IT資格道場

OPCEL認定試験 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 OPCEL認定試験(出題範囲 Ver.3.0・Queens〜Ussuri) 出題範囲 準拠
2026年11月版

SAMPLE

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。

SAMPLE

目次

この教材群の使い方（3点セット）

第1章 OpenStack のアーキテクチャとデプロイメント [250.1/250.2/250.3]

- クラウドコンピューティングの概念 [250.1]
- OpenStack のアーキテクチャと設計 [250.2]
- OpenStack のインストレーションとデプロイメント [250.3]

サンプルはここまでです

SAMPLE

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。

- 対象試験: OPCEL認定試験（OpenStack Professional Certification Exam by LPI-Japan）出題範囲 Ver.3.0（対応バージョンは Queens〜Ussuri）
- 試験形式: 約60問・90分（受験時間85分、アンケート5分）。択一問題と複数選択問題が中心で、キーボード入力問題も少し出ます
- 構成: 全3章。第1章=アーキテクチャとデプロイメント（250）／第2章=管理・共有サービス（251。Keystone・Horizon と API・Heat・Glance）／第3章=インフラサービス（254。Neutron・Nova・Cinder・Swift）
- 各章の節の並び: 出題範囲の細目の順。見出しの角かっこの数字は、その節が対応する細目の番号

SAMPLE

- コマンドの表記: `openstack` コマンドに統一。nova・neutron・cinder など個別クライアントのコマンドは範囲の外

SAMPLE

第1章 OpenStack のアーキテクチャとデプロイメント [250.1/250.2/250.3]

OpenStack は、サーバー、ストレージ、ネットワークをまとめて払い出すクラウド基盤です。クラウドの考え方、部品の役割、入れ方の順に進みます。

クラウドコンピューティングの概念 [250.1]

クラウドの基本特性

クラウドは、必要なときに必要な量だけ計算資源を借りる仕組みです。NIST（米国標準技術研究所）が挙げる基本特性は、次の5つ。[^c1-1]

特性	意味
オンデマンド・セルフサービス	利用者が事業者の手を借りず、自分で資源を確保できる
広帯域ネットワークアクセス	標準的な方法でネットワーク越しに使える
リソースプーリング	事業者の資源が複数の利用者で共有される
迅速な伸縮性	需要に合わせて資源を増減できる
従量制サービス	使った量が自動で測られ、監視・制御・報告と課金に使われる

OpenStack に当てはめると、Horizon や openstack コマンドで利用者が自分でインスタンスを作る操作がセルフサービスです。プロジェクトごとにクオータ（資源の上限）を決めて同じ基盤を分け合う仕組みが、リソースプーリングにあたります。

SAMPLE

提供サービスの分類：IaaS・PaaS・SaaS

クラウドが何を提供するかで、3つに分けます。違いは、利用者がどこまで自分で管理するかであり、機能の多さではありません。

SaaS は、事業者が動かすアプリケーションを利用者がそのまま使う形です。利用者は基盤を管理しません。触れられるのは、ユーザーごとの限られたアプリケーションの設定だけ。PaaS は、利用者が作った（または入手した）アプリケーションを、事業者が用意した言語やツールの上に載せる形です。利用者が制御するのは、配備したアプリケーションと、それを動かす実行環境の設定値まで。OS、ミドルウェア、サーバー、ネットワーク、ストレージは制御しません。IaaS は、処理能力、ストレージ、ネットワークなどの基礎資源を利用者が確保し、OS やアプリケーションを自由に動かせる形です。[^c1-1]

区分	利用者が管理する範囲	事業者が管理する範囲
IaaS	OS、ミドルウェア、アプリケーション、データ	物理サーバー、ストレージ、ネットワーク、仮想化層
PaaS	アプリケーション、データ、実行環境の設定値	IaaS の範囲に加えて OS、ミドルウェア、実行環境そのもの
SaaS	利用するデータや設定	アプリケーションを含むすべて

OpenStack が提供するのは IaaS です。仮想マシン（インスタンス）、ブロックストレージ、仮想ネットワークを API で払い出します。OS の更新やアプリケーションの運用は、IaaS では利用者の仕事です。「OpenStack でアプリケーションの実行環境そのものが手に入る」という理解は、PaaS との取り違え。Magnum のようにコンテナの基盤を払い出すサービスはありますが、中心は IaaS です。

SAMPLE

NIST の IaaS の定義は、利用者の制御範囲をもう少し細かく書いています。利用者は、基盤（物理サーバー、ネットワーク機器、仮想化層）を管理も制御もしません。制御できるのは、OS、ストレージ、配備したアプリケーション。ネットワークは基本的に事業者の範囲で、利用者が制御できるのは、ホストファイアウォールなど一部の構成要素に限られます。[^f6g1-1]

上の表で IaaS のネットワークを事業者の側に置くのは、データセンターのネットワーク機器全般が事業者の管理だからです。OpenStack では、セキュリティグループの規則が、利用者に開かれた限定的な制御の例にあたります。一方、物理サーバーの電源や冷却、仮想化層の設定は、利用者の制御に入りません。

配置による分類：パブリック・プライベート・ハイブリッド

誰がクラウドを使えるか、どこに置くかで3つに分けます。

- パブリッククラウド：一般の人々が使えるように開かれた基盤。資源を不特定多数の利用者で共有する
- プライベートクラウド：1つの組織だけが専用に使う基盤。所有と運用は、その組織、第三者、両者の組み合わせのどれでもよい
- ハイブリッドクラウド：2つ以上の異なる形態（プライベート、コミュニティ、パブリック）を組み合わせた構成。各基盤は独立したまま、標準または独自の技術で結ばれ、データとアプリケーションを移せる（可搬性）

NIST にはもう1つ、共通の関心を持つ特定の利用者グループだけが使うコミュニティクラウドがあります。[^c1-1] OpenStack は、自社のデータセンターに構築するプライベートクラウドの基盤にも、パブリッククラウドを運営する事業者の基盤にもなります。ソフトウェアの種類と配置の分類は、別の軸で考えるもの。

SAMPLE

ハイブリッドクラウドは、自社の OpenStack で普段の負荷を受け、ピーク時だけパブリッククラウドに溢れさせる、といった使い方が典型です。

置き場に制約があるデータを扱う処理（個人情報や、決済のように安全基準がある処理）は自社側に残し、制約の無い処理や変動の大きい処理だけを外へ逃がす分け方もあります。

基盤の置き場所で見ると、パブリッククラウドはクラウド事業者の敷地、プライベートクラウドは利用者の組織の敷地が典型。所有、管理、運営をするのは、企業、学術機関、政府機関のいずれか、またはその組み合わせです。[^f6g1-1] ただしプライベートクラウドは、敷地の内外には左右されず、1つの組織の専用であることで決まります。コミュニティクラウドは、敷地の内にも外にも置けます。

クラウドが向くケースと向かないケース

クラウドが適切で有利になるのは、負荷が大きく変動するシステムです。ピークに合わせてサーバーを買っても、普段はその台数を使い切れません。クラウドなら、ピークの時間だけ台数を増やして終われば返せるので、初期費用を抑えたい場合や、新しいサービスを素早く立ち上げて様子を見たい場合にも向いています。

