IT資格道場

GCP PDE 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 Data Engineer 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 Data Engineer 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。

試験の基本情報

項目	内容
試験名	Google Cloud Certified – Professional Data Engineer
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	**40〜50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）**
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 —— 5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	データ処理システムの設計	**22%**	第1章

セクション	分野	採点対象に占める割合（公式）	本書
2	データの取り込みと処理	**25%**	第2章
3	データの保存	**20%**	第3章
4	分析のためのデータ準備と活用	**15%**	第4章
5	データワークロードの維持と自動化	**18%**	第5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	何を切り分けるか	分かれ方
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 データ処理システムの設計(ドメイン1・採点対象の 22%)

1-1 セキュリティとコンプライアンスを考慮した設計 [1.1]

本節で扱う出題範囲の項目(公式 exam guide の原文)は次のとおりです。以下の本文は、この各項目に対応します。

- Identity and Access Management (e.g., Cloud IAM and organization policies)
- Data security (encryption and key management)
- Privacy (e.g., strategies to handle personally identifiable information)
- Regional considerations (data sovereignty) for data access and storage
- Legal and regulatory compliance
- Designing the project, dataset, and table architecture to ensure proper data governance
- Multi-environment use cases (development vs. production)

Professional Data Engineer 試験のセキュリティ問題は、「この機能は何か」ではなく「この要件のとき、どの仕組みを選ぶのが最小権限かつ運用可能か」を問います。ですから本節は、機能の一覧ではなく、要件から機能へ降りる道筋で説明します。似た機能の取り違えが失点の大半なので、対比表を多めに置きます。

1-1-1 Identity and Access Management(Cloud IAM と organization policies)[1.1]

Google Cloud の権限設計は、**Cloud IAM**(誰が何をできるか)と**organization policies**(そもそも何を許すか)の二階建てです。この二つを混同すると、ほぼ確実に選択肢を取り違えます。

仕組み	問いの形	効き方	典型例
Cloud IAM(allow policy)	「この principal に、このリソースで、この role を与えるか」	与える(許可の付与)	`roles/bigquery.dataViewer` を分析チームに付与
Organization Policy Service(制約)	「この組織では、そもそもこの設定を許すか」	禁じる(構成の制限)	外部公開バケットを組織全体で禁止
IAM Deny policy	「この principal には、たとえ role があっても拒否するか」	allow より優先して拒否	委託先アカウントに削除系権限を一律拒否
VPC Service Controls	「このデータは、この境界の外へ出てよいか」	データ持ち出し経路を遮断	BigQuery からの外部プロジェクトへのコピーを遮断

IAM の allow policy は、**principal(誰)・role(何ができる)・resource(どこで)** の三点セットで表します。principal にはユーザー、Google グループ、サービスアカウント、Google Workspace ドメイン、workforce identity pool、workload identity pool が入ります。role には基本ロール(Owner/Editor/Viewer)、事前定義ロール、カスタムロールの三種類があります。

試験での鉄則は次のとおりです。**基本ロール(Owner/Editor/Viewer)は答えにならない**と考えてください。「分析者に BigQuery のデータを読ませたい」

に対する正解は `roles/bigquery.dataViewer` と `roles/bigquery.jobUser` の組み合わせであって、`roles/editor` ではありません。事前定義ロールで足りるなら事前定義ロール、足りないときだけカスタムロールという順序も同じです。

個人にロールを直接付けないのも定番の切り分けです。Google グループ(あるいは workforce identity federation のグループ)に付与し、人の出入りはグループのメンバーシップで管理します。誤答肢は「担当者が増えるたびに IAM でユーザーを追加する」という形で出てきますが、これは監査性と運用負荷の両面で不適です。

継承(inheritance) も頻出です。リソース階層は organization → folder → project → リソース(BigQuery dataset、Cloud Storage bucket など)で、上位で付けた allow policy は下位すべてに継承されます。逆は起きません。つまり「本番プロジェクトだけ権限を絞りたい」のに organization レベルで付与するのは誤りで、逆に「全プロジェクト共通で監査役に閲覧させたい」なら organization か folder レベルで一度付けるのが正解です。

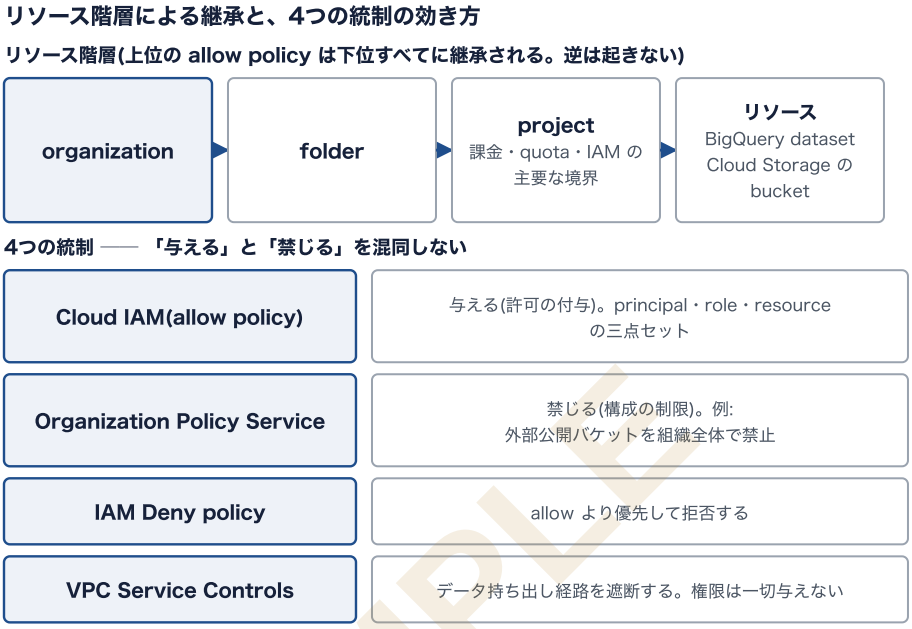


図 1-1　リソース階層による継承と、4つの統制の効き方

BigQuery では IAM の粒度が細かく、これが 1.1 の中心です。

粒度	手段	使う場面
プロジェクト	IAM allow policy	組織横断の管理者・監査役
dataset	dataset レベルの IAM(アクセス制御リスト)	部門ごとの読み書き分離
table / view	table レベルの IAM	同じ dataset 内で一部テーブルだけ見せる

粒度	手段	使う場面
列(column)	policy tag(Data Catalog のタクソノミー)+ column-level security	給与列・メールアドレス列だけ隠す
行(row)	row-level security(row access policy)	営業担当に自分の担当リージョンの行だけ見せる

column-level security は、**taxonomy** の中に **policy tag** を作り、その policy tag をテーブルの列に付け、`roles/datacatalog.categoryFineGrainedReader` を持つ principal だけがその列を読めるようにする仕組みです。列マスキングを併用すると、権限のない人には NULL やハッシュ値を返せます(**dynamic data masking**)。**row-level security** は `CREATE ROW ACCESS POLICY` で述語(例: `region = 'JP'`)を定義します。

「一部の列だけ隠したい」という要件に対して、**authorized view** と **column-level security** のどちらを選ぶかはよく出ます。

手法	仕組み	向く場面	弱点
authorized view	別 dataset のビューに、元テーブルへのアクセスを代理させる。閲覧者は元テーブルの権限を持たない	提供する形(列の選択・集計)を固定したい	ビューの数だけ管理対象が増える
authorized dataset	dataset 単位でまとめて authorized 化	ビューが多い場合の運用簡素化	粒度は dataset

手法	仕組み	向く場面	弱点
column-level security	元テーブルのまま、列に policy tag で権限をかける	同じテーブルを多数の役割が使う	タクソノミー設計が必要
row-level security	行に述語で権限をかける	マルチテナント・地域別の切り出し	述語の維持コスト

authorized routine と **authorized stored procedure** も同じ発想で、UDF やプロシージャに元データへのアクセスを代理させます。

図 1-2　BigQuery のアクセス制御 —— 粒度と手段

サービスアカウントの扱いも問われます。パイプラインは人ではなくサービスアカウントで動かすのが原則で、Dataflow のワーカーには **worker service account**、Cloud Composer には環境のサービスアカウントを割り当てます。**サービスアカウントキー(JSON キー)をダウンロードして配布するのは、試験では基本的に誤答**です。代わりに次を選びます。

