

Azure AI Search での AI エンリッチメント

[アーティクル] • 2024/01/30

Azure AI Search では、AI エンリッチメントとは、未加工の形式で検索できないコンテンツを処理するための Azure AI サービスとの統合を指します。エンリッチメントにより、分析と推論を使用して、以前は存在しなかった検索可能なコンテンツや構造を作成します。

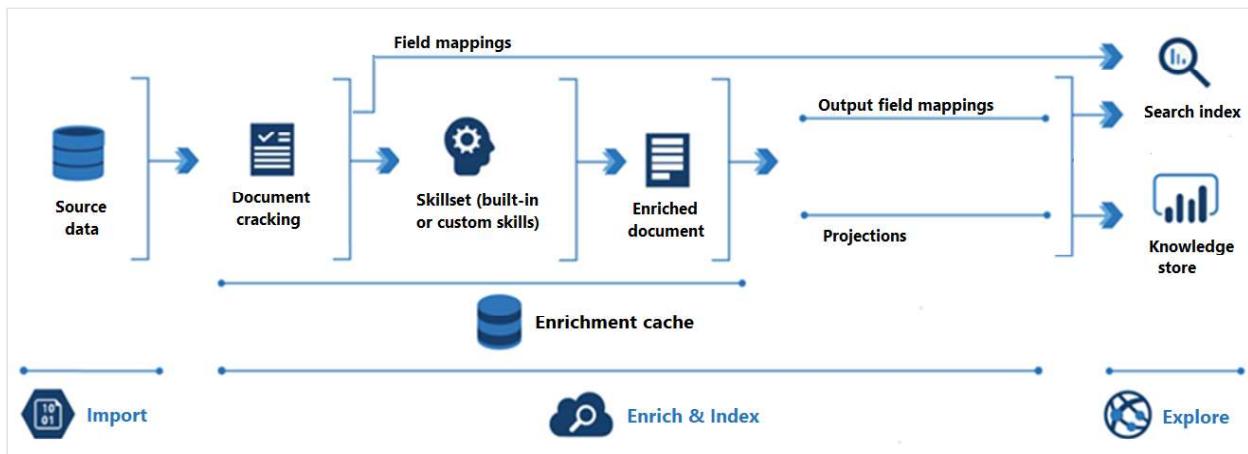
Azure AI Search はテキストおよびベクター検索ソリューションであるため、AI エンリッチメントの目的は、検索関連のシナリオでコンテンツの有用性を向上することです。ソースコンテンツはテキスト形式である必要がありますが（ベクターをエンリッチすることはできません）、エンリッチメント パイプラインによって作成されたコンテンツは、チャネル用の Text Split スキルやエンコード用の AzureOpenAiEmbedding スキルなどのスキルを使用してベクター ストアでベクター化およびインデックス付けできます。

組み込みのスキルは、生のコンテンツに次の変換と処理を適用します。

- 多言語検索の場合の翻訳と言語検出
- テキストの大きなチャネルからユーザー名、場所、およびその他のエンティティを抽出するエンティティ認識
- 重要な用語を識別して出力するためのキー フレーズ抽出
- バイナリ ファイル内の印刷されたテキストと手書きのテキストを認識する光学式文字認識 (OCR)
- 画像の内容を説明し、説明を検索可能なテキスト フィールドとして出力する画像分析

AI エンリッチメントは、Azure データ ソースに接続するインデクサー パイプラインの拡張機能です。エンリッチメント パイプラインには、インデクサー パイプラインのすべてのコンポーネント（インデクサー、データ ソース、インデックス）に加えて、アトミック エンリッチメント ステップを指定するスキルセットが含まれます。

次の図に、AI エンリッチメントの進行を示します。



インポートは最初の手順です。ここで、インデクサーがデータソースに接続し、コンテンツ(ドキュメント)を検索サービスにプルします。AIエンリッチメントシナリオで使用される最も一般的なリソースは [Azure Blob Storage](#) ですが、サポートされている任意のデータソースがコンテンツを提供できます。

エンリッチ&インデックスは、ほとんどのAIエンリッチメントパイプラインを対象とします。

- エンリッチメントは、インデクサーが "[ドキュメントを解読](#)" し、画像とテキストを抽出したときに開始されます。次に発生する処理の種類は、データと、スキルセットに追加したスキルによって異なります。画像がある場合は、画像処理を実行するスキルに転送できます。テキストコンテンツは、テキストと自然言語の処理のためにキューに入れられます。内部的には、スキルによって、変換が発生したときにそれを収集する "[エンリッチされたドキュメント](#)" が作成されます。
- エンリッチ済みコンテンツは、スキルセットの実行の間に生成され、ユーザーが保存しない限り一時的なものです。後でスキルセットを実行するときに再利用するため、[エンリッチメントキャッシュ](#)で解読されたドキュメントとスキル出力を保持できます。
- 検索インデックスにコンテンツを取り込むため、インデクサーはエンリッチされたコンテンツをターゲットフィールドに送信するためのマッピング情報を保持している必要があります。[フィールドマッピング](#)(明示的または暗黙的)は、ソースデータから検索インデックスへのデータパスを設定します。[出力フィールドマッピング](#)は、エンリッチされたドキュメントからインデックスへのデータパスを設定します。
- インデックス作成は、生のコンテンツとエンリッチされたコンテンツが[検索インデックス](#)(そのファイルとフォルダー)の物理データ構造に取り込まれるプロセスです。このステップで、字句解析とトーカン化が行われます。

探索は最後の手順です。出力は常に、クライアントアプリからクエリを実行できる[検索インデックス](#)です。必要に応じて出力を、データ探索ツールまたはダウンストリーム

ムプロセスを介してアクセスされる Azure Storage 内の BLOB とテーブルで構成されるナレッジストアにすることができます。ナレッジストアを作成している場合は、エンリッチされたコンテンツのデータパスが[プロジェクト](#)によって決定されます。インデックスとナレッジストアの両方に、同じエンリッチされたコンテンツを含めることができます。

どのような場合に AI エンリッチメントを使用するか

生コンテンツが、非構造化テキスト、画像コンテンツ、または言語検出と翻訳を必要とするコンテンツの場合は、エンリッチメントが有用です。["組み込みのコグニティブスキル"](#) を通じて AI を適用すると、フルテキスト検索とデータサイエンスアプリケーションに対してこのコンテンツのロックが解除されます。

[カスタムスキル](#)を作成して、外部処理を提供することもできます。オープンソース、サードパーティ、またはファーストパーティのコードは、カスタムスキルとしてパイプラインに統合できます。さまざまなドキュメントの種類の顕著な特徴を識別する分類モデルはこのカテゴリに分類されますが、コンテンツの価値を高める任意の外部パッケージを使用できます。

組み込みスキルのユースケース

組み込みのスキルは、[Azure AI Computer Vision](#) と [Language Service](#) などの Azure AI サービス API に基づいています。コンテンツの入力が少ない場合を除いて、より大きなワークフローを実行するために、[課金対象の Azure AI サービス リソースをアタッチする必要があります。](#)

組み込みのスキルを使用して作られた[スキルセット](#)は、次のアプリケーションシナリオに適しています。

- **画像処理**スキルには、[光学式文字認識 \(OCR\)](#) と、[視覚的特徴](#)の識別が含まれます。後者は、顔検出、画像の解釈、画像の認識(有名な人物やランドマーク)、画像の向きのような属性などの識別です。これらのスキルにより、Azure AI Search でフルテキスト検索用の画像コンテンツのテキスト表現が作成されます。
- **機械翻訳**は、多くの場合、多言語ソリューション用の[言語検出](#)と組み合わせて、[テキスト翻訳](#)スキルによって提供されます。
- **自然言語処理**では、テキストのチャunkが分析されます。このカテゴリのスキルには、[エンティティ認識](#)、[センチメント検出 \(オピニオンマイニングを含む\)](#)、[個人を特定できる情報の検出](#)などがあります。これらのスキルによって、構造化さ

れていないテキストが、インデックスで検索可能およびフィルター可能なフィールドとしてマップされます。

カスタム スキルのユース ケース

カスタム スキルは、指定した外部コードを実行し、**カスタム スキル Web インターフェイス**でラップします。カスタム スキルのいくつかの例は、[azure-search-power-skills](#) の GitHub リポジトリにあります。

カスタム スキルは常に複雑とは限りません。たとえば、パターン マッチングまたはドキュメント分類モデルを提供する既存のパッケージがある場合は、カスタム スキルで完成させることができます。

出力を格納する

Azure AI Search では、インデクサーによって作成された出力が保存されます。1回のインデクサー実行で、エンリッチされてインデックス付けされた出力を含む最大3つのデータ構造を作成できます。

[+] テーブルを展開する

| データ ストア | 必 須 | 場所 | 説明 |
|--------------------------------|-----------------------|------------------|--|
| 検索可 能なイ ンデッ クス | 必 須 | 検索サ ービス | フルテキスト検索やその他のクエリ フォームに使用されます。インデックスの指定はインデクサーの要件です。インデックスの内容は、スキルの出力に加えて、インデックス内のフィールドに直接マップされるすべてのソース フィールドから設定されます。 |
| ナレッ ジス トア | オ プ シ ヨ ン | Azure Storage | ナレッジ マイニングやデータ サイエンスなどのダウンストリーム アプリに使用されます。ナレッジストアは、スキルセット内で定義されています。その定義により、エンリッチされたドキュメントが Azure Storage でテーブルとオブジェクト (ファイルと BLOB) のどちらとして投影されるかが決まります。 |
| エンリ ッチメ ント キャッ シュ: | オ プ シ ヨ ン | Azure Storage | 後続のスキルセットの実行で再利用するためのエンリッチメントのキャッシュに使用されます。キャッシュには、インポートされた未処理のコンテンツ (解読されたドキュメント) が格納されます。スキルセットの実行中に作成されたエンリッチされたドキュメントも格納されます。キャッシュは、画像解析または OCR を使用していて、画像ファイルの再処理にかかる時間と費用を回避したい場合に役立ちます。 |