向かないのは、負荷がほぼ一定で長期間動かし続けるシステムです。使い続けるなら、従量課金の合計が購入費用を上回ることがあります。データの置き場に法令や社内規程の制約があり、特定の場所から出せない場合もそうです。専用の特殊なハードウェアに強く依存するシステムも、仮想化された基盤とは相性がよくありません。

判断の型は、負荷の変動、初期費用、データの置き場の3点を見ることです。選択肢に「負荷が一定で5年間動かし続ける」と「月末だけ処理が10倍になる」

SAMPLE

が並んだら、後者がクラウド向きです。

データの置き場や規制に制約がある組織は、パブリッククラウドを使う前に、事業者が必要な認証や規制への対応を備えているかを確かめます。基盤が事業者の敷地にあり、データの扱いが事業者の運用に左右される点が理由。確かめる対象は、事業者が持つ認証（ISO 27001 など）や監査の報告と、その適用範囲。[^f6g1-2]

建物の外観、画面の見やすさ、起動の速さ、単価の安さ、請求書の形式は、規制への対応を示す材料になりません。インスタンスの増設や契約は、この確認が済んだあとの段階です。

IaaS とアプリケーションの組み合わせ方

IaaS の上のアプリケーションを、1台のインスタンスに頼る作りにすると、そのインスタンスが止まった時点でサービスも止まります。伸縮性を生かすには、アプリケーションの側も作りを合わせなければなりません。

スケールアップは、1台のインスタンスの CPU やメモリを増やして性能を上げる方法です。スケールアウトは、インスタンスの台数を増やして負荷を分散する方法です。クラウドで伸縮性を使うのは、主にスケールアウトです。台数を足し引きしても動くアプリケーションにするには、次の3つが要ります。

- 状態の外出し：セッション情報やファイルなど、消えては困る状態を、インスタンスの外（データベース、ブロックストレージ、オブジェクトストレージ）に置く
- 負荷分散：複数のインスタンスの前にロードバランサーを置き、リクエストを振り分ける
- 冗長化：同じ役割のインスタンスを複数用意し、1台が止まっても残りで動き続ける

SAMPLE

インスタンスは、壊れたら作り直す前提の使い捨ての部品として扱います。インスタンスのローカルディスクに状態を置くと、作り直した時点で状態は残りません。データはボリュームやオブジェクトストレージに置き、インスタンスは同じイメージから何度でも起動できる状態に保ちます。

スケーラビリティは、負荷の増加に対して台数や容量を増やして対応できる性質です。高可用性は、一部が故障してもサービスを続けられる性質です。2つは別の性質です。台数を増やしても、1つの共有部品（単一のデータベースなど）が止まれば、全体は動けません。単一障害点を残さないことが、高可用性の条件です。単一障害点（SPOF）とは、故障するとシステムの停止やデータの損失を起こす、単独の機器やソフトウェア。[^f6g1-4]

OpenStack のサービスは、高可用性の面で2つの型に分かれます。

型	意味	例
ステートレス	要求に応答したら、以後の世話が要らない。要求ごとに独立している	nova-api、glance-api、keystone-api、neutron-api、nova-scheduler
ステートフル	後の要求が、最初の要求の結果に依存する	データベース、メッセージキュー

ステートレスなサービスは、同じものを複数用意してロードバランサーで振り分ければ、可用性が上がります。ステートフルなサービスは、台数を増やして振り分けるだけでは解決しません。状態をそろえる仕組みが要り、構成はアクティブ/パッシブかアクティブ/アクティブのどちらか。[^f6g1-3]

- アクティブ/パッシブ：故障したときに起動する冗長な待機系を持つ。待機系は、平常時は動かさずに備える

SAMPLE

- アクティブ/アクティブ：本体と冗長系を同時に動かす。負荷を分け合い、1台が落ちても利用者には見えにくい
- クォーラム：冗長なノードのクラスターが動き続けるのに、最低限必要な稼働ノード数。過半数に設定する（7ノードなら 4）[^f6g1-4]

アクティブ/パッシブは「待機系がある」、アクティブ/アクティブは「全員が働く」と覚えると、名前の取り違えが減ります。スケールアップは1台の性能を上げる方法で、この2つの構成とは別の話です。

高可用性の目標は、99.99% 以上が目安です。OpenStack の基盤そのものは、この水準を実現できますが、個々のゲストインスタンスの可用性は保証されません。インスタンスの上で動くアプリケーションの可用性は、利用者が冗長化や負荷分散で対処する領域。基盤が 99.99% でも、インスタンスに同じ数字を約束する根拠にはなりません。[^f6g1-4]

永続ボリュームの共有では、Cinder のボリュームを、既定では1つのインスタンスに接続して使います。複数のインスタンスから同時に使う道は、2つ。1つは、ボリュームタイプに `multiattach=<is> True` を付けた multiattach 対応のボリュームです（Cinder API のマイクロバージョン 3.50 以上）。もう1つは、NFS や Manila のような共有ファイルシステムです。multiattach のボリュームでは、複数のインスタンスの書き込みを調整するクラスター対応のファイルシステムを、利用者が用意します。調整が無いと、データが壊れる恐れがあります。[^f6g1-5]

非機能要件と SLA

非機能要件は、機能そのもの以外の、システムに求める品質です。IaaS では次のようなものがあります。

- 可用性：どれだけ止まらずに動くか

SAMPLE

- 性能：処理の速さやネットワークの転送速度
- 拡張性：負荷や利用者の増加に対応できるか
- 保守・運用：保守作業の時間、バックアップ、監視
- セキュリティ：アクセス制御、暗号化、データの保護

SLA（Service Level Agreement、サービス品質保証）は、事業者と利用者の間で、どの水準までサービス品質を保証するかを合意した文書です。結ぶ意義は、品質の水準を数字で決めておくことで、基準を満たさなかった場合の対応（料金の返還など）を事前に取り決められる点にあります。口頭の約束では、障害が起きたときの基準がありません。

SLA に明記される項目は、OPCEL の例題が3つ挙げています。年間や月間の障害件数の上限、データ転送速度の下限、システム監視間隔の上限です。障害発生時の停止時間で保証するのは「上限」で、「下限」ではありません。停止時間を一定以上にする約束は、利用者にとって意味がないからです。[^c1-2]

可用性は稼働率で表します。稼働率が99.9%なら、止まってよい時間は年に8.76時間（8,760時間の0.1%）、30日の月では43.2分です。

稼働率	年間の停止時間の上限	30日の月の上限
99%	87.6時間	7.2時間
99.9%	8.76時間	43.2分
99.99%	52.56分	4.32分
99.999%	5.256分	約26秒

HA ガイドは、99.99% を保つには、障害後の復旧を 1〜2 分程度に抑える必要があるとしています。[^f6g1-4]

SAMPLE

構成全体の稼働率は、つなぎ方で変わります。直列（どれか1つが止まれば全体が止まる）なら、稼働率を掛け合わせます。例：99.9%の部品2つを直列にすると、0.999×0.999＝0.998001 で約99.8%。並列（1つでも動いていれば全体が動く）なら、止まる確率を掛け合わせます。99%の部品2つの並列は 1−0.01×0.01＝0.9999 で、99.99%に上がります。冗長化は、この並列の計算で稼働率を押し上げます。部品1つの質を上げることだけが、稼働率を上げる手ではありません。