- Google Cloud 上のワークロード → そのリソースに **attached service account** を割り当てる(キー不要)
- 他クラウド・オンプレミス → **Workload Identity Federation** で外部 ID を短命の Google 資格情報に交換する
- GKE 上のワークロード → **Workload Identity Federation for GKE**
- 一時的な代理実行 → `roles/iam.serviceAccountTokenCreator` による **短命トークンの発行**(service account impersonation)

organization policies(組織のポリシー)は「構成の禁止」を担います。データ基盤でよく出る制約は次のとおりです。

制約	効果	要件文の言い回し
`constraints/gcp.resourceLocations`	リソースを作れるロケーションを限定	「データは EU から出してはならない」
`constraints/iam.allowedPolicyMemberDomains`(domain restricted sharing)	自社ドメイン以外への付与を禁止	「社外アカウントへの共有を防ぎたい」
`constraints/storage.publicAccessPrevention`	バケットの公開を禁止	「バケットが誤って全世界公開されるのを防ぐ」

制約	効果	要件文の言い回し
`constraints/iam.disableServiceAccountKeyCreation`	サービスアカウントキーの作成を禁止	「鍵の流出事故を構造的に無くす」
`constraints/compute.requireOsLogin`、`constraints/compute.vmExternalIpAccess`	外部 IP や SSH の制限	「Dataproc クラスタに外部 IP を付けさせない」

organization policies は organization/folder/project のどこにでも設定でき、既定では継承されます。子で緩めたいときは継承を上書きしますが、`enforce` 型の制約では上書き自体を禁じる運用が普通です。

よく出る取り違えを並べます。第一に、「IAM で禁止する」という発想です。IAM は許可を与える仕組みで、禁止は organization policy か IAM Deny policy の役目です。第二に、「VPC Service Controls で権限を与える」という誤りです。VPC Service Controls は境界を作りデータの持ち出しを止めるだけで、権限は一切与えません。第三に、「監査ログを見るために Owner を付ける」という誤りで、正しくは `roles/logging.viewer` と `roles/logging.privateLogViewer` です。

監査面では **Cloud Audit Logs** の四種類を押さえます。**Admin Activity**(既定で有効・無効化不可)、**Data Access**(既定で無効。BigQuery のデータ読み取りのみ既定で記録)、**System Event**、**Policy Denied** です。「誰がこのテーブルを読んだかを追跡したい」なら Data Access ログを明示的に有効化する、が正解の形です。

1-1-2 Data security —— encryption と key management [1.1]

Google Cloud では保存データ(data at rest)は**既定で常に暗号化**され、転送中(data in transit)も Google のネットワーク内および外部との通信で暗号化されます。ですから「暗号化を有効にする」という選択肢は多くの場合ダミーです。問われるのは**鍵を誰が管理するか**です。

方式	鍵の所有・保管	使える場面	選ぶ理由の言い回し
Google-managed encryption key(既定)	Google が生成・保管・ローテーション	何もしなくてよい	規制要件が無い
Customer-managed encryption key(CMEK)	Cloud KMS 上の鍵を顧客が作成・管理	BigQuery、Cloud Storage、Pub/Sub、Dataflow、Dataproc など	「鍵のローテーション周期を自社で決めたい」「鍵を無効化して読めなくしたい(crypto-shredding)」
Customer-supplied encryption key(CSEK)	顧客が鍵の実体をリクエストごとに渡す。Google は保管しない	Cloud Storage、Compute Engine の一部	「鍵をクラウドに置きたくない」
Cloud External Key Manager(Cloud EKM)	鍵は外部の鍵管理サービスに置き、参照だけする	BigQuery、Cloud Storage 等	「鍵をクラウド事業者の外に保持する規制」
Cloud HSM	FIPS 140-2 Level 3 の HSM 上の鍵	高保証が要る領域	「ハードウェア保護の証跡が要る」

CMEK を選ぶときの実務的な注意が試験に出ます。**鍵は保護対象リソースと同じロケーションに置く**必要があります(BigQuery の US マルチリージョンの

データセットなら、鍵も対応するロケーション)。また、各サービスの **service agent**(例:BigQueryなら `bq-<project-number>@bigquery-encryption.iam.gserviceaccount.com`)に `roles/cloudkms.cryptoKeyEncrypterDecrypter` を付与しないと、テーブル作成やジョブが失敗します。「CMEKにしたらジョブがPermission deniedで落ちる」というトラブルの原因はほぼこれです。

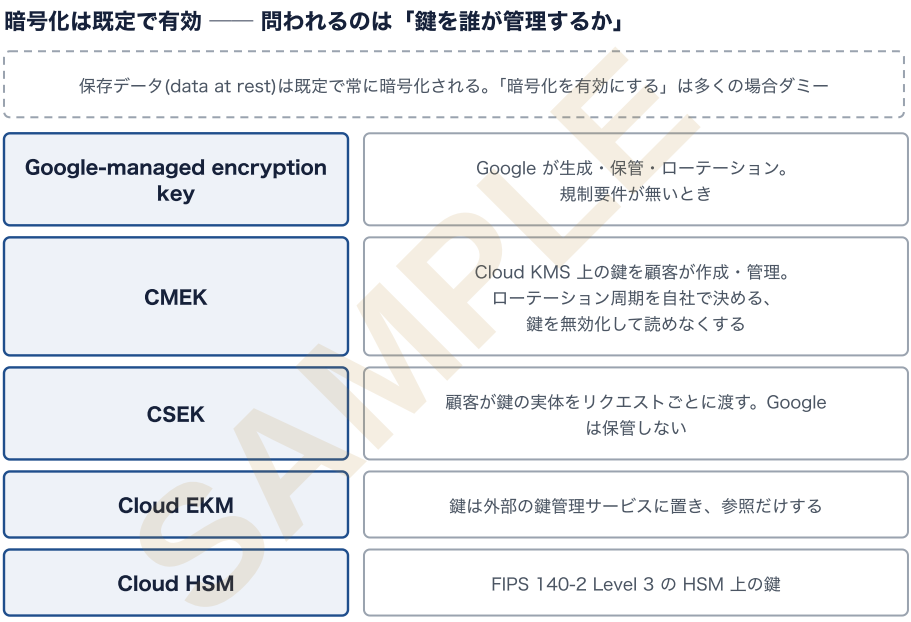


図 1-3　暗号化は既定で有効 —— 問われるのは「鍵を誰が管理するか」

鍵の**ローテーション**はCloud KMSの自動ローテーション期間で設定します。ローテーションしても既存データが再暗号化されるわけではなく、新しいバージョンは以後の書き込みに使われる点に注意します。既存データを新しい鍵に載せ替えたいなら、テーブルの再作成やオブジェクトの書き直しが要りま

す。**鍵バージョンを無効化・破棄すると、そのデータは復号できなくなる**ため、これが「保持期限を過ぎたデータを確実に読めなくする」手段(crypto-shredding)として使えます。

アプリケーション層の暗号化としては、BigQuery の **AEAD 暗号化関数**(`AEAD.ENCRYPT` 、 `AEAD.DECRYPT_STRING` 、 `KEYS.NEW_KEYSET` 、 `DETERMINISTIC_ENCRYPT`)があります。これは「同じテーブルの中で、顧客ごとに別の鍵で列を暗号化し、鍵を渡した相手だけが復号できるようにしたい」という要件で選びます。決定的暗号(`DETERMINISTIC_ENCRYPT`)は暗号文のまま等値結合やグループ化ができる代わりに、同じ平文が同じ暗号文になるので頻度分析に弱い、という対比も押さえます。

転送中の保護では、**Private Service Connect**、**VPC Service Controls**、**Cloud Interconnect / Cloud VPN** が道具になります。「BigQuery へのアクセスをインターネット経由にしたくない」なら **Private Google Access** または **Private Service Connect endpoint** を選び、「データの持ち出し自体を止めたい」なら VPC Service Controls の境界を作る、という切り分けです。

1-1-3 Privacy —— personally identifiable information(PII)の扱い [1.1]

