

# Azure AI Search の検索インデックス

[アーティクル] • 2024/02/16

Azure AI Search の "検索インデックス" は検索可能なコンテンツであり、検索エンジンでインデックス作成、全文検索、ベクトル検索、ハイブリッド検索、フィルターされたクエリに使用できます。インデックスは、スキーマによって定義され、検索サービスに保存されます。2番目のステップとしてデータのインポートが続けます。このコンテンツは検索サービス内に存在します。これは、最新の検索アプリケーションで想定されるミリ秒単位の応答時間に必要な、プライマリデータストアとは別のものです。インデクサー主導のインデックス作成シナリオを除き、検索サービスがソースデータに接続したり、クエリを実行したりすることはありません。

検索インデックスを作成して管理する場合、この記事は次の点を理解するのに役立ちます。

- コンテンツ (ドキュメントおよびスキーマ)
- 物理データ構造
- 基本操作

今すぐに使いたいですか? 代わりに、[検索インデックスの作成](#)に関する記事を参照してください。

## 検索インデックスのスキーマ

Azure AI Search のインデックスには検索ドキュメントが格納されます。概念的に、ドキュメントはインデックス内で検索可能なデータの1つの単位です。たとえば、小売業者に製品ごとのドキュメントがあり、ニュース組織に記事ごとのドキュメントがある場合、旅行サイトにはホテルと目的地ごとのドキュメントがある場合があります。これらの概念をなじみのあるデータベースの同等のものに対応させるなら、検索インデックスはテーブルと同じで、ドキュメントはテーブルにおける行とほぼ同じです。

次の例に示すように、ドキュメントの構造は "インデックス スキーマ" によって決まります。"フィールド" コレクションは通常、インデックスの最大の部分であり、各フィールドには、名前、[データ型](#)の割り当て、および使用方法を決定する許容される動作を示す属性が設定されます。

JSON

```
{  
  "name": "name_of_index, unique across the service",  
  "fields": [  
    {
```

```

        "name": "name_of_field",
        "type": "Edm.String | Collection(Edm.String) | Collection(Edm.Single)
| Edm.Int32 | Edm.Int64 | Edm.Double | Edm.Boolean | Edm.DateTimeOffset |
Edm.GeographyPoint",
        "searchable": true (default where applicable) | false (only Edm.String
and Collection(Edm.String) fields can be searchable),
        "filterable": true (default) | false,
        "sortable": true (default where applicable) | false
(Collection(Edm.String) fields cannot be sortable),
        "facetable": true (default where applicable) | false
(Edm.GeographyPoint fields cannot be facetable),
        "key": true | false (default, only Edm.String fields can be keys),
        "retrievable": true (default) | false,
        "analyzer": "name_of_analyzer_for_search_and_indexing", (only if
'searchAnalyzer' and 'indexAnalyzer' are not set)
        "searchAnalyzer": "name_of_search_analyzer", (only if 'indexAnalyzer'
is set and 'analyzer' is not set)
        "indexAnalyzer": "name_of_indexing_analyzer", (only if
'searchAnalyzer' is set and 'analyzer' is not set)
        "normalizer": "name_of_normalizer", (applies to fields that are
filterable)
        "synonymMaps": "name_of_synonym_map", (optional, only one synonym map
per field is currently supported)
        "dimensions": "number of dimensions used by an embedding models",
(applies to vector fields only, of type Collection(Edm.Single))
        "vectorSearchProfile": "name_of_vector_profile" (indexes can have many
configurations, a field can use just one)
    }
],
"suggesters": [ ],
"scoringProfiles": [ ],
"analyzers":(optional)[ ... ],
"charFilters":(optional)[ ... ],
"tokenizers":(optional)[ ... ],
"tokenFilters":(optional)[ ... ],
"defaultScoringProfile": (optional) "...",
"corsOptions": (optional) { },
"encryptionKey":(optional){ },
"semantic":(optional){ },
"vectorSearch":(optional){ }
}

```

簡潔にするために他の要素は折りたたまれていますが、次のリンク先で詳細を確認できます。

- `suggesters` は、オートコンプリートのような先行入力クエリをサポートします。
- `scoringProfiles` は関連性のチューニングに使われます。
- `analyzers` は、アナライザーがサポートする言語規則やその他の特性に従って文字列をトークンに処理するために使われます。
- `corsOptions`、つまりクロスオリジンリモートスクリプト (CORS) は、さまざまなものから要求を発行するアプリに使われます。

- `encryptionKey` は、インデックス内の機密コンテンツの二重暗号化を構成します。
- `semantic` は、全文検索とハイブリッド検索でのセマンティック再ランク付けを構成します。
- `vectorSearch` は、ベクトルフィールドとクエリを構成します。

## フィールド定義

検索ドキュメントは、[インデックス要求の作成](#)に関する記事の本文の "フィールド" コレクションによって定義されます。ドキュメントの識別のためのフィールド(キー)、検索可能なテキストの格納、フィルター、ファセット、並べ替えをサポートするためのフィールドが必要になります。ユーザーに表示しないデータのフィールドが必要になる場合もあります。たとえば、検索スコアを上げるためにスコアリングプロファイルで使用できる、利益率やマーケティングプロモーションのフィールドが必要になることがあります。

受信データが階層化された性質を持つ場合は、入れ子構造に使われる複合型として、インデックス内でそれを表すことができます。あらかじめ登録されているサンプルデータセットである Hotels は、各ホテルとの一対一のリレーションシップを持つ Address(複数のサブフィールドを含む) と、各ホテルに複数の部屋が関連付けられている複合型コレクションの Rooms を使用した複合型を示しています。

