IT資格道場

DP-420 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)

DP-420 出題範囲(Skills measured)2026-10-06版 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

この教材の使い方

試験の基本情報

試験の設計図：3 分野と配点

構成

4 ステップ学習法

第1章 Design and develop database solutions [1.1.1/1.1.2/1.1.3/1.1.4/1.1.5/1.1.6/1.2.1/1.2.2/1.2.3/1.2.4/1.2.5/1.2.6/1.3.1/1.3.2/1.3.3/1.3.4/1.3.5/1.3.6/1.3.7/1.3.8/1.4.1/1.4.2/1.4.3/1.4.4/1.4.5/1.5.1/1.5.2]

整合性レベルを評価する [1.1.1]

要求ユニット (RU) の消費要件を評価する [1.1.2]

適切なスループットの評価と構成 [1.1.3]

アカウント レベルとコンテナー レベルの設定を構成する [1.1.4]

データベースとコンテナーの作成と管理 [1.1.5]

SDK クライアントと接続を初期化する [1.1.6]

クエリ、パラメーター化クエリ、ページング、継続トークンを実装する [1.2.1]

読み取り操作（ポイント読み取り）を実装する [1.2.2]

作成、置換、更新、修正、削除の各操作を実装する [1.2.3]

一括操作、バッチ操作、トランザクション操作を実装する [1.2.4]

Time to Live (TTL) の構成 [1.2.5]

コンカレンシー制御メカニズムを実装する [1.2.6]

アクセス パターンと運用要件に合わせてデータ モデルを設計する [1.3.1]

カーディナリティ、アクセス パターン、書き込みディストリビューションの要件に基づいてパーティション キーを設計して選択する [1.3.2]

関連データを埋め込む、または参照するタイミングを決定する [1.3.3]

SAMPLE

パーティション キーの選択がスケーラビリティとパフォーマンスに与える影響を評価する [1.3.4]

階層パーティション キー (HPK) とマルチテナント パーティション分割戦略を実装する [1.3.5]

合成パーティション キーを実装する [1.3.6]

ソリューション レビュー中にパーティション分割のアンチパターンとパーティション分割の問題を特定する [1.3.7]

ワークロードのコスト要件を見積もる [1.3.8]

変更フィード操作を実装する [1.4.1]

変更フィード プロセッサ モデルとプル モード モデルを区別する [1.4.2]

リース コンテナーを構成する [1.4.3]

最新モードまたはすべてのバージョンを使用して変更フィード処理を実装し、削除する [1.4.4]

コンテナー のコピー ジョブを実装する [1.4.5]

エージェント キットを実装して、Azure Cosmos DB開発に推奨されるソリューションに従う [1.5.1]

エージェント キットを実装して AI アプリケーション開発を高速化する [1.5.2]

サンプルはここまでです

SAMPLE

3 冊の使い方は、ダウンロード欄の案内を参照してください。

SAMPLE

この教材の使い方

このテキストの目的は、Microsoft の Associate 資格への合格です。

Microsoft Certified: Azure Cosmos DB AI Developer Associate(試験コード DP-420、試験名 Building AI Applications with Azure Cosmos DB。旧名は Azure Cosmos DB Developer Specialty)

DP-420 は、**Azure Cosmos DB for NoSQL を使うアプリの開発者の視点**で問われる資格です。製品名の暗記は不要。問われるのは、「この要件なら、どの設定・API・クエリか」です。

出題は 3 分野。

- データ モデルとパーティション キー、読み書きの API、変更フィード
- 認証・ネットワーク・バックアップ・性能の監視・フリート管理
- フルテキスト検索、ベクター検索、エージェントのメモリ、Fabric との連携

深い NoSQL の経験は要りません。押さえるのは 2 点です。

- 各機能の「何をする道具か」
- 「選ぶ条件・設定の勘どころ」

問われる中心は、この 2 点です。C# や Python、SQL のコード例は出ますが、**丸暗記は不要。**「どの設定やメソッドが何をするか」が読めれば十分です。

SAMPLE

試験の基本情報

項目	内容
試験名	Microsoft Certified: Azure Cosmos DB AI Developer Associate(DP-420)
出題数	50 問
出題形式	択一・複数選択のほか、並べ替え、ホットエリア、アクティブ画面、ケーススタディ、「問題と解決策」(同じ状況に対して解決策だけを差し替え、それぞれに「はい」「いいえ」で答える形式)が含まれることがあります。試験ごとの形式は事前に公表されません
試験時間	100 分
合格ライン	1,000 点満点中 **700 点**(スケールスコア)
受験料	20,300 円(日本での価格。税の扱いは申込時に確認してください)
受験方法	ピアソン VUE。テストセンターまたはオンライン監督試験
言語	日本語で受験可(英語ほか多言語提供)
資格の有効期限	取得後 12 か月(Microsoft の無料オンライン評価で更新)

分からない問題も、必ずどれかを選んで埋めること。受験料や実施形式は変わることがあります。申し込み時に、公式ページの最新情報を必ず確認してください。

SAMPLE

試験の設計図：3 分野と配点

このテキストが準拠する Skills measured は **2026 年 10 月 6 日版**です。この版から、資格名は Azure Cosmos DB AI Developer Associate、段は Associate です。

分野	配点	一言でいうと
① データベース ソリューションの設計と開発	**40〜45%**	設計して作る —— モデル・パーティション キー・読み書き・変更フィード
② データベース ソリューションのセキュリティ保護・最適化・デプロイ	**30〜35%**	守って速く保つ —— 認証・バックアップ・性能・監視・フリート
③ データベース ソリューションでの AI と分析機能の実装	**20〜25%**	AI につなぐ —— 検索・エージェントのメモリ・Fabric

①が最大で、全体の約 4 割。②が約 3 割、③が約 2 割で続きます。

配点の偏りは小さくありません。①を先に固め、次に②、最後に③の順で仕上げると効率的です。ただし③は新しい版で加わった範囲で、設定名や上限の数字がそのまま問われます。捨てずに、名前を読んで中身が浮かぶ状態にしてください。

構成

このテキストは 3 分野を 3 つの章に分けた構成です。同じ機能が複数の分野にまたがることもあります(TTL は①の設定にも③のメモリにも出ます)。並べ方は**公式の分野順そのまま**。

- 第 1 章 … 分野①(データベース ソリューションの設計と開発)

SAMPLE

- 第 2 章 … 分野②(セキュリティ保護・最適化・デプロイ)
- 第 3 章 … 分野③(AI と分析機能の実装)

見出しの末尾の [1.1.1] などの数字は、公式の出題範囲の番号です。

4 ステップ学習法

購入者限定テキストは 3 冊で 1 セット。次の順番で使うと最短です。

- **STEP 1(理解する)**:「学習用テキスト」を通読し、Cosmos DB の地図と「選ぶ条件・設定の勘どころ」を頭に入れる
- **STEP 2(解き方を掴む)**:「頻出度別ノート」で頻出度 A の論点を解いて、解き方を掴む
- **STEP 3(腕試し)**: 本サイトの問題演習で出題形式に慣れる
- **STEP 4(再確認)**:「頻出度別ノート」に戻って頻出度 A の論点を再確認する(試験直前もここ)

「暗記用ノート」は STEP には数えず、空き時間や用語の再確認に使ってください。

DP-420 は操作と設定を問う試験です。可能なら Azure の無料アカウントかエミュレーターで Cosmos DB を作り、実際に触ると定着が段違い。時間がなければ、先に「頻出度別ノート」で頻出度 A の論点を確認。それを含む章から読み始めても構いません。

SAMPLE

第1章 Design and develop database solutions [1.1.1/1.1.2/1.1.3/1.1.4/1.1.5/1.1.6/1.2.1/1.2.2/1.2.3/1.2.4/1.2.5/1.2.6/1.3.1/1.3.2/1.3.3/1.3.4/1.3.5/1.3.6/1.3.7/1.3.8/1.4.1/1.4.2/1.4.3/1.4.4/1.4.5/1.5.1/1.5.2]

この章は試験の 40〜45% を占め、Azure Cosmos DB for NoSQL を対象にします。データ モデルとパーティション キー、読み書きの API、変更フィード、AI コーディング支援を扱います。

整合性レベルを評価する [1.1.1]

整合性レベルは、複数のレプリカ（複製）があるデータを読んだ時に、どこまで新しい値が返るかを決める設定です。強い順に 5 つあります。Strong、Bounded staleness、Session、Consistent prefix、Eventual の順です[1]。

レベル	読み取りの保証	読み取りの RU
Strong	線形化可能性。常に最新のコミット済みの版を返す	他の約 2 倍
Bounded staleness	遅れを K 版（更新回数）か T 時間のどちらか先に届いた方で抑える	他の約 2 倍
Session	1 つのクライアント セッションの中で、自分の書き込みを読め（read-your-writes）、読んだ後の書き込みが順序どおりになる（write-follows-reads）	基準

SAMPLE

レベル	読み取りの保証	読み取りの RU
Consistent prefix	更新が書いた順序のまま見える。トランザクション内の複数項目は常にまとまって見える	基準
Eventual	順序の保証なし。4 つのレプリカのうち 1 つだけに読みを出すので、古いデータや無いデータが返りうる	基準

新しいアカウントの既定は Session で、アカウント内のすべてのデータベースとコンテナーに効きます[2]。書き込みの RU は 5 レベルすべて同じです。Strong と Bounded staleness だけは、読み取りが 2 つのレプリカを見ます。そのため、同じ RU で出せる読み取りのスループットは、他のレベルの半分です[1]。

Bounded staleness の遅れの下限は構成で変わります。単一リージョンのアカウントは、最小 10 書き込みまたは 5 秒です。複数リージョンのアカウントは、最小 100,000 書き込みまたは 300 秒です[1]。

Strong には 2 つの制約があります。書き込みリージョンが複数あるアカウントでは使えません。リージョン間の距離が 5,000 マイル（8,000 km）を超えるアカウントでは、既定でブロックされます（解除はサポートへの連絡）[1]。

Session の保証はセッション トークンに乗っています。トークンは 1 つのパーティションにだけ結び付きます[1]。複数のクライアント（Web サーバーが複数台など）が同じ流れで読み書きする時は、トークンを渡し合います。渡さないと、別のクライアントでは自分の書き込みが見えないことがあります[1]。

要求ごとの上書きは、弱くする方向だけです。SDK のインスタンスか個々の要求で、既定より弱いレベルを指定できます。強くしたい時はアカウントの既定を

SAMPLE

変えます。上書きが効くのは読み取りだけで、書き込みのレプリケーション方式は変わりません[2]。.NET v3 では `ItemRequestOptions` の `ConsistencyLevel` に指定します。例は `ConsistencyLevel.Eventual` です。強くする方向も指定できる `ReadConsistencyStrategy` もあります。Java 4.69 以上と .NET 3.46 以上のプレビュー機能で、Direct モードだけで動きます[2]。

