IT資格道場

GCP PCA 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)

Google Cloud 公式 Professional Cloud Architect Exam Guide（2026-09-07 取得）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

3点セットの使い方

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。

はじめに ── この教材の使い方

このテキストは、Google Cloud が実施する **Professional Cloud Architect 認定** の合格を目的に作られています。

Professional 級の試験が問うのは、個々の製品の説明ではありません。**要件と制約を読み取り、複数の選択肢の中から最も適切な設計・運用・トラブルシューティングの判断を選べること**です。本書は Exam guide の各セクションに沿って、「どの場面でどれを選ぶか・なぜ他は不適か」を本文に書き込んでいます。

本書は Google Cloud の公式教材ではなく、IT資格道場が独自に書き下ろしたオリジナルの教材です。実際に出題された問題を再現したものではありません。

ケーススタディについて: Google Cloud は公式ケーススタディ4本（Altostrat Media／Cymbal Retail／EHR Healthcare／KnightMotives Automotive）を公開し、本番の各回はそのうち2本が出題され、ケーススタディの設問が全体の 20〜30% を占めると案内しています。本書は各事例の要件と設計判断の対応を付章にまとめ、当サイトの事例演習（長文シナリオ＋4択）で練習できるようにしています。本番の画面や設問そのものの再現ではありません。

試験の基本情報

項目	内容
試験名	Google Cloud Certified – Professional Cloud Architect
実施	Google Cloud（テストセンター、またはオンライン監督試験）
問題数	**50〜60問**（公式の案内）
試験時間	**120分**
出題形式	**択一（multiple choice）、複数選択（multiple select）、ケーススタディ（case study）**
合格ライン	非公表
受験言語	英語・日本語（日本語提供あり）

出題数・試験時間・試験ガイドは改定されることがあります。 お申し込みの前に、Google Cloud 公式サイトで最新の情報をご確認ください。

出題範囲の全体像 —— 6つのセクションと本書の章

セクション	分野	採点対象に占める割合（公式）	本書
1	クラウドソリューションアーキテクチャの設計と計画	**25%**	第1章
2	クラウドソリューションインフラの管理とプロビジョニング	**17.5%**	第2章
3	セキュリティとコンプライアンスの設計	**17.5%**	第3章
4	技術プロセスとビジネスプロセスの分析と最適化	**15%**	第4章
5	実装の管理	**12.5%**	第5章
6	ソリューションと運用のエクセレンスの確保	**12.5%**	第6章

本書の章の並びは、Exam guide の並びとまったく同じです。節見出しの末尾にある `[1.1]` のような番号は、Exam guide のセクション番号です。Google Cloud はセクションごとの割合だけを公表しており、細目単位の出題数は公表していません。本書もそれ以上の数字は書きません。

サービス名・機能名は英語のまま覚えてください。本文では、製品名・機能名・用語を英語表記のまま書いています。本番の設問文と選択肢に出るのはこの表記であり、日本語訳だけを覚えても画面では出会いません。

全設問に効く判断軸 ——「3つの問い」

問い	何を切り分けるか	分かれ方
問い1: 要件は何を最優先しているか	可用性／性能／費用／セキュリティ／運用負荷	同じ「移行したい」でも、何を最優先に置くかで正解の製品と構成が変わります
問い2: マネージドで足りるか、自分で管理するか	サーバーレス／マネージド／セルフマネージド	「運用したくない」ならマネージド、「細かな制御が要る」ならセルフマネージド、と制約が構成を決めます
問い3: それは Google の責任か、自分の責任か	責任共有モデル	どの層まで自分で守り・運用するかは全章で一貫して効きます。迷ったらここへ戻ってください

第1章 クラウドソリューションアーキテクチャの設計と計画（ドメイン1・採点対象の 25%）

本章は Professional Cloud Architect の中で最も比重の大きい領域です。個々の製品名を覚えているかではなく、与えられたビジネス要件と技術要件から「どの構成を選び、なぜ他を選ばないか」を言えるかが問われます。試験の設問は、ケーススタディ（Altostrat Media・Cymbal Retail・EHR Healthcare・KnightMotives Automotive）や短い場面設定の形で要件を並べ、その中の一語（可用性の数値・データ所在地・予算・移行期限など）が正解を一意に決める作りになっています。読むときは「制約を決めている一語はどれか」を探す癖をつけてください。

1-1 ビジネス要件を満たすクラウドソリューションインフラの設計 [1.1]

1-1-1 Business use cases and product strategy —— 何のために作るのかを先に固定する [1.1]

アーキテクトの最初の仕事は、技術選定ではなく Business use cases（そのシステムが誰のどんな業務を成立させるのか）と product strategy（その事業がどこで勝とうとしているのか）の把握です。ここがずれると、以降の判断がすべてずれます。

場面で考えます。Cymbal Retail のようなオンライン小売が「商品カタログの属性生成を自動化したい」と言うとき、product strategy は「品揃えの広さで勝つ」ことです。したがってカタログ登録のリードタイム短縮が最優先で、生成品質の最後の数%を人手で詰めるより、Human-in-the-Loop（HITL）レビュ

ー付きで早く回す設計が正解になります。一方 EHR Healthcare のような医療 SaaS の product strategy は「規制準拠を保ったまま医療機関を素早くオンボードする」ことなので、同じ「早く回す」でも、監査証跡を落とす短縮策は選べません。

試験での使い方は次のとおりです。選択肢が技術的にはどれも動く場合、business use case に照らして「事業として意味のある方」を選びます。逆に、技術的に優れていても事業の勝ち筋と無関係なもの（例: 小売のカタログ改善の場面で、いきなり自前 LLM のフルスクラッチ学習を提案する選択肢）は誤答です。

判断の軸	正解に寄る選択肢	誤答になりやすい選択肢
事業の勝ち筋に直結するか	収益・顧客体験・規制遵守に効く構成	技術的に高度だが事業指標に効かない構成
意思決定の可逆性	後から変えられる構成を先に取る	初手で不可逆な巨大投資をする
時間軸	段階的に価値を出す	全面刷新が完了するまで何も出さない

1-1-2 Identifying functional and non-functional requirements —— 機能要件と非機能要件を分ける [1.1]

Functional requirements（機能要件）は「システムが何をするか」、non-functional requirements（非機能要件）は「どの程度の品質で行うか」です。Professional 級の設問は、ほぼ必ず non-functional requirements の側に正解の鍵を置きます。

非機能要件は次の観点で拾います。

観点	要件文に現れる語	設計に効く形
可用性	99.9%、24/7、無停止	ゾーン冗長かリージョン冗長か、SLA の積み上げ
性能・遅延	ミリ秒、リアルタイム、Performance and latency	リージョン配置、キャッシュ、Global Load Balancer
拡張性	急成長、スパイク、Scalability	オートスケール、サーバーレス、シャーディング
耐久性	データを失えない	レプリケーション、バックアップ、保持期間
保守性	少人数運用	マネージドサービス優先
規制	HIPAA、GDPR、データ所在地	リージョン制約、暗号化、監査ログ
コスト	予算、OpEx 削減	Cost optimization の各手法

よく出る取り違えは、可用性の数値を「どこまで冗長化するか」に翻訳しないことです。EHR Healthcare の「minimum 99.9% availability for all customer-facing systems」は、単一ゾーンの構成では満たせません。regional な Managed Instance Group または regional GKE cluster、そして複数ゾーンにまたがるロードバランサが必要になります。逆に「99.999%」や「リージョン障害でも継続」と書かれていれば、multi-region の構成（Spanner の multi-region 構成、Cloud Storage の multi-region バケット、複数リージョンのバックエンドを持つ Global External Application Load Balancer）まで踏み込みます。

もう一つの取り違えは、SLA と SLO と SLI の混同です。SLI（Service Level Indicator）は測る値そのもの、SLO（Service Level Objective）は自分たちが

守ると決めた目標値、SLA（Service Level Agreement）は顧客との契約で違反時に補償が発生するものです。設問が「社内で運用の判断基準にしたい」と言えば SLO、「契約上の保証」と言えば SLA です。SLO には error budget（許容される逸脱量）が伴い、これを使い切ったらリリースを止めて信頼性作業に回す、という運用判断につながります。

1-1-3 Business continuity plan —— 事業継続計画を RPO と RTO で表す [1.1]

Business continuity plan（BCP）は「重大な障害が起きても事業を続けるための計画」です。技術に落とすときは、必ず二つの数値に翻訳します。

- RPO（Recovery Point Objective）: どこまでのデータ損失を許容するか。バックアップ頻度・レプリケーション方式を決める。
- RTO（Recovery Time Objective）: どれだけの時間で復旧するか。待機構成のレベルを決める。

待機構成	概要	RTO の目安	コスト
Backup and restore	バックアップだけ保持し、被災後に環境を作る	時間〜日	最小
Cold standby	環境定義（Terraform）だけ用意し、平時は動かさない	時間	小
Warm standby（pilot light）	縮小した構成を常時稼働させ、被災時に拡大	分	中
Hot standby / active-active	両リージョンで本番トラフィックを処理	秒〜ゼロ	大