インデックスとナレッジストアは相互に完全に独立しています。インデクサーの要件を満たすにはインデックスをアタッチする必要がありますが、ナレッジストアだけが目的の場合は、作成された後でインデックスを無視してかまいません。

コンテンツを探索する

検索インデックスまたはナレッジストアを定義して読み込んだら、そのデータを探索できます。

検索インデックスに対してクエリを実行する

クエリを実行して、パイプラインによって生成されたエンリッチされたコンテンツにアクセスします。 インデックスは、Azure AI Search 用に作成する他のインデックスと同様です。 カスタム アナライザーを使用してテキスト解析を補完したり、あいまい検索クエリを呼び出したり、フィルターを追加したり、スコアリングプロファイルを実験して検索の関連性を調整したりできます。

ナレッジストアに対してデータ探索ツールを使用する

この Azure Storage ナレッジストアでは、JSON ドキュメントの BLOB コンテナー、イメージオブジェクトの BLOB コンテナー、または Table Storage のテーブルの形式を想定できます。 [Storage Explorer](#)、[Power BI](#)、または Azure Storage に接続する任意のアプリを使用して、コンテンツにアクセスできます。

- BLOB コンテナーは、エンリッチされたドキュメント全体をキャプチャします。これは、他のプロセスにフィードを作成する場合に便利です。
- テーブルは、エンリッチされたドキュメントのスライスが必要な場合、または出力の特定の部分を含めるか除外する場合に便利です。 Power BI での分析には、表がデータの探索や可視化に推奨されるデータソースです。

可用性と料金

エンリッチメントは、Azure AI サービスが利用できるリージョンで利用できます。 エンリッチメントをご利用いただけるかどうかは、「[リージョン別の利用可能な Azure 製品](#)」ページをご確認いただけます。

課金は、従量課金制モデルに従います。 スキルセットで複数リージョンの Azure AI サービスキーが指定されている場合、組み込みスキルを使用するためのコストが転嫁されます。 Azure AI Search によって測定される画像抽出に関するコストもあります。 ただし、テキスト抽出とユーティリティのスキルは請求されません。 詳細については、「[Azure AI Search の課金方法](#)」を参照してください。

一般的なワークフローのチェックリスト

エンリッチメントパイプラインは、"スキルセット" を含む "インデクサー" で構成されます。インデックス作成の後で、インデックスのクエリを実行して結果を検証できます。

サポートされるデータソース内のデータのサブセットから始めます。インデクサーとスキルセットの設計は、反復的なプロセスです。代表的なデータセットが小さいと作業が速くなります。

1. データへの接続が指定されているデータソースを作成します。
2. スキルセットを作成します。プロジェクトが小さい場合を除き、Azure AI マルチサービスリソースをアタッチする必要があります。ナレッジストアを作成する場合は、スキルセット内でそれを定義します。
3. 検索インデックスを定義するインデックススキーマを作成します。
4. インデクサーを作成して実行し、上記のすべてのコンポーネントをまとめます。この手順では、データの取得、スキルセットの実行、インデックスの読み込みを行います。
インデクサーでは、検索インデックスへのデータパスを設定するフィールドマッピングと出力フィールドマッピングも指定します。
必要に応じて、インデクサーの構成でエンリッチメントキャッシュを有効にします。この手順により、後で既存のエンリッチメントを再利用できるようになります。
5. クエリを実行して結果を評価するか、デバッグセッションを開始してスキルセットの問題を解決します。

上記の手順を繰り返す場合は、インデクサーを実行する前にリセットします。または、実行ごとにオブジェクトを削除して再作成します (Free レベルを使用している場合は推奨)。インデクサーのキャッシュを有効にした場合、ソースでデータが変更されていない場合、およびパイプラインに対する編集によってキャッシュが無効にされない場合は、インデクサーがキャッシュからプルされます。

次のステップ

- クイックスタート: AI エンリッチメントのスキルセットを作成する
- チュートリアル: AI エンリッチメント REST API について学習する
- スキルセットの概念
- ナレッジストアの概念
- スキルセットを作成する
- ナレッジストアを作成する

Azure AI Search での増分 エンリッチメントとキャッシュ

[アーティクル] • 2024/02/18

① 重要

この機能はパブリック プレビュー段階にあり、[追加使用条件](#) の下で提供されます。この機能は、[プレビュー REST API](#) でサポートされます。

"インクリメンタル エンリッチメント" とは、[スキルセットの実行](#)中にキャッシュされたエンリッチメントを使うことを指します。こうすることで、新規および変更されたスキルとドキュメントにのみ、Azure AI サービスへの API 呼び出しの従量課金処理料金が発生します。キャッシュには、[ドキュメント解析](#)の出力に加え、ドキュメントごとの各スキルの出力が格納されます。キャッシュは課金対象 (Azure Storage を使用) です。ただし、ストレージのコストは画像抽出や AI 処理よりも低いため、強化全体コストは削減されます。

キャッシュを有効にすると、インデクサーによって更新内容が評価され、既にあるエンリッチメントをキャッシュから取得できるかどうかが判断されます。ドキュメント解析フェーズから出力される画像やテキスト コンテンツに加え、編集の上流のスキル出力や、編集に対して直交するスキル出力は再利用できる可能性が高くなります。

スキルセットの処理が完了した後、更新された結果は、キャッシュだけでなく、検索インデックスまたはナレッジ ストアにも書き戻されます。

制限事項

⊗ 注意事項

[SharePoint Online インデクサー \(プレビュー\)](#) を使用している場合は、増分エンリッチメントを避ける必要があります。特定の状況では、キャッシュが無効になり、キャッシュを再ロードする場合は、[インデクサーをリセットして実行](#)する必要があります。

キャッシュの構成

物理的には、ご使用の Azure ストレージ アカウントの BLOB コンテナーに、インデクサーごとに 1 つキャッシュが格納されます。 それぞれのインデクサーには、使っているコンテナーに対応する一意で不变のキャッシュ識別子が割り当てられます。

キャッシュは、"cache" プロパティを指定してインデクサーを実行すると作成されます。 キャッシュできるのは、エンリッチされたコンテンツのみです。 インデクサーにスキルセットがアタッチされていない場合、キャッシュは適用されません。

次の例は、キャッシュが有効になっているインデクサーを示しています。 詳細な手順については、[エンリッチメント キャッシュの有効化](#)に関するページを参照してください。 cache プロパティを追加するときの要求では、[プレビュー API バージョン](#) (2020-06-30-Preview 以降) を使うことに注意してください。

JSON

```
POST https://[search service name].search.windows.net/indexers?api-version=2020-06-30-Preview
{
    "name": "myIndexerName",
    "targetIndexName": "myIndex",
    "dataSourceName": "myDatasource",
    "skillsetName": "mySkillset",
    "cache": {
        "storageConnectionString": "<Your storage account connection string>",
        "enableReprocessing": true
    },
    "fieldMappings": [],
    "outputFieldMappings": [],
    "parameters": []
}
```

キャッシュ管理

キャッシュのライフサイクルは、インデクサーによって管理されます。 インデクサーが削除されると、そのキャッシュも削除されます。 インデクサーの cache プロパティが null に設定されるか、接続文字列が変更された場合、既存のキャッシュはインデクサーの次回の実行時に削除されます。

インクリメンタル エンリッチメントはユーザーの介入なしに変更を検出して対応するように設計されています。 その一方で、特定の動作を呼び出すために使用できるパラメーターが用意されています。

- 新しいドキュメントを優先する
- スキルセット チェックをバイパスする

- データ ソース チェックをバイパスする
- スキルセットの評価を強制的に実行する

新しいドキュメントを優先する

cache プロパティには、`enableReprocessing` パラメータが含まれます。キャッシュ内で既に表現されている受信ドキュメントの処理を制御する目的で使用されます。true(既定値)の場合、インデクサーを再実行すると、キャッシュ内に既にあるドキュメントが再処理されます。これは、スキルの更新によってそのドキュメントが影響を受けると仮定されるためです。

false の場合、既存のドキュメントは再処理されず、実質的に新しい受信コンテンツが既存のコンテンツより優先されます。`enableReprocessing` を false に設定するのは、あくまで一時的な措置としてください。ほとんどの場合は `enableReprocessing` を true に設定しておくと、新規と既存両方のすべてのドキュメントが、最新のスキルセット定義に従って確実に有効になります。

スキルセットの評価をバイパスする

通常、スキルの変更とそのスキルの再処理は連動します。ただし、スキルを変更しても再処理が行われてはならない場合がいくつかあります(たとえば、新しい場所に、または新しいアクセスキーを使って、カスタムスキルをデプロイする場合)。ほとんどの場合、これらは、スキル出力の実体そのものに実際の影響を与えない末梢的な変更です。