## フィールド属性

フィールド属性は、フィールドがどのように使用されるか(フルテキスト検索、ファセットナビゲーション、並べ替えなどの操作で使用されるかどうか)を決定します。

文字列フィールドは多くの場合、"検索可能" および "取得可能" としてマークされます。検索結果を絞り込むために使用されるフィールドには、"並べ替え可能"、"フィルター可能"、および "ファセット可能" が含まれます。

 テーブルを展開する

属性	説明
"検索可能"	全文またはベクトル検索可能。テキストフィールドは、インデックス作成時に単語分割などの字句解析の対象になります。検索可能フィールドを "sunny day" などの値に設定した場合、その値は内部的に個別のトークン "sunny" と "day" に分割されます。 詳細については、「 <a href="#">フルテキスト検索のしくみ</a> 」を参照してください。
"フィルター可能"	\$filter クエリで参照されます。型 <code>Edm.String</code> または <code>Collection(Edm.String)</code> のフィルター可能フィールドは単語分割されないため、比較は完全に一致するかどうかだけになります。たとえば、このようなフィールドを "sunny day" に設定した場合、 <code>\$filter=f</code>

<b>属性</b>	<code>eq 'sunny'</code> では一致が見つかりませんが、 <code>\$filter=f eq 'sunny day'</code> では見つかります。
"並べ替え可能"	既定では、システムは結果をスコアで並べ替えますが、ドキュメント内のフィールドに基づいて並べ替えを構成できます。型 <code>Collection(Edm.String)</code> のフィールドを "並べ替え可能" にすることはできません。
"フル可"能"	通常、カテゴリ (たとえば、特定の市にあるホテル) ごとのヒットカウントを含む検索結果のプレゼンテーションで使用されます。このオプションは、型 <code>Edm.GeographyPoint</code> のフィールドでは使用できません。"フィルター可能"、"並べ替え可能"、または "フル可" 能" である型 <code>Edm.String</code> のフィールドの長さは、最大 32 キロバイトです。詳細については、「 <a href="#">Create Index (REST API) (インデックスの作成 (REST API))</a> 」を参照してください。
"キー"	インデックス内のドキュメントの一意識別子。キー フィールドとして正確に 1 つのフィールドを選択する必要があります、それは型 <code>Edm.String</code> である必要があります。
"取得可能"	検索結果でこのフィールドを返すことができるかどうかを決定します。これは、あるフィールド ("利幅" など) をフィルター、並べ替え、またはスコア付けのメカニズムとして使用するが、このフィールドをエンドユーザーには表示したくない場合に役立ちます。 <code>true for key</code> である必要があります。

いつでも新しいフィールドを追加できますが、既存のフィールド定義はインデックスの有効期間の間ロックされます。このため、開発者は通常、単純なインデックスを作成したり、アイデアをテストしたり、ポータルページを使用して設定を検索したりするためのポータルを使用します。インデックスを容易に再構築できるようにコードベースのアプローチに従う場合は、インデックス設計を頻繁に反復する方がより効率的です。

### ① 注意

インデックスの作成に使用する API には、さまざまな既定の動作があります。 REST API の場合、ほとんどの属性は既定で有効であり (たとえば、文字列フィールドの "searchable" および "retrievable" は `true` です)、無効にする場合は、単にそれらを設定するだけです。.NET SDK の場合は、逆のことが言えます。明示的に設定していないプロパティの場合、既定では、特に有効にしない限り、対応する検索動作は無効にされています。

## 物理的な構造とサイズ

Azure AI Search におけるインデックスの物理的な構造は、主に内部実装です。そのスキーマにアクセスし、そのコンテンツにクエリを実行し、そのサイズを監視し、容量を

管理することができますが、クラスター自体(インデックス、[シャード](#)、その他のファイルとフォルダー)は、Microsoft が内部で管理します。

インデックス サイズを監視するには、Azure portal の [インデックス] タブを使用するか、検索サービスに対して [GET INDEX 要求](#)を発行します。 [サービス統計情報要求](#)を発行し、ストレージ サイズの値を確認することもできます。

インデックスのサイズは、次の条件によって決まります。

- ドキュメントの数量と構成
- 個々のフィールドの属性
- インデックスの構成(具体的には、`suggester` を含めるかどうか)

ドキュメントの構成と数量は、インポートに選択した内容によって決まります。検索インデックスには検索可能なコンテンツのみを含める必要があります。ソースデータにバイナリ フィールドが含まれている場合は、AI エンリッチメントを使用してコンテンツを解読して分析し、テキスト検索可能な情報を作成する場合を除き、それらのフィールドは除外してください。

フィールドの属性によって動作が決まります。これらの動作をサポートするために、インデックス作成プロセスによって、必要なデータ構造が作成されます。たとえば、`Edm.String` 型のフィールドの場合、"検索可能" によって、トークン化された用語の転置インデックスをスキャンする[全文検索](#)が呼び出されます。これに対して、"フィルター可能" または "並べ替え可能" の属性では、未変更の文字列に対する反復処理がサポートされます。次のセクションの例では、選択した属性に基づくインデックス サイズのバリエーションを示しています。

[suggester](#) は、先行入力またはオートコンプリートのクエリをサポートするコンストラクトです。そのため、`suggester` を含めると、インデックス作成プロセスによって、逐語的な文字の照合に必要なデータ構造が作成されます。`suggester` はフィールド レベルで実装されるため、先行入力にふさわしいフィールドのみを選択してください。

## 属性と `suggester` がストレージに与える影響を示す例

次のスクリーンショットは、属性のさまざまな組み合わせの結果であるインデックス格納パターンを示しています。このインデックスは[不動産サンプル インデックス](#)に基づいています。これは、データのインポート ウィザードと組み込みのサンプル データを使用して簡単に作成できます。インデックスのスキーマは表示されませんが、インデックス名に基づいて属性を推測できます。たとえば、`realestate-searchable` インデックスでは "searchable" 属性が選択されていて他には何もなく、`realestate-retrievable` インデックスでは "retrievable" 属性が選択されていて他には何もなく、以下同様です。