PII(personally identifiable information)の扱いは、「見つける・分類する・隠す・記録する」の四段階で整理すると、選択肢の切り方が安定します。

段階	主な手段	補足
見つける(discovery)	Sensitive Data Protection(旧 Cloud Data Loss Prevention / Cloud DLP)の inspection job、discovery scan	BigQuery テーブル、Cloud Storage、Datastore を走査

段階	主な手段	補足
分類する(classification)	infoType detector(`EMAIL_ADDRESS`、`PHONE_NUMBER`、`CREDIT_CARD_NUMBER`、`JAPAN_INDIVIDUAL_NUMBER` など)、custom infoType	結果を policy tag に反映して column-level security へ接続
隠す(de-identification)	masking、redaction、bucketing、date shifting、format-preserving encryption、cryptographic hashing、tokenization	再識別が必要かどうかで選ぶ
記録する(governance)	Cloud Audit Logs の Data Access、Knowledge Catalog(旧 Dataplex Universal Catalog)の lineage	誰がいつ触ったかを残す

Cloud Data Loss Prevention(Cloud DLP) は現在 **Sensitive Data Protection** という名称ですが、exam guide と多くの教材は Cloud DLP 表記のままです。本書では両方の呼び方を併記します。

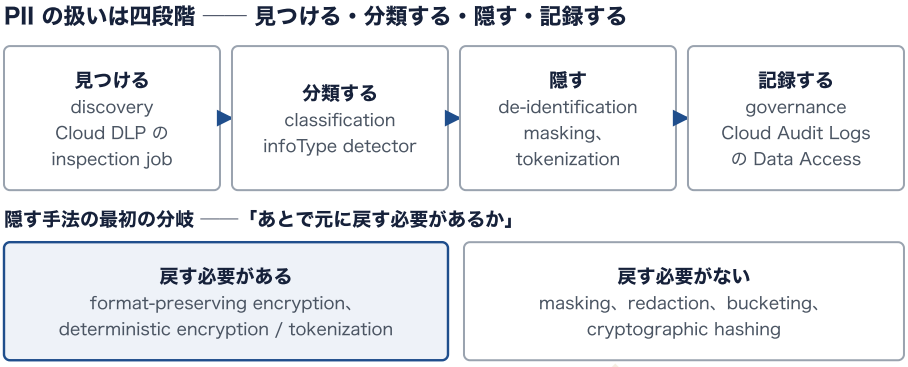
de-identification の手法選びが最頻出です。

手法	何をするか	再識別	向く場面
masking(character masking)	文字を `*` などで置換	不可	画面表示でカード番号の下4桁だけ残す

手法	何をするか	再識別	向く場面
redaction	該当部分を削除	不可	自由記述ログから氏名を消す
bucketing / generalization	値を区間に丸める(年齢 34 → 30〜39)	不可	統計分析用のデータセット
date shifting	日付を個人ごとに一定量ずらす	条件付き	医療データで受診間隔だけ保ちたい
cryptographic hashing	鍵付きハッシュに置換	不可	結合キーとしてだけ使いたい
format-preserving encryption(FPE)	桁数・文字種を保ったまま暗号化	可(鍵があれば)	既存システムの入力チェックを壊さない
deterministic encryption / tokenization	同じ入力を同じトークンに	可(鍵があれば)	部門をまたいだ突合が必要

「あとで元に戻す必要があるか」が最初の分岐です。戻す必要があるなら FPE か tokenization、戻す必要がないなら masking・hashing・bucketing です。「別テーブルと結合したいが実値は見せたくない」なら決定的な手法(deterministic encryption か hashing)を選びます。

再識別リスクの評価では **risk analysis** の指標が出ます。**k-anonymity**(同じ準識別子の組み合わせが最低 k 件ある)、**l-diversity**(各グループ内の機微属性が l 種類以上ある)、**k-map**、**δ-presence** です。「匿名化したデータを外部に提供してよいか判断したい」という文脈では、これらを Cloud DLP の risk analysis job で測る、が正解の形になります。



「別テーブルと結合したいが実値は見せたくない」なら決定的な手法(deterministic encryption か hashing)。
生データを一度も保存したくないなら、書き込み前の Dataflow 段階で de-identify する。

図 1-4　PII の扱いは四段階 —— 見つける・分類する・隠す・記録する

パイプラインへの組み込み方も問われます。取り込み時に隠すなら Dataflow のテンプレート(Cloud Storage / Pub/Sub から Cloud DLP を呼んで BigQuery に書く)を使い、保存後に隠すなら BigQuery 側の column-level security と dynamic data masking を使います。**生データを一度も保存したくない**という要件なら、書き込み前の Dataflow 段階で de-identify するのが唯一の正解になります。逆に「監査のため生データは保持しつつ、分析者には見せない」なら、生データは制限された dataset に置き、分析者には authorized view か masking 済みの列を見せます。

1-1-4 Regional considerations(data sovereignty)—— データの所在 [1.1]

data sovereignty(データ主権)は「データがどの国・地域の法の下に置かれるか」の問題で、**保存場所**と**処理場所**と**アクセス元**の三つを分けて考えます。

観点	主な統制手段
保存場所	BigQuery dataset のロケーション、Cloud Storage bucket のロケーション、`constraints/gcp.resourceLocations`
処理場所	Dataflow ジョブの `--region` と `--workerZone`、Dataproc のリージョン、BigQuery のジョブ実行ロケーション
アクセス元	VPC Service Controls、IAM の条件（`request.time`、アクセスレベル）、Access Context Manager
運用者のアクセス	Access Transparency、Access Approval、Assured Workloads

BigQuery のロケーション規則は覚えるべき事実です。**dataset のロケーションは作成後に変更できません。クエリは、参照するすべての dataset が同じロケーションにある場合にだけ実行できます。**US の dataset と EU の dataset を単一のクエリで結合することはできません。移す場合は BigQuery Data Transfer Service の **dataset copy** か、Cloud Storage 経由の export/load を使います。

「EU の個人データを EU 外へ出してはならない」という要件が出たら、次のセットが正解の骨格になります。

1. dataset と bucket を `europe-west1` 等の EU ロケーションに作る
2. `constraints/gcp.resourceLocations` を組織/フォルダに設定し、EU 以外にリソースを作れなくする
3. VPC Service Controls の境界を作り、EU プロジェクト群から外へのデータ持ち出しを遮断する

4. CMEK の鍵も EU ロケーションの Cloud KMS キーリングに置く
5. 必要なら Assured Workloads で EU 管理者のみの運用(data residency + sovereignty)を強制する

マルチリージョン(US、EU)とリージョン(asia-northeast1 など)の違いも問われます。マルチリージョンは可用性が高く、BigQuery ではマルチリージョン内で自動的に冗長化されますが、「国内に限定せよ」という要件には使えません。日本国内限定なら `asia-northeast1`(東京)や `asia-northeast2`(大阪)を選びます。

1-1-5 Legal and regulatory compliance —— 法令・規制対応 [1.1]

規制要件は、試験では「その規制が何を求めるか」よりも「Google Cloud のどの機能で満たすか」の形で問われます。代表的な対応表を持っておくと素早く切れます。

要件の言い回し	満たす機能
「一定期間、誰にも改変・削除させない」(WORM)	Cloud Storage の bucket lock と retention policy、BigQuery の table 有効期限とは別物
「監査証跡をすべて残す」	Cloud Audit Logs(Data Access を明示的に有効化)、Log bucket の保持期間、log sink で BigQuery / Cloud Storage へ
「監査ログを改ざんされたくない」	log sink 先の bucket に retention policy と bucket lock、別プロジェクトへ集約
「Google の運用担当者のアクセスを可視化・承認したい」	Access Transparency(可視化)、Access Approval(事前承認)

要件の言い回し	満たす機能
「規制準拠のコントロールパッケージを一括適用」	Assured Workloads(FedRAMP、CJIS、EU Regions and Support など)
「削除要求(忘れられる権利)に応える」	行削除 + BigQuery の time travel と fail-safe 期間の理解、CMEK の crypto-shredding
「保持期間を過ぎたデータを自動で消す」	Cloud Storage の Object Lifecycle Management、BigQuery の dataset/table/partition expiration