スキルに対する変更が実際は表面的なものであることがわかっている場合は、`disableCacheReprocessingChangeDetection` パラメータを true に設定して、スキルの評価をオーバーライドする必要があります。

1. [スキルセットの更新](#)を呼び出し、スキルセット定義を変更します。
2. 要求に "disableCacheReprocessingChangeDetection=true" パラメーターを追加します。
3. 変更を送信します。

このパラメーターを設定すると、スキルセット定義に対する更新だけがコミットされ、変更が既存のキャッシュに及ぼす影響は評価されません。プレビューの API バージョンである 2020-06-30-Preview 以降を使用してください。

HTTP

```
PUT https://[servicename].search.windows.net/skillsets/[skillset name]?api-version=2020-06-30-Preview&disableCacheReprocessingChangeDetection
```

データ ソースの検証チェックをバイパスする

ほとんどの場合、データ ソース定義を変更すると、キャッシングが無効になります。ただし、接続文字列の変更やストレージ アカウントのキーのローテーションなど、変更によってキャッシングが無効になってはならないことがわかっているシナリオでは、[データ ソースの更新](#)で `ignoreResetRequirement` パラメータを追加します。このパラメーターを `true` に設定すると、リセット条件 (結果的にすべてのオブジェクトが最初から再構築されて設定される条件) をトリガーすることなく、コミットを実行できます。

HTTP

```
PUT https://[search service].search.windows.net/datasources/[data source name]?api-version=2020-06-30-Preview&ignoreResetRequirement
```

スキルセットの評価を強制的に実行する

キャッシングの目的は不必要的処理を回避することにあります。ここで、インデクサーによって検出されない変更 (たとえば、カスタム スキルなどの外部コード内の変更) をスキルに加えるケースを考えてみましょう。

この場合は、[スキルのリセット](#)を使用して、特定のスキル (そのスキルの出力に依存するダウンストリームのスキルも含まれます) を強制的に再処理することができます。この API は、無効にして再処理用にマークする必要があるスキルのリストが含まれた POST 要求を受け取ります。スキルのリセット後、[インデクサー実行](#)要求を行ってパイプライン処理を呼び出します。

特定のドキュメントを再キャッシングする

[インデクサーのリセット](#)を使用すると、検索コーパス内のすべてのドキュメントが再処理されます。再処理を必要とするドキュメントがごく少数のシナリオでは、[ドキュメントのリセット \(レビュー\)](#)を使用して、特定のドキュメントを強制的に再処理します。ドキュメントがリセットされると、インデクサーによってそのドキュメントのキャッシングが無効にされ、ドキュメントはその後データ ソースから読み取ることによって再処理されます。詳細については、[インデクサー、スキル、ドキュメントの実行またはリセット](#)に関するページを参照してください。

特定のドキュメントをリセットする場合は、検索インデックスから読み取るドキュメントキーのリストを要求で指定します。キーが外部データソースのフィールドにマップ

されている場合、指定する値は検索インデックスで使用されているものであることが必要です。

API の呼び出し方法に応じて、要求ではキー リストを追加、上書き、またはキューに登録します。

- 異なるキーを使用して API を複数回呼び出すと、ドキュメントキーのリセットの一覧に新しいキーが追加されます。
- "overwrite" クエリ文字列パラメーターを true に設定して API を呼び出すと、リセットされるドキュメントキーの現在のリストが要求のペイロードで上書きされます。
- API を呼び出しても、ドキュメントキーはインデクサーによって実行される処理のキューに追加されるだけです。スケジュールに従ってまたはオンデマンドでインデクサーが次に呼び出されると、データ ソースからの他の変更よりも、リセットされたドキュメントキーの処理が優先されます。

次の例は、ドキュメントのリセット要求を示しています。

HTTP

```
POST https://[search service name].search.windows.net/indexers/[indexer
name]/resetdocs?api-version=2020-06-30-Preview
{
    "documentKeys" : [
        "key1",
        "key2",
        "key3"
    ]
}
```

キャッシングが無効になる変更

キャッシングを有効にすると、インデクサーによってパイプライン構成の変更が評価され、再利用できるコンテンツと再処理が必要なコンテンツが特定されます。このセクションでは、キャッシングが完全に無効になる変更を示した後、増分処理がトリガーされる変更を示します。

無効化につながる変更とは、キャッシング全体の有効性が失われる変更をいいます。たとえばデータ ソースの更新は、無効化につながる変更です。次に示すのは、インデクサー パイプラインのいずれかの部分に対する変更で、キャッシングが無効になるすべてのものの一覧です。

- データ ソースの種類の変更

- データソースコンテナーの変更
- データソースの資格情報の変更
- データソースの変更検出ポリシーの変更
- データソースの削除検出ポリシーの変更
- インデクサーのフィールドのマッピングの変更
- インデクサーのパラメーターの変更:
 - 解析モード
 - ファイル名拡張子を除外
 - ファイル名拡張子のインデックスを作成
 - サイズの大きいドキュメントのストレージメタデータのみのインデックスを作成
 - 区切りテキストのヘッダー
 - 区切りテキストの区切り記号
 - ドキュメントのルート
 - 画像操作(画像の抽出方法に対する変更)

増分処理がトリガーされる変更

増分処理では、対象のスキルセット定義が評価された後、再実行する必要があるスキルが特定され、ドキュメントツリーの影響を受ける部分が選択的に更新されます。結果的にインクリメンタルエンリッチメントが発生する変更の完全な一覧を次に示します。

- スキルの種類を変更する(スキルの OData 型が更新された)。
- スキルに固有のパラメーター(URL、defaults など)が更新された。
- スキルの出力が変更(スキルから返される出力が追加または変更)された。
- 先祖の変更を伴うスキルの入力の変更があった(スキルのチェーンが変更された)。
- アップストリームのスキルが無効化された(このスキルへの入力となっているスキルが更新された場合)。
- ドキュメントの再プロジェクトを伴う更新がナレッジストアのプロジェクト場所に生じた。
- ドキュメントの再プロジェクトを伴う変更がナレッジストアのプロジェクト場所に生じた。
- インデックスに対するドキュメントの再プロジェクトを伴う変更が、インデクサーの出力フィールドのマッピングに生じた。

キャッシュに使用される API

REST API バージョン 2020-06-30-Preview 以降では、インデクサーの追加のプロパティを使用して、インクリメンタル エンリッチメントが提供されます。スキルセットとデータ ソースは、一般公開されているバージョンを使用できます。操作の順序について詳しくは、リファレンス ドキュメントに加えて、[インクリメンタル エンリッチメント用のキャッシュの構成](#)に関するページを参照してください。

- インデクサーの作成または更新 (api-version=2020-06-30-Preview)
- スキルセットの更新 (api-version=2020-06-30) (要求での新しい URI パラメーター)
- スキルのリセット (api-version=2020-06-30)
- データ ソースの更新: プレビュー API バージョンで呼び出されるときは、"ignoreResetRequirement" という名前の新しいパラメータを指定し、更新アクションによってキャッシュが無効になってはならないときはそれを true に設定する必要があります。"ignoreResetRequirement" は、簡単に検出できない不整合が意図せずデータに発生する可能性があるため、慎重に使ってください。

次のステップ[°]

インクリメンタル エンリッチメントは、変更の追跡をスキルセットと AI エンリッチメントに拡張する強力な機能です。インクリメンタル エンリッチメントを使用すると、スキルセットの設計を反復処理する際に、既存の処理済みコンテンツを再利用できます。次のステップで、インデクサーのキャッシュを有効にします。

[インクリメンタル エンリッチメントのキャッシュを有効にする](#)

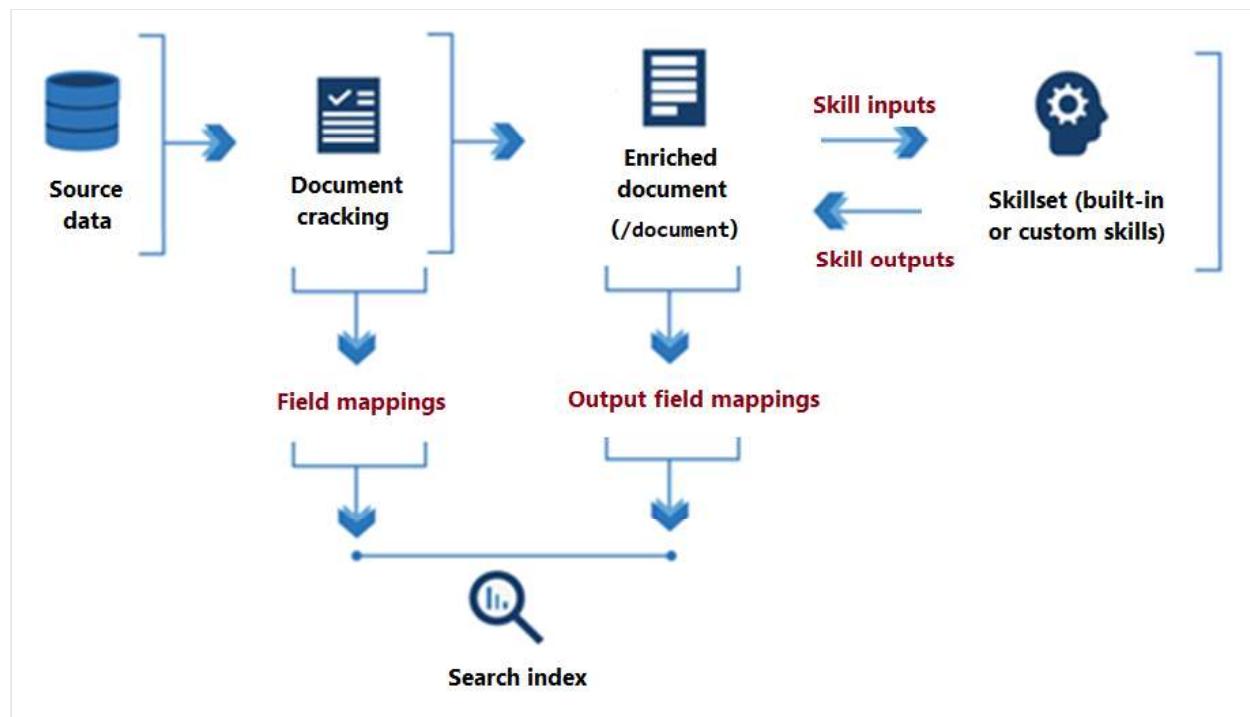
Azure Cognitive Search のスキルセットの概念

[アーティクル] • 2023/08/08

この記事は、スキルセットの概念と構成について理解を深める必要がある開発者を対象としています。ここでは、AI エンリッチメントの高度な概念に精通していることを前提としています。