RPO をほぼゼロにしたいなら、同期レプリケーションを持つ製品（Cloud SQL の高可用性構成は同一リージョン内、Spanner は multi-region で強整合）を選びます。「リージョン障害でも RPO=0、RTO≒0」と書かれたら Spanner が正解に寄り、Cloud SQL の cross-region read replica（非同期＝RPO>0、昇格に時間がかかる＝RTO>0）は誤答になります。

BCP は技術だけではありません。誰が復旧を宣言するか（意思決定の権限）、連絡経路、DR 演習の頻度も計画に含めます。「手順書はあるが一度も試していない」は BCP として不十分で、定期的な failover テストの実施が推奨事項です。

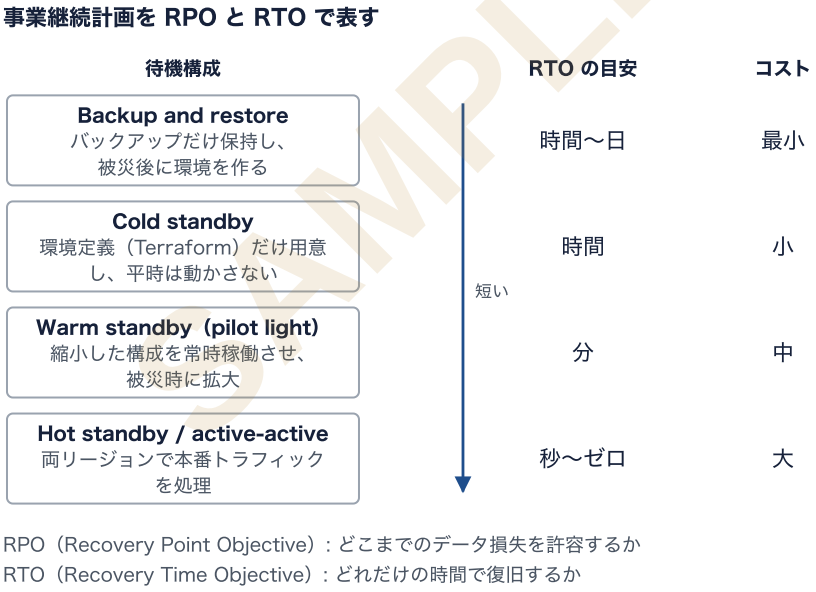

図 1-1　事業継続計画を RPO と RTO で表す

1-1-4 Cost optimization —— 費用最適化の手札を並べる [1.1]

Cost optimization は Altostrat Media（ストレージ費用の最適化）でも KnightMotives（新技術投資の原資づくり）でも中心の要件です。手札を整理します。

手法	効く場面	注意点
Committed use discounts（CUD）	1年・3年ずっと動く基盤	使わなくても課金される。resource-based と spend-based がある
Sustained use discounts	Compute Engine を月内で長時間動かす	自動適用。申し込み不要
Spot VMs	中断可能なバッチ、レンダリング、CI	24時間以内に停止しうる。ステートを外に出す
Custom machine types	vCPU とメモリの比率が標準型と合わない	過剰なサイズ指定を避ける
Autoscaling / サーバーレス	トラフィックの波が大きい	最小インスタンス数の設定でコールドスタートと費用を調整
Cloud Storage のクラス分け	アクセス頻度が下がるデータ	早期削除の最低保存期間に注意
BigQuery の課金モデル選択	分析基盤	on-demand と capacity（スロット予約）の切り替え
リソースのラベル付けと予算アラート	全社	Billing export to BigQuery で分析

Cloud Storage のクラスは頻出です。

クラス	想定アクセス頻度	最低保存期間	主な用途
Standard	高頻度	なし	配信中のメディア、ホットデータ
Nearline	月1回程度	30日	直近のバックアップ
Coldline	四半期に1回程度	90日	過去データ
Archive	年1回未満	365日	長期保管、法定保存

Altostrat Media の「Optimize cloud storage costs while maintaining high availability and scalability for media content」に対する正解の形は、Object Lifecycle Management のルールで一定日数経過後に Nearline・Coldline・Archive へ自動遷移させ、アクセスパターンが読めない部分は Autoclass に任せる、という組み合わせです。誤答として出やすいのは「全部 Archive にする」（配信中コンテンツの取り出し費用と最低保存期間で逆に高くつく）と「バケットを毎回手で作り替える」（運用負荷）です。

費用最適化を語るときは、単価だけでなく総所有コストで見ます。マネージドサービスは単価が高く見えても、運用人員・パッチ適用・障害対応を含めると安いことが多く、EHR Healthcare の「Decrease infrastructure administration costs」のような要件では、セルフマネージドを選ぶ選択肢はまず誤答です。

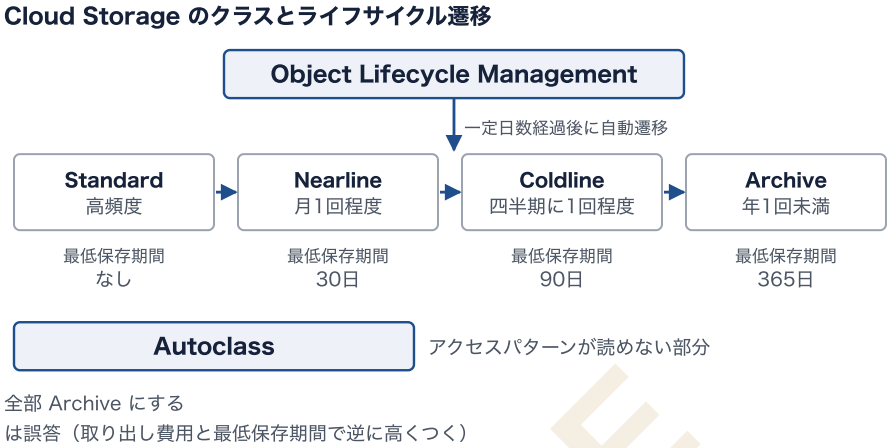

図 1-2　Cloud Storage のクラスとライフサイクル遷移

1-1-5 Supporting the application design —— アプリ設計を支えるインフラを選ぶ [1.1]

Supporting the application design とは、アプリケーションの作りに合ったインフラを与えることです。逆に言えば、アプリの作りを無視した「良いインフラ」は誤答になります。

アプリの作り	適したプラットフォーム	理由
ステートレスな HTTP サービス、コンテナ化済み	Cloud Run	リクエスト単位のスケール、ゼロスケール、運用がほぼ不要
多数のマイクロサービス、細かい制御が必要	GKE	サービスメッシュ、カスタムスケジューリング、DaemonSet
イベント駆動の小さな処理	Cloud Run functions	Pub/Sub や Cloud Storage の通知で起動

アプリの作り	適したプラットフォーム	理由
OS レベルの依存、ライセンス、レガシー	Compute Engine	完全な制御。sole-tenant nodes でライセンス要件にも対応
VMware のまま持っていきたい	Google Cloud VMware Engine	再構築せずに移設できる

ステートの置き場所を分離することが設計の鍵です。セッション情報を VM のローカルディスクに置いていると、オートスケールもゾーン障害対応も成り立ちません。Memorystore（Redis）や Firestore に外出しし、コンピュートは使い捨てにできる状態（cattle であって pet ではない状態）にします。

Cymbal Retail の既存環境には MySQL、Microsoft SQL Server、Redis、MongoDB が混在しています。この場合の移行先の対応は、MySQL と SQL Server は Cloud SQL、Redis は Memorystore、MongoDB は Firestore もしくは MongoDB Atlas（Google Cloud 上のパートナーサービス）です。ここで「全部 Spanner に統合する」という選択肢は、アプリの改修量を無視しているため誤答になりやすい形です。

1-1-6 Integration patterns with external systems —— 外部システムとの統合パターン [1.1]

Integration patterns with external systems は、EHR Healthcare の「legacy file- and API-based integrations with insurance providers」や Cymbal Retail の「SFTP file transfers, ETL batch processing」のように、必ず一つのケーススタディに出てきます。

パターン	使う製品	向く場面
同期 API 呼び出し	Apigee、Cloud Endpoints、Application Load Balancer	即時応答が要る。認可・レート制限・課金を挟みたい
非同期メッセージング	Pub/Sub	送り手と受け手を疎結合にする。スパイクの吸収
ファイル連携	Cloud Storage + Storage Transfer Service、SFTP を Cloud Storage へ	既存のバッチ連携をそのまま生かす
イベント連携	Eventarc	サービス間のイベントを標準形式で配る
ワークフロー	Workflows、Cloud Composer	複数サービスをまたぐ手順の制御
データ統合 ETL/ELT	Dataflow、Datastream、BigQuery Data Transfer Service	継続的なデータ取り込み

Apigee は API management の中心製品です。外部の保険会社やディーラーに API を出すとき、認証（API キー、OAuth 2.0、JWT 検証）、レート制限（quota、spike arrest）、バージョニング、開発者ポータル、分析を一箇所で持てます。「パートナーごとに利用量を計測して課金したい」「公開 API に一貫したセキュリティポリシーを当てたい」と書かれたら Apigee です。単に負荷分散したいだけなら Load Balancer で足り、Apigee は過剰です。

