てる
スナップショット	**形式そのものが対応**（内部スナップショット）	形式としては非対応。外部の仕組み（LVM、ストレージ側）に頼る
差分イメージ（バッキングファイル）	対応。親イメージを共有して差分だけ持てる	非対応
圧縮・暗号化	対応	非対応
性能	変換の一段があるぶんオーバーヘッドがある	**最速**。オーバーヘッドが無い

選び方の軸は「機能を取るか、速度を取るか」です。テンプレートから多数の VM を薄く作りたい、スナップショットで戻したい、という運用なら QCOW2。データベースのように I/O 性能が直接効く用途なら RAW、というのが基本の判断です。

1-7 既存資産の移行 —— virt-p2v と virt-v2v [5.1]

名前が似ていて、**入れ替え型のひっかけが最も多い**箇所です。中央の文字で覚えます。

ツール	変換の向き	使い方の概要
virt-p2v	**P**hysical → **V**irtual（物理マシンを仮想マシンへ）	移行対象の**物理マシンを専用の起動媒体でブート**し、そのディスク内容を変換先のホストへ転送する
virt-v2v	**V**irtual → **V**irtual（他の仮想化基盤の VM を KVM／libvirt 形式へ）	変換元の仮想マシンのディスクイメージや管理基盤を指定して変換する

どちらも、単にディスクをコピーするのではなく、**移行先で起動できるようにゲストの中身に手を入れます。**具体的には、ゲストのデバイスドライバを移行先に合った構成（virtio）へ入れ替え、初期 RAM ディスクを作り直し、ブートローダの設定を整えます。**「ブロックコピーするだけのツール」と書かれていたら誤り**です。

移行時に見落とされやすいのは次の点です。

- 移行元のネットワークインターフェース名が変わり、ネットワーク設定が当たらなくなる
- MAC アドレスが変わることでライセンス認証や DHCP 予約が外れる
- 移行元に固有のハードウェア監視エージェントや管理ツールが残り、起動時にエラーを出し続ける

仮想ディスクのサイズ変更 [5.1]

拡大は「器を広げる → 中身を広げる」の順で、**必ず外側から内側へ**進めます。

1. **イメージ自体を拡大**する（`qemu-img resize <イメージ> +50G`）
2. ゲスト内で**パーティションを拡大**する
3. LVM を使っているなら**物理ボリューム、論理ボリュームの順に拡大**する

4. 最後に**ファイルシステムを拡大**する（オンラインで拡大できるものが多い）

縮小は**この逆順**で、ファイルシステム → 論理ボリューム → パーティション → イメージ、と内側から詰めます。**順番を間違えると、まだ使われている領域を切り落としてデータを失います。** `qemu-img resize` で縮小するには、事故防止のため縮小を意図していることを明示するオプションが必要です。

いずれの場合も、**作業前にバックアップかスナップショットを取る**のが前提です。設問で「無停止で安全に縮小できる」と書かれていたら疑ってください。

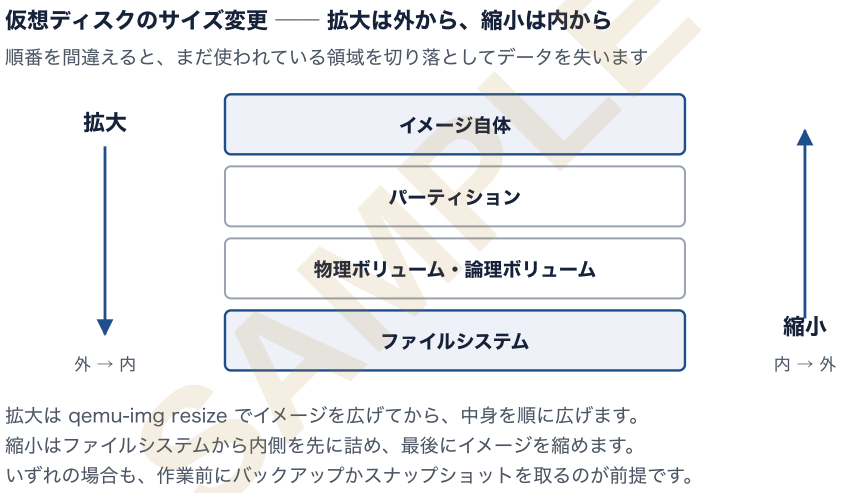

図 1-3　仮想ディスクのサイズ変更 —— 拡大は外から、縮小は内から

1-8 性能問題の切り分け —— 症状から原因へ [5.1]

ここまでの要素を、実際の設問の形（症状が先に示される形）から逆に引けるように並べ直します。

症状	まず疑う原因	確かめ方	対処
ゲスト内の CPU 使用率は低いのに処理が進まない	**steal time**（順番が回ってきていない）	ゲストで CPU の内訳を見て steal の割合を確認する	CPU オーバーコミット率を下げる／同居 VM を移す／上限（quota）が掛かっていないか確認する
特定の VM だけメモリアクセスが遅い	**NUMA ノードまたぎ**	`numastat` でリモートアクセスの割合を見る	メモリの確保先ノードを固定し、vCPU を同じノードへピニングする
ホスト全体が同時に遅くなり、ディスクの読み書きが常時発生する	**メモリオーバーコミットの破綻**（ホストのスワップ）	ホストのスワップ使用量とページイン／ページアウトを見る	割り当てを見直す／バルーニングで回収する／VM を他ホストへ退避する
ディスクのスループットがホストの実力に遠く届かない	デバイスが**完全エミュレーション**のまま	ゲスト内でディスクのドライバ名・デバイス名を確認する	virtio へ切り替える（起動できるようドライバの導入を先に済ませる）
平均は速いが応答時間のばらつきが大きい	vCPU が**コア間を移動している**	ホストのスケジューリング統計、キャッシュミス率	vCPU をピニングし、`isolcpus` などで他プロセスを追い出す
仮想マシンをまったく起動できない、または極端に遅い	**仮想化支援機能が無効**	ホストで `kvm_intel` ／ `kvm_amd` が読み込まれているか、`/dev/kvm` があるかを見る	BIOS／UEFI で Intel VT-x／AMD-V を有効にする