スキルセットは、インデクサーにアタッチされている Azure Cognitive Search の再利用可能なリソースです。これには、外部のデータ ソースから取得したドキュメントに対して、組み込みの AI や外部カスタム処理を呼び出す 1 つ以上のスキルが含まれています。

次の図は、スキルセット実行の基本的なデータ フローを示しています。



スキルによって行われるのは、スキルセット処理の開始から終了まで、"エンリッチされたドキュメント"に対する読み取りと書き込みです。最初は、エンリッチされたドキュメントは、データ ソースから抽出された生コンテンツだけです ("`/document`" ルートノードとして明確に表現されています)。スキルが実行されるたびに、スキルによってその出力がグラフのノードとして書き込まれるため、エンリッチされたドキュメントは、構造と実質的な内容を取得していきます。

スキルセットの実行が完了すると、エンリッチされたドキュメントの出力は、"出力フィールド マッピング" を介してインデックスに組み込まれます。ソースからインデッ

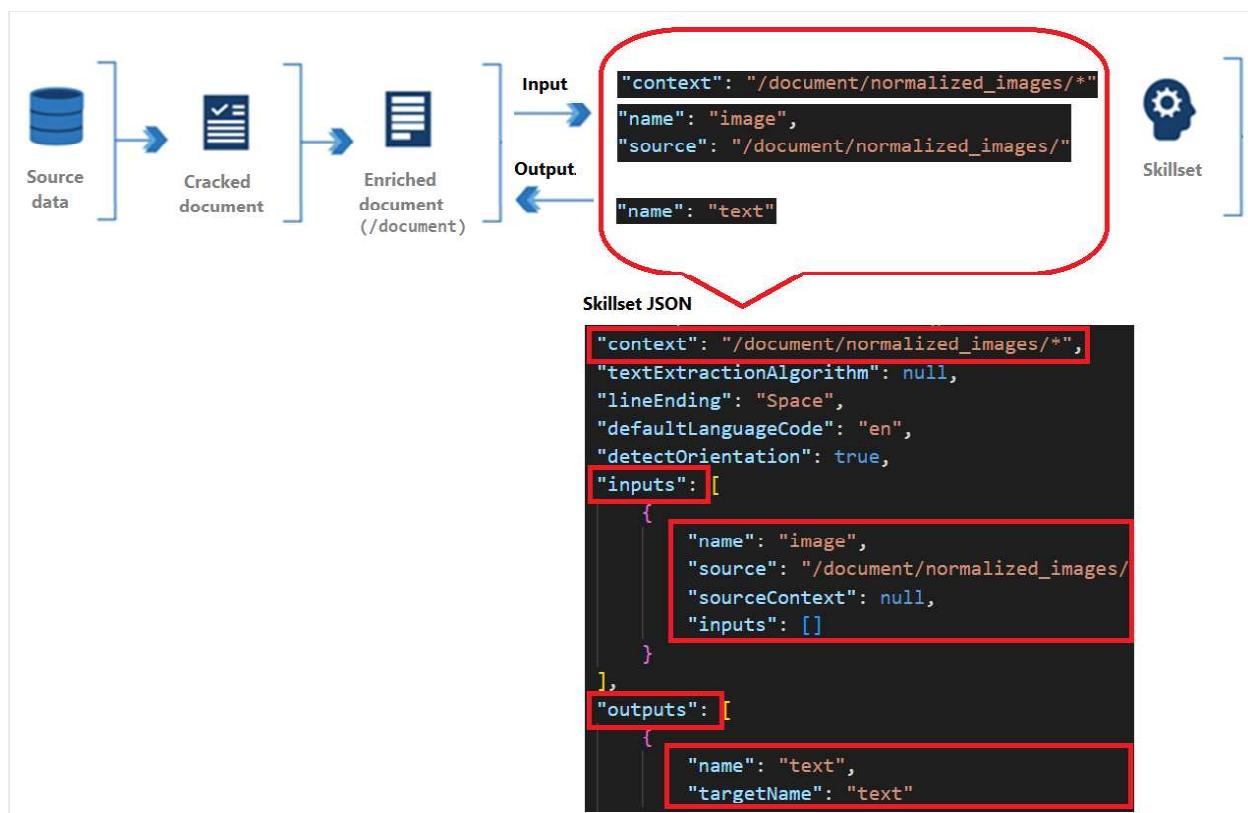
クスにそのまま転送する生コンテンツは、"フィールド マッピング" によって定義されます。

エンリッチメントを構成するには、スキルセットとインデクサーで設定を指定します。

スキルセットの定義

スキルセットとは、画像ファイル上のテキストや OCR の翻訳など、エンリッチメントを実行する 1 つ以上の "スキル" の配列です。スキルは、Microsoft の組み込みスキル、または外部でホストするロジックを処理するためのカスタムスキルのいずれかになります。スキルセットにより、インデックスを付けるときに使用される、またはナレッジストアにプロジェクトされるエンリッチされたドキュメントが生成されます。

スキルには、コンテキスト、入力、出力があります。



- コンテキストは、ドキュメントごとに 1 回、またはコレクション内のアイテムごとに 1 回の操作のスコープを指します。
- 入力はエンリッチされたドキュメント内のノードから発信されます。ここで、"source" と "name" は特定のノードを識別します。
- 出力は、エンリッチされたドキュメントに新しいノードとして返送されます。値は、ノードの "name" とノードの内容です。ノード名が重複している場合は、あいまいさを解消するためにターゲット名を設定できます。

スキル コンテキスト

各スキルにはコンテキストがあります。コンテキストはドキュメント全体 (`/document`) であったり、ツリー内の下位のノード (`/document/countries/*`) であったりします。コンテキストによって次のことが決まります。

- 1つの値に対してスキルが実行される回数 (フィールドごと、ドキュメントごとに1回)。型コレクションのコンテキスト値の場合、`/*` を追加すると、コレクション内のインスタンスごとにスキルが1回呼び出されます。
- 出力の宣言。スキルの出力が追加される強化ツリー内の場所。出力は、常にコンテキストノードの子としてツリーに追加されます。
- 入力の形状。複数レベルのコレクションの場合、コンテキストを親コレクションに設定すると、スキル入力の形状に影響します。たとえば、国または地域のリストが含まれる強化ツリーがあり、それぞれが郵便番号のリストを含む州のリストでエンリッチメント処理されている場合、コンテキストをどのように設定するかによって、入力がどのように解釈されるかが決まります。

| Context | 入力 | 入力の形状 | スキルの呼び出し |
|------------------------------------|--|-----------------------|----------|
| <code>/document/countries/*</code> | <code>/document/countries/*/states/*/zipcodes/*</code> | 国または地域ごとのすべての郵便番号のリスト | 1回 |

| Context | 入力 | 入力の形状 | スキルの呼び出し |
|--------------------------------|---|-------------|---------------------|
| | | スト | |
| /document/countries/*/states/* | /document/countries/*/states/*/zipcodes/* | 州内の郵便番号のリスト | 国または地域と州の組み合わせごとに1回 |

スキルの依存関係

スキルは、独立して並列に実行することも、あるスキルの出力を別のスキルにフィードする場合は順番に実行することもできます。次の例は、順番に実行される 2 つの組み込みスキルを示しています。

- スキル #1 は、"reviews_text" ソースフィールドのコンテンツを入力として受け取り、そのコンテンツを出力として 5,000 文字の "pages" に分割する[テキスト分割スキル](#)です。大きなテキストを小さなチャunkに分割すると、センチメント検出などのスキルの結果が向上する可能性があります。
- スキル #2 は、"pages" を入力として受け入れ、センチメント分析の結果が含まれる "Sentiment" という名前の新しいフィールドを作成する[センチメント検出スキル](#)です。