疎結合の判断はこう切ります。相手システムが落ちても自分は受け付け続けたい、相手の処理速度に引きずられたくない、再処理したい —— これらが要件なら Pub/Sub を挟みます。逆に「呼び出し元にその場で結果を返す必要がある」なら同期 API です。Pub/Sub は at-least-once 配信が基本なので、受け

手を冪等に作る必要があります。厳密に一度だけ処理したい場合は Pub/Sub の exactly-once delivery を有効にするか、Dataflow で重複排除します。

1-1-7 Movement of data —— データの移動を設計する [1.1]

Movement of data は「どれだけの量を、どこからどこへ、どの期限で、どの回線で運ぶか」の問題です。

手段	対象	目安
gcloud storage / gsutil	少量、都度	数十 GB
Storage Transfer Service	他クラウド（S3、Azure Blob）、オンプレ、URL リスト。定期実行可	TB 級
Transfer Appliance	回線が細い、期限が短い	数百 TB〜PB
BigQuery Data Transfer Service	SaaS・他 DWH から BigQuery へ定期取り込み	分析データ
Datastream	データベースの変更データキャプチャ（CDC）	ほぼリアルタイム同期
Database Migration Service	MySQL、PostgreSQL、SQL Server、Oracle の移行	最小停止時間の移行
Dataflow	変換を伴うストリーム・バッチ	Apache Beam

回線の見積もりは試験で問われます。必要日数は「データ量 ÷ 実効帯域」で概算し、期限に間に合わなければ Transfer Appliance か、回線増強（Dedicated Interconnect）を選びます。「10 PB を 1 か月で」という設問で、

既存回線が 1 Gbps なら物理的に不可能なので Transfer Appliance が正解です。

データ移動には方向による課金の非対称性があります。Google Cloud への流入（ingress）は原則無料、外への流出（egress）とリージョン間・ゾーン間の転送には料金がかかります。設計上は「データの近くで計算する」のが原則で、BigQuery のデータを毎回オンプレに引き出して集計する構成は、性能でもコストでも誤りです。

1-1-8 Design decision trade-offs —— 設計判断のトレードオフを言語化する [1.1]

Design decision trade-offs は、Professional 級の中心です。すべてを最大化する構成は存在せず、必ず何かを差し出します。

対立軸	一方を取ると	典型的な決め手
強整合 vs 低遅延・低コスト	Spanner の multi-region は強整合だが高価	「金額の不整合が許されない」なら強整合
マネージド vs 制御	Cloud Run は運用が軽いが、実行環境の自由度は低い	「少人数運用」「OS 依存あり」のどちらが書かれているか
単一リージョン vs マルチリージョン	可用性は上がるが、コストとレイテンシと整合性の難度が上がる	要求 SLA とデータ所在地
リフト&シフト vs リファクタ	早いが最適化されない／遅いが恩恵が大きい	移行期限（データセンターのリース満了）の有無
内製 vs 既製（build vs buy）	差別化領域は内製、非差別化はマネージド	事業の勝ち筋にあるか

対立軸	一方を取ると	典型的な決め手
密結合 vs 疎結合	疎結合は障害を局所化するが、結果整合と運用の複雑さを招く	相手の可用性に引きずられてよいか

回答の作法として、まず「捨ててよい要件」を特定します。EHR Healthcare のデータセンターのリース満了が迫っているという記述は、「時間」が最も希少な資源であることを示しています。この場合、まずリフト&シフト（あるいはコンテナ化済みの部分だけ GKE へ）で期限を守り、最適化は移行後に段階的に行う、という順序が正解です。逆に期限の記述がなく「運用負荷を根本から下げたい」だけなら、時間をかけたマネージド化が正解に寄ります。

1-1-9 Workload disposition strategies —— build、buy、modify、deprecate [1.1]

Workload disposition strategies は、既存の各ワークロードに対して「作る（build）・買う（buy）・直す（modify）・捨てる（deprecate）」のどれを当てるかの判断です。移行の文脈では次の分類とほぼ重なります。

戦略	別名	内容	選ぶ条件
Rehost	lift and shift	VM をそのまま移す	期限が厳しい、改修予算がない
Replatform	lift and optimize / modify	一部をマネージドに置換（自前 MySQL → Cloud SQL）	少ない改修で大きな運用削減
Refactor	re-architect / build	作り直す（モノリス → コンテナ、サーバーレス）	差別化領域、拡張性の限界

戦略	別名	内容	選ぶ条件
Repurchase	buy	SaaS に置き換える	非差別化領域（人事、経費、CRM）
Retire	deprecate	廃止する	利用者がいない、重複機能
Retain	—	当面残す	規制・依存・移行不能

EHR Healthcare の「legacy file- and API-based integrations with insurance providers ... There is no plan to upgrade or move these systems at the current time」は Retain の明示です。したがって設問で「これらのレガシーをどうするか」と聞かれたら、移行や作り直しではなく、ハイブリッド接続（Cloud Interconnect / Cloud VPN）で当面つなぐのが正解です。KnightMotives の老朽化したメインフレームと ERP は、業務を止められないので段階的な replatform か、周辺から API 化していく strangler fig パターンが妥当です。

Cymbal Retail の call center agents による手作業の受注入力は、Conversational Commerce の導入で置き換えられるため deprecate（あるいは業務の再配置）の候補です。「人員コスト削減」という business requirement がその根拠になります。

1-1-10 Success measurements —— KPI、ROI、metrics で成功を定義する [1.1]

Success measurements（KPI、ROI、metrics）を先に決めておかないと、移行が成功したかどうかを誰も判定できません。

指標の種類	例	誰が見るか
Business KPI	売上、コンバージョン率、解約率、コールセンター応答時間、新規保険会社のオンボード日数	経営
Technical metrics	レイテンシ（p50/p95/p99）、エラー率、スループット、可用性、デプロイ頻度、変更失敗率、MTTR	開発・運用
Financial	ROI（投資回収率）、TCO、単位あたりコスト（1リクエストあたり、1ユーザーあたり）	財務

ROI（return on investment）は「（得られた利益 − 投資額）÷ 投資額」で表され、移行の稟議を通す言語です。試験では「経営層への報告に適した指標はどれか」という形で、technical metrics ではなく business KPI や ROI を選ばせる設問が出ます。逆に「SRE チームがリリース判断に使う指標」なら SLO と error budget です。

DevOps の 4 指標（DORA metrics: deployment frequency、lead time for changes 、change failure rate 、time to restore service ）は 、EHR Healthcare の「roll out new continuous deployment capabilities to update their software at a fast pace」のような要件の達成度を測るのに適します。

KPI は測定可能でなければ意味がありません。定義した KPI は Cloud Monitoring のカスタム指標や BigQuery のダッシュボードとして実装し、ベースライン（移行前の値）を必ず取ってから移行します。ベースラインがないと改善量を主張できません。

1-1-11 Security and compliance —— 要件段階での組み込み [1.1]

Security and compliance は後付けできません。要件定義の段階で、次を確定させます。

- データの分類（公開・社内・機密・規制対象）と、分類ごとの取り扱い規則
- 適用される規制（HIPAA、GDPR、PCI DSS など）とデータ所在地の制約
- 認証・認可の方式（既存の Active Directory や外部 IdP との連携方法）
- 暗号化の要件（保存時・転送時、鍵を誰が持つか）
- 監査の要件（誰が何をしたかを何年保持するか）

EHR Healthcare は医療データを扱うため、リージョン選択、暗号化、監査ログの保持、アクセス制御の粒度がすべて要件から導かれます。KnightMotives は EU のデータ保護規制への準拠が明記されており、EU 域内リージョンへのデータ配置と、Organization Policy による resource location の強制が設計に入ります。詳細は第3章で扱います。

設計段階の原則は least privilege（最小権限）と defense in depth（多層防御）です。「一つの防御が破られても次がある」構成、すなわち IAM の絞り込みに加えて VPC Service Controls の境界、Organization Policy の制約、監査ログの三重を置きます。

1-1-12 Observability —— 要件としての可観測性 [1.1]

Observability（可観測性）は、logs・metrics・traces の三本柱で「システムの内部状態を外から推測できる」状態を指します。設計段階で決めるのは、何を出すか（計装）、どこに集めるか（集約先）、どれだけ保持するか（保持期間）、誰に知らせるか（アラート経路）です。

EHR Healthcare の「Monitoring is currently being done via various open source tools. Alerts are sent via email and are often ignored」は、二つの欠陥を示しています。ツールが分散していること（centralized visibility がない）と、アラートが無視されていること（アラート疲れ）です。したがって正解の方向は、Cloud Monitoring と Cloud Logging に集約し、SLO ベースのアラートに絞り、通知を Pub/Sub や PagerDuty など実際に見られる経路へ出すことです。Altostrat Media も同様に Cloud Monitoring と Prometheus（Google Cloud Managed Service for Prometheus）が混在し、メール通知に依存しています。

観測対象は組織の階層に合わせて設計します。複数プロジェクトを横断して見たいなら、metrics scope に対象プロジェクトを追加した scoping project を作ります。ログは organization レベルの aggregated sink で一括して集約先（Cloud Storage、BigQuery、Pub/Sub、別プロジェクトの Log Bucket）へ流します。詳細は第6章で扱います。