症状	まず疑う原因	確かめ方	対処
移行した VM が起動途中で止まる	ドライバ・ブート設定の不整合	コンソールで停止位置を見る（ルートデバイスが見つからない、など）	virt-v2v／virt-p2v による変換をやり直す／初期 RAM ディスクを作り直す

切り分けの順序は「ホストが先、ゲストが後」です。 ホスト側が飽和していれば、ゲスト内をいくらチューニングしても改善しません。設問でも「ゲスト内のパラメータを調整する」という選択肢は、ホスト側の飽和が示唆されているときには誤りになります。

また、**症状は消えるが原因が残る対処**が誤り肢として頻出します。監視の閾値を上げる、タイムアウトを延ばす、VM を再起動する、といった選択肢は、原因が特定できていない段階では「暫定対応」であって「解決」ではありません。この区別は第13章（障害時の基本手順）にも直結します。

第1章の混同しやすい対の概念 [5.1]

	オーバーコミット	NUMA 調整・ピニング
扱うもの	リソースの**量**	リソースの**配置**
目的	集約率を上げる	レイテンシを安定させる
破綻の形	スワップ、steal time	ノードまたぎ、キャッシュミス

	CPU オーバーコミット	メモリ オーバーコミット
破綻時の影響範囲	その VM が遅くなる	**ホスト上の全 VM が遅くなる**
劣化の仕方	ほぼ線形	**閾値を越えると急激**

	重み（shares）	上限（quota／period）
効くとき	**競合しているときだけ**	**常に**
できること	優先順位づけ	絶対的な頭打ち

	virtio	完全エミュレーション	パススルー
ゲストのドライバ	専用ドライバが要る	標準ドライバで動く	実機用ドライバが要る
ライブマイグレーション	可能	可能	**困難**

	QCOW2	RAW
スナップショット	形式が対応	**形式としては非対応**
速度	一段のオーバーヘッド	**最速**

	virt-p2v	virt-v2v
変換元	**物理マシン**	**他の仮想化基盤の仮想マシン**
起動媒体でブートするか	する	しない

第1章の試験ポイント [5.1]

- **p2v と v2v の入れ替え（①型）**：「VMware 上の VM を KVM へ移すので virt-p2v を使う」は誤り。**physical から始まるのが p2v**。物理サーバーの移行だけが p2v です

- **移行ツールを単なるコピーと書く（⑦型）**：「ディスクイメージをそのまま複製するだけなので、移行後の起動設定は変わらない」は誤り。**ドライバの入れ替えと初期 RAM ディスクの再作成を伴います**
- **NUMA を量の問題に使う（③型）**：「メモリ不足なので NUMA ノードを増やす」「NUMA を無効化すると使えるメモリが増える」はいずれも誤り。**配置の話**です
- **ピニングを万能薬にする（⑥型）**：「遅いので vCPU をピン留めした」は、隔離を併用していなければ改善しないどころか悪化します。**ピン先の混雑を確認**してください
- **重みと上限のすり替え（①型）**：「他の VM がいなくても CPU を使いすぎないようにしたい」に shares を当てるのは誤り。**上限は quota／period**です
- **リングプロテクションの捏造（⑦型）**：「Ring 1 と Ring 2 に仮想マシン専用の特権が定義されている」「VT-x はゲストを Ring 3 で動かす」は誤り。**VT-x／AMD-V はリングの内側にホスト用／ゲスト用のモードを足した**もので、ゲストカーネルはゲスト用モードの Ring 0 で動きます
- **VT-x にゲスト改変を持ち込む（④型）**：「ハードウェア支援を使うにはゲストカーネルの改変が必要」は誤り。改変が要るのは準仮想化の側です
- **virtio の欠点を速度と書く（①型）**：virtio の欠点は**ゲスト側にドライバが要る**こと。「エミュレーションより遅い」は誤りです
- **パススルーとライブマイグレーション（④型）**：「性能を出すためにパススルーし、あわせて無停止移行も行う」は両立しません。**物理デバイスに縛られる**ためです
- **QCOW2 と RAW の入れ替え（①型）**：「RAW はスナップショットを内部に持てる」「QCOW2 のほうが I/O が速い」はいずれも誤りです

- **サイズ変更の順序反転（⑤型）**：拡大は**外→内**、縮小は**内→外**。「イメージを縮めてからファイルシステムを縮める」は**データ消失の手順**です
- **VM Exit の見落とし（②型）**：ゲスト内の CPU 使用率が低いのにホストの CPU 使用率が高い、という症状にゲスト側のチューニングを当てるのは誤り。**ホストへ抜ける回数**を疑い、デバイスを virtio へ寄せます
- **綴りを問われうる語**：`virt-p2v`、`virt-v2v`、`qemu-img resize`、`numactl`、`numastat`、`kvm_intel` ／ `kvm_amd`、`/dev/kvm`

サンプルはここまでです

続き(第2章〜巻末)は、LinuC 4SA02 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。

LinuC レベル4 4SA02試験 学習用テキスト(無料サンプル)

版

2026年7月版(LPI-Japan公式 LinuC レベル4 システムアーキテクト 4SA02試験(LinuC-4SA Version 1.0) 出題範囲 準拠)

発行

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

お問い合わせ

info@shikaku-doujou.com