最初のスキル ("pages") の出力が感情分析でどのように使用されているかに注目してください。ここで、"/document/reviews_text/pages/*" はコンテキストと入力の両方です。パスの構文の詳細については、[スキルセットで注釈を参照する方法](#)に関するページを参照してください。

```
JSON

{
  "skills": [
    {
      "@odata.type": "#Microsoft.Skills.Text.SplitSkill",
      "name": "#1",
      "description": null,
      "context": "/document/reviews_text",
      "defaultLanguageCode": "en",
      "textSplitMode": "pages",
      "maximumPageLength": 5000,
      "inputs": [
        {
          "name": "text",
          "source": "/document/reviews_text"
        }
      ],
      "outputs": [
        {
          "name": "textItems",
          "targetName": "pages"
        }
      ]
    },
    {
      "@odata.type": "#Microsoft.Skills.Text.SentimentSkill",
      "name": "#2",
      "description": null,
      "context": "/document/reviews_text/pages/*",
      "defaultLanguageCode": "en",
      "inputs": [
        {
          "name": "text",
          "source": "/document/reviews_text/pages/*"
        }
      ],
      "outputs": [
        {
          "name": "sentiment",
          "targetName": "sentiment"
        },
        {
          "name": "confidenceScores",
          "targetName": "confidenceScores"
        },
        {
          "name": "sentences",
          "targetName": "sentences"
        }
      ]
    }
  ]
}
```

```

        "targetName": "sentences"
    }
]
}
...
]
}

```

強化ツリー

エンリッチされたドキュメントは、スキルセットの実行中に作成された一時的なツリー状のデータ構造で、スキルを通じて行われたすべての変更を収集します。集合的に、エンリッチメントはアドレス指定可能なノードの階層として表されます。ノードには、外部データ ソースから逐語的に渡されたエンリッチされていないフィールドも含まれます。

エンリッチされたドキュメントが存在するのは、スキルセットの実行中ですが、[キャッシュ](#)したり、[ナレッジストア](#)に送信したりできます。

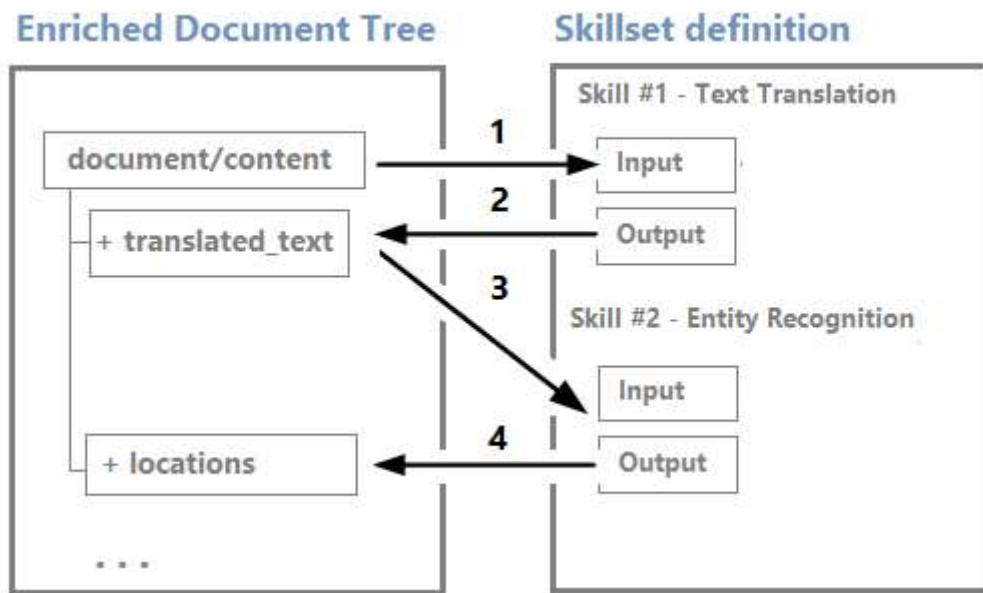
エンリッチされたドキュメントは、最初は、"ドキュメント解析" 中にデータ ソースから抽出されたコンテンツに過ぎません。ここでテキストやイメージがソースから抽出され、言語解析やイメージ解析に利用できるようになります。

最初のコンテンツはメタデータと "ルート ノード" (`document/content`) です。ルート ノードは、通常、ドキュメント全体であるか、ドキュメント解析中にデータ ソースから抽出された正規化されたイメージです。強化ツリーでの表現方法は、データ ソースの種類によって異なります。次の表は、サポートされているいくつかのデータ ソースについて、エンリッチメントパイプラインに入ったドキュメントの状態を示しています。

| データ ソース/解析モード | Default | JSON、JSON 行、および CSV |
|-----------------|---|---|
| Blob Storage | /document/content /document/normalized_images/* ... | /document/{key1} /document/{key2} ... |
| Azure SQL | /document/{column1} /document/{column2} ... | 該当なし |
| Azure Cosmos DB | /document/{key1} /document/{key2} ... | 該当なし |

スキルが実行されると、出力が新しいノードとしてエンリッチメントツリーに追加されます。スキルの実行がドキュメント全体に対して行われる場合、ノードはルートの下の最初のレベルに追加されます。

ノードは、ダウンストリームスキルの入力として使用できます。たとえば、翻訳された文字列などのコンテンツを作成するスキルは、エンティティを認識したり、キーフレーズを抽出したりするスキルの入力になる可能性があります。



[デバッグセッションのビジュアルエディター](#)を使用してエンリッチメントツリーを視覚化して操作できますが、ほとんどは内部構造です。

エンリッチメントは変更できません。ノードは一度作成されたら編集できません。スキルセットや強化ツリーの複雑さが増しても、強化ツリー内のすべてのノードをインデックスやナレッジストアにする必要はありません。

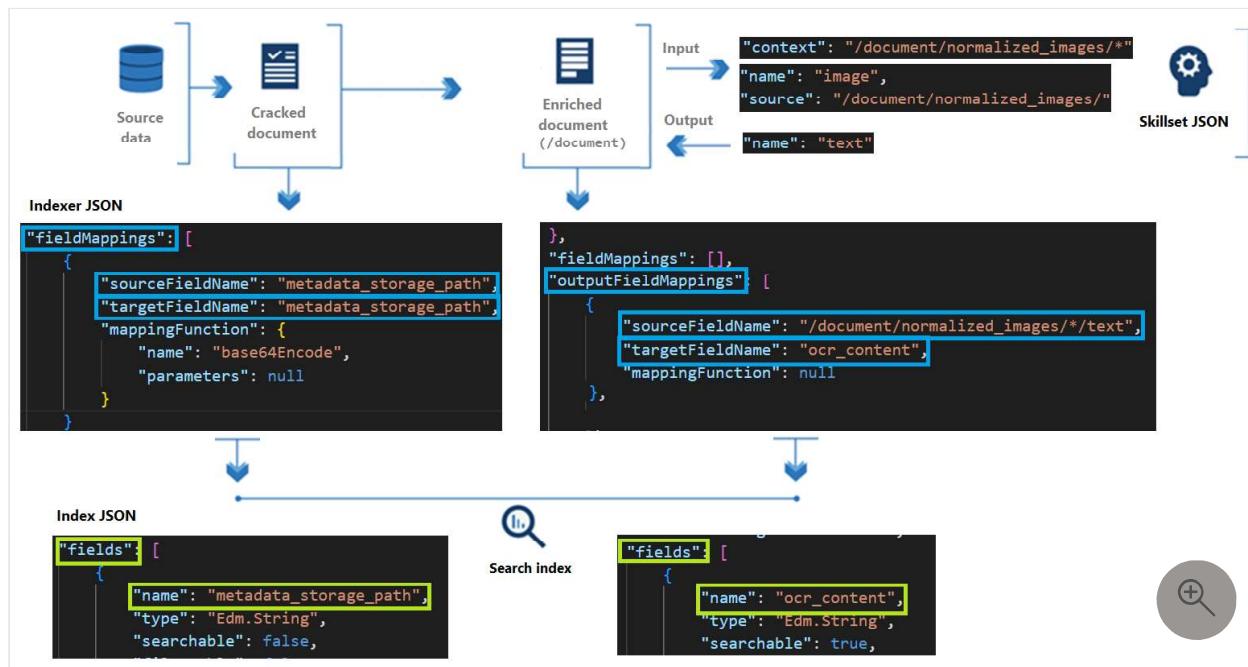
エンリッチメント出力のサブセットのみを選択的に保持して、使用する情報のみを保持することができます。インデクサー定義の[出力フィールドマッピング](#)によって、検索インデックスに実際に取り込まれるコンテンツが決まります。同様に、ナレッジストアを作成する場合は、プロジェクトに割り当てられている[シェイプ](#)に、出力をマップできます。

① 注意

強化ツリー形式により、エンリッチメントパイプラインでは、メタデータをプリミティブデータ型にもアタッチできます。メタデータは有効な JSON オブジェクトになりませんが、ナレッジストア内のプロジェクト定義で有効な JSON 形式にプロジェクトできます。詳細については、[Shaper スキル](#)に関するページをご覧ください。

インデクサーの定義

インデクサーには、インデクサーの実行を構成するために使用されるプロパティとパラメーターがあります。これらのプロパティの中には、検索インデックス内のフィールドにデータパスを設定するマッピングがあります。



マッピングには次の 2 つのセットがあります。

- "fieldMappings" は、ソース フィールドを検索フィールドにマップします。
- "outputFieldMappings" は、エンリッチされたドキュメント内のノードを検索フィールドにマップします。