1-2 技術要件を満たすクラウドソリューションインフラの設計 [1.2]

1-2-1 Familiarity with the Google Cloud Well-Architected Framework —— 5つの柱 [1.2]

Google Cloud Well-Architected Framework は、設計の良し悪しを判断するための共通言語です。5 つの柱で構成されます。

柱	問い	代表的な実装
Operational excellence	運用し、改善し続けられるか	IaC、CI/CD、SLO、ポストモーテム、Cloud Monitoring

柱	問い	代表的な実装
Security, privacy, and compliance	守れているか	IAM、暗号化、VPC Service Controls、監査ログ
Reliability	落ちないか、落ちても戻るか	冗長化、グレースフルデグラデーション、DR 演習
Cost optimization	無駄がないか	CUD、Spot VMs、ライフサイクル管理、可視化
Performance optimization	速いか、伸びるか	適切なマシンタイプ、キャッシュ、リージョン配置

Framework には AI and ML perspective の視点も加わっており、モデルのライフサイクル管理、責任ある AI、データガバナンスを同じ 5 つの柱に当てはめて考えます。

試験での使い方は、選択肢の優劣を柱で説明できるかです。たとえば「単一の巨大 VM に全部載せる」構成は Reliability（単一障害点）と Operational excellence（デプロイのたびに全停止）で劣ります。「本番と開発を同じプロジェクトに同居させる」構成は Security（権限分離の失敗）で劣ります。柱どうしが衝突する場合（Reliability を上げると Cost が上がる）は、ビジネス要件のどちらが明示されているかで決めます。

1-2-2 High availability and fail-over design —— 高可用性とフェイルオーバー [1.2]

High availability の設計は、障害の単位（ドメイン）を意識するところから始めます。

障害ドメイン	対策	代表製品の挙動
インスタンス単位	複数インスタンス＋ヘルスチェック	Managed Instance Group の autohealing
ゾーン単位	複数ゾーンに分散	regional MIG、regional GKE cluster、Cloud SQL の HA 構成（同期スタンバイを別ゾーンに）
リージョン単位	複数リージョンに分散	multi-region の Cloud Storage、Spanner multi-region、複数リージョンのバックエンドを持つグローバル LB
依存サービス単位	タイムアウト、リトライ、サーキットブレーカー、graceful degradation	部分機能を落として全体を守る

Fail-over の方式は、自動か手動か、そして切り替え時にデータをどう保つかで分かれます。

- Cloud SQL の high availability configuration: 同一リージョンの別ゾーンに同期スタンバイを持ち、自動フェイルオーバーします。リージョン障害には対応しません。
- Cloud SQL の cross-region read replica: 非同期のため RPO > 0。手動で昇格（promote）します。DR 用です。
- Spanner: multi-region 構成なら、リージョン障害でも強整合を保ったまま継続します。ただし書き込みレイテンシは上がります。
- Bigtable: replication を有効にして複数クラスタを持ち、アプリケーションプロファイルで single-cluster routing（整合性優先）か multi-cluster routing（可用性優先）を選びます。

- Filestore: エンタープライズティアはリージョン内の冗長性を持ちます。
- Memorystore for Redis: Standard tier がレプリカを持ち自動フェイルオーバーします。

ヘルスチェックの設計も問われます。ロードバランサのヘルスチェックは、単に TCP ポートが開いているかではなく、依存先（データベース、外部 API）まで含めた「実際にリクエストを処理できるか」を返すエンドポイントにします。ただし依存先の一時的な不調で全インスタンスが同時に unhealthy になると、かえって全断を招くため、readiness と liveness を分けます。

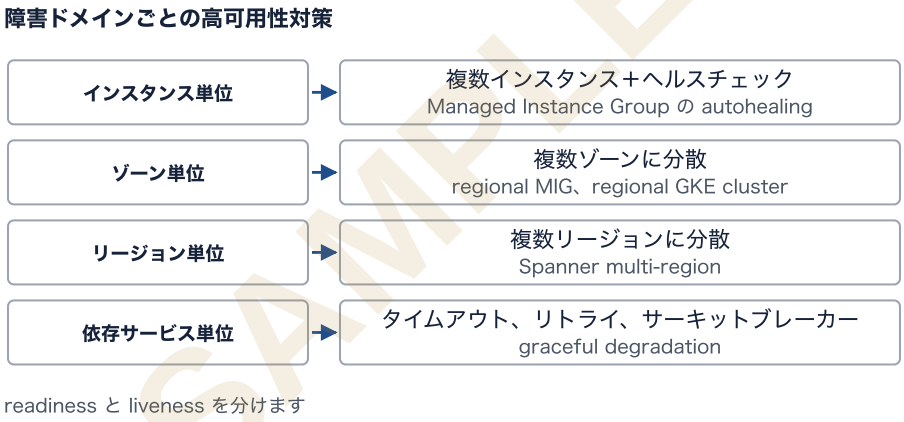

図 1-3　障害ドメインごとの高可用性対策

1-2-3 Flexibility of cloud resources —— リソースの柔軟性 [1.2]

Flexibility of cloud resources は、クラウドの利点である「必要な時に必要な量だけ」を設計に織り込むことです。

- 垂直方向: machine type の変更（停止して変更、または一部は稼働中に変更可）、custom machine types で vCPU とメモリを個別指定、GPU や Local SSD の追加。

- 水平方向: Managed Instance Group の autoscaling（CPU 使用率、ロードバランサの処理量、Cloud Monitoring のカスタム指標、スケジュールベース）、GKE の Horizontal Pod Autoscaler と Cluster Autoscaler、Cloud Run の同時実行数ベースの自動スケール。
- 時間方向: Spot VMs、preemptible な性質を前提としたバッチ設計、Dynamic Workload Scheduler による一括確保。

柔軟性を殺す設計を避けることが重要です。ライセンスが物理コアに紐づく、IP アドレスが固定である前提でコードが書かれている、セッションがインスタンスローカルに保存されている —— これらはいずれもスケールを妨げます。sole-tenant nodes は、ライセンス上どうしても物理サーバーを占有する必要がある場合の解ですが、柔軟性とコスト効率を犠牲にします。

1-2-4 Scalability to meet growth requirements —— 成長に耐える設計 [1.2]

Scalability の設問は「10 倍になったら壊れる箇所はどこか」を問います。よくあるボトルネックと対処は次のとおりです。

ボトルネック	症状	対処
リレーショナル DB の書き込み	単一プライマリの限界	読み取りは read replica へ、書き込みは Spanner へ移行、またはシャーディング
ホットスポット	特定キーに集中	Bigtable・Firestore で連番やタイムスタンプ先頭のキーを避け、ハッシュやフィールドの並べ替えでキーを分散
同期処理の連鎖	一箇所の遅延が全体に波及	Pub/Sub で非同期化、バックプレッシャー

ボトルネック	症状	対処
ファイル共有	NFS の帯域	Filestore の上位ティア、Cloud Storage への置き換え
割り当て（quota）	API の上限に到達	事前に quota 引き上げを申請、リトライに指数バックオフ

Cymbal Retail の「The solution must handle Cymbal's extensive product catalog and accommodate their anticipated growth without compromising performance」に対しては、商品検索を RDB の LIKE 検索から Vertex AI Search（Discovery AI）のようなマネージド検索へ移し、カタログデータの実体は BigQuery と Cloud Storage に置き、配信は Cloud Run でスケールさせる形が素直です。

キャパシティプランニングは、ピーク値ではなく成長率と季節変動で行います。ブラックフライデーのような予測可能なピークには、事前のスケジュールベースオートスケーリングと quota の引き上げ、予約（reservations）の確保を組み合わせます。

1-2-5 Performance and latency —— 性能と遅延 [1.2]

遅延は主に、物理距離・処理時間・待ち行列の三つから生まれます。

原因	対処
ユーザーとサーバーの距離	ユーザーに近いリージョンへ配置、Global External Application Load Balancer の Anycast IP、Cloud CDN
静的コンテンツの反復配信	Cloud CDN、Cloud Storage のキャッシュ制御

原因	対処
データベースの往復	Memorystore によるキャッシュ、接続プール、N+1 クエリの排除
サービス間の距離	同一リージョン・同一ゾーンへの配置、compact placement policy
分析クエリの遅さ	BigQuery のパーティション分割・クラスタリング、BI Engine
ストレージ I/O	SSD Persistent Disk、Hyperdisk、Local SSD、Managed Lustre

EHR Healthcare の「Reduce latency to all customers」は、グローバルな配置の問題です。正解は、複数リージョンにバックエンドを置いた Global External Application Load Balancer で最寄りに振り分け、静的資産は Cloud CDN、データベースは読み取りをリージョンごとの replica に寄せる、という組み合わせです。誤答として出やすいのは「単一リージョンのままインスタンスを増やす」（距離の問題は解けない）です。

測るときは平均ではなくパーセンタイル（p95、p99）で見ます。平均が良くても p99 が悪ければ、一部のユーザーは常に遅い体験をしています。Cloud Trace で分散トレースを取り、どのスパンが時間を食っているかを特定します。