SLA の稼働率の計算は、稼働できる時間と、計画停止を除いた停止時間が材料。計画停止は、保守のために事前に告知した停止。障害による予期しない停止は、稼働率を下げる要因として数えます。復旧が早くても、計画停止でなければ停止時間に入ります。[^f6g1-6]

OpenStack のアーキテクチャと設計 [250.2]

OpenStack の全体像：サービスの束

OpenStack は単一のソフトウェアではなく、役割ごとに分かれた独立のサービスの集まりです。各サービスにはコード名（プロジェクト名）があり、Nova、Neutron、Cinder のように呼びます。サービスどうしは、次の3つの決まりでつながります。[^c1-3]

- すべてのサービスは、共通の Identity サービス（Keystone）で認証する
- サービスどうしは、特権の管理者コマンドが要る場合を除いて、公開された REST API で通信する。1つのサービスの内部のプロセスどうしは、AMQP のメッセージブローカー（RabbitMQ など）を使う
- 各サービスは状態をデータベースに保存する

利用者は、Horizon（ダッシュボード）、コマンドラインのクライアント、curl などで直接 API を呼ぶ方法、SDK を使うアプリケーションのどれからでも操作で

SAMPLE

きます。どの方法も、最後は各サービスへの REST API 呼び出しになります。[^c1-3]

エンドユーザーが使えるサービスと使い方

エンドユーザー（一般の利用者）は、プロジェクトの中でインスタンスの作成と削除、イメージの登録、ボリュームの作成とインスタンスへの接続ができます。仮想ネットワークとルーターの作成、フローティング IP の割り当て、キーペアとセキュリティグループの管理も、利用者の操作です。

操作の入口は、次の3つです。[^c1-4]

- Horizon：Web 画面。プロジェクト、管理（管理者のみ）、ID の3つのパネル群に分かれる
- openstack コマンド：コンピュート、アイデンティティ、イメージ、オブジェクトストレージ、ブロックストレージを、1つのシェルと統一した書式にまとめたクライアント
- API の直接呼び出し：Keystone で取ったトークンを HTTP ヘッダーに付けて、各サービスに要求を送る

コアコンポーネントの役割とコード名

試験で最も問われるのが、次の6つの対応です。

コード名	サービス	役割
Keystone	Identity（アイデンティティ）	認証、認可、サービスカタログ
Nova	Compute（コンピュート）	インスタンスの作成と管理
Glance	Image（イメージ）	起動に使うディスクイメージの登録と管理

SAMPLE

コード名	サービス	役割
Neutron	Networking（ネットワーク）	仮想ネットワーク、ルーター、DHCP など
Cinder	Block Storage（ブロックストレージ）	インスタンスに接続するボリューム
Horizon	Dashboard（ダッシュボード）	Web 画面による操作

OPCEL の例題には、Horizon を「ネットワークサービスを提供するコンポーネント」とした誤りの選択肢が出ています。Horizon はダッシュボードで、ネットワークは Neutron の担当。Nova の役割を問う例題では、選択肢の「起動イメージの管理」は Glance、「ブロックストレージの管理」は Cinder、「オブジェクトストレージの管理」は Swift の役割だと解説しています。[^c1-2]

各コンポーネントの内部を、公式文書の言い方で見ます。

Nova は、いくつものプロセスで構成されます。[^c1-5]

プロセス	役割
nova-api	エンドユーザーのコンピュート API を受け付けて応答する
nova-api-metadata	インスタンスからのメタデータ要求を受け付ける
nova-scheduler	キューから受け取ったインスタンスの要求について、どのコンピュートホストで動かすかを決める
nova-conductor	nova-compute とデータベースの間を仲介する。コンピュートノードがデータベースを直接問い合わせないようにする

SAMPLE

プロセス	役割
nova-compute	ハイパーバイザーの API を通じてインスタンスを作成、終了するワーカー
nova-novncproxy	VNC 接続でインスタンスにアクセスするためのプロキシ
nova-spicehtml5proxy	SPICE 接続でインスタンスにアクセスするためのプロキシ

nova-conductor は台数を横に増やせますが、nova-compute が動くノードには置きません。コンピュートノードがデータベースに直接つながらない作りにするための部品です。nova-compute が操作するハイパーバイザーには、KVM/QEMU、XenServer、VMware などがあります。[^c1-5] ハイパーバイザーの API は、KVM か QEMU なら libvirt、XenServer なら XenAPI、VMware なら VMwareAPI。[^c1-5]

Nova は、主要な部品のほとんどを複数のサーバーで動かせる作りです。コンピュートの台数は、他の部品よりずっと多くなるのが普通。データベースは論理的に全部品で共有します。データベースへのアクセスはオブジェクト層を通り、nova-compute のデータベースの要求は、RPC で nova-conductor が代わりに処理します。アップグレードの途中でも、新しい制御側と、前のリリースの nova-compute が通信できるようにするための作りです。[^c1-12]

Nova と組んで使うのが Placement です。Placement は Stein（19.0.0）で Nova から切り出された独立のサービスで、リソースプロバイダー（コンピュートノードなど）の在庫と使用量を管理します。Nova はその API で資源の空きを調べます。[^c1-6] 資源の種類はリソースクラスで表し、標準のクラスに VCPU、MEMORY_MB、DISK_GB があります。[^f7-2]

Cinder の構成は次のとおりです。[^c1-7]

SAMPLE

プロセス	役割
cinder-api	API 要求を受け付け、cinder-volume に回す
cinder-scheduler	ボリュームを作る最適なストレージのノードを選ぶ。nova-scheduler に似た役割
cinder-volume	ブロックストレージの読み書き要求に応じ、状態を保つ
cinder-backup	あらゆる種類のボリュームをバックアップ先に保存する。cinder-volume と同じく、ドライバーの仕組みで様々な保存先と連携する

Cinder のバックエンドには、NFS、iSCSI、Ceph など様々なストレージを使えます。[^c1-2]

Glance は、glance-api がイメージの検索、取得、保存の API を受け付けます。glance-registry はイメージのメタデータを扱うプロセスで、Queens リリースで非推奨になりました。[^c1-8] OPCEL の例題には、Glance が glance-api と glance-registry で構成されると書いた解説があります。Queens〜Ussuri が範囲なので、registry は非推奨という事実と合わせて覚えます。イメージファイルの置き場所（ストア）には、通常のファイルシステム、Object Storage、RADOS ブロックデバイス、VMware のデータストア、HTTP が使えます（一部は読み取り専用）。[^c1-8]

Neutron は、neutron-server が API を受け付けます。ノード上で動くエージェントは、次の4つです。

- neutron-dhcp-agent：DHCP を担当する
- neutron-metadata-agent：インスタンスのメタデータ要求を中継する
- neutron-linuxbridge-agent：ブリッジを操作する（手順書の例）

SAMPLE

- neutron-l3-agent：セルフサービスネットワークでルーターを担当する [^c1-9]