"sourceFieldName" プロパティは、データ ソース内のフィールドまたはエンリッチメントツリー内のノードのいずれかを指定します。 "targetFieldName" プロパティは、コンテンツを受け取るインデックス内の検索フィールドを指定します。

エンリッチメントの例

この例では、[ホテル レビュースキルセット](#) を参照ポイントとして使用し、スキルの実行によって強化ツリーがどのように発展するかを概念図を使って説明します。

また、この例では以下もわかります。

- スキルの実行回数の決定に対して、スキルのコンテキストと入力がどのように働くか
- コンテキストに基づく入力の形状

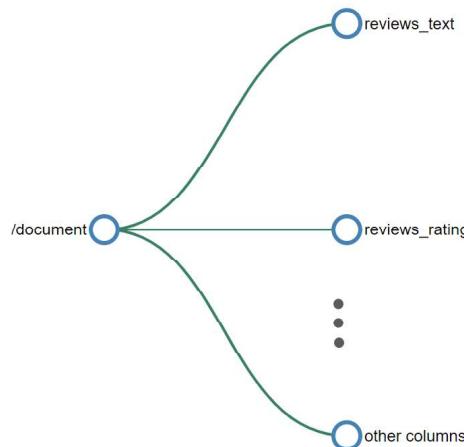
この例では、CSV ファイルのソース フィールドに、ホテルに関する顧客レビュー ("reviews_text") と評価 ("reviews_rating") が含まれています。インデクサーによって、BLOB ストレージからメタデータ フィールドが追加され、スキルによって、翻訳されたテキスト、センチメント スコア、キーフレーズ検出が追加されます。

ホテルレビューの例で、エンリッチメントプロセス内の "ドキュメント" は 1 つのホテルレビューを表します。

💡 ヒント

このデータの検索インデックスやナレッジ ストアを作成するには、[Azure portal](#)、[Postman](#)、または [REST API](#) を使用します。 [デバッグ セッション](#)を使用して、スキルセットの構成、依存関係、強化ツリーへの影響の分析情報を確認することもできます。 この記事のイメージは、デバッグ セッションからプルされました。

概念的には、最初の強化ツリーは次のようにになります。



すべての強化のルート ノードは `"/document"` です。 BLOB インデクサーを使用すると、`"/document"` ノードに子ノード `"/document/normalized_images"` と `"/document/content"` ができます。この例のように、データが CSV の場合、列名は `"/document"` の下のノードにマップされます。

スキル #1: 分割スキル

ソース コンテンツが大量のテキストで構成されている場合は、言語、センチメント、キーフレーズ検出の精度を高めるために、より小さいコンポーネントに分割すると便利

です。 使用できるグレインは、ページと文の 2 つです。 ページは約 5,000 文字で構成されます。

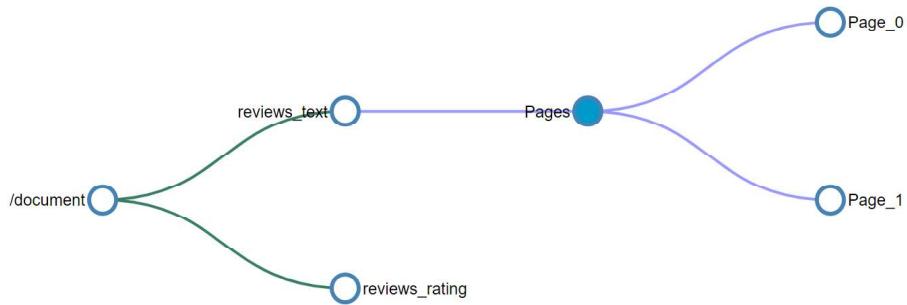
通常、テキスト分割スキルは、スキルセットにおいて最初に選ばれます。

JSON

```
"@odata.type": "#Microsoft.Skills.Text.SplitSkill",
"name": "#1",
"description": null,
"context": "/document/reviews_text",
"defaultLanguageCode": "en",
"textSplitMode": "pages",
"maximumPageLength": 5000,
"inputs": [
{
    "name": "text",
    "source": "/document/reviews_text"
}
],
"outputs": [
{
    "name": "textItems",
    "targetName": "pages"
}]
```

`/document/reviews_text` のスキルコンテキストでは、この分割スキルは `reviews_text` に対して 1 回実行されます。 スキルの出力はリストであり、`reviews_text` が 5000 文字のセグメントに分割されています。 分割スキルからの出力は、`pages` という名前が付けられ、エンリッチメントツリーに追加されます。`targetName` 機能を使用することで、強化ツリーに追加される前に、スキル出力の名前を変更できます。

これで、強化ツリーのスキルのコンテキストの下に、新しいノードが配置されました。 このノードは、任意のスキル、プロジェクト、または出力フィールド マッピングで使用できます。

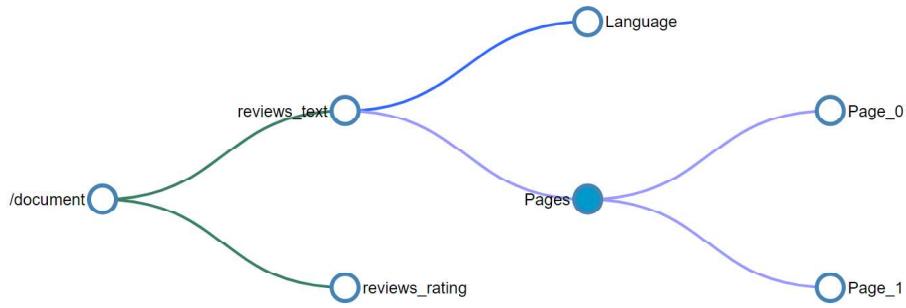


スキルによってノードに追加されたいずれかの強化にアクセスするには、強化の完全なパスが必要です。たとえば、`pages` ノードのテキストを別のスキルへの入力として使用する場合は、`"/document/reviews_text/pages/*"` として指定する必要があります。パスの詳細については、[注釈の参照](#)に関するページをご覧ください。

スキル #2: 言語検出

ホテルレビュー ドキュメントには、複数の言語で表される顧客フィードバックが含まれています。言語検出スキルによって、どの言語が使用されているか判断されます。結果は、キー フレーズ抽出とセンチメント検出に渡され(非表示)、センチメントと語句を検出するときに言語が考慮されます。

言語検出スキルは、スキルセットで定義されている 3 番目のスキル(スキル #3)ですが、次に実行されるスキルです。入力は必要ないので、前のスキルと並行して実行されます。先行する分割スキルと同様に、言語検出スキルもドキュメントごとに 1 回呼び出されます。強化ツリーには言語用の新しいノードが追加されています。

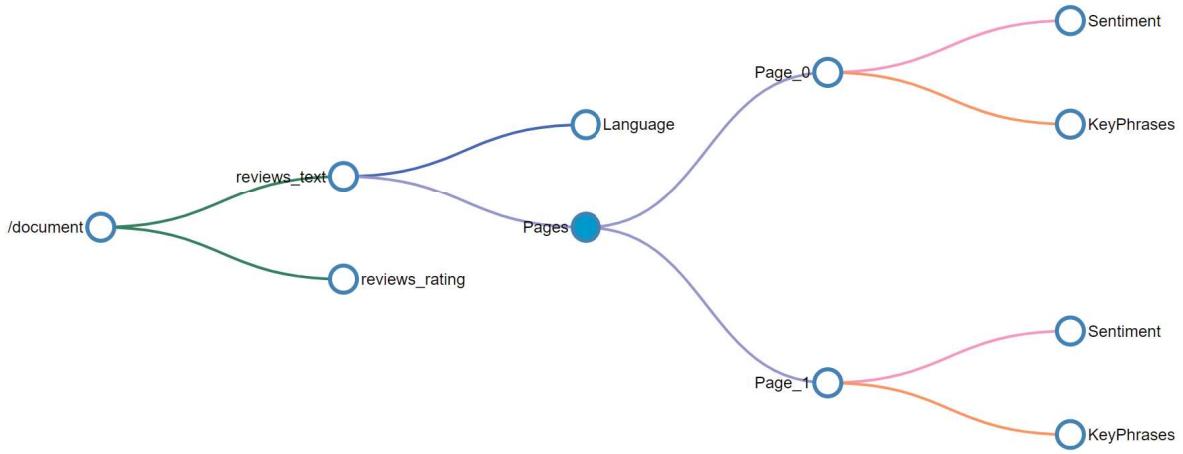


スキル #3 と #4 (感情分析とキー フレーズ検出)

お客様からのフィードバックには、ポジティブなものからネガティブなものまで、さまざまなエクスペリエンスが反映されています。フィードバックは、感情分析スキルによって分析され、負数から正数の範囲内で一連のスコアが割り当てられます。センチメントがはっきりしない場合は、中立が割り当てられます。感情分析と同時に、結果的に重要であると思われる単語と短いフレーズが、キー フレーズ検出によって特定され、抽出されます。

コンテキストとして `/document/reviews_text/pages/*` が指定された感情分析スキルとキー フレーズ スキルは両方とも、`pages` コレクション内の項目ごとに 1 回ずつ呼び出されます。スキルからの出力は、関連付けられている `page` 要素の下のノードになります。

スキルセットに含まれる残りのスキルを表示し、各スキルの実行によって強化のツリーがどのように成長し続けるかを視覚化できるようになるはずです。マージスキルや Shaper スキルなどの一部のスキルでも新しいノードが作成されますが、既存のノードのデータのみが使用され、新しい強化は作成されません。