1-2-6 Gemini Cloud Assist —— 設計と運用を支える AI コラボレーター [1.2]

Gemini Cloud Assist は、Google Cloud のコンソールに統合された AI 支援機能で、設計・運用・費用の各局面を横断します。試験では「どの場面でこれを使うか」が問われます。

機能	できること
Investigations	ログ・指標・構成を横断して分析し、複数の仮説を並行検証して根本原因の候補を提示する
Application Design Center	技術要件とビジネス要件を自然言語で伝えると、アーキテクチャ案を生成し、編集してデプロイまでつなげる
Cost optimization	費用を自然言語で問い合わせる、コスト異常の原因分析、データベースの最適化推奨
Troubleshooting	Cloud Run などでクラウドのシグナルからアプリケーションコードまで追跡、Cloud SQL・BigQuery・Spanner の診断

利用の入口は、Google Cloud コンソールとモバイルアプリの Cloud Assist パネル、コンソール各所の quick prompts、そして Cloud Hub・Database Center・Application Design Center といったハブです。Slack や ServiceNow との連携もあります。

設問での切り分けはこうです。「障害の根本原因を素早く絞り込みたい」なら Investigations、「新しいアーキテクチャの草案を作りたい」なら Application Design Center、「先月から費用が跳ねた理由を知りたい」なら Cost optimization の機能です。Gemini Cloud Assist は判断を代行するものではなく、提示された案はアーキテクトが検証してから採用します。

1-2-7 Backup and recovery —— バックアップと復旧 [1.2]

Backup and recovery は BCP の実装面です。製品ごとに手段が異なります。

対象	手段	要点
Compute Engine の VM	Persistent Disk snapshot、snapshot schedule、machine image	スナップショットは増分。別リージョンへの保存を指定できる
複数リソースの一括保護	Backup and DR Service	VM、Cloud SQL、VMware Engine などをポリシーで一元管理し、バックアップボールトで保護
Cloud SQL	自動バックアップ＋point-in-time recovery（PITR）	PITR にはバイナリログ（または WAL）の保持が必要
Cloud Storage	Object Versioning、soft delete、bucket lock（retention policy）	誤削除・ランサムウェア対策には versioning と retention policy
Firestore	マネージドの export/import、PITR	エクスポート先は Cloud Storage
BigQuery	time travel（既定 7 日）、table snapshot、table clone	誤 UPDATE の巻き戻しは time travel
Spanner	backup、export	バックアップはインスタンス内、export は Cloud Storage へ
GKE	Backup for GKE	クラスタ構成と Persistent Volume の両方を保護

三つの原則を押さえます。第一に、バックアップは本番と障害ドメインを分ける（別リージョン、別プロジェクト、可能なら別の権限主体）。第二に、復元を定期的にテストする（取れているが戻せないバックアップは無価値）。第三に、保持期間を規制と業務から決め、Object Lifecycle Management や retention policy で自動化する。

ランサムウェアや内部不正への備えとしては、バックアップの削除を防ぐ仕組み（bucket lock による retention policy、Backup and DR のバックアップボールト、削除権限の分離）が問われます。同じ管理者がバックアップも削除できる構成は、監査で指摘される典型です。

1-3 ネットワーク・ストレージ・コンピューティングリソースの設計 [1.3]

1-3-1 Integration with on-premises/multicloud environments —— ハイブリッドとマルチクラウド [1.3]

Integration with on-premises/multicloud environments は 、EHR Healthcare・Altostrat Media・KnightMotives のいずれにも登場します。接続手段の選択が最初の判断です。

手段	帯域	SLA	経路	向く場面
Cloud VPN（HA VPN）	1トンネルあたり最大 3 Gbps 程度	99.99%（HA VPN・2インターフェース構成）	インターネット経由・IPsec 暗号化	小規模、早く始めたい、バックアップ経路
Dedicated Interconnect	10 Gbps または 100 Gbps 単位	99.9% / 99.99%（冗長構成による）	Google との物理直結	大容量、安定した低遅延
Partner Interconnect	50 Mbps〜50 Gbps	99.9% / 99.99%	サービスプロバイダ経由	コロケーションが Google の PoP にない
Cross-Cloud Interconnect	10/100 Gbps	あり	Google と他クラウドの直結	マルチクラウド間の大容量転送

Interconnect は既定では暗号化されません。規制で経路上の暗号化が要る場合は、HA VPN over Cloud Interconnect または MACsec（Cloud Interconnect の暗号化機能）を併用します。EHR Healthcare の「secure and high-performance connection between on-premises systems and Google Cloud」は、まさにこの組み合わせが答えになります。

DNS の相互解決も設計項目です。オンプレから Google Cloud の内部名を引けるように inbound server policy を、Google Cloud からオンプレの名前を引けるように Cloud DNS の forwarding zone または outbound server policy を設定します。IP アドレス空間は重複しないように設計します（重複するとルーティングが成立せず、NAT が必要になります）。

マルチクラウドの選択肢としては、BigQuery Omni（AWS・Azure 上のデータをそのまま BigQuery のインターフェースで分析）、Anthos / GKE Enterprise（複数環境で一貫した Kubernetes 運用と Config Sync によるポリシー配布）があります。Altostrat Media の「Provide scalable, performant kubernetes environments both on-premises and in the cloud」と「Modernize CI/CD for containerized deployments with a centralized management platform」の二つを同時に満たすのが GKE Enterprise（fleet 管理・Config Sync・Policy Controller）です。

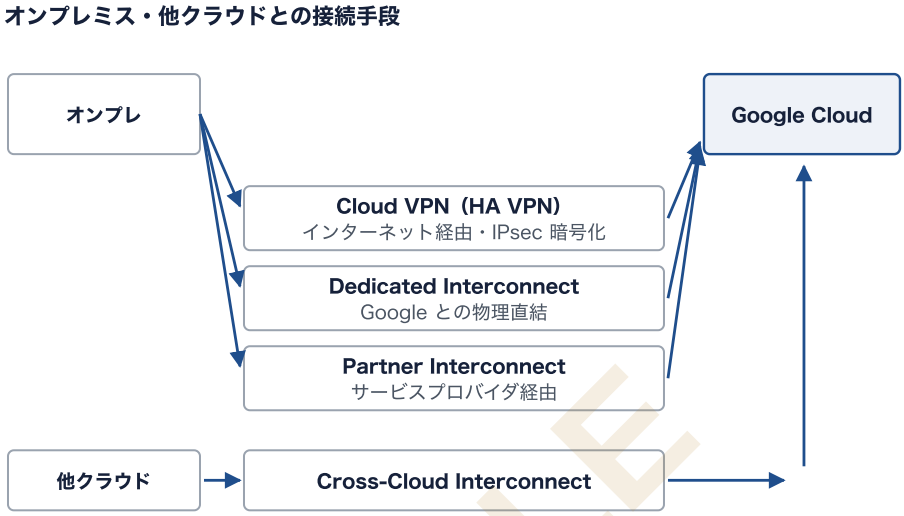

図 1-4　オンプレミス・他クラウドとの接続手段

1-3-2 Google Cloud AI and machine learning solutions —— AI の手札 [1.3]

Google Cloud AI and machine learning solutions は、Gemini LLMs、Agent Builder、Model Garden、Gemini models、AI Hypercomputer で構成されます。用途に応じた選び分けが問われます。

手段	内容	選ぶ場面
事前構築の API	Vision API、Speech-to-Text、Text-to-Speech、Translation、Natural Language、Video Intelligence、Document AI	汎用のタスク。学習データも ML の専門家も不要

手段	内容	選ぶ場面
Gemini models（Gemini LLMs）	マルチモーダルの大規模言語モデル。要約・抽出・生成・推論	自然言語の理解と生成、画像・音声・動画を含む入力
Model Garden	100 種類以上の基盤モデル（Google 製・サードパーティ・オープンモデル）のカタログ。探して試して、そのままデプロイやチューニングへ	「どのモデルを使うか」を比較検討したい
Agent Builder	エージェントを構築するための枠組み。データストアへのグラウンディング、ツール呼び出し、会話フローの設計	社内文書に基づく問い合わせ対応、業務エージェント
Gemini Enterprise Agent Platform	ML/生成 AI の統合プラットフォーム（旧 Vertex AI）。Pipelines、Feature Store、Model Registry、Workbench、Training、Prediction、Evaluation	自前のモデル開発・運用まで含む一気通貫
AI Hypercomputer	GPU・TPU・高速ネットワーク・Managed Lustre / Cloud Storage を統合した AI 向けスーパーコンピューティング基盤	大規模な学習、高スループット推論
BigQuery ML	SQL でモデルを作る。Gemini を SQL から呼ぶこともできる	データが BigQuery にあり、分析者が SQL しか使わない

判断の順序は「既製の API で足りるか → 基盤モデルをそのまま使えるか（プロンプトのみ） → グラウンディングや RAG で足りるか → チューニングが要るか → 自前学習が要るか」です。上に行くほど早く安く、下に行くほど自由度が高くなります。試験では、要件が汎用的なのに自前学習を選ぶ選択肢は誤答です。