NAME	DOCUMENT COUNT	STORAGE SIZE
realestate-all-attributes-no-suggester	4,959	26.55 MiB
realestate-all-attributes-plus-suggester	4,959	49.4 MiB
realestate-filterable-facetable-sortable	4,959	20.89 MiB
realestate-no-attributes	4,959	4.99 MiB
realestate-retrievable	4,959	5.04 MiB
realestate-searchable	4,959	9.95 MiB

これらのインデックスのバリエーションはやや人為的なものですが、属性がストレージに与える影響の広範な比較のために参照できます。

- "取得可能" は、インデックスのサイズに影響しません。
- "フィルター可能"、"並べ替え可能"、"ファセット可能" は、より多くのストレージを消費します。
- `suggester` では、インデックス サイズが大きくなる可能性が大きいにありますが、スクリーンショットで示すほどではありません。`(suggester` 対応になる可能性のあるすべてのフィールドが選択されていますが、ほとんどのインデックスではこのようなシナリオになる可能性はありません)。

また、上記の表に反映されていない事柄に、[アナライザー](#)の影響があります。  
edgeNgram トークナイザーを使って逐語的な文字シーケンス (`a, ab, abc, abcd`) を格納した場合、インデックスは、標準アナライザーを使用した場合よりも大きくなります。

## 基本的な操作と相互作用

インデックスの概要を理解したので、このセクションでは、1つのインデックスに接続してセキュリティを保護するなどの、インデックスの実行時操作について説明します。

### ① 注意

インデックスを管理する際、インデックスの移動やコピーに関して、ポータルや API のサポートはないことに注意してください。代わりに、ユーザーは通常、アプリケーション デプロイソリューションを別の検索サービスでポイントする(同じインデックス名を使用している場合)か、名前を変更して現在の検索サービスにコピーを作成してからビルドします。

## インデックスの分離性

Azure AI Search で操作の対象となるインデックスは一度に 1 つです。インデックスに関連したすべての操作は、単一のインデックスが対象となります。関連するインデックスや、インデックス作成またはクエリのための独立したインデックスの結合の概念はありません。

## 継続的に使用可能

インデックスは、最初のドキュメントのインデックスが作成されるとすぐにクエリで使用できますが、すべてのドキュメントのインデックスが作成されるまでは完全には機能しません。内部的には、検索インデックスは [パーティション間で分散され、レプリカ上で実行されます](#)。物理インデックスは内部で管理されます。論理インデックスはユーザーが管理します。

インデックスは継続的に使用可能であり、一時停止したり、オフラインにしたりすることはできません。継続的な操作のために設計されているので、コンテンツの更新やインデックス自体への追加はリアルタイムで行われます。その結果、要求がドキュメントの更新と一致する場合、クエリは一時的に不完全な結果を返す可能性があります。

ドキュメント操作 (更新または削除) や、現在のインデックスの既存の構造と整合性に影響しない変更 (新しいフィールドの追加など) に対しては、クエリの継続性が存在します。構造上の更新 (既存のフィールドの変更) を行う必要がある場合は、通常、開発環境での削除と再構築のワークフローを使用して、または運用サービスでインデックスの新しいバージョンを作成することによってそれらが管理されます。

[インデックスの再構築](#)を避けるため、小規模な変更を行っている一部のお客様は、以前のバージョンと共に新しいものを作成することによって、フィールドの "バージョン管理" を選択しています。これは、時間の経過と共に、特にレプリケートに負荷のかかる運用環境のインデックスで、古いフィールドまたは古いカスタム アナライザ一定義の形式の、孤立したコンテンツになります。インデックス ライフサイクル管理の一部として、インデックスの計画更新に関する問題に対処することができます。

## エンドポイントの接続とセキュリティ

すべてのインデックス作成とクエリの要求は、インデックスを対象とします。エンドポイントは、通常、次のいずれかになります。

[+] [テーブルを展開する](#)

エンドポイント	接続とアクセスの制御
<code>&lt;your-service&gt;.search.windows.net/indexes</code>	インデックスのコレクションを対象とします。インデックスを作成、一覧表示、または削除するときに使用します。これらの操作には管理者権限が必要です。管理者 API キーまたは Search 共同作成者ロールを通じて使用できます。
<code>&lt;your-service&gt;.search.windows.net/indexes/&lt;your-index&gt;/docs</code>	1 つのインデックスのドキュメントコレクションを対象とします。インデックスまたはデータ更新に対してクエリを実行するときに使用します。クエリには、読み取り権限で十分であり、クエリ API キーまたはデータ閲覧者ロールを通じて使用できます。データ更新の場合は、管理者権限が必要です。

検索サブスクライバー (つまり検索サービスの作成者) は、Azure portal で検索サービスを管理できます。サービスを作成または削除するには、Azure サブスクリプションに共同作成者以上のアクセス許可が必要です。検索サービスに直接接続するには、[Azure portal にサインイン](#) します。

他のクライアントの場合は、接続手順のクイックスタートを確認することをお勧めします。

- [クイックスタート: REST](#)
- [クイックスタート: Azure SDK](#)

## 次のステップ

Azure AI Search のほぼすべてのサンプルまたはチュートリアルを使用して、インデックスを作成する実践的な体験ができます。まず、目次から任意のクイックスタートを選択できます。

ただし、データを使用してインデックスを読み込む方法についても理解しておく必要があります。インデックスの定義とデータのインポートの方法は、連携して定義されます。次の記事では、インデックスの作成および読み込みの詳細について説明します。

- [検索インデックスの作成](#)
- [ベクトルストアを作成する](#)
- [インデックスの別名を作成する](#)
- [データインポートの概要](#)