コンピュートノードでも Networking のエージェント（手順書の例は neutron-linuxbridge-agent）が動き、インスタンスを仮想ネットワークにつなぎ、セキュリティグループでファイアウォールの機能を提供します。[^f6g1-10]

ネットワークの形には2種類あります。プロバイダーネットワークは、主にレイヤー2のサービスと VLAN で物理ネットワークに橋渡しする形です。セルフサービスネットワークは、レイヤー3のルーティングを足し、VXLAN などのオーバーレイでテナントごとの仮想ネットワークを作って NAT で外に出す形です。[^c1-10] プロバイダーネットワークだけの構成（手順書の Networking Option 1）では、セルフサービス（プライベート）ネットワーク、ルーター、フローティング IP を使えません。LBaaS や FWaaS のような高度なサービスの土台になるのは、セルフサービスネットワークの構成（Option 2）です。[^f7-3]

Keystone の内部は、Identity、Resource、Assignment、Token、Catalog、Policy などの機能に分かれます。Identity はユーザーとグループのデータと認証情報の検証、Resource はプロジェクトとドメインのデータ、Assignment はロールとロール割り当てのデータを扱います。Token はトークンを検証して管理し、Catalog はエンドポイントの登録簿です。Glance などの各サービスは、受け取ったトークンを Keystone に問い合わせて確かめ、利用者の身元とロールを得ます。[^f7-1] ロール割り当ては、ロール、リソース、アイデンティティの3つ組。[^c1-28] 詳しい使い方は、第2章で扱います。

インスタンスを起動するまでの連携

openstack コマンドでインスタンスを作る例を見ます。フレーバー（CPU・メモリ・ディスクの大きさ）、イメージ、ネットワークを指定します。

SAMPLE

```
openstack server create --flavor m1.tiny --image cirros \
  --network provider --key-name mykey provider-instance
```

この要求が、どの部品をどの順で通るかを手順で示します。

1. クライアントが Keystone で認証し、トークンを得る。以後の API 呼び出しにそのトークンを付ける
2. nova-api が要求を受け、トークンを検証し、データベースに記録する
3. nova-api が、メッセージキューを通じて nova-conductor に構築を依頼する。nova-conductor が nova-scheduler に置き先を問い合わせ、nova-scheduler は Placement に資源の空きを問い合わせ、フィルターで候補のホストを絞ってから重み付けで1台を選ぶ
4. 選ばれたホストの nova-compute に、nova-conductor からメッセージキューで起動の指示が届く
5. nova-compute が Glance からイメージを取得する
6. nova-compute が Neutron に、インスタンスの仮想ポートの作成とネットワークへの接続を依頼する
7. ボリュームを使う場合は、Cinder にボリュームの接続を依頼する
8. nova-compute がハイパーバイザーに仮想マシンの作成を指示する

3のスケジューラーは、まずフィルターで候補を絞ります。フィルターは、ホストを受け入れるか除外するかの二択です。受け入れられたホストを、別のアルゴリズムが重み付けして選びます。重みは正規化してから足し合わせ、最も大きいホストが選ばれます。[^c1-11]

覚えておく分け方は、Keystone は認証、Glance はイメージ、Neutron はネットワーク、Cinder はボリューム、Nova が全体を束ねて仮想マシンを作る、とい

SAMPLE

う役割分担です。

起動に失敗したときは、症状から止まっている部品を絞れます。

症状	疑う部品
認証でトークンが得られない	Keystone
条件に合うホストが見つからず、インスタンスを置けない	nova-scheduler と Placement（資源の空き、フィルターの条件）
起動に使うイメージを取得できない	Glance
仮想ポートを作れず、ネットワークに接続できない	Neutron
ボリュームを接続できない	Cinder
コンピュートノードを増やしたのにインスタンスが載らない	discover_hosts の未実行（250.3 を参照）

サービスの内部の型：API・スケジューラー・ワーカー・エージェント

OpenStack の多くのサービスは、同じ型で作られています。

- API サービス：REST の要求を受け付け、認証と検査をして、データベースに記録し、他のプロセスに依頼する（nova-api、cinder-api、glance-api、neutron-server）
- スケジューラー：どのノードに仕事を割り当てるかを決める（nova-scheduler、cinder-scheduler）
- ワーカー：実際の作業をするプロセス（nova-compute、cinder-volume）
- エージェント：ノード上で動き、そのノードの設定を現実に反映する。Neutron の各エージェントが代表例

SAMPLE

プロセスどうしの内部の呼び出しは RPC（遠隔手続き呼び出し）です。Nova では、oslo.messaging というライブラリが RPC をメッセージキューの上で動かします。API サーバーは REST 要求を処理したあと、メッセージキューを通じて、対象のサービス（たとえばコンピュート）に RPC メッセージを送ります。受け取ったサービスのマネージャーがそのメッセージを処理します。[^c1-12]

窓口は REST、内部は RPC（メッセージキュー）という二重の構造。ユーザーは REST、サービスの中のプロセスどうしは RPC です。

インストールガイドは、メッセージキューを「デーモンの間でメッセージを受け渡す中央のハブ」と呼び、サービスの間の操作と状態の情報の調整もここで行うと書いています。[^c1-5]

各サービスは、複数のプロセスでできています。どのサービスにも、API の要求を待ち受けて前処理をし、サービスの他の部分に渡す API プロセスが最低1つあります。Identity サービスを除いて、実際の作業は別のプロセスが行います。Identity サービスは例外で、API プロセスが作業も受け持つ作り。[^f6g1-8]

同じサービスのプロセスどうしは、AMQP のメッセージブローカー（メッセージキュー）で連絡します。サービスの状態を保存するのは、データベース。ブローカーはメッセージを運ぶだけで、状態の置き場ではありません。

支えるサービス①：データベース・メッセージキュー・memcached

OpenStack の各サービスを支えるミドルウェアは、次の3つ。

部品	役割	手順書での例
SQL データベース	各サービスの状態を保存する	MariaDB（MySQL 互換）。ポート 3306

SAMPLE

部品	役割	手順書での例
メッセージキュー	サービス内のプロセスどうしのメッセージを中継する	RabbitMQ。ポート 5672
memcached	Keystone のトークンをキャッシュする	設定ファイル /etc/memcached.conf

データベースは、ほとんどの OpenStack サービスが状態を保存する先で、Ubuntu の手順書の例では MariaDB を入れます。手順書の例では、MariaDB、RabbitMQ、memcached をコントローラーノードに入れます。設定ファイル `/etc/mysql/mariadb.conf.d/99-openstack.cnf` には、`bind-address`（コントローラーの管理用 IP）、`default-storage-engine = innodb`、`max_connections = 4096` などを書きます。[^c1-13] `bind-address` を管理用 IP にするのは、他のノードから接続させるためです。データベースはサービスごとに分け、`keystone`、`glance`、`placement`、`nova_api`、`nova`、`nova_cell0`、`neutron`、`cinder` を作ります（Heat を入れるなら `heat` も）。

メッセージキューは、RabbitMQ や Qpid を含むいくつかの製品に対応しています。手順書の例で入れるのは RabbitMQ だけで、Qpid は入れません。多くのディストリビューションが RabbitMQ に対応しているためです。準備は、次の2つのコマンドです。[^c1-14]

```
rabbitmqctl add_user openstack RABBIT_PASS
rabbitmqctl set_permissions openstack ".*" ".*" ".*"
```