Altostrat Media の要件との対応を示します。「Automatically generate concise summaries of media content」は Gemini models による要約、「Extract rich metadata from media assets using NLP and computer vision」は Video Intelligence API や Vision API と Gemini の組み合わせ、「Detect and filter inappropriate content」は Responsible AI のセーフティフィルタと Model Armor、「Develop advanced chatbots with natural language understanding」は Agent Builder、「Ensure that AI systems are auditable and their decisions can be explained」は Explainable AI と Model Registry によるモデル系統の記録が該当します。

Cymbal Retail では、商品属性の生成（Attribute Generation）が Gemini のマルチモーダル入力、画像バリエーション生成（Image Generation and Enhancement）が Imagen、自然言語による商品検索（Automate Product Discovery）が Vertex AI Search（Discovery AI）、会話型の接客が Agent Builder のエージェントに対応します。

1-3-3 Cloud-native networking —— VPC、peering、firewalls、load balancers、routing [1.3]

Cloud-native networking の要素を整理します。

VPC（virtual private cloud）は Google Cloud ではグローバルなリソースで、サブネットがリージョンに属します。したがって、同一 VPC 内であれば異なるリ

ージョンのリソースが内部 IP で通信できます（他クラウドとの大きな違いです）。サブネットの IP レンジは後から拡張できます。

概念	要点
Subnet	リージョン単位。プライマリレンジとセカンダリレンジ（GKE の Pod・Service 用）
Routing	システム生成ルート（サブネットルート、既定のインターネットゲートウェイ）とカスタムルート、Cloud Router による BGP の動的ルート
Firewall	VPC firewall rules（方向・優先度・ソース/ターゲットをタグやサービスアカウントで指定）と hierarchical firewall policies（組織・フォルダで一括適用）、network firewall policies
VPC Network Peering	二つの VPC を内部 IP で接続。推移的でない（A-B、B-C があっても A-C は通らない）。IP レンジの重複不可
Shared VPC	ホストプロジェクトがネットワークを持ち、サービスプロジェクトがそれを使う。ネットワーク管理を中央に集約しつつ、リソースの権限は分ける
Private Service Connect	消費者側の VPC に private endpoint を作り、Google API やパートナー・自社のサービスへ内部 IP で到達。IP レンジの重複や推移性の問題を回避できる
Private Google Access	外部 IP を持たない VM から Google API へ到達させる
Cloud NAT	外部 IP を持たない VM からの外向き通信

概念	要点
Cloud DNS	公開ゾーン・限定公開ゾーン・転送ゾーン・ピアリングゾーン
Network Connectivity Center	ハブアンドスポークで複数の VPC・オンプレ・他クラウドを一元的につなぐ

Shared VPC と VPC Network Peering の使い分けは頻出です。同一組織内で「ネットワークは中央のチームが管理し、各部門はサブネット単位で使う」なら Shared VPC。別組織や、独立して管理される VPC どうしを単純につなぐなら Peering。サービスの提供者と消費者という関係（自社の SaaS を他部門や他社に出す）なら Private Service Connect です。Private Service Connect は IP レンジが重複していても使え、公開範囲を絞れるため、近年は Peering の代替として推奨される場面が増えています。

ロードバランサの選択は次の表で切ります。

種類	レイヤ	スコープ	主な用途
Global External Application Load Balancer	L7（HTTP/HTTPS）	グローバル・Anycast IP	世界中のユーザー向け Web、Cloud CDN 併用、URL マップによる振り分け、Cloud Armor 適用
Regional External Application Load Balancer	L7	リージョン	データ所在地の要件でリージョン内に閉じたい
External proxy Network Load Balancer	L4（TCP/SSL）	グローバル/リージョン	HTTP 以外のプロキシ

種類	レイヤ	スコープ	主な用途
External passthrough Network Load Balancer	L4	リージョン	クライアント IP を保つ、UDP、非 HTTP
Internal Application Load Balancer	L7	リージョン/クロスリージョン	内部のマイクロサービス間
Internal passthrough Network Load Balancer	L4	リージョン	内部の TCP/UDP、ネクストホップとして使う

Container networking は GKE で扱います。VPC-native cluster（alias IP）が既定で、Pod と Service にサブネットのセカンダリレンジを割り当てます。これにより Pod の IP が VPC のルーティングに乗り、ファイアウォールルールや Private Service Connect と自然に噛み合います。Network Policy で Pod 間の通信を制御し、Gateway API または Ingress で外部公開します。GKE Dataplane V2（eBPF ベース）はネットワークポリシーのログ取得にも対応します。

1-3-4 Choosing data processing solutions —— データ処理基盤の選択 [1.3]

Choosing data processing solutions は、処理の形（バッチかストリームか）、開発言語、運用負荷で決めます。

製品	種類	選ぶ場面
Dataflow	Apache Beam のフルマネージド実行。ストリームとバッチを同じコードで	ストリーミング ETL、ウィンドウ集計、Pub/Sub → BigQuery
Dataproc	マネージド Hadoop / Spark	既存の Spark・Hive 資産をそのまま移したい。エフェメラルクラスタで費用削減
BigQuery	サーバーレス DWH。SQL での大規模分析、BigQuery ML、BI Engine	分析、レポート、機械学習の前処理
Pub/Sub	メッセージング	取り込み口、疎結合、スパイク吸収
Cloud Composer	マネージド Apache Airflow	複数ジョブの依存関係を持つワークフロー
Workflows	サーバーレスのオーケストレーション	API 呼び出しの手順制御。軽量
Dataform	BigQuery 内の SQL 変換の管理（ELT）	データモデリング、テスト、依存管理
Datastream	サーバーレス CDC	運用 DB の変更を BigQuery へ継続反映
Data Fusion	GUI ベースの ETL（CDAP）	コードを書かずにパイプラインを組みたい

判断の型は次のとおりです。既存の Spark ジョブがある → Dataproc。新規でストリーム処理を書く → Dataflow。SQL で完結する → BigQuery。ジョブの依存関係と再実行の管理が主題 → Cloud Composer。単純な API の連携 → Workflows。

Altostrat Media は BigQuery を主要な DWH として使い、Cloud Run functions でイベント駆動のトランスコードやメタデータ抽出を行っています。ここに「動画がアップロードされたら自動で要約とメタデータを付ける」を足すなら、Cloud Storage のオブジェクト通知 → Eventarc → Cloud Run（または Cloud Run functions）→ Gemini API → BigQuery、というイベント駆動の流れになります。

1-3-5 Choosing appropriate storage types —— object、file、databases [1.3]

Choosing appropriate storage types は、データの形とアクセスパターンで一意に決まります。

分類	製品	選ぶ条件
Object	Cloud Storage	非構造化データ、メディア、バックアップ、データレイク。POSIX は不要
File	Filestore	NFS が要る既存アプリ、共有ファイルシステム
File（AI/HPC）	Google Cloud Managed Lustre	高スループットの並列ファイルアクセス、大規模学習
Block	Persistent Disk、Hyperdisk、Local SSD	VM のディスク。Local SSD は揮発性だが最速
リレーショナル（垂直）	Cloud SQL（MySQL、PostgreSQL、SQL Server）	既存 RDB の移行、単一リージョンで足りる規模
リレーショナル（水平・グローバル）	Spanner	無停止のスケール、グローバルで強整合、高い可用性

分類	製品	選ぶ条件
PostgreSQL 互換の分析寄り	AlloyDB for PostgreSQL	PostgreSQL 互換のまま、分析クエリとトランザクションの両方を速くしたい
ドキュメント	Firestore	モバイル・Web のバックエンド、リアルタイム同期、オフライン対応
ワイドカラム	Bigtable	時系列・IoT・大量書き込み、ミリ秒未満の読み書き、ペタバイト級
インメモリ	Memorystore（Redis、Valkey、Memcached）	キャッシュ、セッション、レートリミッタ
DWH	BigQuery	分析クエリ、集計、BI

よく出る取り違えを挙げます。

- Bigtable と BigQuery: 名前が似ていますが別物です。Bigtable は低遅延の運用系 NoSQL、BigQuery は分析用 DWH です。「1 秒あたり数十万件のセンサーデータを書き込み、直近の値を数ミリ秒で読む」なら Bigtable、「過去 3 年分を集計してレポートを出す」なら BigQuery。
- Cloud SQL と Spanner: 「単一リージョンで、既存の MySQL をそのまま」なら Cloud SQL。「グローバルに強整合、書き込みも水平にスケール、99.999% の可用性」なら Spanner。Spanner は高価なので、要件にその必要性が書かれていない場合は誤答です。
- Firestore と Bigtable: 階層的なドキュメントとリアルタイムリスナーが要るなら Firestore、単純なキーでの超大量アクセスなら Bigtable。
- Filestore と Cloud Storage: アプリが POSIX のファイルシステムを要求するなら Filestore。要求しないなら Cloud Storage（Cloud Storage

FUSE で疑似的にマウントもできますが、性能特性は異なります）。

KnightMotives の車両テレメトリ（走行データ、路面状況）は、書き込み量が膨大で時系列であるため Bigtable が中心になり、分析は BigQuery へ流します。Cymbal Retail の商品カタログはリレーショナルのまま Cloud SQL、検索インデックスは Vertex AI Search、画像は Cloud Storage という分担になります。