場面で考えます。既定が Session のアカウントで、一部の読み取りだけ RU を下げたい時は、その要求だけ Eventual に弱めます。逆に、特定の読み取りだけ Strong にしたい時は、要求では強められません。アカウントの既定を Strong にして、それ以外を要求ごとに弱めるか、`ReadConsistencyStrategy` を検討します。複数リージョン書き込みのアカウントでは Strong が選べないので、選べる最も強いレベルは Bounded staleness です。ただし公式は、複数リージョン書き込みのアカウントでの Bounded staleness をアンチパターンとしています。アプリ サーバーは、自分と同じリージョンで読み書きします[1]。

要求ユニット (RU) の消費要件を評価する [1.1.2]

要求ユニット（RU）は、CPU・IOPS・メモリを 1 つの単位にまとめた性能の通貨です。スループットは 1 秒あたりの RU（RU/s）で測ります[3]。ID とパーティション キーで 1 項目を読むポイント読み取りは、約 1 KB の項目で 1 RU です。100 KB の項目なら 10 RU になります。ポイント読み取りの RU を決めるのは、整合性レベルを別にすれば項目のサイズだけです[4]。

RU を増やす要素は 4 つです。項目が大きいこと、索引の対象が多いこと、Strong か Bounded staleness の読み取り、クエリが複雑なことです。索引は既定ですべてのプロパティが対象なので、絞ると書き込みの RU が下がります。クエリの複雑さは、述語の数、UDF の数、結果の大きさで決まります[3]。同じ

SAMPLE

データへの同じクエリは、繰り返しても毎回同じ RU になります。ユニーク キーのポリシーがあると、作成・更新・削除の RU がやや増えます[5]。

読み取りの効率は、ポイント読み取り、単一パーティションのフィルター付きクエリ、フィルターなしのクエリの順に悪くなります[4]。複数リージョンでは、コンテナーに R RU/s を割り当てると N 個のリージョンのそれぞれで R が使えます。合計は R × N で、リージョンごとに別々の値は割り当てられません[3]。

RU の測り方は 3 つです。.NET v3 では応答の `RequestCharge` プロパティを読みます。HTTP の応答ヘッダーなら `x-ms-request-charge` です[6]。ポータルでは Data Explorer で新しい SQL クエリを実行し、Query Stats で実際のリクエスト チャージを見ます。

429 と再試行。割り当てた RU を超えると、サービスは HTTP 429（要求が多すぎる）を返します。SDK は 429 を既定で自動再試行し、応答の `x-ms-retry-after-ms` の時間だけ待ちます。再試行の上限を超えたら、エラーがアプリに返ります[7]。429 は再試行して安全なエラーです。404（見つからない）や 409（競合）は再試行しても結果が変わらないので、SDK は再試行しません[7]。

モニターの指標は**正規化 RU 使用量**（Normalized RU Consumption）です。0〜100% の値で、全パーティション キー範囲（物理パーティション 1 つに対応）のうち、最も使っているものの割合を 1 分ごとに出します。たとえば最大 20,000 RU/s のオートスケールで物理パーティションが 2 つなら、それぞれ 1,000〜10,000 RU/s で動きます。ある秒に P1 が 6,000、P2 が 8,000 を使うと、P1 は 60%、P2 は 80%、コンテナー全体は大きい方の 80% です[8]。

100% に張り付いても、すぐに増設とは限りません。本番の負荷で 429 の割合が 1〜5% で、全体の待ち時間が許容範囲なら、RU/s を使い切れている健全

SAMPLE

な状態です。100% が複数のパーティション キー範囲で続き、429 が 5% を超える時に、スループットを増やします[8]。

適切なスループットの評価と構成 [1.1.3]

スループットには 3 つの種類があります。手動（Provisioned）は RU/s を固定で決め、100 RU/s 刻みで増減します[3]。Autoscale は最大 RU/s（Tmax）だけ決め、負荷に応じて 0.1 × Tmax から Tmax の間を自動で動きます。Serverless は RU/s を決めず、使った RU に応じて課金されます。

項目	手動	Autoscale	Serverless
最小	400 RU/s	Tmax は 1,000 RU/s から（100〜1,000 の範囲で動く）	決めない
刻み	100 RU/s	Tmax は 1,000 RU/s 刻み	なし
課金	割り当てた RU/s を 1 時間ごと	その 1 時間内にスケールした最高の RU/s を 1 時間ごと	使った RU
リージョン	複数可	複数可	単一リージョンのみ

手動の最小値は、400、現在のストレージ GB × 1 RU/s、過去の最大 RU/s ÷ 100 のうち最大のものです。Autoscale の最小の Tmax は、次の 3 つのうち最大のものです。1,000、ストレージ GB × 10 RU/s、過去の最大 RU/s ÷ 10 です[9]。Autoscale は Tmax につき 0.1 × Tmax GB まで保管でき、超えると Tmax が自動で上がります[10]。

SAMPLE

Autoscale の単価は手動の 1.5 倍です。そのため、月のうち最大値を使う時間が 66% 以下なら Autoscale が得で、66% を超えるなら手動が得です[11]。負荷が読めない、または山と谷が大きいなら Autoscale、一定なら手動です。Serverless は、アクセスが少ない開発やテストの用途向けです。使った RU とストレージの分だけ課金されます[12]。コンテナーの作成時にスループットを渡すとエラーになります。共有スループットのデータベースも作れません[12]。Serverless のコンテナーは 5,000 RU/s から始まり、物理パーティションあたり最大 5,000 RU/s です[12]。

専用と共有。スループットは、コンテナー単位（専用）か、データベース単位（共有）で割り当てます。専用はそのコンテナーに SLA が付き、ほとんどのワークロードで推奨されます。共有は、データベース内のコンテナーで RU/s を分け合い、個々のコンテナーへの保証はありません。共有のデータベースは最初の 25 コンテナーまでです。それを超えるコンテナーは、専用のスループットを付ければ追加できます。データベース単位の Autoscale では、最大 RU/s の最小値 1,000（100〜1,000 RU/s で動く）のデータベースに、最大 25 コンテナーまで置けます[13]。専用のコンテナーを共有に変える、またはその逆はできません。必要ならデータを新しいコンテナーへコピーします[13]。

上限は、コンテナーとデータベースで最大 1,000,000 RU/s です（サポートで引き上げ可）。物理・論理のパーティションは 1 つあたり 10,000 RU/s までです[9]。2024 年 9 月 25 日以降に作ったアカウントでは、動的スケーリングが既定で有効です。リージョンやパーティション単位で動きます。Serverless は対象外です[10]。

バースト容量は、使わなかった分を物理パーティションが最大 5 分ためておき、最大 3,000 RU/s で使う仕組みです。保証はなく、Serverless では使えず、アカウント単位で有効にします[14]。

SAMPLE

優先度ベースの実行は、負荷が高い時に低優先度の要求を先に抑える仕組みです。ベストエフォートで SLA はありません。.NET では `ItemRequestOptions.PriorityLevel` に High か Low を指定します。Python は `priority="High"/"Low"` の引数です。優先度を付けない要求は、既定では高優先度として動きます（アカウントの既定の優先度は変えられます）。Serverless では使えず、Python SDK ではプレビュー扱いです。共有スループットのデータベースのコンテナーでは、動きが決まりません[15]。バックグラウンドの一括処理を Low にして、画面用の読み取りを守る使い方です。

スループット バケットは、要求をグループに分けて、グループごとに使える RU/s の上限を決めるプレビュー機能です。コンテナーあたり最大 5 バケットで、`ItemRequestOptions.ThroughputBucket` で割り当てます[16]。優先度は「どちらを先に抑えるか」、バケットは「各グループに上限を付ける」仕組みです。

SDK の操作は 3 つです。`Container.ReadThroughputAsync` で現在値を読みます。`ThroughputResponse.MinThroughput` で最小値を知ります。`Container.ReplaceThroughputAsync` で変更します[13]。引き上げに時間がかかっている最中にもう一度変更すると、HTTP 423 が返ります[13]。

アカウント レベルとコンテナー レベルの設定を構成する [1.1.4]

アカウントの設定は、容量モード、リージョン、複数リージョン書き込み、既定の整合性、Free Tier などです。容量モード（Provisioned throughput か Serverless）はアカウントの作成時に選びます[17]。NoSQL では、Serverless から Provisioned throughput へは後から変えられます。逆には戻せません[18]。API は NoSQL、MongoDB、Gremlin、Table、Cassandra の 5 種類です。アカウント名は 3〜44 文字で、小文字・数字・ハイフンだけ使え、グローバルで一意です。複数リージョン書き込みは、ポータルの Replicate data globally の

SAMPLE

構成で有効にします。Multi-region writes にチェックを入れます[2]。1 つのサブスクリプションに作れるアカウントは既定で 250 で、サポートへの依頼で 1,000 まで増やせます[9]。

Free Tierは、アカウントごとに最初の 1,000 RU/s と 25 GB が無料です。超えた分は通常の料金です。サブスクリプションあたり 1 アカウントまでで、作成時に選びます（後から変えられません）。有効化の指定は、作成の道具ごとに違います。CLI は `--enable-free-tier true` を使います。PowerShell は `-EnableFreeTier $true`、ARM は `enableFreeTier` です。Serverless のアカウントには使えません[19]。

コンテナーの設定は、パーティション キー、索引ポリシー、既定の TTL、ユニーク キー、競合解決です。試験が突くのは、**作った後に変えられる設定と変えられない設定**の区別です。

設定	作成後の変更
パーティション キー	変更できない。新しいコンテナーにコピーして移す[20]
ユニーク キー	変更できない（作成時だけ定義）[5]
競合解決（Custom）	作成時だけ設定できる[21]
階層パーティション キー	作成時だけ設定できる（1.3.5）
索引ポリシー・既定の TTL・スループット	変更できる

ユニーク キーは論理パーティションの範囲で一意性を保ちます。ポリシーは最大 16 パス、制約（組み合わせ）は最大 10 個です。値が無い項目は null として扱われ、null の項目は 1 つだけ許されます（スパースなユニーク キーは非対応）[5]。重複すると 409 になります。

SAMPLE

競合解決のポリシーは、複数リージョン書き込みで同じ項目が衝突した時の勝ち方を決めます。既定は Last Write Wins で、システムが持つ `_ts` の大きい方が勝ちます。NoSQL API だけ、数値のカスタム パス（conflict resolution path）を指定できます。Custom（マージのストアド プロシージャ）も NoSQL だけで、未登録や例外の時は競合フィードに書かれます[21]。索引は、追加したデータが既定ですべて自動で索引されます[22]。

その他の上限は、コンテナーあたりのユニーク キー数が 10、TTL の最大値が 2,147,483,647 です[9]。

データベースとコンテナーの作成と管理 [1.1.5]