上のツリーのコネクタの色は、エンリッチメントが異なるスキルで作成されたことを示しています。ノードは個別に指定する必要があり、親ノードを選択したときに返されるオブジェクトの一部にはなりません。

スキル #5: Shaper スキル

出力にナレッジストアが含まれる場合は、最後のステップとして **Shaper スキル**を追加します。 Shaper スキルによって、強化ツリー内のノードからデータ シエイプが作成されます。たとえば、複数のノードを 1 つのシェイプに統合することができます。その後、このシェイプをテーブルとして射影し(ノードはテーブル内の列になります)、名前によってシェイプをテーブル プロジェクションに渡します。

整形が 1 つのスキルにまとめられているため、Shaper スキルは簡単に操作できます。また、個々のプロジェクト内でインライン整形を選ぶこともできます。 Shaper スキルによってエンリッチメントツリーの加減が行われることはありません。したがって、これは視覚化されません。代わりに、Shaper スキルは、既にあるエンリッチメントツリーを再現する手段と考えることができます。概念的には、これはデータベース内のテーブルからビューを作成するのと似ています。

```

JSON

{
  "@odata.type": "#Microsoft.Skills.Util.ShaperSkill",
  "name": "#5",
  "description": null,
  "context": "/document",
  "inputs": [
    {
      "name": "name",
      "source": "/document/name"
    },
    {
      "name": "text",
      "source": "/document/reviews/text"
    }
  ],
  "outputs": [
    {
      "name": "sentiment",
      "target": "Sentiment"
    },
    {
      "name": "keyphrases",
      "target": "KeyPhrases"
    }
  ]
}
  
```

```
{
  "name": "reviews_date",
  "source": "/document/reviews_date"
},
{
  "name": "reviews_rating",
  "source": "/document/reviews_rating"
},
{
  "name": "reviews_text",
  "source": "/document/reviews_text"
},
{
  "name": "reviews_title",
  "source": "/document/reviews_title"
},
{
  "name": "AzureSearch_DocumentKey",
  "source": "/document/AzureSearch_DocumentKey"
},
{
  "name": "pages",
  "sourceContext": "/document/reviews_text/pages/*",
  "inputs": [
    {
      "name": "Sentiment",
      "source": "/document/reviews_text/pages/*/Sentiment"
    },
    {
      "name": "LanguageCode",
      "source": "/document/Language"
    },
    {
      "name": "Page",
      "source": "/document/reviews_text/pages/*"
    },
    {
      "name": "keyphrase",
      "sourceContext": "/document/reviews_text/pages/*/Keyphrases/*",
      "inputs": [
        {
          "name": "Keyphrases",
          "source": "/document/reviews_text/pages/*/Keyphrases/*"
        }
      ]
    }
  ],
  "outputs": [
    {
      "name": "output",
      "targetName": "tableprojection"
    }
  ]
}
```

]
}

次のステップ°

概要と例を参考にしながら、[組み込みスキル](#)を使用して、最初のスキルセットを作成してみてください。

[最初のスキルセットを作成する](#)

Azure AI Search 内の統合データのチャンキングと埋め込み

[アーティクル] • 2024/03/27

① 重要

この機能はパブリック プレビュー段階にあり、[追加使用条件](#) の下で提供されます。この機能は、[2023-10-01-Preview REST API](#) でサポートされます。

統合ベクター化によって、データ チャンキングとテキスト-to-ベクター埋め込みがインデクサーベース インデックス作成のスキルに追加されます。テキスト-to-ベクター変換もクエリに追加されます。

この機能は、プレビューのみ段階です。一般提供バージョンのベクトル検索と以前のプレビュー バージョンでは、データのチャンキングとベクター化が外部のチャンキングとベクターのコンポーネントに依存しており、アプリケーション コードが各手順を操作し、調整する必要があります。このプレビュー版では、チャンキングとベクター化がスキルおよびインデクサーを通じてインデックス作成に組み込まれています。テキスト分割スキルを使用してデータをチャunkするスキルセットをセットアップして、それから AzureOpenAIEmbedding スキルまたはカスタム スキルを使用して埋め込みモデルを呼び出すことができます。インデックス作成時に使用されるあらゆるベクトル化を、テキストをベクターに変換するクエリで呼び出すこともできます。

インデックス作成の場合、統合ベクター化では以下が必要です。

- サポートされるデータ ソースからデータを取得するインデクサー。
- テキスト分割スキルを呼び出してデータをチャunkするスキルセットと、AzureOpenAIEmbedding スキルとデータをベクター化するためのカスタム スキルのいずれか。
- チャunkおよびベクター化した内容を受け取るための1つ以上のインデックス。

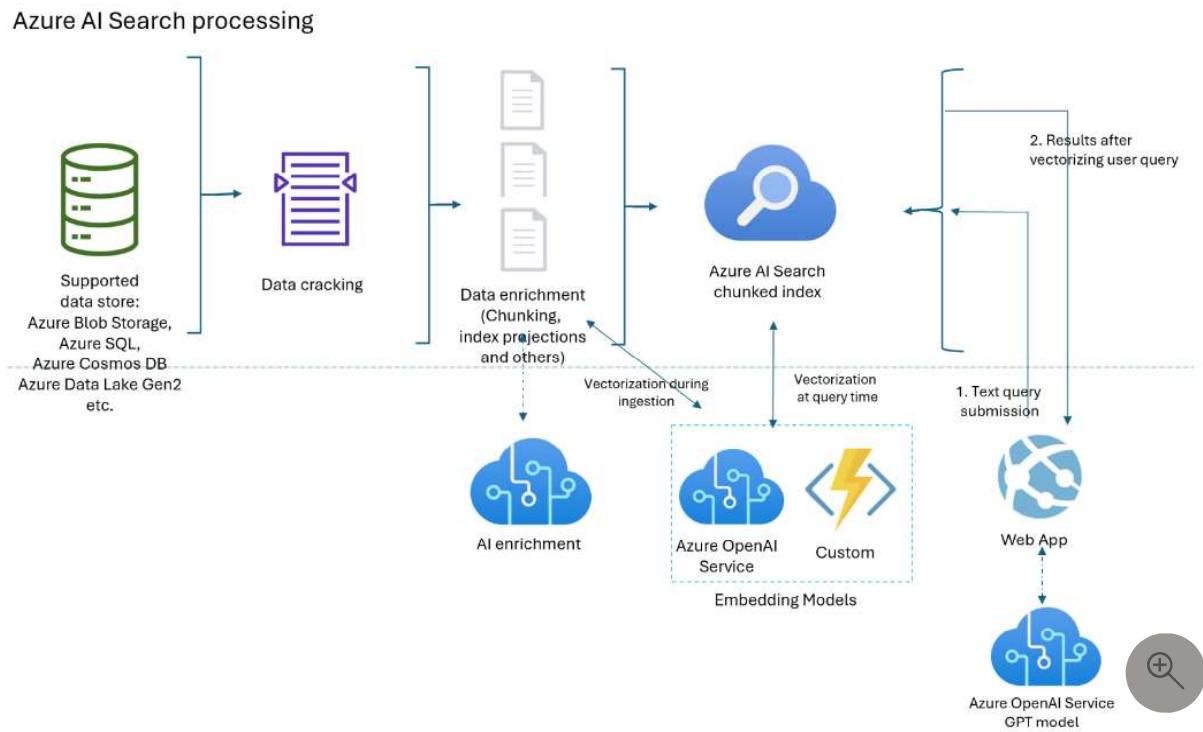
クエリの場合:

- インデックススキーマで定義され、ベクター フィールドに割り当てられて、自動的にクエリ時に使用されてテキスト クエリをベクターに変換するベクター化。

ベクター変換は、テキスト-to-ベクターの一方向です。クエリと結果にはベクター-to-テキスト変換がありません(たとえば、ベクター結果を人間が読み取り可能な文字列に変換することはできません)。

コンポーネント図

次の図は、統合ベクター化の構成要素を示しています。



こちらが統合ベクター化のための構成要素のチェックリストです。

- インデクサーベースのインデックス作成でサポートされているデータソース。
- ベクター フィールドを指定するインデックスと、ベクター フィールドに割り当てられたベクター化定義。
- データ チャンキングのためのテキスト分割スキルを提供するスキルセットと、ベクター化のスキル (AzureOpenAiEmbedding スキルと、外部埋め込みモデルをポイントするカスタム スキルのいずれか)。
- オプションとして、チャックしたデータをセカンダリ インデックスにプッシュするインデックス プロジェクション (スキルセットにも定義される)
- 埋め込みモデル (Azure OpenAI でデプロイされているか、HTTP エンドポイントを通じて提供される)。
- プロセスをエンドツーエンドで進めるためのインデクサー。インデクサーでは、変更検出のスケジュール、フィールド マッピング、優先度も指定されます。

このチェックリストは統合ベクター化に重点を置いていますが、お持ちのソリューションはこのリストに限定されません。AI エンリッチメントのためのスキルを増やし、ナレッジストアを作成し、セマンティック ランク付けを追加し、関連性チューニングや他のクエリ機能を追加することができます。