BigQuery の **time travel** は既定で 7 日間(2〜7 日で設定可)、その後に **fail-safe** 期間が約 7 日間あり、この間は Google 側にデータが残ります。「削除したはずのデータが time travel で読める」ため、厳密な削除要求では time travel window の設定と、必要なら CMEK の鍵破棄を併用する、という説明ができるようにしておきます。

コンプライアンス設計の全体像としては、**組織のポリシーで禁じる → IAM で最小権限を与える → VPC Service Controls で境界を作る → Cloud DLP で機微を検出・秘匿する → Cloud Audit Logs で証跡を残す** という五層で覚えると、どの選択肢が抜けているかを見つけやすくなります。

1-1-6 project・dataset・table のアーキテクチャ設計(data governance)[1.1]

BigQuery の「project、dataset、table architecture」をどう切るかは、この objective の核心です。切り方は権限境界・課金境界・ロケーション境界の三つで決まります。

階層	何の境界か	設計の指針
project	課金・quota・IAM の主要な境界。予約(reservation)の割り当て単位	環境(dev/prod)、事業部、機密度で分ける
dataset	ロケーションの境界。アクセス制御の主要単位	データの層(raw/staging/curated/mart)とドメインで分ける
table	データそのもの。パーティション・クラスタリングの単位	命名規則を統一し、列・行レベルの制御を載せる

実務でよく使われ、試験でも正解になりやすいのが**層(layer)で dataset を分ける**設計です。

- `raw_*`：取り込んだそのままのデータ。書き込みはパイプラインのサービスアカウントのみ、読み取りもデータエンジニアのみ
- `staging_*`：クレンジング・正規化の途中。人は原則触らない
- `curated_*`：検証済みの正データ。分析者は読み取りのみ
- `mart_*` / `analytics_*`：部門ごとの集計・ビュー。BI ツールが参照する

さらに**プロジェクトを分ける**判断は、次のいずれかに当たるときです。第一に環境の分離(後述)、第二に課金を部門別に分けたい場合、第三に quota や reservation を独立させたい場合、第四に機密度が違い IAM の事故を構造的に防ぎたい場合です。逆に、単に「テーブルが多いから」では project を分けません。

data mesh 的な設計(3.4 で詳述)では、ドメインごとに project を持たせ、Knowledge Catalog(Dataplex)で横断カタログを作り、Analytics Hub で共有する、という形になります。ここでの governance の要点は、「誰がデータの所有者か」をリソース階層と IAM で明示することです。

命名規則も governance の一部です。`<環境>_<ドメイン>_<層>` のように機械可読な規則にしておくと、organization policy や Knowledge Catalog の aspect による自動分類、コストの label 集計がやりやすくなります。**label**(リソースラベル)を project・dataset・table・ジョブに付けておくと、Cloud Billing のエクスポート先 BigQuery でコストをドメイン別に切れます。これは 5.1 のコスト最適化にも直結します。

1-1-7 Multi-environment use cases(development vs. production)[1.1]

開発環境と本番環境の分離は、「別 project にする」が原則の答えです。理由を三つ言えるようにしておきます。第一に IAM の境界が project 単位で効くこと、第二に quota と reservation が project 単位であること、第三に誤操作のブラスト半径を限定できることです。

分離の粒度	実現	評価
別 organization	完全分離	過剰。共通の組織ポリシーが効かなくなる
別 folder + 別 project(推奨)	`dev` フォルダと `prod` フォルダ、その下に project	組織ポリシーをフォルダ単位で出し分けできる
同一 project 内で dataset を分ける	`sales_dev` と `sales_prod`	権限事故が起きやすい。試験では通常不正解

環境間の差分は**コードで表現**します。Terraform や Cloud Deployment Manager で同じモジュールを変数違いで適用し、Dataform や dbt では同じ SQL を環境変数でスキーマ切り替えします。**Dataform** には development workspace と release configuration / workflow configuration があり、開発中は各自の workspace で `dataform_dev_<user>` のようなスキーマに出力

し、本番は release configuration からスケジュール実行する、という形が標準です。

データの扱いも設計が要ります。本番データをそのまま開発環境にコピーするのは、PII の観点で通常は不可です。次のいずれかを選びます。

- Cloud DLP で de-identify したサブセットを開発環境に配る
- 合成データ(synthetic data)を生成して使う
- 本番の読み取り専用 authorized view を、マスキング済みの列だけ公開する

CI/CD の観点は 2.3 で詳述しますが、1.1 の文脈では「dev から prod への昇格が人手のコピーではなく、パイプラインのコード変更 → レビュー → 自動デプロイになっているか」が問われます。誤答肢は「本番の BigQuery コンソールで直接ビューを編集する」「本番の Composer DAG を Cloud Storage に手でアップロードする」といった形で出ます。

多環境での鍵と接続情報は、Secret Manager に環境ごとのシークレットを置き、サービスアカウントに `roles/secretmanager.secretAccessor` を環境ごとに付与します。設定値をコード内や Composer の DAG にベタ書きするのは誤りです。

1-2 信頼性と忠実性を考慮した設計 [1.2]

本節で扱う出題範囲の項目(公式 exam guide の原文)は次のとおりです。以下の本文は、この各項目に対応します。

- Preparing and cleaning data (e.g., Dataform, Dataflow, and Cloud Data Fusion, prompting LLMs for query generation)
- Monitoring and orchestration of data pipelines

- Disaster recovery and fault tolerance
- Making decisions related to ACID (atomicity, consistency, isolation, and durability) compliance and availability
- Data validation

「信頼性(reliability)」は落ちないこと・戻せること、「忠実性(fidelity)」は入ってきたデータが正しく・欠けも重複もなく届くことです。この二つは別々の対策を要求します。

1-2-1 Preparing and cleaning data —— Dataform、Dataflow、Cloud Data Fusion、LLM によるクエリ生成 [1.2]

データの前処理をどのツールでやるかは、この objective の最頻出です。まず対比表を頭に入れます。

ツール	実体	得意	選ぶ理由	選ばない理由
Dataform	BigQuery 内の SQL ワークフロー(SQLX)。ELT	BigQuery に入った後の変換、依存関係、assertion によるテスト、バージョン管理	「変換を SQL で書きたい」「Git で管理したい」	BigQuery 外のデータは扱えない
Dataflow	Apache Beam のフルマネージド実行基盤。ETL	バッチとストリーミングを同じコードで、任意の変換、大規模	「ストリーミングが要る」「複雑な処理ロジック」「Java/Python で書く」	単純な SQL 変換には重い

ツール	実体	得意	選ぶ理由	選ばない理由
Cloud Data Fusion	CDAP ベースの GUI 型 ETL。プラグインで多数のソースに接続	ノーコード/ローコード、非エンジニアの参加、豊富なコネクタ	「コードを書かずに」「多様な外部ソース」	実行は裏で Dataproc。起動が遅くコストが高め
Dataprep(Alteryx)	対話的なデータ整形	探索的なクレンジング、レシピ	「まずデータを見ながら整えたい」	大規模な定常運用には不向き
BigQuery SQL / スケジュールドクエリ	BigQuery 単体	最小構成の変換	「他に何も要らない」	依存関係管理・テストが弱い

Dataform の押さえどころは次です。SQLX ファイルで `config { type: "table" }`、`type: "view"`、`type: "incremental"` を宣言し、`ref()` でテーブル間の依存を書くと DAG が自動生成されます。**assertion**(`assertions: { uniqueKey: ["id"], nonNull: ["created_at"] }` や `type: "assertion"` の SQL)が **data validation** の役割を果たし、失敗すると下流を止められます。開発は development workspace、本番は release configuration と workflow configuration でスケジュールします。

Cloud Data Fusion はエディション(Developer / Basic / Enterprise)があり、Enterprise でないと本番向けの機能(高可用、複数パイプラインの並列)が制限されます。裏で **Dataproc** クラスタを起動して実行するため、起動時間とコストがかかります。**Wrangler** という対話的なクレンジング UI があり、ここで作った変換(directives)をパイプラインに組み込めます。

prompting LLMs for query generation(LLM にクエリを書かせる)は v4.2 で追加された話題です。BigQuery では **Gemini in BigQuery** による SQL 生成・補完・説明、および `ML.GENERATE_TEXT` による行単位の生成が使え