データベースは名前空間のようなもので、コンテナーをまとめる管理の単位です。コンテナーの作成にはパーティション キーが必須で、共有スループットのデータベース内でも同じです[2013]。

.NET の作成メソッドは 2 組あります。`CreateDatabaseAsync` と `Database.CreateContainerAsync` は、すでに同じ名前があると例外になります[23]。`CreateDatabaseIfNotExistsAsync` と `CreateContainerIfNotExistsAsync` は、無ければ作ります。あればそれを返します。戻り値はそれぞれ `DatabaseResponse` と `ContainerResponse` です。実体は `.Database` と `.Container` で取ります[2423]。コンテナーの作成では、パーティション キーのパスとスループットを渡します。Python は `DatabaseProxy.create_container(id=..., partition_key=PartitionKey(path=...), default_ttl=...)` で作ります。クライアントの階層は、`CosmosClient` から `get_database_client()`、そして `get_container_client()` へと下ります[2526]。

SAMPLE

Azure CLI では、`az cosmosdb sql database create` に 3 つ渡します。`-a` がアカウント、`-g` がリソース グループ、`-n` が名前です。`--throughput` を付けると共有スループットのデータベースになります。コンテナーは `az cosmosdb sql container create` で作ります。`-p` にパーティション キーのパスを渡し、`--throughput`（手動）か `--max-throughput`（Autoscale）を付けます[27]。

作成後の変更は、スループットが `ReplaceThroughputAsync` で、索引ポリシーが `ReplaceContainerAsync` です[28]。変えられないものは、パーティション キー、専用か共有かの区別、名前です。変えたい時は新しいコンテナーを作ってコピーします（1.4.5 のコピー ジョブは、名前の変更もその用途に挙げています）。

`CreateIfNotExists` はセットアップの時に 1 回だけ呼びます。毎回の要求の前に呼ぶと、メタデータの操作がシステム予約の RU を消費します。アカウントのメタデータ操作の上限は 240 RU/s です[9]。データベース名・コンテナー名は、起動時にキャッシュするか設定から取ります[29]。名前の長さの上限は、SDK のガイドが 256 文字以下、limits のページの表が 255 です。255 までなら両方に収まります[239]。アカウントあたりのデータベースとコンテナーの合計は 500 までで、増やせません[9]。

SDK クライアントと接続を初期化する [1.1.6]

`CosmosClient` は、アプリの生存期間で 1 つだけ作って使い回します。接続はクライアント単位で張られるため、要求のたびに作ると接続が増えて遅くなります[29]。Python も同じです。非同期は `azure.cosmos.aio.CosmosClient` で、明示的に閉じます（`async with` を推奨）。`aiohttp` は別にインストールします[30]。

認証は Microsoft Entra ID の資格情報が推奨で、.NET では `DefaultAzureCredential` を渡します。Python は、先に `pip install azure-`

SAMPLE

`cosmos` を実行します。その後、`CosmosClient(ENDPOINT, credential)` に `azure-identity` の `DefaultAzureCredential` を渡します[3132]。

接続モードは 2 つです。

	Direct	Gateway
通信	TCP	HTTPS
対応 SDK	.NET と Java	すべての SDK
特徴	高性能でホップが少ない	ホップが 1 つ多い。ファイアウォールが厳しい環境向け
必要なポート	10000〜20000（プライベート エンドポイントは 0〜65535）	SQL は 443

.NET v3 の既定は Direct です。切り替えには `CosmosClientOptions.ConnectionMode = ConnectionMode.Gateway` を設定します[33]。Direct のポートが閉じていると 503 になります[34]。Direct でも、コンテナーの情報（パーティション キーの定義など）とルーティング情報は、ゲートウェイのノードから HTTP で取ります[3429]。Gateway のポートは API で違います。SQL・Table・Graph は 443、MongoDB は 10255、Cassandra は 10350 です[34]。

優先リージョンの指定は、.NET v3 が `ApplicationPreferredRegions`（または `ApplicationRegion`）、Python が `preferred_locations` です。Python の既定は None です。フェイルオーバー時の読み取りは、先頭のリージョンへ向かいます[2930]。

クライアントのオプションで押さえるのは 4 つです。429 の再試行は、`MaxRetryAttemptsOnRateLimitedRequests`（回数）と

SAMPLE

`MaxRetryWaitTimeOnRateLimitedRequests`（待ち時間）で設定します[35]。バルクは`AllowBulkExecution = true`（ビルダーでは`WithBulkExecution(true)`）で有効にします。Direct の詳細設定の既定は、次のとおりです。`MaxRequestsPerTcpConnection` は 30、`MaxTcpConnectionsPerEndpoint` は 65535 です。`OpenTcpConnectionTimeout` は 5 秒、`IdleTcpConnectionTimeout` は無期限です。Gateway の `GatewayModeMaxConnectionLimit` の既定は 50 です[33]。Python SDK は、読み取りとクエリは再試行しますが、書き込みは冪等ではないため、一時的なエラーでも再試行しません[30]。

クエリ、パラメーター化クエリ、ページング、継続トークンを実装する [1.2.1]

NoSQL のクエリは ANSI SQL に近い書き方で、**SELECT 句が必須**、FROM と WHERE は省略できます[36]。`SELECT *` が使えるのは、FROM 句で別名がちょうど 1 つ宣言されている時だけです。`SELECT VALUE` は、結果をオブジェクトで包まず、JSON の値そのものを返します。たとえば `SELECT VALUE p.name FROM products p` は文字列の配列を返します。`SELECT p.name FROM products p` は `{"name": ...}` の配列を返します[36]。

JOIN は、1 つの項目の中でだけ働きます。ルート項目と入れ子の配列を結合する自己結合で、コンテナーや別の項目をまたぐ結合はできません。まとめて読みたい関係は、データ モデルの段階で埋め込みにします[37]。サブクエリは相関と非相関の両方があり、`EXISTS` は、サブクエリが 1 行でも返せば true を返す式です。SELECT と WHERE のどちらにも置けます[38]。

組み込み関数は、数学、配列、集計、文字列、日付と時刻、項目、全文検索、条件、型の判定、空間の各区分にあります。出題で見かける関数は次のとおりです。`CONTAINS`・`STARTSWITH`・`ENDSWITH` は、文字列が含む・前方一致・後方一致

SAMPLE

するかを返します。`ARRAY_CONTAINS` は、配列に値があるかを返します。集計は `COUNT` や `MAX` です[39]。

パラメーター化クエリは、値を `@` で始まる名前で書き、値は別に渡します。SQL インジェクションを防げます。名前を渡す時も先頭の `@` を含めます。パラメーターの型は検証されません[40]。.NET は、`QueryDefinition` に `.WithParameter("@name", value)` を足します。それを `GetItemQueryIterator` に渡します[41]。Python は `container.query_items(...)` に、`query=`、`parameters=`、`enable_cross_partition_query=` の 3 つを渡します。`parameters` は `[dict(name="@category", value=...)]` の形のリストです[26]。

ページング。`GetItemQueryIterator<T>` は `FeedIterator` を返します。外側の `while (iterator.HasMoreResults)` で 1 ページずつ `ReadNextAsync` します。内側の `foreach` で、そのページの項目を回します[41]。LINQ の `GetItemLinqQueryable` で作った `IQueryable` を直接回すと同期になります。非同期にするには `ToFeedIterator` を使います[41]。

1 ページの項目数は `QueryRequestOptions.MaxItemCount` で決め、-1 なら SDK が最適値を選びます。ページングを実装するためだけに使うものではありません[42]。Python の `max_item_count` の既定は 100 項目 / 4 MB です[30]。1 回のクエリ ページは最大 5 秒、応答は最大 4 MB で、どちらかに達すると、そこまでの結果と**継続トークン**が返ります。クライアントはそのトークンを渡して再開します[9]。

`OFFSET <件数> LIMIT <件数>` は、先頭の指定件数を飛ばして、続く指定件数を返す構文です。ORDER BY と組み合わせて使います[43]。ページ送りの手段は、要件で選びます。「前回の続き」だけが要るなら継続トークンです。「N 件目から」という位置指定が要るなら OFFSET LIMIT です。

SAMPLE

パーティションをまたぐクエリは、WHERE にパーティション キーの条件が無く、全物理パーティションに問い合わせるクエリです。パーティション キーを含めないクエリは、物理パーティション 1 つにつき 2〜3 RU の上乗せがあります[20]。クロスパーティションのクエリは、`MaxConcurrency` で並列に訪ねるパーティションの最大数を決めて動かします（-1 なら SDK が最適な並列度を決めます）。`MaxBufferedItemCount` で、先読みする件数を抑えます[42]。単一パーティションのクエリは、WHERE でパーティション キーを等価で指定します。

読み取り操作（ポイント読み取り）を実装する [1.2.2]

ポイント読み取りは、項目の **id とパーティション キー値の両方**を渡して 1 件だけ読む操作です[44]。id しか渡さなければ、全パーティションを探すクロスパーティションのクエリになり、効率が落ちます。RU は 1 KB の項目で 1 RU、100 KB なら 10 RU です。決めるのは項目サイズと整合性レベルです。整合性が Strong か Bounded staleness の時は 2 倍になります[4]。

間違えやすいのが、`SELECT * FROM c WHERE c.id = "x" AND c.pk = "y"` のようなクエリです。1 件を id とパーティション キーで絞っても、ポイント読み取りとは見なされず、RU は高くなります。ポイント読み取りは REST か SDK の読み取り操作だけです[4]。

.NET のメソッドは 3 つです。`ReadItemAsync<T>(id, partitionKey)` は、`ItemResponse<T>` を返します。この応答の `.RequestCharge` が RU、`.StatusCode` が HTTP の状態、`.Resource` が項目の本体です[44]。`ReadItemStreamAsync<T>` は、デシリアライズせずに `Stream` で返すので、性能の面で有利です。`ReadManyItemsAsync<T>` は、(id, PartitionKey) の組のリストで複数の項目をまとめて読みます。多数の独立した項目を取る時に、IN 句のクエリより待ち時間の点で有利になるよう作られています[44]。存在しない項目

SAMPLE

を `ReadItemAsync` で読むと、404（Resource not found）になります。404 は SDK が再試行しない状態コードです[7]。Python は `container.read_item(item=id, partition_key=pk)` です[15]。

id の決まりは 2 つあります。大文字と小文字を区別します（`ID` や `Id`、`_id` は id とは別の普通のプロパティです）。長さは最大 1,023 バイトです[459]。`_etag` を `if-none-match` ヘッダーと組み合わせると、再取得が要るかどうかを判定できます[46]。

作成、置換、更新、修正、削除の各操作を実装する [1.2.3]

新しい JSON 項目で必須のプロパティは **id だけ**です[45]。

操作	動き	既存の id の時
`CreateItemAsync`	新規作成	409 Conflict
`ReplaceItemAsync`	全体を置き換え。渡す id が項目の id と一致する必要がある	無いと失敗
`UpsertItemAsync`	無ければ作成、あれば置換	置換する
`DeleteItemAsync`	削除（id と `PartitionKey`）	