可用性と料金

統合ベクター化の可用性は、埋め込みモデルに基づきます。 Azure OpenAI を使用している場合は、「[リージョン別の提供状況](#)」を確認してください。

カスタム スキルと Azure ホスティング メカニズム (Azure 関数アプリ、Azure Web アプリ、Azure Kubernetes など) を使用している場合は、[リージョン別の製品ページ](#)で機能の可用性について確認してください。

データ チャンкиング (テキスト分割スキル) は無料で、すべての地域のすべての Azure AI サービスでご利用になれます。

① 注意

2019 年 1 月 1 日より前に作成された一部の古い検索サービスは、ベクトルワーカー ロードをサポートしないインフラストラクチャにデプロイされています。ベクトル フィールドをスキーマに追加しようとしてエラーが表示された場合、それはサービスが古いためです。このような場合は、ベクトル機能を試すために新しい検索サービスを作成する必要があります。

統合ベクター化をサポートできるのはどんなシナリオですか？

- 大きなドキュメントをチャunkに再分割すると、ベクターおよび非ベクターシナリオに便利です。ベクターの場合、埋め込みモデルの入力制約に合わせるのにチャunkが役立ちます。非ベクター シナリオの場合、チャットスタイルの検索アプリで GPT がインデックス作成したチャunkからの応答をアセンブルしています。ベクトル化 (または非ベクトル化) されたチャunkをチャットスタイルの検索に使用できます。
- フィールドのすべてがベクター フィールドであり、ドキュメント ID (検索インデックスに必要) が唯一の文字列フィールドであるベクター ストアを構築します。ベクター ストアにクエリを実行してドキュメント ID を取得し、ドキュメントのベクター フィールドを別のモデルに送信します。
- ベクターおよびテキスト フィールドを組み合わせて、セマンティック ランク付けを使用した (または使用しない) ハイブリッド検索にします。統合ベクター化によってベクター検索でサポートされるシナリオのすべてが簡略化されます。

統合ベクター化を使用するのはどのようなときか

組み込み統合ベクター化サポートの Azure AI Studio を使用することをお勧めします。この方法でお客様のニーズが満たされない場合は、Azure AI Search のプログラマティックインターフェイスを使用して統合ベクター化を呼び出すインデクサーとスキルセットを作成することができます。

統合ベクター化の使用方法

クエリ専用ベクター化の場合:

1. インデックスにベクター化を追加します。インデックスにベクターを生成するために使用したのと同じ埋め込みモデルになるはずです。
2. ベクター プロファイルにベクター化を割り当て、それからベクター プロファイルをベクター フィールドに割り当てます。
3. ベクター化するテキスト文字列を指定するベクター クエリを作成します。

より一般的なシナリオ - インデックス作成時のデータのチャンкиングとベクター化:

1. インデクサーベースのインデックス作成でサポートされているデータ ソースへのデータ ソース接続を作成します。
2. チャンкиング用のテキスト分割スキルと、[AzureOpenAIEmbeddingModel](#) またはチャンクをベクター化するカスタムスキルを呼び出すスキルセットを作成します。
3. クエリ時のベクター化を指定し、それをベクター フィールドに割り当てるインデックスを作成します。
4. データの取得からスキルセット実行まで、インデックス作成を通してすべてを進めるためのインデクサーを作成します。

オプションとして、チャンクしたコンテンツが一方のインデックス上にあり、チャンクされていないコンテンツが別のインデックスにある高度なシナリオのためのセカンダリインデックスを作成します。チャンクしたインデックス(セカンダリインデックス)は RAG アプリで役立ちます。

💡 ヒント

Azure portal で新しい [データのインポートとベクトル化] ウィザードを試して、コードを記述する前に統合ベクター化を探索します。

あるいは、同じワークフローを実行するための Jupyter ノートブックをセルごとに構成して、各手順がどう機能するかを調べます。

制限事項

Azure OpenAI の埋め込みモデルのクオータと制限について理解します。 Azure AI Search には再試行ポリシーがありますが、クオータを使い果たすと、再試行が失敗します。

Azure OpenAI の 1 分あたりトーカンの制限は、モデルごと、サブスクリプションごとに設けられています。 埋め込みモデルをクエリとインデックス作成の両ワークフローで使用している場合は、このことを覚えておいてください。 可能であれば、[ベストプラクティスに従ってください](#)。 ワークフローごとに埋め込みモデルを用意して、それらを別々のサブスクリプションでデプロイするようにしてください。

Azure AI Search では、[サービスの制限](#)がレベルおよびワークフロー別にあることを忘れないでください。

最後に、次の機能は現在サポートされていません。

- [カスタマー マネージド暗号化キー](#)
- ベクター化への[共有プライベート リンク接続](#)
- 現在は、統合型データ チャンキングおよびベクター化のためのバッチ処理がありません

統合ベクター化のメリット

統合ベクター化の重要メリットのいくつかを紹介します。

- データ チャンキングとベクター化の分離したパイプラインがありません。 コードの書き込みと維持がより簡単です。
- エンドツーエンドのインデックス作成を自動化します。 ソース (Azure Storage、 Azure SQL、 Cosmos DB など) でデータが変更されると、インデクサーはこれらの更新を、パイプライン全体 (取得からドキュメントの解読まで) で、オプションの AI エンリッチメント、データ チャンキング、ベクター化、インデックス作成を通じて進めることができます。
- チャンクしたコンテンツをセカンダリ インデックスに射影します。 セカンダリ インデックスは他の検索インデックス (フィールドや他のコンストラクトを持つスキーマ) のように作成されますが、インデクサーによりプライマリ インデックスと並行して作成されます。 各ソース ドキュメントのコンテンツが、同じインデックス作成実行中に、プライマリおよびセカンダリ インデックスのフィールドへ流れていきます。

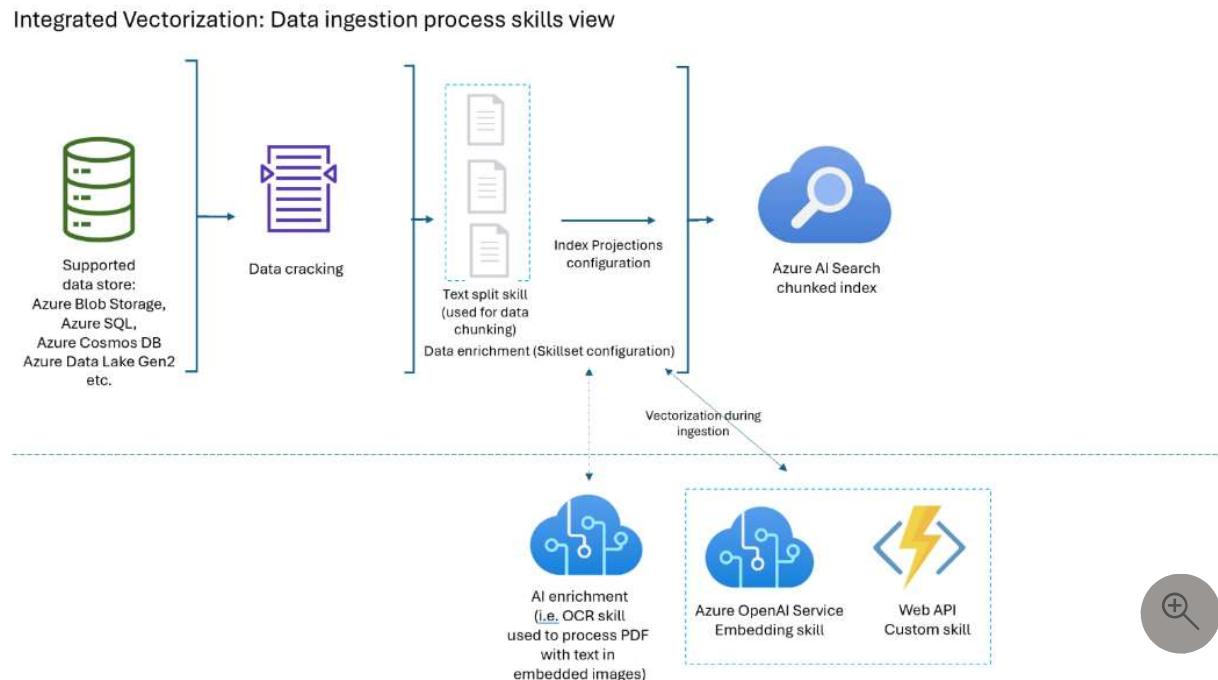
セカンダリ インデックスの目的は、データ チャンキングおよび取得拡張生成 (RAG) アプリです。 サイズの大きな PDF をソース ドキュメントとして想定すると、プライマリ インデックスには基本情報 (タイトル、日付、作成者、説明) があ

り、セカンダリ インデックスにはコンテンツのチャンクがあります。データ チャンク レベルのベクター化によって、関連する情報を見つけて (各チャンクが検索可能である) 関連する応答を返すのが、特にチャットスタイルの検索アプリでは簡単になります。

チャンク後のインデックス

チャンキングとは、コンテンツをより小さな管理可能部分 (チャンク) に分割することで、それらを別々に処理できるようにするプロセスです。チャンキングが必要になるのは最大入力サイズの埋め込みモデルや大型言語モデルでソース ドキュメントが大きすぎるけれども、それによって [RAG パターン](#) やチャットスタイル検索でインデックス構造がよくなると考えられる場合です。

次の図は、チャンク後インデックス作成の構成要素を示しています。



次のステップ[°]

- 検索インデックスにベクター化を構成する
- スキルセットにインデックス投影を構成する