1つ目がユーザー `openstack` の作成で、2つ目が設定、書き込み、読み取りの権限の付与です。各サービスの設定ファイルには、`transport_url = rabbit://openstack:RABBIT_PASS@controller:5672/` の形で接続先を書きます。

SAMPLE

memcached は、Identity サービスの認証の仕組みがトークンをキャッシュするために使います。設定ファイル `/etc/memcached.conf` の `-l` には、既定の `127.0.0.1` ではなくコントローラーの管理用 IP アドレスを書きます。他のノードから届かせるための設定で、既定のままでは他のノードが memcached に接続できません。[^c1-15]

3つの役割は、永続する状態の保存がデータベース、プロセスどうしの連絡がメッセージキュー、認証の一時記憶が memcached、と一言で分けられます。

支えるサービス②：共有ストレージ・NFS・仮想化

仮想化（ハイパーバイザー）は、1台の物理サーバーを複数の仮想マシンに分ける層です。nova-compute は各コンピュートノードで動き、ハイパーバイザーの API を通じてインスタンスを作成、終了します。対応するハイパーバイザーには、KVM/QEMU、XenServer、VMware などがあります。[^c1-5] ハイパーバイザーを介さず物理サーバーを丸ごと払い出す役目は、Ironic の担当。

CPU やメモリで、実際の容量より多くのリソースを割り当てる機能が、オーバーコミットです。代表は、物理 CPU を超える仮想 CPU の割り当て。OPCEL の例題では、CPU のオーバーコミット率が8なら、物理コア1個に最大8個の仮想コアを割り当てられると解説しています。[^c1-2] Nova の公式文書によると、Stein より前のリリースで設定が未指定のとき、既定値は次のとおりです。`cpu_allocation_ratio` が 16.0、`ram_allocation_ratio` が 1.5、`disk_allocation_ratio` が 1.0 です（Stein 以降、新しいコンピュートノードの初期値は `initial_cpu_allocation_ratio`、`initial_ram_allocation_ratio`、`initial_disk_allocation_ratio` で決まり、既定は CPU 16.0、メモリ 1.5、ディスク 1.0）。[^c1-11][^f7-4] スケールアップ（1台の性能を上げる）、スケールアウト（台数を増やす）、オーバーレイ（ネットワークのトンネル）は、オーバーコミットの問いでは正解になりません。

SAMPLE

共有ストレージは、複数のノードから同じデータに届くストレージです。NFS（ネットワークファイルシステム）は、その代表的な方式で、ネットワーク越しにファイルシステムをマウントして共有します。OpenStack では次のような使い方があります。

- Cinder のバックエンドとして、NFS、iSCSI、Ceph などのストレージを使う
- Glance のイメージの置き場として、ファイルシステムや RADOS ブロックデバイスを使う
- Manila が提供する共有ファイルシステムを、複数のインスタンスからマウントして共有する

取り違えやすい3つは、使い方で見分けます。

- Cinder（ブロックストレージ）：インスタンスに接続して、ディスクのように使う
- Swift（オブジェクトストレージ）：HTTP でオブジェクトを出し入れする
- Manila（共有ファイルシステム）：複数のインスタンスでファイルを共有する

追加コンポーネント①：Heat・Ceilometer・aodh・Swift

Heat はオーケストレーションサービスで、テンプレートに書いた構成（インスタンス、ネットワーク、ボリュームなど）をまとめて自動で作ります。Ussuri の手順書では `heat-api`、`heat-api-cfn`、`heat-engine` の3つを入れます。API のエンドポイントは、8004 番（orchestration）と 8000 番（cloudformation）です。[^c1-16] heat-api-cfn は、AWS CloudFormation 形式の API に対応する窓口。

Ceilometer は、OpenStack のコアコンポーネントの利用状況を集めて正規化するデータ収集サービスで、課金、リソースの追跡、アラームに使えるデータ

SAMPLE

を出します。[^c1-17] aodh はアラームサービスで、Ceilometer や Gnocchi が集めた測定値やイベントのデータに対して定義したルールに基づき、アクションを起動します。[^c1-18] 計測は Ceilometer、その値を見て通知や処理を起動するのは aodh、と分けて覚えます。

公式文書は Swift を、冗長でスケーラブルなデータストレージであり、更新や取得ができる大量の静的データの長期保存システムだと説明しています。複数のマシンにデータのコピーを分散し、中心となる単一のノードはありません。容量を増やすにはノードを足します。[^c1-19] Ussuri の手順書では、プロキシのエンドポイントが 8080 番の `/v1/AUTH_(プロジェクトID)` です。[^c1-20]

追加コンポーネント②：Ironic・Designate・Manila・Barbican・Magnum

サービス	役割	取り違えやすい相手
Ironic	物理サーバー（ベアメタル）を、仮想マシンのように払い出す	Nova（仮想マシン）
Designate	DNS を API で管理する DNSaaS	Neutron の DHCP
Manila	共有ファイルシステムを提供する	Cinder（ブロック）、Swift（オブジェクト）
Barbican	パスワード、暗号鍵、X.509 証明書など秘密情報の保管と管理	Keystone（認証）
Magnum	コンテナオーケストレーションエンジンを払い出す	Heat（テンプレート）
Octavia	ロードバランサーを提供する	Neutron（ネットワーク全般）

SAMPLE

Ironic は、仮想ではなく物理のマシンを払い出します。間にハイパーバイザーは入りません。単独でも OpenStack の一部としても使え、Identity、Compute、Networking、Image、Object の各サービスと連携します。PXE と IPMI などの管理プロトコルでハードウェアを管理し、コンピュートサービスに対して、物理サーバーを仮想マシンと同じように扱うためのインターフェースを提供します。[^c1-21] 内部のプロセスは、OPCEL の例題が次の2つを挙げています。

- ironic-conductor：IPMI でベアメタルノードの電源を入れ、iSCSI でノードにイメージをコピーする
- ironic-python-agent：ラムディスクに組み込まれ、ブートローダーや OS イメージのインストールなどを行う REST API を提供するエージェント

「ironic-iscsi」や「ironic-image-copy」という名前のサービスはありません。[^c1-2]

Designate は、マルチテナント対応の DNSaaS です。Keystone の認証を組み込んだ REST API を持ち、Nova と Neutron の操作に応じてレコードを自動生成する設定もできます。DNS サーバーは Bind9、PowerDNS 4 などに対応します。[^c1-22]

Barbican は鍵管理サービス（Key Manager）で、対称鍵、非対称鍵、証明書、任意のバイナリデータといった秘密情報の安全な保管、プロビジョニング、管理を担当します。[^c1-23] Keystone が確かめるのは「誰か」、Barbican が守るのは秘密そのものです。両者は守る対象が違うので、取り違えません。

Magnum は、コンテナオーケストレーションエンジンを OpenStack の第一級のリソースとして、コンテナの展開と管理に使えるようにします。扱うオブジェクトには、クラスター、クラスターテンプレートなどがあります。[^c1-24] クラスター（Cluster）はノードの集まり。クラスターテンプレート（ClusterTemplate）