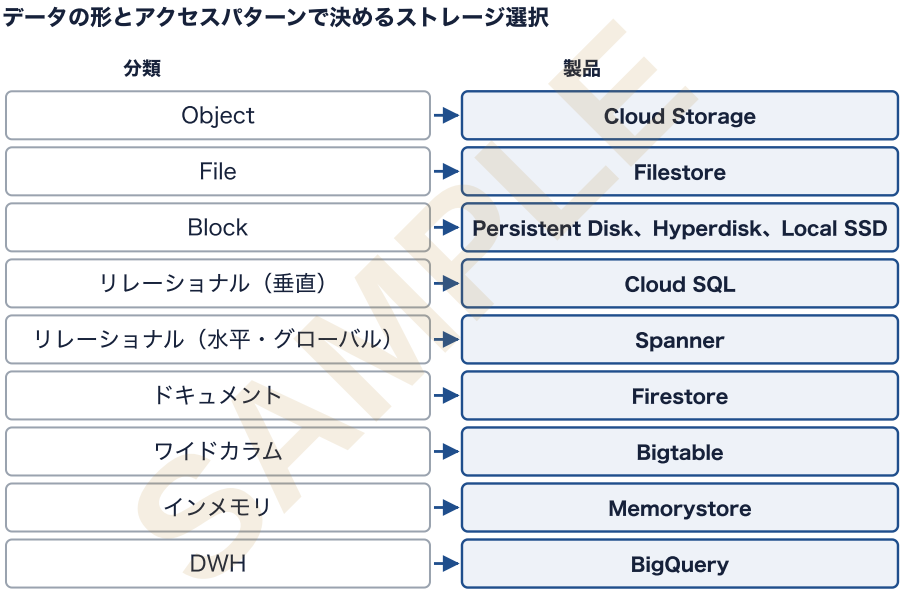

図 1-5　データの形とアクセスパターンで決めるストレージ選択

1-3-6 Mapping compute needs to platform products ―― GKE、Cloud Run、Cloud Run functions [1.3]

Mapping compute needs to platform products は、抽象度の階段を降りるか登るかの判断です。

製品	抽象度	向く条件	向かない条件
Cloud Run functions	最も高い	単一の関数、イベント駆動、短時間	複雑な依存、長時間の処理
Cloud Run	高い	コンテナ化されたステートレスなサービス、HTTP/gRPC、ジョブ	特殊なカーネル要件、DaemonSet 的な常駐
GKE Autopilot	中	Kubernetes の API は要るが、ノード管理はしたくない	ノードへの特権アクセスが要る
GKE Standard	中	ノード構成の細かい制御、GPU/TPU、カスタムスケジューリング	運用の手間を避けたい
Compute Engine	低い	OS・カーネル・ライセンスの制約、レガシー	運用負荷を下げたい
Google Cloud VMware Engine	低い	VMware のまま移設	クラウドネイティブ化が目的

原則は「要件を満たす中で最も抽象度の高いものを選ぶ」です。運用負荷が下がり、スケールが自動になるためです。試験では、要件に「Kubernetes の細かい制御」「ノードへの特権」「特定の CNI」などが書かれていない限り、GKE より Cloud Run が正解になる場面が増えています。

逆に GKE が必要になる典型は、既に Kubernetes で運用している（EHR Healthcare、Cymbal Retail、Altostrat Media がいずれも該当）、複数環境で一貫した運用が要る（GKE Enterprise の fleet）、GPU や TPU を細かく割り

当てたい、ステートフルなワークロード（StatefulSet）がある、といった場合です。

GKE Autopilot と Standard の選択は、「ノードを意識するか」で切ります。Autopilot は Pod 単位の課金でノード管理が不要、セキュリティ既定値も強化されています。Standard はノードプール・マシンタイプ・taint/toleration を自分で設計します。「運用チームが小さい」「Kubernetes の運用経験が浅い」なら Autopilot が正解に寄ります。

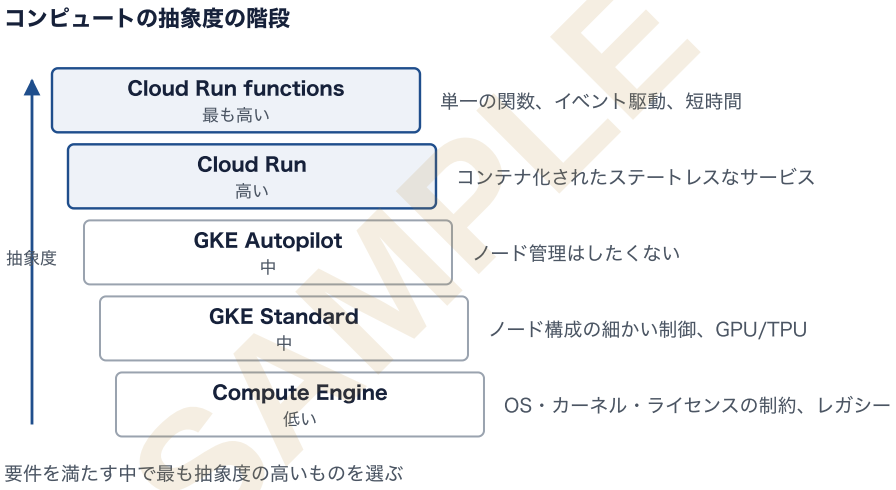

図 1-6　コンピュートの抽象度の階段

1-3-7 Choosing compute resources —— spot VMs、custom machine types、特殊ワークロード [1.3]

Choosing compute resources は、マシンファミリーと購入形態の選択です。

マシンファミリー	特徴	用途
E2	低コスト、汎用	開発、小規模 Web

マシンファミリー	特徴	用途
N シリーズ（N2、N2D、N4）	バランス型	一般的なアプリ、DB
C シリーズ（C3、C4）	高い一貫した性能	レイテンシに敏感なサービス、HPC
M シリーズ	大容量メモリ	SAP HANA、インメモリ DB
C2D / H シリーズ	高い演算性能	HPC、シミュレーション
A シリーズ（A3、A4）	GPU 搭載	学習・推論
TPU（Cloud TPU）	行列演算に特化	大規模なモデル学習・推論

購入形態は次のとおりです。

- On-demand: 既定。中断なし。
- Spot VMs: 大幅に安いが、いつでも中断されうる（30 秒の preemption 通知）。ステートレスなバッチ、CI、レンダリング、GKE のノードプールの一部に使う。中断されると困る処理には使わない。
- Reservations: 特定ゾーンで容量を確保する。ピーク時に確実に確保したい場合。
- Committed use discounts: 1 年・3 年の利用コミットで割引。
- Dynamic Workload Scheduler: 相互接続されたノードのブロックをまとめて確保する方式（flex-start や calendar mode）。GPU/TPU が不足しがちな大規模学習で使う。

Custom machine types は、標準のマシンタイプでは vCPU とメモリの比率が合わない場合に使います。「メモリだけ大量に必要で CPU は少なくてよい」というワークロードで、標準型を選ぶとメモリに合わせて CPU も過剰になり

無駄が出ます。extended memory を使えば、標準の比率を超えたメモリも指定できます。

特殊ワークロード（specialized workload）の手札も押さえます。GPU が要る推論には A シリーズか、GKE の GPU ノードプール。ライセンスが物理サーバー単位なら sole-tenant nodes。機密性が極めて高い処理には Confidential VM（使用中のメモリも暗号化）。Windows Server や SQL Server のライセンスは、bring your own license（BYOL）と従量課金の比較が financial impact の論点になります。

1-4 移行計画（ドキュメントとアーキテクチャ図）の作成 [1.4]

1-4-1 Integrating solutions with existing systems —— 既存システムとの統合 [1.4]

Integrating solutions with existing systems は、移行が「全部一度に切り替わらない」という前提から始まります。移行期間中は、クラウド側とオンプレ側が並走し、両方を見る仕組みが要ります。

並走期の設計項目は次のとおりです。

項目	論点
ネットワーク	Interconnect / HA VPN、IP レンジの重複回避、DNS の相互解決、ファイアウォールの穴
認証	既存の Active Directory と Cloud Identity の同期（Google Cloud Directory Sync）、SAML/OIDC のフェデレーション
データ	双方向同期か片方向か、正はどちらか、切り戻しの手順

項目	論点
トラフィック	段階的な切り替え（DNS の重み付け、ロードバランサのトラフィック分割）
監視	移行前後で同じ指標を見て比較できるか

段階的な切り替えの型としては strangler fig パターン（既存システムの前にプロキシを置き、機能単位で新システムへ振り替えていく）が代表です。KnightMotives の老朽化した ERP とメインフレームは、この型で周辺から剥がしていくのが現実的です。

1-4-2 Assessing and migrating systems and data —— Migration Center による評価 [1.4]