409 は、同じ id とパーティション キーの項目が既にある時と、ユニーク キーに違反した時の両方で返ります[7]。項目のサイズは最大 2 MB（JSON の UTF-8 の長さ）で、入れ子は最大 128 階層です[9]。パーティション キーの値は、作成後に変えられません。置換でも別のパーティションへ移れないので、変えるなら削除してから作り直します[20]。

SAMPLE

Patch（部分更新）は、変えるフィールドだけを送る API です。JSON Patch の RFC に基づき、パスの中の `~` と `/` は `~0` と `~1` にエスケープします[47]。操作は 6 種類です。

操作	動き
Add	無ければ追加。配列では挿入
Set	値を設定。Add と同じく、無ければ追加。違うのは配列で、既存の要素を更新
Replace	置き換え。対象が無いとエラー（必須プロパティの検証向き）
Remove	削除
Increment	正負の数だけ増減。フィールドが無ければ作る
Move	元の位置から値を取り除き、移動先に加える。`from` が要る

1 回の Patch で実行できる操作は最大 10 です[47]。.NET は `Container.PatchItemAsync<T>(id, partitionKey, patchOperations)` です。.NET v3 SDK の 3.23.0 以降で使えます。Python の `patch_item` は 4.4.0b2 以降で、ガイドは「Preview」と表記しています[48]。REST は HTTP の PATCH メソッドを使います。`Content-Type` は `application/json_patch+json` で、本文に `operations` 配列を入れます。各操作の `op` は、`add`・`set`・`replace`・`remove`・`incr`・`move` の小文字の名前で書きます（例 `{ "op": "incr", "path": "/inventory/quantity", "value": 10 }`）[47][49]。複数リージョン書き込みでは、項目単位でなく Patch 操作のパス単位で競合を検出して解決できます[47]。

SAMPLE

条件付き Patchは、SQL 風のフィルター述語（`FilterPredicate`）を付け、満たさなければ操作を失敗させます。たとえば `"FROM products p WHERE p.used = false"` です[48]。REST の状態コードは 2 つです。条件を満たさない時は **412 Precondition Failed** です。本文が不正な時は 400 Bad Request です[50]。条件を同時実行制御に使う例は 1.2.6 で扱います。

一括操作、バッチ操作、トランザクション操作を実装する [1.2.4]

Bulk は、大量の操作をまとめて高スループットで流す仕組みです。.NET v3 では、`CosmosClientOptions` の `AllowBulkExecution = true` で有効にします[35]。使い方は、操作ごとに `Task` を作り、`Task.WhenAll` で待ちます（リストを渡す専用のメソッドはありません）。一括の取り込みは `CreateItemAsync` を、更新は `ReplaceItemAsync` を、削除は `DeleteItemAsync` を、並行に呼びます[35]。待ち時間の最適化が要らない大量のデータ投入では、有効化が推奨されています[29]。旧 bulk executor ライブラリから移る先は、.NET v3 SDK の Bulk です。v3 には隠れた再試行の挙動が無いので、`MaxRetryAttemptsOnRateLimitedRequests` と `MaxRetryWaitTimeOnRateLimitedRequests` を直接設定します[35]。

トランザクション バッチは、同じパーティション キーの点操作を 1 つにまとめる仕組みです。結果は、全部成功するか、全部取り消す（ロールバック）かのどちらかです。スコープは 1 つの論理パーティションで、スナップショット分離の ACID トランザクションです[51]。

上限	値
1 バッチの操作数	最大 100
ペイロード	2 MB 以内

SAMPLE

上限	値
実行時間	最大 5 秒

.NET の手順は 3 つです。

1. `container.CreateTransactionalBatch(partitionKey)` でバッチを作る
2. `batch.CreateItem<T>` などで操作を足す
3. `ExecuteAsync()` で `TransactionalBatchResponse` を得て、`IsSuccessStatusCode` を確かめる

失敗した時は、失敗した操作がそれ自身のエラー コードを持ちます（既存の項目の作成なら 409）。他の操作は **424 Failed Dependency** になります[51]。Python は `container.execute_item_batch(batch_operations=..., partition_key=...)` です。操作は、操作名の文字列、引数のタプル、kwargs の辞書の組で渡します。失敗すると `CosmosBatchOperationError` が出ます[51]。同じパーティション キーの複数の項目への Patch もバッチに含められ、1 つ失敗すれば全体がロールバックされます[47]。

	Bulk	トランザクション バッチ
目的	大量の操作の高スループット	小さな操作群の原子性
パーティション	複数可	1 つの論理パーティションだけ
失敗の扱い	操作ごとに成否が違う	全部成功か全部取り消し

操作が複数のパーティションにまたがる、または数が 100 を超えるなら Bulk、同じキーの少数の項目を全部成功か全部失敗で扱いたいならバッチです。

SAMPLE

Time to Live (TTL) の構成 [1.2.5]

TTL は、項目を自動で期限切れにして消す仕組みです。単位は秒で、最大値は 2,147,483,647 です。期限は、**項目の最終更新時刻**からの経過で数えます[52]。

コンテナーの `DefaultTimeToLive` は 3 つの状態を持ちます。

コンテナーの値	状態
未設定（null）	TTL は Off。項目は自動では期限切れにならない
-1	TTL は On（既定の期限なし）。項目の `ttl` で個別に期限を付けられる
n（0 以外の整数）	TTL は On。最終更新から n 秒で期限切れ

項目の `ttl` は、コンテナーの `DefaultTimeToLive` が null でない時だけ効き、効く時はコンテナーの値を上書きします。コンテナーが未設定なら、項目に `ttl` があっても効果がありません。項目の `ttl` は 1〜2,147,483,647 の正の整数か -1（期限なし）で、null は入れられません。コンテナーの既定を使いたい項目には、`ttl` プロパティ自体を入れません[52]。

例です。コンテナーが 1000 の時、`ttl` の無い項目は 1,000 秒で期限切れになります。`ttl = -1` の項目は期限切れせず、`ttl = 2000` の項目は 2,000 秒で期限切れになります[52]。設定の場所は 3 つあります。Python は `create_container(default_ttl=-1 または秒数)` です。CLI はコンテナー作成時に指定します。ポータルは Settings の Time to Live で、Off・On (no default)・On の 3 つから選び、On では秒数を入れます[25]。

削除はバックグラウンドで行われ、期限を過ぎた項目は、まだ消えていなくてもクエリの結果に出ません。TTL の削除の RU は、プロビジョニングなら未使

SAMPLE

用の RU の余りで処理されます（足りなければ遅れます）。Serverless なら、削除操作と同じ料金で RU が課金されます[52]。TTL の削除は、All versions and deletes モードの変更フィードに記録されます。`timeToLiveExpired` で見分けられます（1.4.4）。

コンカレンシー制御メカニズムを実装する [1.2.6]

オプティミスティック同時実行制御（OCC）は、更新の紛失（lost update）を防ぐ仕組みです。2 つのクライアントが同じ項目を同時に更新して、片方の更新が消える事故を防ぎます[46]。

すべての項目にはシステムが管理する `_etag` があり、更新のたびにサーバーが新しい値を作ります。クライアントは更新の要求に `if-match` ヘッダーで読んだ時点の `_etag` を付けます。現在の `_etag` と一致すれば更新され、古ければサーバーは **412 Precondition Failed** で拒否します[46]。置換（replace）では、`if-match` を要求のオプションとして明示する必要があります[46]。.NET は `ItemRequestOptions.IfMatchEtag`（`RequestOptions` から引き継ぐプロパティ）で渡します[53]。Python のトランザクション バッチでは、操作の kwargs に `if_match_etag` を入れます[51]。

412 を受けた時の流れは次のとおりです。

1. 最新の項目を読み直す
2. 変更をやり直す
3. 新しい `_etag` で再度要求する

412 は SDK が自動では再試行しないので、アプリ側で読み直して再試行します（または、自分の `_etag` で上書きします）[7]。409 は、既存の id やユニーク キーの違反です。449 は、書き込みだけで起きる一時的なエラーで、再試行して

SAMPLE

安全です。449 が多い時は、同じ項目へ同時の更新が集中する設計の問題かもしれません[7]。

条件付き Patch も、条件を満たさなければ失敗する条件付き更新なので、同時実行制御に使えます[47]。`_etag` は既定では索引の対象外です（含めるパスに追加した場合を除く）[22]。複数書き込みリージョンの更新競合は、Insert・Replace・Delete の 3 種類で、削除は挿入と置換の競合で常に勝ちます[21]。

アクセス パターンと運用要件に合わせてデータ モデルを設計する [1.3.1]

Cosmos DB のモデリングは、リレーショナルの正規化とは逆の発想で、**エンティティを JSON の自己完結した項目**にします[54]。設計の順序は、先に「アプリが出す要求」（アクセス パターン）を洗い出し、それをテスト一覧として、反復のたびに RU と拡張性を確かめることです。要求は、書き込みのコマンドと、読み取り専用のクエリに分けて考えます（CQRS）[55]。

一般に、非正規化（埋め込み）は読み取りが速く、正規化（参照）は書き込みが速くなります。参照は、サーバーへの往復が増えます[54]。読み取りが多いシステムでは、書き込み時に `countOfBooks` のような集計値を計算して持たせると、読み取りが軽くなります。

同じコンテナーに複数種類の項目を混在させるなら、種類を示すフィールド（例 `type`）を持たせます。JOIN は 1 つの項目の中でしか働かないので、一緒に読みたい項目は同じ論理パーティションに置く設計になります（1.3.3）[54]。複数項目を原子的に更新できる範囲は、同じ論理パーティションのトランザクション バッチです。履歴を残す方法は、保持の要件、クエリの待ち時間、コストで選びます。選択肢は、イベント ソーシング（変更フィード）、スナップショットと

SAMPLE

差分、TTL による版の失効です[54]。IEEE 754 の範囲を超えうる数値は、文字列にします。項目の大きさの上限は 2 MB です。

非正規化した値が元のデータとずれる問題は、**変更フィード**で解決します。元のコンテナーの変更フィードを読み、複製先の項目を更新する処理を置きます（1.4 章）。別のパーティションやコンテナーに非正規化した複製の同期にも、この方法を使います[55]。

カーディナリティ、アクセス パターン、書き込みディストリビューションの要件に基づいてパーティション キーを設計して選択する [1.3.2]

パーティション キーは、パスと値の 2 要素から成ります（例 パス `/userId`、値 `Andrew`）[20]。パスは英数字とアンダースコアを使え、入れ子は `/` で区切ります。値は文字列か数値で、長さは通常 2,048 バイトまでです（large partition key が有効でないと 101 バイトまで）[20]。同じ値を持つ項目の集まりが**論理パーティション**で、1 つにつき最大 20 GB です。論理パーティションは、物理パーティションに載ります。