SAMPLE

はクラスターの雛形で、同じ構成のクラスターを繰り返し作るのに使います。[^c1-24]

Manila は共有ファイルシステムサービスで、作ったファイルシステムを複数のインスタンスからマウントして共有します。[^c1-2][^c1-25]

OPCEL の例題には、Murano（アプリケーションカタログ）と Zaqar（マルチテナント対応のクラウドメッセージング）が、誤りの選択肢として登場します。[^c1-2] 上の表のどれにも当たらない機能の説明は、除外してかまいません。

追加コンポーネント③：Cyborg と Kolla

Cyborg は、特定の処理を高速化する専用のハードウェアであるアクセラレーター（GPU や FPGA など）を管理するための、汎用の管理フレームワークです。[^c1-26] 扱うのは、リソースの発見（Resource Discovery）とライフサイクル管理（Life Cycle Management）です。[^f7-5]

Kolla は、OpenStack を運用するための、本番向けのコンテナとデプロイツールを提供することを使命とするプロジェクトです。Kolla-Ansible は、Kolla が作るコンテナイメージを使って、Ansible で OpenStack のサービスを展開するツールです。[^c1-27]

API のエンドポイントとサービスカタログ

各サービスの API の窓口の URL を、エンドポイントと呼びます。Keystone は、どのサービスがどこにあるかを一覧にしたサービスカタログを持ち、クライアントはそこから接続先を引きます。Keystone の内部で、エンドポイントの登録簿の役割を担うのが Catalog サービス。[^c1-28]

エンドポイントには、公開（public）、内部（internal）、管理（admin）の3種類（インターフェース）があります。Ussuri の Keystone の手順書では、`keystone-`

SAMPLE

`manage bootstrap` に `--bootstrap-admin-url`、`--bootstrap-internal-url`、`--bootstrap-public-url` を渡します。3つとも `http://controller:5000/v3/` を指します。[^c1-29] 3種類は役割の違いです。外部の利用者が使う窓口が public、クラウドの内側のサービスどうしが使う窓口が internal、管理操作用が admin。

各サービスの手順書の既定のポートは、次の表のとおり。

サービス	ポート	備考
Keystone	5000	Queens より前は、v2 API 用の管理専用サービスが 35357 番で別に動いていた。v2 API の廃止後は5000だけで動かせる。設定項目 admin_port の既定値には35357が残る
Nova（Compute）	8774	`http://controller:8774/v2.1`
Nova メタデータ	8775	
Placement	8778	Ussuri の手順書のエンドポイント例
Glance	9292	
Neutron	9696	
Cinder	8776	v2 は `/v2/`、v3 は `/v3/` に `%(project_id)s` が付く
Heat	8004	CFN 互換 API は 8000

SAMPLE

サービス	ポート	備考
Swift プロキシ	8080	`/v1/AUTH_(プロジェクトID)`
noVNC プロキシ	6080	`novncproxy_base_url = http://controller:6080/vnc_auto.html`
RabbitMQ	5672	

ポートの値は、ディストリビューションや設定で変わります。公式のファイアウォール一覧は、Placement を 8003 番と載せていて、手順書のエンドポイントの値とは違います。公式のファイアウォール一覧は、Swift（Object Storage）に 6000、6001、6002 番を載せています。ただし Swift の値も 6000〜6002 番だけではありません。別の表には、プロキシの 8080 番が載っています。Placement の手順書も、環境によっては 8780 番やポート無しになると書いています。[^c1-30] 問題文が手順書の例を指すなら、エンドポイントの値で答えます。ファイアウォール一覧の値は使いません。

OpenStack のインストレーションとデプロイメント [250.3]

クライアントのインストール

openstack コマンドを提供するのは、python-openstackclient パッケージです。入れ方は2つあり、PyPI から入れる方法と、ディストリビューションのパッケージで入れる方法があります。Compute、Identity、Image、Object Storage、Block Storage の操作は、これ1つで足ります。Heat の `openstack stack` のように、プラグイン（python-heatclient など）を足して使うコマンドもあります。[^c1-4]

SAMPLE

方法	コマンド
pip	`pip install python-openstackclient`
pip3	`pip3 install python-openstackclient`
Ubuntu のパッケージ	`apt install python3-openstackclient`
CentOS/RHEL のパッケージ	`dnf install python3-openstackclient`

pip は Python のパッケージ管理ツールで、pip3 は Python 3 用の pip です。手順書は、環境や Python のバージョンによっては pip を pip3 と書く必要があると注記しています。[^f7-7] OPCEL の出題範囲が重要なユーティリティに挙げるのも、pip コマンドと pip3 コマンドの両方。[^c1-40] Ubuntu と CentOS/RHEL のパッケージ名は、公式のインストールガイドでは `python3-openstackclient` です。[^c1-31] 入れたクライアントには、認証情報（環境変数、openrc、clouds.yaml）を渡して使います。

旧来の個別のクライアント（サービスごとに別名のコマンド）を1つにまとめた書式が、openstack コマンド。公式文書には旧クライアントとの対応表がありますが、公式自身がこの表を不完全（incomplete）だと書いています。[^c1-4] 試験に出るのは openstack コマンドの書式で、個別のクライアントのコマンドは範囲に入りません。

旧クライアントで行っていた操作	openstack コマンド
Cinder の上限値の表示	`openstack limits show --absolute`
Glance のイメージ一覧	`openstack image list`

SAMPLE

旧クライアントで行っていた操作	openstack コマンド
Neutron のネットワーク一覧	`openstack network list`
Nova のインスタンス作成	`openstack server create`
Nova のフレーバー一覧	`openstack flavor list`

手動インストールの利点と欠点

手動インストールは、パッケージを入れて設定ファイルを1つずつ書き、サービスを自分で連携させる方法です。

	内容
利点	各部品の役割と連携がわかる。構成を完全に自分で決められる。障害時の原因の切り分けにも役立つ
欠点	手順が多く、時間がかかる。設定の書き間違いが起こりやすい。同じ環境を再現しにくい。台数が増えるほど手作業の負担も増える

手動デプロイの一般的な順序

公式のインストールガイドは、環境の準備から始めます。環境には、セキュリティ、ホストのネットワーク、NTP（時刻同期）、OpenStack のパッケージ、SQL データベース、メッセージキュー、memcached、etcd が含まれます。[^c1-13] 時刻がずれるとトークンの有効期限の判定などがずれ、認証が通らなくなるため、NTP でノードの時刻をそろえておかなければなりません。手順書が入れる NTP の実装は Chrony です。[^f7-8]

そのあと、最低限のサービスを次の順に入れます。[^c1-32]

SAMPLE

1. Identity サービス（Keystone）
2. Image サービス（Glance）
3. Placement サービス
4. Compute サービス（Nova）
5. Networking サービス（Neutron）

最低限の構成のあとに、Dashboard（Horizon）と Block Storage（Cinder）の導入が勧められています。[^c1-32]

順序は依存関係で決まります。Keystone が最初なのは、他のすべてのサービスが認証に使うからです。Keystone は認証情報を SQL データベースに保存するので、データベースが先に動いている必要があります。OPCEL の例題は、最初に入れるべきものを「Keystone」と「SQLデータベース」と答えています。LDAP サーバーは Keystone の認証元に使えますが、必須ではありません。Horizon は Web 画面を提供するだけのサービスなので、他のサービスが動いてから入れます。[^c1-2]