ます。設計上の注意を三つ押さえます。第一に、生成された SQL は必ず人がレビューし、`--dry_run` でスキャン量を確認してから本番に載せます。第二に、プロンプトに機微データを混ぜない(必要なら Cloud DLP で de-identify する)ことです。第三に、生成物は非決定的なので、本番パイプラインには「生成された SQL を固定してコード管理する」形で組み込み、実行時に毎回生成させないのが原則です。

クレンジングの典型的な作業と対応する手段は次のとおりです。

作業	BigQuery SQL / Dataform	Dataflow(Beam)
重複排除	`ROW_NUMBER() OVER (PARTITION BY key ORDER BY ts DESC) = 1`	`Distinct`、`Deduplicate`、Pub/Sub の message ID による exactly-once
欠損補完	`COALESCE`、`IFNULL`、直近値の window 関数	`ParDo` 内で既定値、side input で参照表
型・書式の正規化	`SAFE_CAST`、`PARSE_DATE`、`REGEXP_REPLACE`	スキーマ定義 + `SAFE` な変換、失敗は dead-letter へ
外れ値の除去	`APPROX_QUANTILES` で分位点、`WHERE` で除外	統計を side input で配り `Filter`
参照整合性の検査	`LEFT JOIN` して NULL を数える、assertion	`CoGroupByKey` して片側欠損を dead-letter へ

1-2-2 Monitoring and orchestration of data pipelines [1.2]

オーケストレーション(いつ何を実行するか)と監視(動いているか・正しいか)は設計段階で決めます。ここでは選定基準だけ整理し、実装は 2.3 と 5.2/5.4

で扱います。

手段	実体	向く場面
Cloud Composer	マネージド Apache Airflow。Python の DAG	依存関係が複雑、多数のシステムをまたぐ、リトライやバックフィルが要る
Workflows	サーバーレスのステップ実行(YAML/JSON)	軽量な API の連鎖、イベント駆動、常時起動のコストを避けたい
Cloud Scheduler	cron	単純な定時起動
BigQuery scheduled queries	BigQuery 内蔵	単一のクエリを定期実行するだけ
Dataform workflow configuration	Dataform 内蔵	Dataform の DAG のみで完結する場合
Eventarc / Pub/Sub	イベント駆動	「ファイルが置かれたら動く」

選定の勘所は、**Composer は常時稼働の環境費がかかる**点です。「1 日 1 回、クエリを 1 本流すだけ」に Composer を選ぶのは過剰で、scheduled query か Workflows が正解になります。逆に「20 個のタスクが依存関係を持ち、失敗時に部分再実行したい」なら Composer です。

監視の設計では、**何を測るか**を先に決めます。

- 鮮度(freshness): 最新レコードのタイムスタンプと現在時刻の差
- 網羅性(completeness): 期待件数に対する実件数、欠損キーの数
- 正確性(accuracy): 参照値との突合、集計値の許容誤差
- 処理の健全性: Dataflow の system lag と data watermark、backlog(Pub/Sub の未確認メッセージ数と oldest unacked message

age)、エラー率

- コスト: スキャンバイト数、slot 使用時間

これらを **Cloud Monitoring** の指標にし、**alerting policy** を作ります。ログから指標を作るときは **log-based metrics** を使います。SLO を定義するなら Cloud Monitoring の service monitoring を使います。

1-2-3 Disaster recovery と fault tolerance [1.2]

災害復旧の設計は、**RPO(どこまでのデータ損失を許すか)** と **RTO(どれだけの時間で復旧するか)** から逆算します。

サービス	主な保護手段	RPO/RTO の目安
BigQuery	マルチリージョン冗長、time travel(既定 7 日)、snapshot、table clone、`bq cp` によるクロスリージョンコピー、dataset replication と managed disaster recovery(Enterprise Plus)	Enterprise Plus のマネージド DR は自動フェイルオーバー
Cloud Storage	multi-region / dual-region、object versioning、soft delete、turbo replication(dual-region)	turbo replication は RPO 15 分目標
Cloud SQL	自動バックアップ、point-in-time recovery(PITR)、high availability(リージョン内の別ゾーンにスタンバイ)、cross-region read replica	HA はゾーン障害、read replica の昇格はリージョン障害

サービス	主な保護手段	RPO/RTO の目安
Spanner	multi-region 構成で自動的に複数リージョンに複製	実質的に無停止
Bigtable	replication(複数クラスタ)+app profile によるルーティング	single-cluster routing と multi-cluster routing の選択
Pub/Sub	メッセージ保持(既定 7 日まで)、snapshot と seek	seek で過去時点から再処理
Dataflow	チェックポイント、`--enableStreamingEngine`、リージョン指定	ストリーミングは再起動時に状態を保つ設計が要る
Dataproc	ジョブを冪等に、データは Cloud Storage に(ephemeral cluster)	クラスタを捨てて作り直す
Composer	DAG を Cloud Storage に、環境は再作成可能に	別リージョンに環境を用意する構成もある

fault tolerance(耐障害性) の設計原則は、試験では次の形で問われます。第一に、**処理を冪等(idempotent)にする**ことです。同じ入力を二度処理しても結果が変わらないようにすれば、リトライが安全になります。BigQuery なら `MERGE` による upsert、Cloud Storage なら決定的なオブジェクト名です。第二に、**失敗したレコードを捨てない**ことです。Dataflow の **dead-letter queue**(処理できなかった要素を別の Pub/Sub トピックや BigQuery テーブルに退避)がその答えです。第三に、**部分再実行できる粒度に切る**ことです。日次パーティション単位で書き、失敗したパーティションだけ作り直します。

「Dataflow のストリーミングジョブが失敗した場合の復旧」は定番です。**drain**(新規入力を止めて処理中のデータを完了させる)と **cancel**(即座に破

棄)の違い、および **update**(既存ジョブを新しいコードに置き換えて状態を引き継ぐ)を区別します。パイプラインのロジックを変えたいなら update、止めたいがデータを失いたくないなら drain です。

1-2-4 ACID(atomicity, consistency, isolation, durability)と可用性の判断 [1.2]

ACID compliance と availability のトレードオフは、ストレージ選定(3.1)と直結します。まず何が ACID を満たすかを整理します。

サービス	トランザクション	整合性	可用性の性格
Cloud SQL	単一インスタンス内で完全な ACID	強整合	HA 構成でゾーン障害に耐える。リージョンをまたぐ書き込みは不可
AlloyDB for PostgreSQL	完全な ACID(PostgreSQL 互換)	強整合	リージョン内 HA、cross-region replication
Spanner	分散環境で ACID。外部整合性(external consistency)	グローバルに強整合	multi-region で 99.999% を狙える
Bigtable	行単位でアトミック。複数行のトランザクションは無し	単一クラスタでは強整合、レプリケーション時は結果整合	クラスタ追加で可用性を上げる
Firestore	ドキュメント/複数ドキュメントのトランザクション対応	強整合(Native mode)	マルチリージョン構成あり

サービス	トランザクション	整合性	可用性の性格
BigQuery	単一ステートメントはアトミック。マルチステートメントトランザクションも可	強整合	分析用途。OLTP には使わない
Cloud Storage	オブジェクト単位でアトミック。上書きは強整合	メタデータも強整合	ロケーション種別で可用性が変わる

判断の型を三つ持っておきます。第一に、「グローバルに分散した書き込みで、強整合と ACID の両方が要る」なら **Spanner** です。これは Cloud SQL では実現できません。第二に、「秒間数十万の書き込みがあり、行キーで引くだけ。複数行トランザクションは不要」なら **Bigtable** です。第三に、「既存の PostgreSQL/MySQL アプリをそのまま移す」なら **Cloud SQL**、性能と分析(columnar engine)も要るなら **AlloyDB** です。