- インデックスを読み込む

# Azure AI Search でのベクター ストレージ

[アーティクル] • 2024/03/04

Azure AI 検索には、[ベクトル検索](#)と[ハイブリッド検索](#)用のベクトルストレージと構成が用意されています。サポートはフィールド レベルで実装されます。つまり、同じ検索コーコードでベクター フィールドと非ベクター フィールドを組み合わせることができます。

ベクターは検索インデックスに格納されます。[Create Index REST API](#) または同等の Azure SDK メソッドを使用して、[ベクター ストアを作成します。](#)

ベクター ストレージの主な考慮事項は以下のとおりです。

- 目的とするベクトル取得パターンに基づいて、ユース ケースに合うスキーマを設計します。
- インデックス サイズを見積もり、検索サービスの容量を確認します。
- ベクトルストアを管理する
- ベクトルストアをセキュリティで保護する

## ベクター取得パターン

Azure AI Search には、検索結果を操作するための 2 つのパターンがあります。

- 生成検索。言語モデルは、Azure AI Search のデータを使用して、ユーザーのクエリに対する応答を作成します。このパターンには、プロンプトを調整し、コンテキストを維持するためのオーケストレーション レイヤーが含まれます。このパターンでは、検索結果はプロンプト フローにフィードされ、GPT や Text-Davinci などのチャット モデルが受け取ります。このアプローチは、検索インデックスによってグラウンド データが提供される[取得拡張生成 \(RAG\)](#) アーキテクチャに基づいています。
- 検索バー、クエリ入力文字列、レンダリングされた結果を使用した従来の検索。検索エンジンによって、ベクトル クエリが受け入れられて実行され、応答が作成されます。ユーザーはそれらの結果をクライアント アプリにレンダリングします。Azure AI Search では、結果はフラット化された行セットで返され、検索結果を含めるフィールドを選ぶことができます。チャット モデルがないため、応答で人間が判読できる非ベクトル コンテンツをベクトルストア (検索インデックス) に設定することが期待されます。検索エンジンはベクトルに一致しますが、非ベクトル値を使用して検索結果を設定する必要があります。[ベクトル クエリとハイ](#)

**ブリッド クエリ**は、従来の検索シナリオ用に作成できるクエリ要求の型に対応します。

インデックス スキーマには、主なユース ケースが反映されている必要があります。次のセクションでは、生成 AI と従来の検索用に構築されるソリューションのフィールド構成の違いに着目します。

## ベクトルストアのスキーマ

ベクトルストアのインデックス スキーマには、名前、キー フィールド (文字列)、1つ以上のベクトル フィールド、およびベクトル構成が必要です。非ベクター フィールドは、ハイブリッド クエリ、または言語モデルを通過する必要のない、人間が読み取り可能な逐語的なコンテンツを返す場合に推奨されます。ベクター構成の手順については、「[ベクターストアを作成する](#)」を参照してください。

### 基本的なベクター フィールドの構成

ベクトル フィールドは、データ型とベクトル固有のプロパティによって区別されます。フィールド コレクション内のベクトル フィールドの外観を以下に示します。

```
JSON
{
    "name": "content_vector",
    "type": "Collection(Edm.Single)",
    "searchable": true,
    "retrievable": true,
    "dimensions": 1536,
    "vectorSearchProfile": "my-vector-profile"
}
```

ベクトル フィールドの型は `Collection(Edm.Single)` です。

ベクトル フィールドは検索可能で取得可能である必要がありますが、フィルター可能、ファセット可能、並べ替え可能にすることはできません。また、アナライザー、ノーマライザー、シノニム マップの割り当てを持つことはできません。

ベクトル フィールドでは、埋め込みモデルによって生成される埋め込みの数に `dimensions` を設定する必要があります。たとえば、text-embedding-ada-002 では、テキストのチャunkごとに 1,536 個の埋め込みが生成されます。

ベクトル フィールドには、"ベクトル検索プロファイル" によって示されるアルゴリズムを使用してインデックスが作成されます。このプロファイルはインデックス内の他の

場所で定義されているため、例では示されていません。 詳細については、ベクトル検索の構成に関するページを参照してください。

## 基本的なベクトル ワークロードのフィールド コレクション

ベクトルストアでは、ベクトルフィールド以外にもさらにフィールドが必要です。 たとえば、キー フィールド (この例では `"id"`) はインデックス要件です。

JSON

```
"name": "example-basic-vector-idx",
"fields": [
  { "name": "id", "type": "Edm.String", "searchable": false, "filterable": true, "retrievable": true, "key": true },
  { "name": "content_vector", "type": "Collection(Edm.Single)", "searchable": true, "retrievable": true, "dimensions": 1536, "vectorSearchProfile": null },
  { "name": "content", "type": "Edm.String", "searchable": true, "retrievable": true, "analyzer": null },
  { "name": "metadata", "type": "Edm.String", "searchable": true, "filterable": true, "retrievable": true, "sortable": true, "facetable": true }
]
```

`"content"` フィールドなどの他のフィールドでは、人間が判読できる `"content_vector"` フィールドと同等のものが提供されます。 応答の作成専用の言語モデルを使用している場合は、非ベクトルコンテンツ フィールドを省略できますが、クライアントアプリに検索結果を直接プッシュするソリューションには非ベクトルコンテンツが必要です。

メタデータ フィールドは、特にメタデータにソース ドキュメントに関する配信元情報が含まれている場合に、フィルターに役立ちます。 ベクトルフィールド上で直接フィルター処理を行うことはできませんが、ベクトルクエリの実行前または実行後にフィルター処理を行うプリフィルター モードまたはポストフィルター モードを設定することができます。

## データのインポートとベクター化ウィザードによって生成されるスキーマ

評価と概念実証のテストには、[データのインポートとベクター化ウィザード](#)をお勧めします。 ウィザードによって、このセクションのスキーマ例が生成されます。

このスキーマの偏りは、検索ドキュメントがデータ チャンクを中心に構築されていることです。RAG アプリで一般的なように、言語モデルが応答を作成する場合は、データ チャンクを中心に設計されたスキーマが必要です。