良いキーの条件は 3 つです。**カーディナリティが高い**（取りうる値の範囲が広い）こと。RU とストレージを、すべての論理パーティションへ均等に散らすこと。そして、クエリの絞り込みに使われることです。

- 読み取りが多く、同じプロパティの等価フィルターがよく出るなら、そのプロパティをキーにする
- 小さな読み取り中心のコンテナー、または規模を問わない書き込み中心のコンテナーなら、`/id` が自然な選択になる
- 1 つのキーで全クエリを満たせないなら、別のキーを持つグローバル セカンダリ インデックスを検討する。グローバル セカンダリ インデックスは、別

SAMPLE

のパーティション キーを持つ追加のコンテナーで、元のコンテナーのデータと自動で同期される[20]

どのキーでも、項目の作成後にキー値は変えられません。移すなら、新しいキーで作り直して元を削除します（2 つの操作は論理パーティションをまたぐと原子的になりません）。キーそのものの変更は、新しいキーを持つ新コンテナーへデータを移すしかなく、コンテナー コピー ジョブが助けになります[20]。

関連データを埋め込む、または参照するタイミングを決定する [1.3.3]

埋め込みは、関連データを同じ項目の中に入れる形、参照は別の項目に分けて id などでたどる形です。

観点	埋め込み	参照
関係	包含、one-to-few	one-to-many、many-to-many
更新	少ない	頻繁に変わる
増え方	無制限には増えない	無制限に増えうる
読み取り	一緒に読まれる（1 回で済む）	往復が増える

件数に実用上の上限がない配列（コメントなど）の埋め込みは避けます。項目が際限なく大きくなるからです。折衷案は、投稿の項目に `recentComments` だけ埋め込み、残りは `postId` を持つ別の項目にしてクエリする形です[54]。株価のように多くの項目で共有され頻繁に変わる値の埋め込みも避けます。1 回の変更で大量の項目の更新が要るからです[54]。

外部キー制約はありません。項目のリンクは弱く、参照先の存在の確認はアプリ（またはトリガーやストアド プロシージャ）が行います。参照を持たせる側

SAMPLE

は、増え方で決めます。出版社あたりの本が無制限に増えるなら、本の側に出版社の id を持たせます。many-to-many は、結合用の項目を作らず、双方の項目に相手の id の配列を持たせます[54]。表示に必ず要る名前とサムネイルだけを書籍の側へ非正規化する**ハイブリッド**も許されます。リレーショナルからの変換は 3 通りです。one-to-few は親に埋め込みます。無制限の one-to-many は、`parentId` を持つ別項目にします。頻繁に更新される子は分離します[54]。

参照にすると読み取りの回数が増えるので、RU の見積もりも変わります。同じパーティションから 1 回のクエリで取れれば小さく、パーティションをまたぐと物理パーティションごとに 2〜3 RU が上乗せされます[20]。投稿とそのコメント・いいねのように一緒に読みたい項目は、同じ論理パーティションに置きます。文書の例は、posts コンテナーを `postId` で分け、`type`（post・comment・like）で区別します[55]。

パーティション キーの選択がスケーラビリティとパフォーマンスに与える影響を評価する [1.3.4]

パーティションの上限は表のとおりです[209]。

単位	上限
論理パーティション	20 GB。10,000 RU/s（載る物理パーティションの上限）
物理パーティション	10,000 RU/s、50 GB
1 GB あたりの最低値	1 RU/s

物理パーティションは、データ量やスループットが増えると自動で分割されます。可用性には影響せず、1 つの論理パーティションのデータは同じ物理パー

SAMPLE

ティションに残ります。プロビジョニングしたスループットは、物理パーティションに**均等に**割られるため、均等に使われるキーが要ります[20]。物理パーティションごとに、小さなコンテナーでも最低 4 つのレプリカがあります。

キー値はハッシュされて、キー空間が物理パーティションに割り当てられます。要求が一部のキーに偏ると、そのパーティションが**ホット パーティション**になります。スループットを無駄にして、429 とコストの増加を招きます[20]。プロビジョニングが 30,000 RU を超える、またはデータが 100 GB を超える時は、物理パーティションが数個では足りなくなります。そのため、クロスパーティションのクエリを減らす設計が効いてきます[20]。論理パーティションが 20 GB に近づいたかどうかは、Azure Monitor のアラートで見張ります。

論理パーティションが 20 GB に達すると、そのキーへの書き込みがエラーになります。恒久的な対処は階層パーティション キーでの再設計で、猶予が要る時はサポートへ一時的な上限引き上げを申請できますが、SLA は保証されません[9]。

キーを変えたい時は、新しいキーを持つ**新しいコンテナーへ写す**のが唯一の手段で、コンテナー コピー ジョブ（1.4.5）を使います[20]。トランザクション バッチは、論理パーティションを越えて組めません。

階層パーティション キー (HPK) とマルチテナント パーティション分割戦略を実装する [1.3.5]

階層パーティション キー（HPK、サブパーティション化）は、パーティション キーを複数の階層のパスで指定する機能です。階層の数の上限は、文書によって書き方が違います。HPK の主な文書とパーティション分割の文書は「3 階層まで」と書き、最終階層を `/id` にする形の文書は「4 階層まで対応し、4 階層目は 3 階層に加えて `/id` が要るまれな場合だけ」と書いています[5657]。第 1 階

SAMPLE

層から順に絞り込めます[56]。効き目は 2 つです。第 1 階層（プレフィックス）の単位で、論理パーティションの 20 GB と 10,000 RU/s の上限を越えられます。もう 1 つは、プレフィックスを含むクエリが、関係する物理パーティションだけに送られることです[56]。

定義は、`kind` が `MultiHash`、`version` が 2 で、`paths` に階層のパスを順に並べます。.NET は、`ContainerProperties` の `PartitionKeyPaths` にパスのリストを渡します。Python は、`PartitionKey(path=[...], kind="MultiHash")` で指定します。使える SDK のバージョンは次のとおりです[56]。

SDK	バージョン
.NET v3	3.33.0 以上
Java v4	4.42.0 以上
JavaScript v4	4.0.0
Python	4.6.0 以上

項目の書き込みには 2 つの方法があります。SDK がオブジェクトからパスの値を自動で取り出す方法と、`PartitionKeyBuilder` で手動で指定する方法です。大規模では、完全なパスを指定する方が有利です[56]。

クエリのルーティングは前方一致です。`tenantId` だけ、または `tenantId` と `userId` を指定すると、該当する物理パーティションだけに絞られます。`userId` だけのように先頭の階層を欠くと、全物理パーティションに送られます[56]。

制約は 4 つあります。作成時にだけ設定でき、既存のコンテナーには付けられません（既存データはコンテナー コピー ジョブなどで新コンテナーへ移す）。API for NoSQL だけが対応します。ユーザーとアクセス許可の機能では、

SAMPLE

キー パスの部分プレフィックスへの権限付与はできず、完全なパスだけです。最終階層を一意な `/id` にすると、前の階層（`tenantId`、`userId`）がそれぞれ 20 GB を超えられます。ただし、その形ではトランザクション バッチとストアド プロシージャは使えません[57]。アカウントのプロパティ `EnforceHierarchicalPartitionKeyIdLastLevel` を有効にすると、新しいコンテナーは HPK で、最終階層が `/id` であることが必須になります。既存のコンテナーは変わらず、`/id` は自動では付きません[57]。

マルチテナントでは、テナントの分け方が 3 通りあります[58]。

方式	分離	注意
共有コンテナー + テナントごとのパーティション キー	論理的な分離だけ	スループットを共有するのでノイジー ネイバーが起きうる。geo レプリケーション・PITR・顧客管理キー（CMK）などのアカウント単位の機能をテナントごとに変えられない
テナントごとのコンテナー	専用の RU/s をコンテナーに付けられる。Azure RBAC も使える	1 アカウントのコンテナー数に上限があり、複数のアカウントが要りうる。テナント固有の CMK は使えない（CMK はアカウント単位）
テナントごとのデータベース アカウント	最も強い	CMK が要るならこの方式が必須。アカウント間でスループットを共有するには Azure Cosmos DB fleets を使える

通常、B2C の SaaS はテナントごとにパーティション キーを使います。B2B の SaaS はテナントごとにデータベース アカウントを使います[58]。HPK が向くの

SAMPLE

は、テナントが数百から数千以上と多く、使い方が似ている時です。テナントが少数、または規模の偏りが大きいと、ホット パーティションになりえます。テナントごとのカーディナリティが低い時や、書き込み性能を最優先する時は、合成パーティション キーか `/id` だけで分けます[58]。

合成パーティション キーを実装する [1.3.6]

合成パーティション キーは、値が多く散らばるプロパティが無い時に、複数のプロパティなどから人工的に作るキーです。目的は、1 つのキーにワークロードが集中するホット パーティション キーを作らないことです。値の種類は、数百から数千がよいとされています[59]。作り方は 3 つです。

手法	作り方	長所と短所
複数プロパティの連結	`deviceId` と `date` を連結し、`abc-123-2018` のような値を `partitionKey` プロパティに入れる	大量の項目では、連結して挿入するロジックをクライアントに実装する
ランダムな接尾辞	キーの末尾に 1〜400 の乱数を付ける（例 `2018-08-09.1` 〜 `2018-08-09.400`）	書き込みのスループットは大きく上がる。書き込み時の接尾辞が分からないので、特定の項目の読み取りが難しい
事前に計算した接尾辞	読みたい属性（例 VIN）のハッシュから 1〜400 の数値を計算して、日付に付ける	個別の項目の読み取りが簡単になる

作り方は、キーの値の種類を増やす方向で 3 通りに分かれます[59]。

合成キーには代償があります。書き込みは散らせます。ただし、片方のフィールドだけで絞るクエリはクロスパーティションになり、物理パーティションごとに

SAMPLE

2〜3 RU が上乗せされます[20]。`TenantId_UserId` の合成キーは、完全なキーが分かる時はテナント内のユーザーを効率よく探せます。ただし、全ユーザーを事前に指定しない限り、テナント単位のクエリはクロスパーティションになります[56]。

HPK と合成キーの使い分けは、第 1 階層のカーディナリティで決めます。第 1 階層の値が多くなく、20 GB の上限に当たっているなら、HPK でなく合成パーティション キーが勧められます[56]。

ソリューション レビュー中にパーティション分割のアンチパターンとパーティション分割の問題を特定する [1.3.7]

レビューで探すアンチパターンは、次の 6 つです[2054]。

- 低カーディナリティのキー（`status`、`type`、`country`）。論理パーティションの数が値の種類に限られ、RU とストレージが偏ってホット パーティションになりうる
- クエリで使わない高カーディナリティのキー（ランダムな GUID）。書き込みは散るが、読み取りの大半がクロスパーティションのクエリになる
- `/id` を何にでも使うキー。書き込みの分散とポイント読み取りには良いが、id 以外で絞るクエリはクロスパーティションになる。ポイント読み取りと書き込みが中心の時だけ使う
- 件数に上限のない配列の埋め込み。項目が膨らみ、送受信・読み取り・更新が難しくなる
- 頻繁に変わる共有データの埋め込み。1 回の変更で大量の項目の更新が要る
- 日付だけのキー。その日の書き込みが 1 つのキーに集まる