CAP 的なトレードオフの言い回しも押さえます。Bigtable の replication で **multi-cluster routing** を選ぶと可用性は上がりますが、クラスタ間は非同期複製なので結果整合になります。**single-cluster routing** を選ぶと強整合(read-your-writes)が保てますが、そのクラスタが落ちると影響を受けます。「強い整合が必要な処理は single-cluster routing の app profile、可用性優先の読み取りは multi-cluster routing の app profile」と使い分けるのが正解の形です。

durability(耐久性) の観点では、Pub/Sub は ack されるまでメッセージを保持し、既定で少なくとも一度(at-least-once)配信します。**exactly-once delivery** はサブスクリプション設定で有効にできますが、リージョン内に限られます。Dataflow と組み合わせる場合、Beam 側の処理が冪等であれば実質的に exactly-once の結果が得られる、という説明ができるようにします。

1-2-5 Data validation [1.2]

data validation(データ検証)は、忠実性(fidelity)を担保する最後の砦です。どの層で検査するかを設計します。

層	手段	検出できるもの
取り込み時	スキーマ強制(Pub/Sub schema、BigQuery のスキーマ、Avro/Protobuf)、Dataflow の dead-letter	型不一致、必須項目欠落、壊れたレコード
変換時	Dataform の assertion、Dataflow のカウンタ(Metrics.counter)	件数の急変、想定外の値
保存後	Knowledge Catalog(Dataplex)の data quality スキャンと data profiling スキャン、BigQuery のクエリによるチェック	分布の変化、NULL 率、一意性違反、参照整合性
提供前	authorized view 内での制約、BI 側の閾値アラート	集計値の異常

Dataplex(Knowledge Catalog)の data quality スキャンは、ルール(NonNullExpectation、RangeExpectation、SetExpectation、RegexExpectation、UniquenessExpectation、StatisticRangeExpectation、TableConditionExpectation、SQL assertion)を宣言し、BigQuery テーブルに対して定期実行します。結果は BigQuery に書き出せ、Cloud Monitoring や Pub/Sub で通知できます。**data profiling スキャン**は事前にルールを決めず、列ごとの分布・NULL 率・ユニーク数を測るので、ルールを決める前段として使います。

Dataform の assertion との使い分けは、「変換パイプラインの中で止めたい」なら assertion、「保存済みのデータを横断的に定期監視したい」なら Dataplex のスキャン、と覚えます。両方を併用する設計も普通です。

検証で異常が出たときの扱いも設計事項です。**止める(fail fast)** か **通すが隔離する(quarantine)** かの二択で、後者は quarantine 用テーブル/バケットに退避して下流は正常データだけで進める形になります。ストリーミングでは止めると滞留するので、通常は quarantine + アラートが正解です。バッチの財務系集計では fail fast が正解になります。要件文の「遅れても正しさ優先」「多少欠けても止めない」という言い回しで切り分けます。

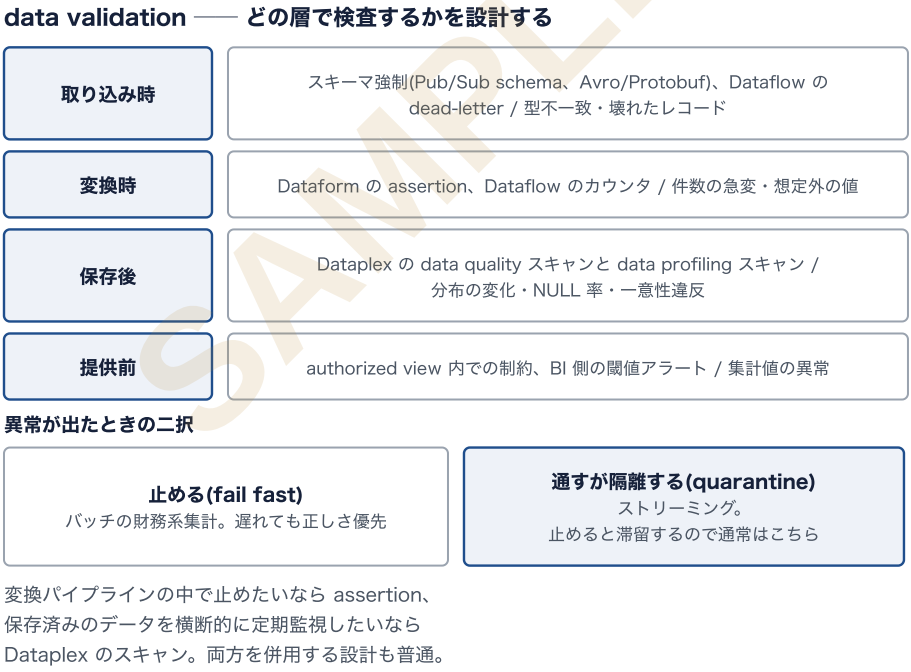


図 1-5　data validation —— どの層で検査するかを設計する

1-3 柔軟性と可搬性を考慮した設計 [1.3]

本節で扱う出題範囲の項目(公式 exam guide の原文)は次のとおりです。以下の本文は、この各項目に対応します。

- Mapping current and future business requirements to the architecture
- Designing for data and application portability (e.g., multi-cloud and data residency requirements)
- Data staging, cataloging, profiling, and discovery (data governance)

1-3-1 Mapping current and future business requirements to the architecture [1.3]

要件をアーキテクチャに写像する作業は、非機能要件を数値に落とすところから始めます。試験のシナリオ文には必ず数値か制約が埋め込まれているので、それを拾って製品を絞ります。

要件の数値・言葉	読み替え	効いてくる選択
「秒間 100 万イベント」	高スループットの取り込み	Pub/Sub + Dataflow、Bigtable
「ダッシュボードは 1 秒以内」	低レイテンシの読み取り	BI Engine、materialized view、Bigtable、Memorystore
「昨日の分が朝までにあればよい」	バッチで十分	スケジュールドクエリ、Dataproc、バッチ Dataflow

要件の数値・言葉	読み替え	効いてくる選択
「数分以内に反映」	ニアリアルタイム	Datastream + BigQuery、Pub/Sub BigQuery subscription、streaming insert
「既存の Spark 資産を活かす」	互換性優先	Dataproc、Dataproc Serverless for Spark、BigQuery Spark stored procedure
「将来 3 倍に増える」	弾力性	サーバーレス(BigQuery、Dataflow、Pub/Sub)を優先
「他社クラウドにも同じ処理を置く」	可搬性	Apache Beam、Apache Spark、オープンフォーマット

将来要件(future business requirements) の扱いが Professional 級の特徴です。「いま必要ないが、将来必要になる」ものに対して、いま何を選んでおくかを問われます。原則は次の三つです。第一に、**データはオープンフォーマットで保持する**(Parquet、Avro、ORC、Apache Iceberg)。第二に、**処理はポータブルな抽象で書く**(Beam、Spark、標準 SQL)。第三に、**いま過剰投資しない**(将来の要件のために今から multi-cloud を作らない)。

「過剰設計を選ばない」ことは、誤答肢の切り方として重要です。「将来グローバル展開の可能性がある」という一文だけで Spanner の multi-region を選ぶのは通常誤りで、現在の要件が単一リージョンの OLTP なら Cloud SQL、明示的に「複数リージョンで強整合な書き込みが要る」と書かれて初めて Spanner です。

1-3-2 Designing for data and application portability(multi-cloud、data residency)[1.3]

portability(可搬性) は、データの可搬性と処理の可搬性に分かれます。

データの可搬性は**フォーマットと保存場所**で決まります。

フォーマット	性格	使いどころ
Avro	行指向、スキーマ同梱、圧縮に強い	BigQuery へのロード、スキーマ変更に強い連携
Parquet	列指向、分析に最適	データレイク、BigLake の外部テーブル
ORC	列指向、Hive 系エコシステム	Hadoop からの移行
Apache Iceberg	テーブル形式(スキーマ進化、time travel、ACID)	オープンなレイクハウス、BigLake tables for Apache Iceberg
CSV/JSON	単純、可読	少量・受け渡し。型の曖昧さと性能が弱点

BigLake は、Cloud Storage(や Amazon S3、Azure Blob Storage)上のオープンフォーマットのデータに、BigQuery と同じきめ細かいアクセス制御(行・列レベル)を適用しつつ、BigQuery・Spark・Dataflow の各エンジンから読めるようにする仕組みです。「データは Cloud Storage に置いたままにしたい、しかし列レベルの権限は要る」という要件の答えが BigLake です。**BigQuery Omni** は、Amazon S3 や Azure Blob Storage 上のデータをその場(データを移さずに)BigQuery の SQL で分析する仕組みで、「data residency の都合で他社クラウドからデータを出せないが分析したい」という要件に対応します。