データ チャンクは、言語モデルの入力制限内を維持するために必要ですが、複数の親 ドキュメントからブルされたコンテンツの小さなチャンクに対してクエリを照合できる場合の類似性検索の精度も向上します。最後に、セマンティック ランク付けを使用している場合、セマンティック ランカーにはトークン制限もあります。これは、データ チャンクがアプローチの一部である場合に、より簡単に満たされます。

次の例では、検索ドキュメントごとに 1 つのチャンク ID、チャンク、タイトル、ベクター フィールドがあります。blob メタデータ (パス) の base 64 エンコードを使用して、chunkID と親 ID がウィザードによって設定されます。チャンクとタイトルは、BLOB コンテンツと BLOB 名から派生します。ベクター フィールドのみが完全に生成されます。これは、ベクトル化されたバージョンのチャンク フィールドです。埋め込みは、指定した Azure OpenAI 埋め込みモデルを呼び出すことによって生成されます。

JSON

```
"name": "example-index-from-import-wizard",
"fields": [
    { "name": "chunk_id", "type": "Edm.String", "key": true, "searchable": true, "filterable": true, "retrievable": true, "sortable": true, "facetable": true, "analyzer": "keyword"}, 
    { "name": "parent_id", "type": "Edm.String", "searchable": true, "filterable": true, "retrievable": true, "sortable": true}, 
    { "name": "chunk", "type": "Edm.String", "searchable": true, "filterable": false, "retrievable": true, "sortable": false}, 
    { "name": "title", "type": "Edm.String", "searchable": true, "filterable": true, "retrievable": true, "sortable": false}, 
    { "name": "vector", "type": "Collection(Edm.Single)", "searchable": true, "retrievable": true, "dimensions": 1536, "vectorSearchProfile": "vector-1707768500058-profile"}]
```

## RAG アプリとチャットスタイル アプリのスキーマ

生成検索用のストレージを設計する場合は、インデックスを作成してベクトル化した静的コンテンツに対して個別のインデックスを作成し、プロンプト フローで使用できる会話用に 2 つ目のインデックスを作成できます。以下のインデックスは、[chat-with-your-data-solution-accelerator](#) アクセラレータから作成されます。

Home > contosochat-search

contosochat-search | Indexes ☆ ...

Search service

Search management

Indexes

Indexers

Data sources

Aliases

Search

Add index

Refresh

Delete

Filter by name...

Name	Document Count	Storage Size
conversations	14	434.61 KB
chat-index	191	5.42 MB

生成検索エクスペリエンスをサポートするチャットインデックスのフィールド:

JSON

```
"name": "example-index-from-accelerator",
"fields": [
    { "name": "id", "type": "Edm.String", "searchable": false, "filterable": true, "retrievable": true },
    { "name": "content", "type": "Edm.String", "searchable": true, "filterable": false, "retrievable": true },
    { "name": "content_vector", "type": "Collection(Edm.Single)", "searchable": true, "retrievable": true, "dimensions": 1536, "vectorSearchProfile": "my-vector-profile" },
    { "name": "metadata", "type": "Edm.String", "searchable": true, "filterable": false, "retrievable": true },
    { "name": "title", "type": "Edm.String", "searchable": true, "filterable": true, "retrievable": true, "facetable": true },
    { "name": "source", "type": "Edm.String", "searchable": true, "filterable": true, "retrievable": true },
    { "name": "chunk", "type": "Edm.Int32", "searchable": false, "filterable": true, "retrievable": true },
    { "name": "offset", "type": "Edm.Int32", "searchable": false, "filterable": true, "retrievable": true }
]
```

オーケストレーションとチャット履歴をサポートする会話インデックスのフィールド:

JSON

```
"fields": [
    { "name": "id", "type": "Edm.String", "key": true, "searchable": false, "filterable": true, "retrievable": true, "sortable": false, "facetable": false },
    { "name": "conversation_id", "type": "Edm.String", "searchable": false, "filterable": true, "retrievable": true, "sortable": false, "facetable": true },
    { "name": "content", "type": "Edm.String", "searchable": true, "filterable": false, "retrievable": true },
    { "name": "content_vector", "type": "Collection(Edm.Single)", "searchable": true, "retrievable": true, "dimensions": 1536,
```

```

"vectorSearchProfile": "default-profile" },
  { "name": "metadata", "type": "Edm.String", "searchable": true,
"filterable": false, "retrievable": true },
    { "name": "type", "type": "Edm.String", "searchable": false,
"filterable": true, "retrievable": true, "sortable": false, "facetable":
true },
      { "name": "user_id", "type": "Edm.String", "searchable": false,
"filterable": true, "retrievable": true, "sortable": false, "facetable":
true },
        { "name": "sources", "type": "Collection(Edm.String)", "searchable":
false, "filterable": true, "retrievable": true, "sortable": false,
"facetable": true },
          { "name": "created_at", "type": "Edm.DateTimeOffset", "searchable":
false, "filterable": true, "retrievable": true },
            { "name": "updated_at", "type": "Edm.DateTimeOffset", "searchable":
false, "filterable": true, "retrievable": true }
]

```

次に示すのは、[Search Explorer](#) における会話インデックスの検索結果のスクリーンショットです。検索に条件がないため、検索スコアは 1.00 となっています。オーケストレーションとプロンプト フローをサポートするために存在するフィールドに注目してください。会話 ID は、特定のチャットを特定します。 "type" は、コンテンツがユーザーとアシスタントのどちらからのものであるかを示します。日付は、古さによって履歴からチャットを削除するために使用されます。