SAMPLE

HPK の落とし穴もあります。第 1 階層のカーディナリティが低いと、取り込み時の書き込みが 1 つの物理パーティションに集中し、50 GB で分割されるまで制限されます。`UserId` のように高カーディナリティに見えても、実際の操作が特定の 1 ユーザーに集中すればホットになります[57]。1 つのテナントが他より桁違いに多い RU を使う時は、HPK でもホットになります。継続的に高い負荷を出す大きなテナントが少数なら、専用のデータベース アカウントに分けて、残りを共有コンテナーに置きます[58]。論理パーティションが 20 GB に達すると、そのキーへの書き込みが拒否されます[57]。

ホット パーティションの見つけ方。ポータルの Insights を開き、Throughput の **Normalized RU Consumption (%) By PartitionKeyRangeID** を見ます。1 つの PartitionKeyRangeId（物理パーティション 1 つ）だけが常に 100% に近く、他が 30% 以下なら、ホット パーティションの兆候です[8]。どのキー値が RU を使っているかは、診断ログで調べます。`PartitionKeyRUConsumption` テーブルの `RequestCharge` を、PartitionKeyRangeId ごとに集計します[8]。全体の 429 が低くても一部の範囲だけ 100% の時は、増設より先にキーの設計を疑います。

コンテナー コピーでキーを変えた結果の失敗も、レビューの材料です。新しいキーで 20 GB を超えると 403（サブステータス 1014）で失敗し、新しいキーと id の組が重複すると挿入エラーになります。組が一意になるようにします[60]。

ワークロードのコスト要件を見積もる [1.3.8]

見積もりの道具は **Azure Cosmos DB Cost Estimator** です。パブリック プレビューで、サインインは不要です。出る金額は概算で、正式な見積書ではありません[61]。入力の経路は 2 つあります。ガイド付きは約 2 分の 9 ステップのウィザード、自分で設定は約 5 分の Configure パネルです[61]。ガイド付きのス

SAMPLE

テップは、Workload、Stage、Data、Traffic、Backup、Read/Write Pattern、Region、Review、Estimate の順です[61]。

操作は、点読み取りが Reads、作成が Writes、更新が Updates、削除が Deletes、クエリが Queries に対応します。Capacity Mode は Serverless、Autoscale、Manual から選びます。Autoscale は変動する負荷に、Manual は持続的で予測できる負荷に向きます。索引は自動・無効・カスタムから選びます[61]。クエリの複雑さは、simple が約 3 RU、filter が約 10 RU、complex が約 30 RU として見積もります。整合性は、Strong や Bounded staleness が Session などの約 2 倍の読み取り RU/s を要します[61]。ベクトルの操作を有効にすると、次元数（500、768、1536、3072）と索引の種類（Flat、QuantizedFlat、DiskANN）も入力します。Advanced の Change Feed Consumers で、変更フィードを読む処理のコストを足せます。Compare ビューでは、スループット・ストレージ・バックアップの内訳と、12 か月の成長予測が見られます。構成の比較（例 Serverless 対プロビジョニング）もできます[61]。

課金の要素は、RU/s とストレージです。ストレージは 1 GB につき最低 1 RU/s が要ります[9]。変更フィードの読み取りに専用の料金はなく、リース コンテナーのスループットと各要求の RU だけです[62]。オンラインのコンテナー コピー中は、ソースでの置換と削除が、基本の書き込み RU に 50〜100% を足します（作成は影響なし）[60]。

3 つの容量の比べ方は、**ピークと平均の差**で考えます。

負荷の形	選ぶ	理由
常に一定で、予測できる	手動	同じ RU/s なら単価が最も低い

SAMPLE

負荷の形	選ぶ	理由
山と谷がある、読めない	Autoscale	単価は 1.5 倍だが、最大値を使う時間が月の 66% 以下なら得（1.1.3）
時々しか使わない、小さい	Serverless	使った RU だけ課金。単一リージョンのみ

変更フィード操作を実装する [1.4.1]

変更フィードは、コンテナーに起きた変更を、**発生した順に並べて保持する永続的な記録**です。すべてのアカウントで既定で有効で、モードによっては追加の設定が要ります[62]。残るのは挿入と更新です。削除と TTL による失効は、All versions and deletes モードだけが記録します（1.4.4）[62]。

順序の保証は、**パーティション キーごと**です。キー値をまたぐ順序は保証されません。トランザクション バッチ、ストアド プロシージャ、バルク モードで書いた項目は変更時刻が同じで、その中の順序は不定です[62]。各変更は変更フィードに 1 回現れます。チェックポイントはクライアントが管理します。変更フィード プロセッサなら自動で、配信は少なくとも 1 回（at-least-once）です[62]。複数の書き込みリージョンでは、競合がないと確認された時刻 `crts` の順で届きます（`_ts` はローカルで書いた時刻）。Latest version モードの応答には、`_etag` と `_ts` に加えて、変更フィード用の `_lsn`（バッチの ID）が付きます。同じ `_lsn` を複数の項目が持つことがあります[63]。

変更フィードの読み取りは、他の操作と同じようにプロビジョニング済みのスループットを使い、アカウントの任意のリージョンで読めます。並列の単位は、コンテナーのパーティション キー範囲です（プロセッサでは物理パーティション、プル モデルでは `FeedRange`）[62]。読む方法は 4 つあります。Azure

SAMPLE

Functions、変更フィード プロセッサ、プル モデル、Apache Spark です。他の API では名前が変わります。API for MongoDB は change stream、API for Cassandra は述語付きのクエリです[62]。

Azure Functions の Cosmos DB トリガーは、変更フィードを読む最も簡単な方法です。プロセッサのスケールと確実なイベントの検出を、ワーカーのインフラを持たずに使えます。API for NoSQL だけが対象です[64]。必要なのは 2 つのコンテナーです。1 つは、変更を生む**監視対象のコンテナー**です。もう 1 つは**リース コンテナー**で、複数の Function インスタンスの状態を保持します。リース コンテナーはトリガーの構成の `CreateLeaseContainerIfNotExists` を設定すると自動で作られ、手動でも作れます。パーティション化されたリース コンテナーは、パーティション キーが `/id` であることが必須です[64]。トリガーは両方のモードに対応します。Latest version モードが既定です。All versions and deletes モードは、.NET isolated のワーカーで使えます。拡張の Microsoft.Azure.Functions.Worker.Extensions.CosmosDB は 4.16.1 以上が要ります[64]。

変更フィード プロセッサ モデルとプル モード モデルを区別する [1.4.2]

変更フィードを自分のコードで読む方法は 2 つです。

	変更フィード プロセッサ	プル モデル
進捗の管理	リース コンテナーのリース	継続トークン（メモリか自分で保存）
新しい変更の確認	`WithPollInterval` に従い自動	自分で呼ぶ

SAMPLE

	変更フィード プロセッサ	プル モデル
並列化	同じコンテナーを読む複数のスレッドやマシンへ自動	`FeedRange` で手動（オーケストレーターが配る）
単一のパーティション キーだけの処理	非対応	可能
対応言語	.NET V3 と Java V4	Python と Node.js はこちら

大抵はプロセッサが最も簡単です。プル モデルは、特定のパーティション キーの読み取り、ペースの制御、既存データの 1 回限りの読み取り（移行など）に向きます。リースと継続トークンは相互に変換できません[65]。

プル モデルでは、`GetFeedRangesAsync` で `FeedRange` を取ります。物理パーティションごとに 1 つ得られ、それぞれに `FeedIterator` を作って並列に処理します。`FeedIterator` の作成時には、`ChangeFeedStartFrom`（Beginning、Now、継続トークンなど）と `ChangeFeedMode` を必ず指定します。新しい変更が無いと HTTP **304 NotModified** が返ります。0 件の 200 OK は終端を意味しないので、ポーリングを続けます[65]。継続トークンを指定すると、開始時刻などより優先されます。Latest version のトークンは、コンテナーが存在する間は失効しません。All versions and deletes のトークンは、継続バックアップの保持期間内だけ有効です[65]。`ChangeFeedRequestOptions` の `PageSizeHint` で、ページあたりの最大件数を設定します。

プロセッサの作り方は次のとおりです（.NET）。

1. `container.GetChangeFeedProcessorBuilder<T>(processorName, onChangesDelegate)` でビルダーを作る

SAMPLE

2. `.WithInstanceName(...)` で、各コンピューティング インスタンスの一意な名前を付ける（Java は `hostName`）
3. `.WithLeaseContainer(leaseContainer)` でリース コンテナーを渡す
4. 必要なら `.WithStartTime(...)`（開始時刻）や `.WithPollInterval(...)` を足し、`.Build()` して `StartAsync()` する

開始時刻は `WithStartTime` で特定の日時を指定します。コンテナーの先頭から読みたい時は `WithStartTime(DateTime.MinValue.ToUniversalTime())` を渡します。この指定が効くのは、リース コンテナーが最初に初期化される時だけです。初期化の後に変えても効果はありません[66]。

リース コンテナーを構成する [1.4.3]

リース コンテナーは、変更フィード プロセッサの**状態の保管場所**で、複数のワーカーの処理を調整します。監視対象と同じアカウントでも別のアカウントでも構いません[66]。パーティション キー範囲ごとの進捗は、リース コンテナーの中のリース ドキュメントで個別に管理され、全部のリースがプロセッサの現在の状態を表します。リースの持ち主を決めることが、インスタンスへの負荷の分散です。インスタンスを増やすと、リースが再分散されて、処理が分担されます[66]。

リース コンテナーのパーティション キーは `/id` が必須です。作り忘れると、プロセッサも Functions のトリガーも動かせません。`CreateLeaseContainerIfNotExists` で自動作成できます[64]。

タイミングの既定は 3 つあります[66]。

SAMPLE

設定	既定	意味
Lease Acquire	17 秒ごと	リースを取りに行く間隔。短くすると再分散は速くなるが、リース コンテナーの RU が増える
Lease Expiration	60 秒	更新のないリースは他のホストが取る。更新間隔より小さくしてはいけない
Lease Renewal	13 秒ごと	新しい変更が無くても、持ち主が Replace で更新する