Nova は Placement に資源の空きを問い合わせ、Glance からイメージを取るので、Placement と Glance が Nova の前に来ます。Neutron は Nova と連携してインスタンスをネットワークに接続するため、Nova のあとです。並べ替えの問いでは、この依存を順にたどれば、迷いません。

データベースの初期化と管理のコマンド

パッケージを入れて設定したあとは、各サービスでデータベースの表を作る初期化が要ります。使うのは、サービスごとの `*-manage` コマンドで、公式の範囲に名前がある管理用のコマンドです。

SAMPLE

コマンド	実行例	役割
keystone-manage	`keystone-manage db_sync`	Keystone のデータベースを作成、更新する
glance-manage	`glance-manage db_sync`	Glance のデータベースを作成、更新する
nova-manage	`nova-manage api_db sync` 、`nova-manage db sync`	Nova のデータベースを作成、更新する
neutron-db-manage	`neutron-db-manage --config-file ... upgrade head`	Neutron のデータベースを更新する
cinder-manage	`cinder-manage db sync`	Cinder のデータベースを作成、更新する
heat-manage	`heat-manage db_sync`	Heat のデータベースを作成、更新する

書式は、サービスごとに少しずつ違います。[^c1-29][^c1-10][^c1-33][^c1-34]

- keystone、glance、heat：`db_sync`（アンダースコア）
- cinder、nova のセル用データベース：`db sync`（スペース）
- nova の API データベース：`api_db sync`（スペース）
- neutron：`upgrade head`

公式の手順書では、これらを `su -s /bin/sh -c "コマンド" サービス名` の形で実行します。サービス用のユーザーとして実行するための書き方です。

SAMPLE

```
su -s /bin/sh -c "keystone-manage db_sync" keystone
```

Neutron は、設定ファイルを2つ指定します。

```
su -s /bin/sh -c "neutron-db-manage --config-file
/etc/neutron/neutron.conf \
  --config-file /etc/neutron/plugins/ml2/ml2_conf.ini upgrade head"
neutron
```

手順書は、Neutron のデータベースを後で作る理由を、このスクリプトがサーバーとプラグインの完全な設定ファイルを必要とするからだと書いています。[^c1-10] 設定が未完のまま実行しても、意図した結果にはなりません。

Keystone には、`db_sync` のほかに初期設定のコマンドがあります。

1. `keystone-manage db_sync`：データベースの表を作る
2. `keystone-manage fernet_setup --keystone-user keystone --keystone-group keystone`：Fernet トークンの鍵置き場を作る
3. `keystone-manage credential_setup --keystone-user keystone --keystone-group keystone`：認証情報の鍵置き場を作る
4. `keystone-manage bootstrap`：最初の管理者、ロール、エンドポイントを作る。`--bootstrap-password`、`--bootstrap-admin-url`、`--bootstrap-internal-url`、`--bootstrap-public-url`、`--bootstrap-region-id RegionOne` を渡す

2 と 3 の `--keystone-user` と `--keystone-group` は、Keystone を動かす OS のユーザーとグループを指定します。[^c1-29] bootstrap は管理者ユーザーをデータベースに書き込むので、先に db_sync で表を作っておかなければ動き

SAMPLE

ません。[^c1-29] 設定ファイルの `[token]` セクションには、`provider = fernet` を書きます。

Nova は、複数のデータベースとセルの設定が加わるので、手順が一番長くなります。Ussuri の手順書の順序は、次のとおりです。[^c1-33]

1. `nova-manage api_db sync`：nova_api データベースに表を作る
2. `nova-manage cell_v2 map_cell0`：cell0 を登録する
3. `nova-manage cell_v2 create_cell --name=cell1 --verbose`：cell1 を作る
4. `nova-manage db sync`：nova データベース（セル用）の表を作る
5. `nova-manage cell_v2 list_cells`：cell0 と cell1 が登録されたことを確かめる

cell0 はコンピュートノードを持たず、コンピュートノードは cell1 に属します。コンピュートノードを追加したときは、コントローラーで `nova-manage cell_v2 discover_hosts` を実行し、新しいノードを cell に登録します。手動の代わりに、`/etc/nova/nova.conf` の `[scheduler]` セクションへ `discover_hosts_in_cells_interval = 300` のように間隔を書くと、定期的に自動で探します。[^c1-35]「新しいコンピュートノードを入れたのにインスタンスが載らない」という場面では、discover_hosts の未実行が原因になります。

DevStack

DevStack は、開発とテスト用の OpenStack を素早く作るスクリプトの集まり。本番運用向けではありません。環境は、git の master ブランチ（開発中の最新）のコードから作ります。公式文書は、インストール中にシステムへ大きな変更が加わることと、この目的専用のサーバーか仮想マシンでだけ実行することを警告しています。[^c1-36]

手順は、次のとおりです。

SAMPLE

1. root ではなく、sudo が使える一般ユーザーを用意する。クラウドイメージでなければ、専用の stack ユーザーを作る（`useradd -s /bin/bash -d /opt/stack -m stack`）。sudo は `stack ALL=(ALL) NOPASSWD: ALL` で設定する。この行で、stack ユーザーはパスワードを聞かれずに sudo を使える。DevStack はシステムに多くの変更を加えるので、sudo の権限が要る
2. `git clone https://opendev.org/openstack/devstack` でリポジトリを取得し、そのディレクトリに入る
3. `local.conf` を作り、4つのパスワード（`ADMIN_PASSWORD`、`DATABASE_PASSWORD`、`RABBIT_PASSWORD`、`SERVICE_PASSWORD`）を書く
4. `./stack.sh` を実行する

local.conf は `[[local|localrc]]` の下に設定を書きます。ネットワークの範囲を決める `FLOATING_RANGE` と `FIXED_RANGE` も、この例で指定されています。`FLOATING_RANGE` は手元のネットワークの範囲と重ならないように選びます。[^c1-37] `FIXED_RANGE` が決めるのは、インスタンスが使う内部のアドレス空間。

```
[[local|localrc]]
FLOATING_RANGE=192.168.1.224/27
FIXED_RANGE=10.11.12.0/24
ADMIN_PASSWORD=supersecret
DATABASE_PASSWORD=iheartdatabases
RABBIT_PASSWORD=flopsymopsy
SERVICE_PASSWORD=iheartksl
```

`./stack.sh` を実行するユーザーは、root ではなく sudo の使える一般ユーザーです。[^c1-36]

SAMPLE

項目	内容
所要時間	15〜20分ほど（回線の速さに左右される）
終了時の表示	URL とアカウントの要約
対応するディストリビューション	最新の2つの Ubuntu LTS、最新の Fedora、CentOS/RHEL 7、Debian、OpenSUSE
環境を作り直したいとき	最後の手段として、`clean.sh` が出てくる[^c1-38]

Packstack

Packstack は、Red Hat 系の OS に OpenStack を入れる簡易ツールで、ソースは opendev.org の x/packstack にあります。RDO プロジェクトのもので、単一のホストにも、複数のホスト（SSH 経由）にも使えます。[^c1-39]