処理の可搬性は**抽象レイヤ**で決まります。**Apache Beam** で書けば、実行基盤を Dataflow から Apache Flink や Apache Spark に差し替えられます(Beam の portability framework)。**Apache Spark** で書けば、Dataproc・Dataproc Serverless・他社クラウドの Spark で動きます。逆に、BigQuery 固有の SQL や Dataflow SQL に寄せると、移行時に書き直しが必要になります。可搬性と生産性のトレードオフを説明できることが求められます。

data residency(データ所在)要件下の multi-cloud では、次の型が出ます。

1. データを動かさず、クエリ側を各地域へ配る → BigQuery Omni、BigLake の外部テーブル
2. 集計結果だけを中央に集める → 各地域で集計し、匿名化・集約済みの結果のみ転送
3. メタデータだけを中央に集める → Knowledge Catalog(Dataplex)で横断カタログ、実データは各地域

「グローバルの売上を見たいが、各国の個人データは国外に出せない」なら 2 か 3 が正解で、「全データを US の BigQuery に集約する」は誤答です。

コンテナ化(GKE、Cloud Run)による application portability も、Dataproc 以外の処理を移す文脈で出ます。ただしデータエンジニアリングの文脈では、マネージドサービスを捨ててまで可搬性を取るのは通常過剰なので、要件文に「他社クラウドでも同じものを動かす」と明記されているかを確認します。

1-3-3 Data staging、cataloging、profiling、discovery(data governance)[1.3]

データガバナンスの四語は、それぞれ別の作業を指します。混同すると選択肢を取り違えます。

語	意味	Google Cloud での道具
staging	取り込んだ生データを、変換前に一時的に置く	Cloud Storage の landing/staging バケット、BigQuery の `staging_*` dataset、外部テーブル
cataloging	どこに何のデータがあるかを登録・検索可能にする	Knowledge Catalog(旧 Dataplex Universal Catalog / Data Catalog)、entry と aspect、business glossary
profiling	データの中身の統計(NULL 率、分布、ユニーク数)を測る	Dataplex の data profiling スキャン、BigQuery のクエリ、Dataprep
discovery	利用者が必要なデータを見つける	Knowledge Catalog の検索(semantic search、faceted search)、tag による絞り込み

staging 設計の勘所は、生データを不変(immutable)に保つことです。取り込んだままの形を Cloud Storage に日付プレフィックス付きで置き(`gs://bucket/raw/source=erp/dt=2026-09-07/`)、そこから変換します。こうしておくと、変換ロジックにバグが見つかったときに再処理(reprocessing)ができます。これが「バックフィルできる設計」の土台です。

cataloging では、Knowledge Catalog(Dataplex)が BigQuery の dataset・table・view・model、Cloud Storage のバケット、Pub/Sub のトピック、Spanner・Bigtable などのメタデータを自動収集します。**aspect**(旧 tag)で「データオーナー」「機密度」「保持期間」「SLA」などの独自メタデータを構造的に付けられます。**aspect type**(旧 tag template)でその形を定義します。

business glossary で用語(「アクティブユーザー」の定義など)を統一します。

data lineage(系統)は、どのテーブルがどのテーブルから作られたかを追う機能で、BigQuery のジョブや Dataflow の処理から自動的に収集されます。「この列の値がおかしい。どこから来たか」「このテーブルを消したら何が壊れるか(影響分析)」に答えるのが lineage です。監査対応でも使います。

discovery の設計では、検索されるためのメタデータの質が要点です。テーブル・列の description を埋める、aspect で機密度とオーナーを付ける、business glossary の用語を紐づける、という三点を運用に組み込みます。誤答肢は「スプレッドシートでデータ辞書を管理する」形で出ます。手作業の台帳は陳腐化するので、自動収集されるカタログが正解です。

なお 2026 年 4 月に **Dataplex Universal Catalog は Knowledge Catalog に名称変更**されましたが、API・CLI・IAM ロールの名称(`dataplex.googleapis.com`、`roles/dataplex.*`、`roles/datacatalog.*`)は変わっていません。exam guide(v4.2)は Dataplex / Dataplex Catalog 表記なので、本書もその表記を主に使います。

1-4 データ移行の設計 [1.4]

本節で扱う出題範囲の項目(公式 exam guide の原文)は次のとおりです。以下の本文は、この各項目に対応します。

- Analyzing current stakeholder needs, users, processes, and technologies, and creating a plan to get to desired state
- Planning migration and validation to Google Cloud (e.g., BigQuery Data Transfer Service, Database Migration Service, Transfer Appliance, Google Cloud networking, Datastream)

1-4-1 現状(stakeholders、users、processes、technologies)の分析と目標状態への計画 [1.4]

移行計画は、**現状分析 → 目標状態の定義 → 移行方式の選定 → 検証 → 切り替え → 撤収**の順で組みます。試験では「最初にやるべきことは何か」がよく問われ、答えは多くの場合**現状の棚卸しと関係者の要件確認**です。いきなり転送方式を選ぶ選択肢は誤りになります。

現状分析で洗い出す項目を挙げます。

観点	具体的に集めるもの
stakeholders(利害関係者)	データオーナー、規制対応の担当、予算の決裁者、業務部門の代表
users(利用者)	誰がどのレポートを毎日見ているか、SLA は何時までか
processes(業務プロセス)	日次バッチの締め時刻、月次決算、監査サイクル
technologies(技術)	既存 DWH の種類と容量、ETL ツール、BI ツール、依存するジョブ、ネットワーク帯域
データそのもの	総量、日次増分、スキーマ、機微データの有無、保持要件
制約	停止可能時間、データ主権、予算、移行期限

目標状態の定義では、**移行の戦略**を選びます。

戦略	内容	向く場面
lift and shift	ほぼそのまま移す(Hadoop → Dataproc、Oracle → Cloud SQL/Spanner)	期限が厳しい、まず動かしたい

戦略	内容	向く場面
move and improve	移しつつ一部をマネージドに置換(Hive → BigQuery、Oozie → Composer)	現実的な折衷
rebuild / rearchitect	サーバーレス前提で作り直す(Spark バッチ → BigQuery + Dataform)	長期の運用コストを下げたい

Hadoop 移行の典型解は次の対応です。HDFS → Cloud Storage(**ephemeral cluster** 化のため)、Hive → BigQuery(または Dataproc Metastore + BigLake)、Spark/MapReduce ジョブ → Dataproc または Dataproc Serverless for Spark、Oozie → Cloud Composer、HBase → Bigtable、Kafka → Pub/Sub(または Managed Service for Apache Kafka)。「まず HDFS を Cloud Storage に置き換え、クラスタをジョブごとに使い捨てにする」が最頻出の第一歩です。

移行は**段階的(phased)** に進めます。優先度の低いデータセットから移し、一定期間は**並行稼働(parallel run)** して結果を突合し、問題が無ければ切り替えます。ロールバック手順を決めておくことも要件に入ります。

1-4-2 Google Cloud への移行と検証の計画 —— BigQuery Data Transfer Service、Database Migration Service、Transfer Appliance、Google Cloud networking、Datastream [1.4]

移行の道具は、**何を・どれだけ・どれくらいの継続性で**動かすかで選びます。

道具	対象	一回きり/継続	選ぶ理由
BigQuery Data Transfer Service	SaaS(Google Ads、Google Analytics、YouTube、Search Ads 360)、Cloud Storage、Amazon S3、Amazon Redshift、Teradata、他 BigQuery dataset	継続(スケジュール)も一回も可	「Redshift/Teradata から BigQuery へ」「SaaS のデータを定期取り込み」
Database Migration Service(DMS)	MySQL、PostgreSQL、Oracle、SQL Server → Cloud SQL / AlloyDB / Spanner	一回 + 継続 CDC(最小停止)	「本番 DB を止めずに移行したい」
Datastream	Oracle、MySQL、PostgreSQL、SQL Server、MongoDB、Spanner、Salesforce 等 → BigQuery、Cloud Storage、Apache Iceberg	継続 CDC + backfill	「運用 DB の変更をニアリアルタイムで分析基盤に流したい」
Storage Transfer Service	他クラウド(Amazon S3、Azure Blob Storage)、HTTP/HTTPS、オンプレの POSIX ファイルシステム → Cloud Storage	一回/定期	「ネットワーク経由で大量のオブジェクトを移す」