複数のデプロイ単位で、1 つのリース コンテナーを共有できます。各単位が別の監視対象コンテナーか別の `processorName` を持てば、状態は独立します（Java は別の `leasePrefix`）[66]。同じリース コンテナーを使うインスタンスが増えるほど RU の消費が増え、スロットリングされると処理が遅れ、止まることもあります。Microsoft Entra ID で接続する時の権限は、コンテナーごとに違います。リース コンテナーには、items の read・create・replace・delete・executeQuery が要ります。監視対象には、readMetadata と readChangeFeed が要ります[66]。リージョンを切り替える時は、リージョン別のエンドポイントを使いません。グローバル エンドポイントに、`ApplicationRegion` か `ApplicationPreferredRegions` を組み合わせます。リース ドキュメントが、設定したエンドポイントにスコープされているためです[66]。

遅れの測定は、変更フィードの**見積もり器（Estimator）**です。リースが示す最後に処理した項目と、コンテナーの最新の変更との差を測ります[67]。プロセッサと同じく通知で受ける形は、`GetChangeFeedEstimatorBuilder(processorName, handler, interval).WithLeaseContainer(...)` です。名前は、測りたいプロセッサと同じにします。必要な時に取りたいなら、

SAMPLE

`GetChangeFeedEstimator(processorName, leaseContainer)` を使います。`FeedIterator<ChangeFeedProcessorState>` を返し、各リースの見積もりの遅れが読めます[67]。遅れが増え続けるなら、インスタンスを足します。

最新モードまたはすべてのバージョンを使用して変更フィード処理を実装し、削除する [1.4.4]

変更フィードのモードは 2 つです[63]。

	Latest version	All versions and deletes
記録	create と update の最新の版だけ	create・update・delete と、間の中間の変更もすべて順番に
削除	記録されない（論理削除で代用する）	記録される。TTL の削除は `timeToLiveExpired` で見分ける
対応するアカウント	Table と PostgreSQL 以外のすべて。既定で有効	NoSQL のアカウントだけ
前提	なし	継続バックアップ
開始位置	先頭・特定の時刻（精度約 5 秒）・Now・チェックポイント	Now か、継続トークン / リースだけ

All versions and deletes を使うには、アカウントで**継続バックアップ**を構成します。保持期間（層により 35 日・30 日・7 日）の中の変更だけが読め、範囲外を読むとエラーになります（保持 7 日なら直近 7 日）[68]。有効化は、アカウントの Features ページで All versions and deletes change feed mode を有効にします。反映に最大 30 分かかり、その間はアカウントを変更できません。既存のアカウントでは、REST API でアカウントのプロパティに `"enableAllVersionsAndDeletesChangeFeed": true` を加える方法もあります。

SAMPLE

preview API の 2024-12-01-preview 以降が対象です。アカウントの作成時には有効化できません[63]。

パーティションをマージしたことがある、またはマージが有効なアカウントは、All versions and deletes に対応しません。削除や更新の前の版は取れません[63]。このモードの要求は、設定した接続モードにかかわらず Gateway モードを使います。

.NET のプロセッサは、Latest が `GetChangeFeedProcessorBuilder` を使います[63]。All versions and deletes は `GetChangeFeedProcessorBuilderWithAllVersionsAndDeletes` です。デリゲートが受ける型も `ChangeFeedItem<T>` になります。`metadata.operationType`（create・replace・delete）で操作を判別します[6663]。プロセッサ側の All versions and deletes は「プレビュー」と書かれ、3.60.0 以降で対応します。カスタムの開始点（`WithStartTime`）は Latest だけです。All versions and deletes は、開始時点か、保持期間内の既存のリースから再開します[6366]。SDK の要件は、.NET が 3.60.0 以上、Java が 4.81.0 以上（どちらもプロセッサとプル）です。Python のプルは 4.9.1b1 以上、Functions は .NET isolated の 4.16.1 以上です。Java は `handleAllVersionsAndDeletesChanges()` を使います。Python のプルは `query_items_change_feed(mode="AllVersionsAndDeletes")` です[63]。

削除を拾いたい時の選び方。継続バックアップを使えない、または対象のアカウントが NoSQL でない場合は、Latest version のまま、削除の代わりに**論理削除**を使います。項目に `deleted: true` のような札を付けて更新し、後で TTL で物理的に消します[63]。継続バックアップを入れられ、削除も中間の更新も取りたいなら、All versions and deletes を選びます。

SAMPLE

コンテナー のコピー ジョブを実装する [1.4.5]

コンテナー コピー ジョブは、あるコンテナーの全項目を別のコンテナーへコピーする機能です。使う場面は 5 つあります。スループットの粒度の変更、**パーティション キーの変更**、ユニーク キーの更新、コンテナーやデータベースの名前の変更、HPK のように新しいコンテナーに限る機能の採用です[60]。

	オフライン	オンライン
仕組み	Latest version の変更フィードで増分を複製	All versions and deletes の変更フィードで増分を複製
前提	なし	ソースのアカウントで継続バックアップと All versions and deletes を有効化。アカウントに `EnableOnlineContainerCopy` を追加し、有効化の後ジョブの作成まで少なくとも 30 分待つ
コピー中のソースへの書き込み	止める（重複や欠落の原因になる）	続けられる
完了のさせ方	完了を待ち、アプリの向き先を切り替える	全ドキュメントのコピー後にソースの更新を止め、completion API（`az cosmosdb copy complete`）で完了にする

オフラインの手順は 5 つです。

1. 対象のコンテナーを作る
2. ソースの操作を止める

SAMPLE

3. ジョブを作る
4. 進捗を監視して完了を待つ
5. アプリの向き先を切り替える

複数のジョブは順に実行し、データベース全体はコンテナーごとにジョブを作ります[60]。Azure CLI の `az cosmosdb copy` には、`create`（`--mode Online` など）、`show`、`complete` があり、`cosmosdb-preview` 拡張に含まれます。別のアカウントへコピーする時は、ソースのアカウントの ID に、宛先アカウントの読み書きのロールを付与します。ロール定義 ID は `00000000-0000-0000-0000-000000000002` です。同じアカウント内なら不要です[69]。

制限もあります。ベスト エフォートで、完了時間の SLA はありません。パーティションのマージが有効なアカウントでは動かないので、ジョブの前に無効にします[60]。速さは、ソースとターゲットのスループットと、サーバー側のインスタンスで決まります。既定は 4 vCPU・16 GB を 2 台で、ターゲットはソースの 2 倍以上を推奨します[60]。ジョブは書き込みリージョンで動き、書き込みリージョンが変わると未完了のジョブは失敗するので、作り直します。項目の TTL は残り時間を計算し直してコピーされますが、コンテナー レベルの TTL 設定はコピーされません[60]。

エージェント キットを実装して、Azure Cosmos DB開発に推奨されるソリューションに従う [1.5.1]

Azure Cosmos DB Agent Kit は、AI のコーディング アシスタントに Cosmos DB の指針を与える、**Agent Skills のコレクション**です。データ モデリング、パーティション キー、クエリ、SDK の使い方などを扱います[70]。ドキュメントは learn.microsoft.com の gen-ai/agent-kit にあります（NoSQL が対象）。リ

SAMPLE

ポジトリは `github.com/AzureCosmosDB/cosmosdb-agent-kit` です[71]。示す推奨の領域は次のとおりです[70]。

- パーティション キーの選択、クエリの最適化、データ モデリング
- ホット パーティションの防止、SDK のパターン、セキュリティ
- 索引、スループット、グローバル分散、監視
- ベクトルと全文の検索、設計パターン

助言だけで、実行権限は持ちません。Azure アカウントへの認証やデータベースへのアクセス権は与えず、生成されたコードは提案として、自分でレビューしテストします。推奨は、測定済みの改善の証拠ではありません[70]。Microsoft が公開している MCP サーバーは、Microsoft の製品ドキュメントを検索して取得する別の機能です[72]。Cosmos DB のデータへのアクセスは与えません[70]。

使い方は 3 つです[73]。

- パーティション キーのレビュー。提案するキー、代表的なドキュメントの形、クエリのフィルター、書き込み量、増加の見込み、テナントの大きさの差を渡し、ホット パーティションとストレージのリスクを評価させる
- クエリ最適化のレビュー。クエリ、パーティション キーの定義、索引ポリシー、代表的なデータ、RU の料金と診断を渡す。改善と言う前に、同じデータで RU と待ち時間を実測する
- SDK コードのレビュー。SDK の言語とパッケージの版、クライアントの初期化、関係するデータベースの操作、認証の方法、サニタイズしたエラー診断を渡す。クライアントの使い回し、非同期の操作、パラメーター化クエリ、ページネーション、認証、リトライ、診断ログを確かめる。デプロイが対応するなら Microsoft Entra ID の認証を勧める

SAMPLE

どのレビューでも、資格情報・アクセス トークン・接続文字列・機微なドキュメントの値は渡しません。構造を保ったまま値を置き換えます[73]。

当てはめ方は、この章の知識で決まります。リクエストごとに `CosmosClient` を作るコードは、1 つを使い回す推奨に反します[73]。id 以外のキーで引く読み取りは、ポイント読み取りにできないか検討します。ステータスのような低カーディナリティのキー案は、ホット パーティションの危険として指摘されます。

エージェント キットを実装して AI アプリケーション開発を高速化する [1.5.2]

導入の基本は 2 段で、先に Agent Skills 対応のアシスタントと Node.js（`npm` 、`npx` ）を用意します[74]。次に `npx skills add AzureCosmosDB/cosmosdb-agent-kit` を実行して、案内に従います。スキルの導入やコードのレビューに、Azure サブスクリプションは要りません。データベースに接続する別のツールを使う時だけ、別に認証と権限が要ります[74]。

アシスタント	導入
Claude Code	公式プラグイン マーケットプレイスから `/plugin install cosmosdb@claude-plugins-official`
GitHub Copilot CLI	`/plugin marketplace add AzureCosmosDB/cosmosdb-agent-kit` の後に `/plugin install cosmosdb@cosmosdb-agent-kit`
Gemini CLI / Codex / APM	`gemini extensions install` / `codex plugin marketplace add`（ChatGPT デスクトップで導入）/ `apm install`

SAMPLE

アシスタント	導入
Grok Build / Kimi Code CLI	`grok plugin marketplace add` と `install cosmosdb --trust` / `/plugins install` の後に `/reload`

導入の後は、必要ならアシスタントを再読み込みして、スキル名 `cosmosdb-best-practices` が見えるか確認します。Cosmos DB に言及するだけでは、ロードされた証明にはなりません。同じスキルを同じホストに複数の方法で入れないようにします[74]。必要なスキルだけをその場で読み込み、全ルールを毎回のプロンプトへ入れない設計なので、コンテキストを節約できます[70]。

AI アプリの開発では、ベクトル検索、全文検索、ハイブリッド検索の設定を支援します。エージェント型のワークフロー、変更フィード、マテリアライズド ビューの設計パターンも対象です[73]。対応するアシスタントは、GitHub Copilot（VS Code / CLI）、Claude Code、Gemini CLI、OpenAI Codex、Cursor、Grok Build、Kimi Code CLI などの Agent Skills 互換ホストです。導入の手順はリポジトリで 2026 年 10 月 16 日に確認された内容で、すべてのホストで実行テストされたわけではありません[75]。`npx` が無いなら、Node.js を入れ直して端末を開き直します。プラグインのコマンドが効かないなら、ホストのバージョンを確認するか、skills CLI を使います[74]。スキルが与えるのは指針だけです。ドキュメントの検索には、Microsoft が公開している文書検索用の MCP サーバーが要ります。データベースの照会と管理には、別に構成したツールと権限が要ります[75]。