目的	コマンド
1台にすべて入れる	`packstack --allinone`
対話式で入れる	`packstack`
回答ファイルの雛形を作る	`packstack --gen-answer-file=ans.txt`
回答ファイルで入れる	`packstack --answer-file=ans.txt`

事前に RDO のリポジトリを有効にします（`yum -y install https://www.rdoproject.org/repos/rdo-release.rpm`）。`packstack --allinone` は、設定を聞かずに、1台に全サービスを入れます。実行すると `packstack-answers-日付-時刻.txt` という回答ファイル（answer file）が作られます。同じ設定で再実行するときは、`--answer-file` でこのファイルを渡します。[^c1-39]

SAMPLE

複数台に分けて入れるには、`--gen-answer-file` で作った回答ファイルを編集します。コンピュートを置くホストは `CONFIG_NOVA_COMPUTE_HOSTS` にカンマ区切りの IP で、Swift のストレージは `CONFIG_SWIFT_STORAGE_HOSTS` に書きます。リモートのマシンに入れる公開鍵は `CONFIG_SSH_KEY` に指定します。[^c1-39]

OPCEL の出題範囲は、DevStack と Packstack を「開発用およびテスト用の OpenStack クラウドをデプロイする」項目に並べています。[^c1-40] DevStack はスクリプトを実行して作る環境、Packstack は回答ファイルで設定を決めて入れる環境と覚えると、取り違えません。

リリースを新しくして古い回答ファイルを使い回すときは、廃止された項目や書き間違いが残りがち。Packstack が認識しない項目が無いかは、コマンドで先に調べます。

目的	コマンド
回答ファイルに認識されない項目が無いかを調べる	`packstack --validate-answer-file=ans.txt`
詳しいログを書かせる	`packstack -d --allinone`

`CONFIG_SWIFT_STORAGE_HOSTS` は、カンマ区切りで Swift のストレージを並べます。1つの要素の書き方は2通りです。

書き方	意味
`192.0.2.51`	IP アドレスだけ。そのホストにテスト用のループバックデバイスを作り、ストレージに使う
`192.0.2.50/sdb`	そのホストの `/dev/sdb` をストレージに使う

SAMPLE

デバイス名は `/sdb` の形で書けますが、IP アドレスだけの指定も有効で、物理ディスクの自動検出は行いません。[^f6g1-9]

導入が途中で失敗したときの調べ先は、`/var/tmp/packstack` の下です。実行した日付と乱数の名前のディレクトリ（例：`/var/tmp/packstack/20131022-204316-Bf3Ek2`）ができ、その中に `openstack-setup.log` と、puppet のマニフェストを入れた manifests ディレクトリが置かれます。マニフェストごとのログも manifests の中。[^f6g1-9]

開発・テスト用の構成と本番の構成

開発・テスト用の構成は、1台のホストにすべてのサービスを載せるオールインワンで、DevStack や `packstack --allinone` が作ります。手早く作れて壊しても影響が小さい代わりに、1台が止まれば全部が止まり、性能にも余裕がありません。

本番の構成は、役割ごとにノードを分けるマルチノードです。サービスを複数台で冗長にし、単一障害点を残しません。

公式のインストールガイドが示す例の構成は、本番向けではありません。OpenStack を学ぶための最小の概念実証という位置づけ。基本のインスタンスを起動するのに要るのは、コントローラーノードとコンピュートノードの最低2台です。ブロックストレージとオブジェクトストレージのノードは、それぞれのサービスを足す時にだけ加える任意のノード。[^f6m-1]

ノード	例の構成で載せる物
コントローラー	Identity、Image、Placement、Compute の管理部分、Networking の管理部分と各エージェント、Dashboard。SQL データベース・メッセージキュー・NTP も。ネットワークインターフェースは最低2つ

SAMPLE

ノード	例の構成で載せる物
コンピュート	ハイパーバイザーと nova-compute、Networking のエージェント。2台目以降も足せる
ブロックストレージ（任意）	Cinder のボリュームを置く。台数も増やせる
オブジェクトストレージ（任意）	Swift。2台以上

例の構成には専用のネットワークノードがありません。L3 や DHCP のエージェントもコントローラーに載ります。ガイドは、本番の環境では Ansible、Chef、Puppet、Salt のような配備の道具で、展開と管理を自動化するよう勧めています。[^f6m-1]

配備の道具の1つである Kolla-Ansible は、コンテナ化したサービスを Ansible で展開し、同じ手順で何台にも同じ環境を作れます。[^c1-27][^c1-41] 手動の弱点である手順の多さと再現しにくさを、手順の自動化で補う道具。

構成	ノード数	使う道具の例	用途
オールインワン	1	DevStack、`packstack --allinone`	開発、テスト、学習
マルチノード	複数	手動、Packstack の回答ファイル、Kolla-Ansible	検証、本番

インストールガイドの構成例は、ノードごとに動かすものが決まっています。

SAMPLE

ノード	動かすもの
コントローラー	Identity、Image、Placement、Compute と Networking の管理部分、Networking のエージェント、Dashboard。SQL データベース、メッセージキュー、NTP も載る
コンピュート	インスタンスを動かすハイパーバイザー部分。Compute の既定は KVM。複数台置ける
ブロックストレージ	任意。Block Storage などがインスタンスに渡すディスクを持つ

インスタンス用のディスクを持つのはブロックストレージノード。KVM が動くのは、コンピュートノードです。ブロックストレージのノードは置かなくても例の構成は成り立つので、置くことが求められているわけではありません。[^f6g1-10]

この例の構成は、最小の本番構成と2点で違います。

- Networking のエージェントがコントローラーノードにあり、専用のネットワークノードが無い
- セルフサービスネットワークのオーバーレイ（トンネル）通信が、専用のネットワークを使わず、管理ネットワークを通る

ブロックストレージとコンピュートの通信も、簡単のために管理ネットワークを使っています。本番では、性能と安全性を上げるために、ストレージ用の別のネットワークを置くのが勧めです。[^f6g1-10]

Kolla-Ansible の Ussuri のクイックスタートは、ベアメタルのサーバーか仮想マシンに、Ansible で OpenStack を配備する手順です。手順を読む前に、Ansible と Docker の基本を学んでおくと進めやすい、と案内しています。

SAMPLE

項目	内容
Ansible の版	2.8 以上、2.9 まで。pip では `pip install 'ansible<2.10'` と指定する
inventory	all-in-one（1台用）と multinode（複数台用）の2種類が付属する

最新版の Ansible をそのまま入れて 2.10 を超えたときは、`ansible<2.10` で入れ直します。inventory は、台数に合う方を編集して使います。[^f6g1-11]

SAMPLE

サンプルはここまでです

続き(第2章〜巻末)は、OPCEL 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

OPCEL認定試験 学習用テキスト(無料サンプル)

版

2026年11月版(LPI-Japan公式 OPCEL認定試験(出題範囲 Ver.3.0・Queens〜Ussuri) 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

OPCEL、LinuC は LPI-Japan の商標または登録商標です。OpenStack は OpenStack Foundation(現 OpenInfra Foundation)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の内容を基に作成したもので、LPI-Japan 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。

SAMPLE