Results	
18	{ "@search.score": 1, "id": "ND1mODY4NjgtOWU2ZC00YTY3LTgwNTITNmI20WUyYzllZjRj", "conversation_id": "01db26eb-f781-462b-8da3-0ec10e551a35", "content": "The Gulf Stream carries a lot of heat from the Equator toward the far North Atlantic, n "metadata": "{\"conversation_id\": \"01db26eb-f781-462b-8da3-0ec10e551a35\", \"sources\": [\"doc_26", "type": "assistant", "user_id": null, "sources": [ "doc_2005da260b463009d5e230b09a55acedf51fbcd7", "doc_541c34b9a54b97ac888034a21c73fc6910243741" ], "created_at": "2024-01-15T05:19:52Z", "updated_at": "2024-01-15T05:19:52Z" }, { "@search.score": 1, "id": "NDM1NjM0MzIzGIMzI00ZmQ4LTk3YTgtY2MyMGU3YmRhNTVm", "conversation_id": "01db26eb-f781-462b-8da3-0ec10e551a35", "content": "what can you tell me about winds", "metadata": "{\"type\": \"user\", \"conversation_id\": \"01db26eb-f781-462b-8da3-0ec10e551a35\", \"", "type": "user", "user_id": null, "sources": [], "created_at": "2024-01-15T19:51:12Z", "updated_at": "2024-01-15T19:51:12Z"

## 物理的な構造とサイズ

Azure AI Search におけるインデックスの物理的な構造は、主に内部実装です。スキーマへのアクセス、コンテンツの読み込みとクエリ実行、サイズの監視、容量の管理を行うことはできますが、クラスター自体 (逆インデックスとベクトルインデックス、シャ

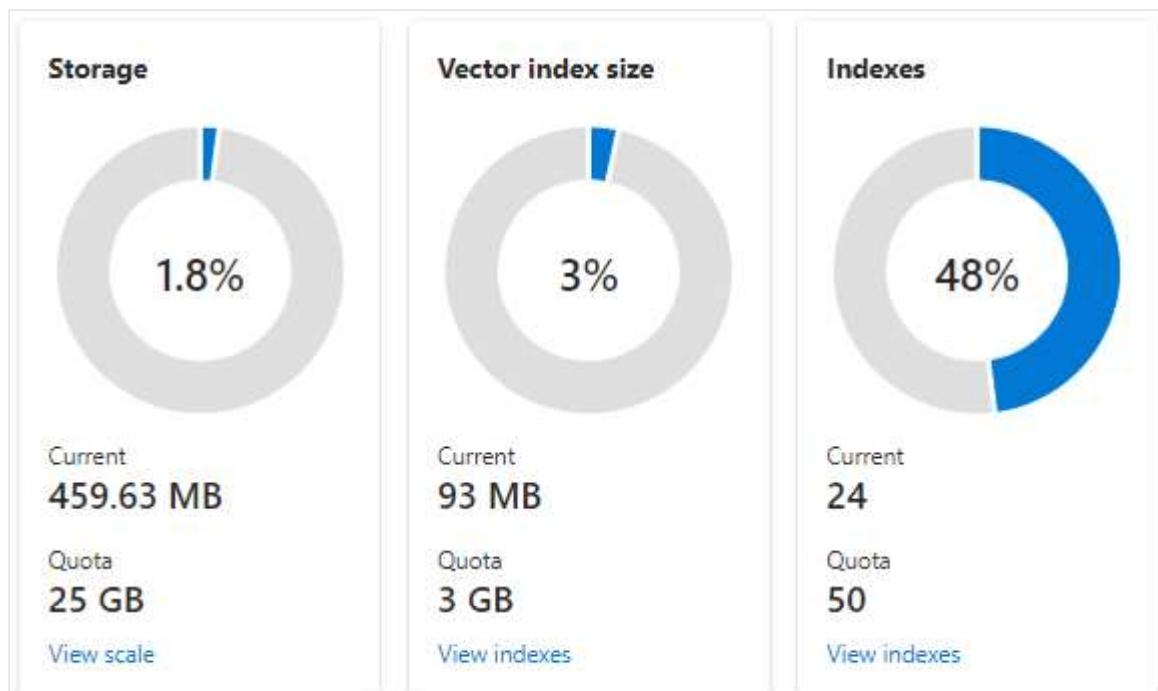
ード、その他のファイルとフォルダー) は、Microsoft によって内部的に管理されます。

インデックスのサイズと内容は、以下によって決まります。

- ドキュメントの数量と構成
- 個々のフィールドの属性。たとえば、フィルター処理可能なフィールドにはより多くのストレージが必要です。
- 類似性検索に HNSW と網羅的 KNN のどちらを選択するかに基づくインデックス構成 (内部的なナビゲーション構造がどのように作成されるかを指定するベクトル構成を含む)。

Azure AI 検索はベクトルストレージに制限を課しており、これは、すべてのワークコードにとってバランスのとれた安定したシステムを維持するのに役立ちます。利用者が制限の範囲内に留まるのを手助けするために、ベクトルの使用状況は Azure portalにおいてと、サービスとインデックスの統計情報を使用したプログラム的な方法とで個別に追跡され報告されます。

次に示すのは、1つのパーティションと1つのレプリカで構成された S1 サービスのスクリーンショットです。この特定のサービスには、平均して1つのベクトルフィールドを含む24個の小さなインデックスがあり、各フィールドは1536個の埋め込みで構成されています。2番目のタイルが示しているのは、ベクトルインデックスのクオータと使用状況です。ベクトルインデックスは、各ベクトルフィールドに対して作成された内部的なデータ構造です。そのため、ベクトルインデックスのストレージは常に、インデックス全体によって使用されるストレージの一部となります。残りの部分は、その他の非ベクトルフィールドとデータ構造によって使用されます。