SAMPLE

サンプルはここまでです

続き(第2章〜巻末)は、DP-420 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。

Footnotes

1. https://learn.microsoft.com/en-us/azure/cosmos-db/consistency-levels ↩ ↩2 ↩3 ↩4 ↩5 ↩6 ↩7
2. https://learn.microsoft.com/en-us/azure/cosmos-db/how-to-manage-consistency ↩ ↩2 ↩3 ↩4
3. https://learn.microsoft.com/en-us/azure/cosmos-db/request-units ↩ ↩2 ↩3 ↩4
4. https://learn.microsoft.com/en-us/azure/cosmos-db/optimize-cost-reads-writes ↩ ↩2 ↩3 ↩4
5. https://learn.microsoft.com/en-us/azure/cosmos-db/unique-keys ↩ ↩2 ↩3
6. https://learn.microsoft.com/en-us/azure/cosmos-db/find-request-unit-charge ↩
7. https://learn.microsoft.com/en-us/azure/cosmos-db/conceptual-resilient-sdk-applications ↩ ↩2 ↩3 ↩4 ↩5 ↩6
8. https://learn.microsoft.com/en-us/azure/cosmos-db/monitor-normalized-request-units ↩ ↩2 ↩3 ↩4

SAMPLE

9. https://learn.microsoft.com/en-us/azure/cosmos-db/concepts-limits ↩ ↩2 ↩3 ↩4 ↩5 ↩6 ↩7 ↩8 ↩9 ↩10 ↩11 ↩12 ↩13
10. https://learn.microsoft.com/en-us/azure/cosmos-db/provision-throughput-autoscale ↩ ↩2
11. https://learn.microsoft.com/en-us/azure/cosmos-db/how-to-choose-offer ↩
12. https://learn.microsoft.com/en-us/azure/cosmos-db/serverless ↩ ↩2 ↩3
13. https://learn.microsoft.com/en-us/azure/cosmos-db/set-throughput ↩ ↩2 ↩3 ↩4 ↩5
14. https://learn.microsoft.com/en-us/azure/cosmos-db/burst-capacity ↩
15. https://learn.microsoft.com/en-us/azure/cosmos-db/priority-based-execution ↩ ↩2
16. https://learn.microsoft.com/en-us/azure/cosmos-db/throughput-buckets ↩
17. https://learn.microsoft.com/en-us/azure/cosmos-db/how-to-create-account ↩
18. https://learn.microsoft.com/en-us/azure/cosmos-db/how-to-change-capacity-mode ↩
19. https://learn.microsoft.com/en-us/azure/cosmos-db/free-tier ↩

SAMPLE

20. https://learn.microsoft.com/en-us/azure/cosmos-db/partitioning ↩ ↩2 ↩3 ↩4 ↩5 ↩6 ↩7 ↩8 ↩9 ↩10 ↩11 ↩12 ↩13 ↩14 ↩15 ↩16

21. https://learn.microsoft.com/en-us/azure/cosmos-db/conflict-resolution-policies ↩ ↩2 ↩3

22. https://learn.microsoft.com/en-us/azure/cosmos-db/index-policy ↩ ↩2

23. https://learn.microsoft.com/en-us/azure/cosmos-db/how-to-dotnet-create-container ↩ ↩2 ↩3

24. https://learn.microsoft.com/en-us/azure/cosmos-db/how-to-dotnet-create-database ↩

25. https://learn.microsoft.com/en-us/azure/cosmos-db/how-to-time-to-live ↩ ↩2

26. https://learn.microsoft.com/en-us/azure/cosmos-db/quickstart-python ↩ ↩2

27. https://learn.microsoft.com/en-us/azure/cosmos-db/manage-with-cli ↩

28. https://learn.microsoft.com/en-us/azure/cosmos-db/how-to-manage-indexing-policy ↩

29. https://learn.microsoft.com/en-us/azure/cosmos-db/best-practice-dotnet ↩ ↩2 ↩3 ↩4 ↩5

30. https://learn.microsoft.com/en-us/azure/cosmos-db/best-practice-python ↩ ↩2 ↩3 ↩4

SAMPLE

31. https://learn.microsoft.com/en-us/azure/cosmos-db/quickstart-dotnet ↩

32. https://learn.microsoft.com/en-us/azure/cosmos-db/how-to-python-get-started ↩

33. https://learn.microsoft.com/en-us/azure/cosmos-db/tune-connection-configurations-net-sdk-v3 ↩ ↩2

34. https://learn.microsoft.com/en-us/azure/cosmos-db/sdk-connection-modes ↩ ↩2 ↩3

35. https://learn.microsoft.com/en-us/azure/cosmos-db/how-to-migrate-from-bulk-executor-library ↩ ↩2 ↩3 ↩4

36. https://learn.microsoft.com/en-us/cosmos-db/query/select ↩ ↩2

37. https://learn.microsoft.com/en-us/cosmos-db/query/join ↩

38. https://learn.microsoft.com/en-us/cosmos-db/query/subquery ↩

39. https://learn.microsoft.com/en-us/cosmos-db/query/functions ↩

40. https://learn.microsoft.com/en-us/cosmos-db/query/parameterized-queries ↩

41. https://learn.microsoft.com/en-us/azure/cosmos-db/how-to-dotnet-query-items ↩ ↩2 ↩3

42. https://learn.microsoft.com/en-us/azure/cosmos-db/performance-tips-query-sdk ↩ ↩2

43. https://learn.microsoft.com/en-us/cosmos-db/query/offset-limit ↩

SAMPLE

44. https://learn.microsoft.com/en-us/azure/cosmos-db/how-to-dotnet-read-item ↩ ↩2 ↩3

45. https://learn.microsoft.com/en-us/azure/cosmos-db/how-to-dotnet-create-item ↩ ↩2

46. https://learn.microsoft.com/en-us/azure/cosmos-db/database-transactions-optimistic-concurrency ↩ ↩2 ↩3 ↩4

47. https://learn.microsoft.com/en-us/azure/cosmos-db/partial-document-update ↩ ↩2 ↩3 ↩4 ↩5 ↩6

48. https://learn.microsoft.com/en-us/azure/cosmos-db/partial-document-update-getting-started ↩ ↩2

49. https://learn.microsoft.com/en-us/rest/api/cosmos-db/patch-a-document ↩

50. https://learn.microsoft.com/en-us/rest/api/cosmos-db/patch-a-document ↩

51. https://learn.microsoft.com/en-us/azure/cosmos-db/transactional-batch ↩ ↩2 ↩3 ↩4

52. https://learn.microsoft.com/en-us/azure/cosmos-db/time-to-live ↩ ↩2 ↩3 ↩4

53. https://learn.microsoft.com/en-us/dotnet/api/microsoft.azure.cosmos.requestoptions.ifmatchetag ↩

SAMPLE

54. https://learn.microsoft.com/en-us/azure/cosmos-db/modeling-data ↩ ↩2 ↩3 ↩4 ↩5 ↩6 ↩7 ↩8 ↩9

55. https://learn.microsoft.com/en-us/azure/cosmos-db/model-partition-example ↩ ↩2 ↩3

56. https://learn.microsoft.com/en-us/azure/cosmos-db/hierarchical-partition-keys ↩ ↩2 ↩3 ↩4 ↩5 ↩6 ↩7 ↩8

57. https://learn.microsoft.com/en-us/azure/cosmos-db/hierarchical-partition-keys-unlimited-scale ↩ ↩2 ↩3 ↩4 ↩5

58. https://learn.microsoft.com/en-us/azure/architecture/guide/multitenant/service/cosmos-db ↩ ↩2 ↩3 ↩4

59. https://learn.microsoft.com/en-us/azure/cosmos-db/synthetic-partition-keys ↩ ↩2

60. https://learn.microsoft.com/en-us/azure/cosmos-db/container-copy ↩ ↩2 ↩3 ↩4 ↩5 ↩6 ↩7

61. https://learn.microsoft.com/en-us/azure/cosmos-db/estimate-ru-with-capacity-planner ↩ ↩2 ↩3 ↩4 ↩5 ↩6

62. https://learn.microsoft.com/en-us/azure/cosmos-db/change-feed ↩ ↩2 ↩3 ↩4 ↩5 ↩6 ↩7

63. https://learn.microsoft.com/en-us/azure/cosmos-db/change-feed-modes ↩ ↩2 ↩3 ↩4 ↩5 ↩6 ↩7 ↩8 ↩9

SAMPLE

64. https://learn.microsoft.com/en-us/azure/cosmos-db/change-feed-functions ↩ ↩2 ↩3 ↩4

65. https://learn.microsoft.com/en-us/azure/cosmos-db/change-feed-pull-model ↩ ↩2 ↩3

66. https://learn.microsoft.com/en-us/azure/cosmos-db/change-feed-processor ↩ ↩2 ↩3 ↩4 ↩5 ↩6 ↩7 ↩8 ↩9

67. https://learn.microsoft.com/en-us/azure/cosmos-db/nosql/how-to-use-change-feed-estimator ↩ ↩2

68. https://learn.microsoft.com/en-us/azure/cosmos-db/continuous-backup-restore-introduction ↩

69. https://learn.microsoft.com/en-us/azure/cosmos-db/how-to-container-copy ↩

70. https://learn.microsoft.com/en-us/azure/cosmos-db/gen-ai/agent-kit/overview ↩ ↩2 ↩3 ↩4 ↩5

71. https://learn.microsoft.com/en-us/azure/cosmos-db/gen-ai/agent-kit ↩

72. https://learn.microsoft.com/en-us/training/support/mcp ↩

73. https://learn.microsoft.com/en-us/azure/cosmos-db/gen-ai/agent-kit/usage ↩ ↩2 ↩3 ↩4

74. https://learn.microsoft.com/en-us/azure/cosmos-db/gen-ai/agent-kit/setup ↩ ↩2 ↩3 ↩4

SAMPLE

75. https://learn.microsoft.com/en-us/azure/cosmos-db/gen-ai/agent-kit/compatibility ↩ ↩2

SAMPLE

DP-420 学習用テキスト(無料サンプル)

版
2026年11月版(DP-420 出題範囲(Skills measured)2026-10-06版 準拠)

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

お問い合わせ
info@shikaku-doujou.com

Microsoft、Azure、Azure Cosmos DB および Microsoft の製品名・サービス名は、Microsoft Corporation 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。

本教材は公式シラバスの内容を基に作成したもので、Microsoft Corporation 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