道具	対象	一回きり/継続	選ぶ理由
Transfer Appliance	物理アプライアンスに書き込んで郵送	一回	「回線が細く、数百 TB〜数 PB を現実的な期間で送れない」
`gcloud storage cp` / `gsutil`	少量	一回	「数 GB のアドホック」
BigQuery Migration Service(SQL translation)	Teradata/Redshift/Snowflake 等の SQL を BigQuery SQL に変換	一回	「既存 SQL 資産の書き換え」

Transfer Appliance を選ぶ判断は、帯域と期間の計算です。試験ではこの計算を要求されます。目安として、1 Gbps の専有回線で 1 日に運べるのは約 10 TB です(実効はこれより落ちます)。「100 TB を 1 週間で」なら約 10 Gbps 相当が要るので、回線がそれ未満なら Transfer Appliance が正解になります。逆に「10 TB を 1 か月で、回線は 1 Gbps」ならネットワーク転送(Storage Transfer Service)で足ります。

Database Migration Service と Datastream の違いが最頻出の取り違えです。

	Database Migration Service	Datastream
目的	データベースそのものの移行(移行後は移行元を止める)	継続的な CDC レプリケーション(移行元は動き続ける)
宛先	Cloud SQL、AlloyDB、Spanner	BigQuery、Cloud Storage、Apache Iceberg

	Database Migration Service	Datastream
典型シナリオ	オンプレ MySQL → Cloud SQL への移行	本番 Oracle → BigQuery でのニアリアルタイム分析
完了	カットオーバーして終わる	継続的に流し続ける

「本番 OLTP を止めずに、分析用に BigQuery へ変更を流し続けたい」なら **Datastream**、「オンプレの PostgreSQL を Cloud SQL に引っ越したい」なら **Database Migration Service** です。

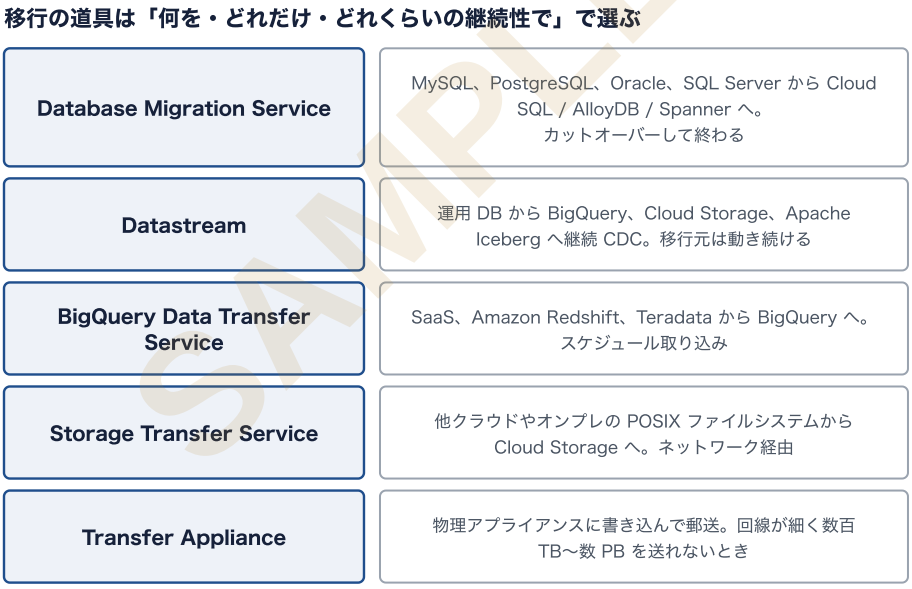


図 1-6　移行の道具は「何を・どれだけ・どれくらいの継続性で」で選ぶ

Google Cloud networking の選択肢も移行計画の一部です。

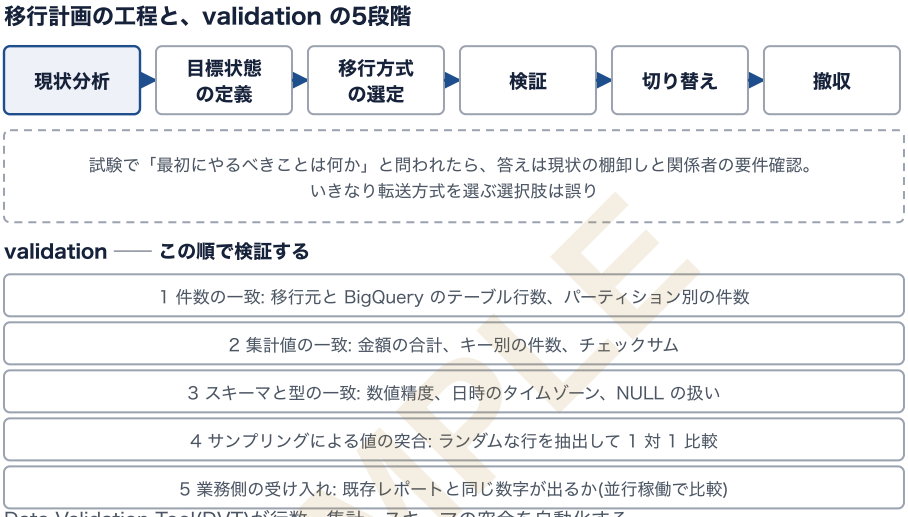
手段	帯域・性質	選ぶ場面
公衆インターネット	可変・共有	少量、機微でない
Cloud VPN(HA VPN)	IPsec、1 トンネルあたり数 Gbps 程度	暗号化した接続がすぐ要る
Dedicated Interconnect	10/100 Gbps の専有	大量・継続・低レイテンシ。コロケーション施設が要る
Partner Interconnect	50 Mbps〜50 Gbps	Interconnect の設備要件を満たせない場合
Cross-Cloud Interconnect	他社クラウドとの専有接続	multi-cloud 構成
Direct Peering / Carrier Peering	Google のエッジと直接	特定用途

加えて、**Private Google Access**(外部 IP を持たない VM から Google API へ)、**Private Service Connect**(サービスへの私設エンドポイント)、**VPC Service Controls**(境界)を組み合わせて、移行トラフィックをインターネットに出さない設計にします。

validation(検証) の計画は、移行設計の採点で必ず問われます。次の順で検証します。

1. 件数の一致: 移行元と BigQuery のテーブル行数、パーティション別の件数
2. 集計値の一致: 金額の合計、キー別の件数、チェックサム(ハッシュの集約)
3. スキーマと型の一致: 数値精度(`NUMERIC` / `BIGNUMERIC`)、日時のタイムゾーン、NULL の扱い
4. サンプリングによる値の突合: ランダムな行を抽出して 1 対 1 比較
5. 業務側の受け入れ: 既存レポートと同じ数字が出るか(並行稼働で比較)

Data Validation Tool(DVT) という Google 提供のオープンソース道具が、移行元 DB と BigQuery の間で行数・集計・スキーマの突合を自動化します。手作業の目視確認を選ぶ選択肢は誤りです。



Data Validation Tool(DVT)が行数・集計・スキーマの突合を自動化する。
手作業の目視確認は誤り。

図 1-7　移行計画の工程と、validation の5段階

タイムゾーンと型の扱いは実務のはまりどころで、試験にも出ます。BigQuery の `TIMESTAMP` は UTC の絶対時刻、`DATETIME` はタイムゾーンを持たない壁時計時刻です。移行元が現地時間で持っていたものを無変換で `TIMESTAMP` に入れると、集計がずれます。金額は浮動小数点(`FLOAT64`)ではなく `NUMERIC` / `BIGNUMERIC` を使う、というのも定番の正解です。

サンプルはここまでです

続き(第2章〜巻末)は、GCP PDE 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 PDE 学習用テキスト(無料サンプル)

版

2026年7月版(Google Cloud 公式 Professional Data Engineer 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