ベクトルインデックスの制限と見積もりについては[別の記事](#)でカバーされていますが、前もって強調しておくべき 2 つの点として、ストレージの最大値はサービス レベルによって異なるということと、検索サービスがいつ作成されたかによっても異なることがあります。同じレベルでも新しいサービスの方が、かなり多いベクトルインデックスの容量を持ちます。このため、以下のアクションを行ってください。

- [検索サービスのデプロイ日を確認する](#)。2023 年 7 月 1 日より前に作成されている場合は、容量を増やすために新しい検索サービスを作成することを検討してください。
- ベクトルストレージ要件の変動が予想される場合、[スケーラブルなレベルを選択する](#)。Basic レベルは、1 つのパーティションに固定されています。柔軟性を高め、パフォーマンスを向上させるには、Standard 1 (S1) 以上を検討してください。

## 基本的な操作と相互作用

このセクションでは、1 つのインデックスへの接続やセキュリティ保護など、ベクトルの実行時操作について紹介します。

### ① 注意

インデックスを管理する際、インデックスの移動やコピーに関して、ポータルや API のサポートはないことに注意してください。代わりに、ユーザーは通常、アプリケーション デプロイ ソリューションを別の検索サービスでポイントする(同じインデックス名を使用している場合)か、名前を変更して現在の検索サービスにコピーを作成してからビルドします。

## 継続的に使用可能

インデックスは、最初のドキュメントのインデックスが作成されるとすぐにクエリで使用できますが、すべてのドキュメントのインデックスが作成されるまでは完全には機能しません。内部的には、インデックスは[複数のパーティションにわたって分散され、レプリカ上で実行されます](#)。物理インデックスは内部で管理されます。論理インデックスはユーザーが管理します。

インデックスは継続的に使用可能であり、一時停止したり、オフラインにしたりすることはできません。継続的な操作のために設計されているので、コンテンツの更新やインデックス自体への追加はリアルタイムで行われます。その結果、要求がドキュメントの更新と一致する場合、クエリは一時的に不完全な結果を返す可能性があります。

ドキュメント操作 (更新または削除) や、現在のインデックスの既存の構造と整合性に影響しない変更 (新しいフィールドの追加など) に対しては、クエリの継続性が存在します。構造上の更新 (既存のフィールドの変更) を行う必要がある場合は、通常、開発環境での削除と再構築のワークフローを使用して、または運用サービスでインデックスの新しいバージョンを作成することによってそれらが管理されます。

[インデックスの再構築](#)を避けるため、小規模な変更を行っている一部のお客様は、以前のバージョンと共に新しいものを作成することによって、フィールドの "バージョン管理" を選択しています。これは、時間の経過と共に、特にレプリケートに負荷のかかる運用環境のインデックスで、古いフィールドまたは古いカスタム アナライザ一定義の形式の、孤立したコンテンツになります。インデックス ライフサイクル管理の一部として、インデックスの計画更新に関する問題に対処することができます。

## エンドポイント接続

ベクトルのインデックス作成とクエリ要求はすべて、インデックスを対象とします。エンドポイントは、通常、次のいずれかになります。

 [テーブルを展開する](#)

エンドポイント	接続とアクセスの制御
<code>&lt;your-service&gt;.search.windows.net/indexes</code>	インデックスのコレクションを対象とします。インデックスを作成、一覧表示、または削除するときに使用します。これらの操作には管理者権限が必要です。管理者 API キーまたは <a href="#">Search 共同作成者ロール</a> を通じて使用できます。
<code>&lt;your-service&gt;.search.windows.net/indexes/&lt;your-index&gt;/docs</code>	1 つのインデックスのドキュメント コレクションを対象とします。インデックスまたはデータ更新に対してクエリを実行するときに使用します。クエリには、読み取り権限で十分であり、クエリ API キーまたはデータ閲覧者ロールを通じて使用できます。データ更新の場合は、管理者権限が必要です。

## Azure AI 検索への接続方法

1. [アクセス許可](#)または [API アクセス キー](#)があることを確認します。既存のインデックスに対してクエリを実行する場合を除き、検索サービスのコンテンツを管理および表示するには、管理者権限または共同作成者ロールの割り当てが必要です。

2. Azure portal から開始します。検索サービスを作成したユーザーは、[アクセス制御 (IAM)] ページを使用して他のユーザーにアクセスを許可するなど、検索サービスを表示および管理できます。
3. プログラムによるアクセスのために他のクライアントに移動します。最初の手順としては、以下のクイックスタートとサンプルをお勧めします。
  - クイックスタート: REST
  - ベクトルのサンプル

## ベクトルデータへのアクセスをセキュリティで保護する

Azure AI 検索では、データ暗号化、インターネットなしのシナリオ用のプライベート接続、Microsoft Entra ID を介した安全なアクセスのためのロールの割り当てが実装されています。エンタープライズセキュリティ機能の全容については、「[Azure AI 検索のセキュリティ](#)」で説明されています。