Google Cloud Migration Center は、移行の入口となる統合ツールです。次のことができます。

- 資産の検出（discovery）: エージェントまたはエージェントレスのコレクタで、オンプレの VM やデータベースの構成・使用状況を収集する。手動アップロード（CSV、RVTools のエクスポート）にも対応。
- 現状の可視化: 資産の一覧、依存関係、使用率。
- 移行先のサイジングと費用試算（total cost of ownership の見積もり）: 現状の使用率から、過剰にプロビジョニングされた資産を適正サイズで見積もる。
- データベースの評価（Database Migration Assessment）: 移行の難易度と互換性の問題を洗い出す。
- 移行グループの作成と計画: どの資産をどの順で動かすか。

実際の移行実行には次を使います。

対象	ツール
VM	Migrate to Virtual Machines（VM の移行）
コンテナ化	Migrate to Containers（VM 上のアプリを GKE / Cloud Run 用のコンテナへ）
データベース	Database Migration Service、Datastream
ファイル・オブジェクト	Storage Transfer Service、Transfer Appliance
VMware	Google Cloud VMware Engine への移設

評価で必ず取るのは、現在の使用率（CPU・メモリ・IOPS・帯域のピークと平均）です。オンプレのスペックをそのまま移すと過剰になり、コスト削減の目的を達成できません。逆に、ピークだけを見て過小に見積もると性能問題を起こします。

1-4-3 Using migration methodologies, workload testing, network planning, and dependency planning [1.4]

Migration methodologies は、Google が示す 4 フェーズで語られます。

フェーズ	内容
Assess	現状の棚卸し、TCO の算定、移行方式の決定、優先順位づけ
Plan	ランディングゾーンの設計（組織階層、ネットワーク、IAM、ログ）、移行の順序と計画
Deploy	実際の移行の実行。小さく始めて繰り返す
Optimize	移行後の最適化。サイジング、マネージド化、コスト、自動化

Workload testing は、移行の可否を判断する material です。

- 機能テスト: 移行後に同じ結果を返すか。
- 性能テスト: 負荷試験（load testing）で、想定ピークの何倍まで耐えるか。移行前のベースラインと比較する。
- 統合テスト: 外部システムとの連携が壊れていないか。
- フェイルオーバーテスト: ゾーン障害・リージョン障害を模擬して RTO/RPO を実測する。
- 切り戻しテスト: 失敗したときに戻せるか。

Network planning では、IP アドレス計画（オンプレと重複しない CIDR の割り当て、将来の拡張余地）、帯域の見積もり、経路の冗長性（Interconnect を 2 本、異なる edge availability domain に）、遅延の実測を行います。移行後にサブネットを切り直すのは高くつくので、ここは最初に固めます。

Dependency planning では、アプリケーション間の依存関係を洗い出し、「一緒に動かさなければならないかたまり（move group）」を特定します。Migration Center の依存関係マップや、ネットワークフローの分析を使います。依存を無視して一部だけ移すと、オンプレとクラウドの間で往復が発生して遅延とエグレス費用が跳ね上がります（これを chatty な通信と呼びます）。移行の順序は、依存の少ない末端（開発環境、参照系）から始め、中核の DB は最後にします。

1-4-4 Determining software license implications and financial impact —— ライセンスと財務影響 [1.4]

Software license implications は見落とされやすく、試験でも問われます。

論点	内容
BYOL（bring your own license）	既存ライセンスをクラウドに持ち込む。ベンダーの条件を確認する。物理コア単位の場合は sole-tenant nodes が必要になることがある
ライセンス込みイメージ	Windows Server、SQL Server などを従量課金で使う。短期・変動する用途に向く
ライセンス単位の変化	物理ソケットから vCPU へ。オートスケールするとライセンス数が動的に増える点に注意
サポート終了	古い OS・ミドルウェアはそのまま移すとサポート対象外になる
オープンソースへの置換	商用 DB から PostgreSQL 互換（Cloud SQL、AlloyDB）への移行はライセンス費用を消せるが、移行コストがかかる

Financial impact の議論では、CapEx（設備投資）から OpEx（運用費）への転換が中心になります。オンプレのサーバーは数年に一度まとめて購入する CapEx でしたが、クラウドは毎月の OpEx です。会計上の扱い、減価償却の残存、データセンターのリース残期間が判断に入ります。EHR Healthcare の「The lease on one of the data centers is about to expire」は、リース満了が移行の締め切りであると同時に、二重コストを避ける経済的な理由でもあります。

移行の費用は、移行そのものの費用（人件費、ツール、データ転送）と、移行後の定常費用に分けて示します。経営層への提示は、ROI と回収期間（payback period）の形にします。

1-5 将来のソリューション改善の構想 [1.5]

1-5-1 Cloud and technology improvements —— 技術の進歩を取り込む [1.5]

Cloud and technology improvements とは、いま作る構成が数年後の新技術を取り込める形になっているか、という視点です。

取り込みやすくする設計	具体例
疎結合	Pub/Sub や API を境界にすると、内側の実装を差し替えられる
標準への準拠	Kubernetes、OpenTelemetry、Apache Beam、SQL。ベンダー固有の作りに閉じない
Infrastructure as Code	Terraform で構成を記述しておけば、新しいマシンタイプやサービスへの移行が差分で済む
抽象化されたモデル呼び出し	Model Garden 経由でモデルを差し替えられるようにしておく
観測の一貫性	指標の定義を安定させておけば、実装を替えても比較できる

具体的な改善の芽は、たとえば次のようなものです。Compute Engine 上の自前 MySQL → Cloud SQL → AlloyDB。VM 上のバッチ → Cloud Run jobs。自前の Prometheus → Google Cloud Managed Service for Prometheus。手作業のトリアージ → Gemini Cloud Assist の Investigations。ルールベースの分類 → Gemini によるゼロショット分類。

重要なのは、改善を「いつ、何をきっかけに実施するか」を決めておくことです。たとえば「この構成は月間 1,000 万リクエストまで。それを超えたら Spanner

を検討する」といった閾値を設計文書に書きます。閾値がないと、限界に達してから慌てて作り直すことになります。

1-5-2 Evolution of business needs —— 事業の変化に追随する [1.5]

Evolution of business needs は、要件そのものが変わることを前提に置く考え方です。

変化の型	設計上の備え
地理的拡大	リージョンを増やせる構成（グローバル LB、マルチリージョン対応 DB）、データ所在地を Organization Policy で制御できる状態
規制の追加	データ分類とタグ付けを最初から。監査ログの保持
取扱データの増加	パーティション設計、ライフサイクル管理、コストの単位あたり監視
M&A・組織変更	フォルダ構造の柔軟性、プロジェクト単位の権限分離、請求先の分割
新チャネルの追加	API ファースト。Apigee で外部提供を標準化

KnightMotives の「monetize corporate data to finance new technology investments」は、将来の事業モデルの変化そのものです。これに備えるには、データを事業部ごとのサイロに置かず、統一されたデータプラットフォーム（BigQuery を中心に、Dataplex Universal Catalog でメタデータとガバナンスを管理）に集め、外部提供の際は Analytics Hub でデータ共有を行い、個人情報は Sensitive Data Protection で匿名化する、という構成になります。

データを外に出す以上、誰がどのデータにアクセスしたかの監査が不可欠です。

Altostrat Media の「unlock new revenue streams through targeted marketing and tailored content offerings」も同様で、視聴ログを BigQuery に集約し、レコメンドと動的価格に使う流れを最初から想定した設計にします。

1-5-3 Cloud-first design approach —— クラウドファーストの設計 [1.5]

Cloud-first design approach は、「オンプレのやり方をそのまま持ち込まず、クラウドの前提で設計し直す」考え方です。オンプレとクラウドで前提が反転する点を押さえます。

オンプレの前提	クラウドの前提
サーバーは貴重で長生きさせる（pet）	インスタンスは使い捨て（cattle）。壊れたら作り直す
容量は事前に買う	必要な時に確保し、不要になれば返す
変更は稀で慎重に	変更は頻繁に、小さく、自動で
手作業の構築手順書	Infrastructure as Code
境界防御（ファイアウォールの内側は安全）	ゼロトラスト（すべての要求を検証する）
障害は例外	障害は前提。設計で吸収する
監視は死活監視	可観測性。SLO と error budget で運用する

クラウドファーストで設計するときの実務的な指針は次のとおりです。第一に、状態を持たないコンピュートと、マネージドな状態ストアに分ける。第二に、すべてを再現可能にする（コードから環境を作り直せる）。第三に、スケールの単

位を小さくする。第四に、セキュリティを構成として記述する（Organization Policy、IAM、ポリシー as code）。第五に、コストを設計時から測る。

ただしクラウドファーストは「常に最新のサーバーレスを選ぶ」という意味ではありません。移行期限が厳しい、改修予算がない、ライセンスや規制の制約がある場合には、まず rehost して期限を守り、Optimize フェーズで段階的にクラウドネイティブへ寄せるのが正しい判断です。設問に「まず何をすべきか（first step）」とあれば、大規模な作り直しではなく、リスクの小さい第一歩（評価、パイロット、非本番からの移行）を選びます。

サンプルはここまでです

続き(第2章〜巻末)は、GCP PCA 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。

GCP PCA 学習用テキスト(無料サンプル)

版

2026年7月版(Google Cloud 公式 Professional Cloud Architect Exam Guide（2026-09-07 取得）準拠)

発行

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

お問い合わせ

info@shikaku-doujou.com