## ベクトルストアを管理する

Azure には、診断ログとアラートを含む[監視プラットフォーム](#)が用意されています。推奨するベストプラクティスを次に示します。

- 診断ログを有効にする
- アラートを設定する
- クエリとインデックスのパフォーマンスを分析する

## 関連項目

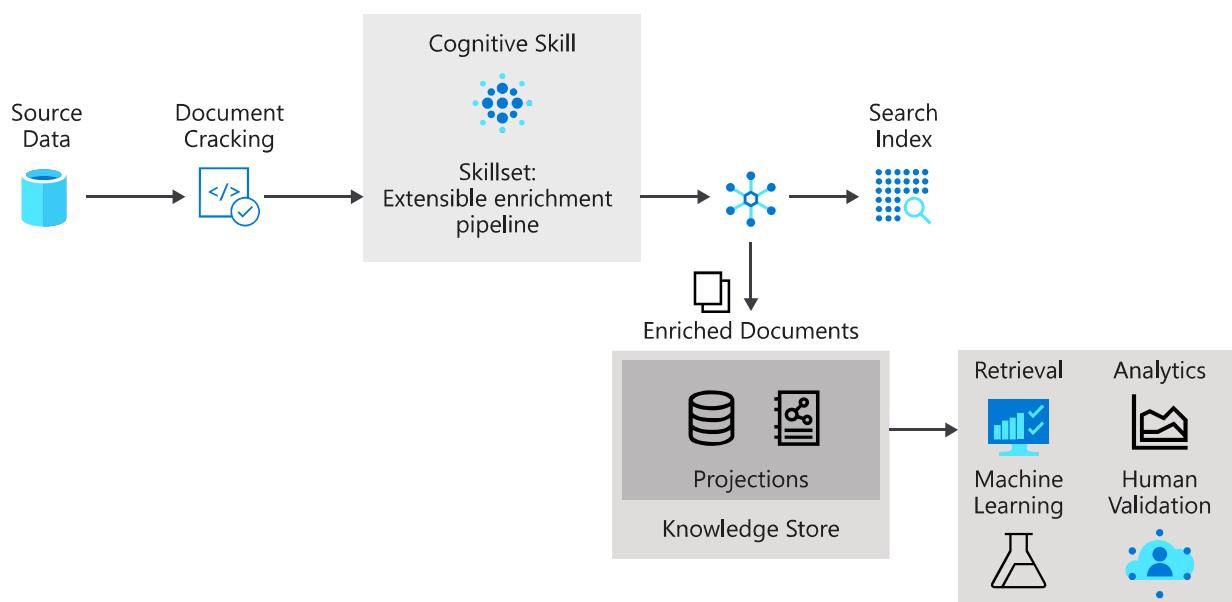
- REST API を使用してベクトルストアを作成する (クイックスタート)
- ベクトルストアを作成する
- ベクトルストアにクエリを実行する

# Azure AI Search 内のナレッジストア

[アーティクル] • 2024/01/10

ナレッジストアは、Azure AI Search のスキルセットによって作成された AI エンリッチコンテンツのセカンダリストレージです。 Azure AI Search では、インデックス作成ジョブは常に出力を検索インデックスに送信しますが、インデクサーにスキルセットをアタッチする場合は、必要に応じて、Azure Storage のコンテナーまたはテーブルに AI エンリッチされた出力を送信することもできます。 ナレッジストアは、ナレッジマイニングなどの検索以外のシナリオで、独立した分析またはダウンストリーム処理に使用できます。

インデックス作成の 2 つの出力 (検索インデックスとナレッジストア) は、同じパイプラインの相互排他的な製品です。 これらは同じ入力から派生し、同じデータを含みますが、その内容は構造化され、格納され、さまざまなアプリケーションで使用されます。



物理的には、ナレッジストアは [Azure Storage](#) です。つまり Azure Table Storage か Azure Blob Storage、またはその両方になります。 Azure Storage に接続できるすべてのツールまたはプロセスは、ナレッジストアのコンテンツを使用できます。

Azure portal を使用して表示すると、ナレッジストアはテーブル、オブジェクト、またはファイルの他のコレクションのようになります。 次のスクリーンショットは、3 つのテーブルで構成されるナレッジストアを示しています。 プレフィックスなどの名前付け規則を `kstore` 採用して、コンテンツをまとめることができます。

The screenshot shows the Azure Storage browser (preview) interface for a storage account named 'demoblobstorage'. The left sidebar includes links for Overview, Activity log, Tags, Diagnose and solve problems, Access Control (IAM), Data migration, Events, and Storage browser (preview). The main area displays a list of tables under the 'Tables' category. A red box highlights the 'Tables' link in the sidebar and the table names in the list. The table names listed are: kstoreProjectionDemoDocument, kstoreProjectionDemoEntities, kstoreProjectionDemoKeyPhrases, and MsAzSearchIndexerCacheIndex33b0d...

## ナレッジストアのメリット

ナレッジストアの主な利点は、コンテンツに柔軟にアクセスできることと、データを形成する機能という2つの点にあります。

Azure AI Search のクエリでのみアクセスできる検索インデックスとは異なり、ナレッジストアには、Azure Storage への接続をサポートする任意のツール、アプリ、またはプロセスからアクセスできます。この柔軟性によって、エンリッチメントパイプラインによって生成された、分析およびエンリッチメントされたコンテンツを消費するための新しいシナリオが開きます。

データをエンリッチする同じスキルセットを、データの形成にも使用できます。Power BIのようなツールは、テーブルの方が適していますが、データ サイエンス ワークフローには BLOB 形式の複雑なデータ構造が必要になる場合があります。スキルセットに [Shaper スキル](#)を追加すると、データのシェイプを制御できるようになります。そして、このシェイプをテーブルや BLOB などのプロジェクトに渡すことで、データの使用目的に沿った物理的なデータ構造を作成することができます。

次のビデオでは、これらの利点の両方について説明します。

<https://www.youtube-nocookie.com/embed/XWzLBP8iWqg?version=3>

## ナレッジストアの定義

ナレッジストアは、スキルセット定義内で定義されており、2つのコンポーネントがあります。

- Azureストレージの接続文字列

- ナレッジストアがテーブル、オブジェクト、ファイルのいずれで構成されているかを決定する[プロジェクト](#)。プロジェクト要素は配列です。1つのナレッジストア内に、テーブル、オブジェクト、ファイルの組み合わせを複数セット作成することができます。

JSON

```
"knowledgeStore": {
    "storageConnectionString": "<YOUR-AZURE-STORAGE-ACCOUNT-CONNECTION-STRING>",
    "projections": [
        {
            "tables": [ ],
            "objects": [ ],
            "files": [ ]
        }
    ]
}
```

この構造体で指定するプロジェクトの種類は、ナレッジストアが使用するストレージの種類を決定しますが、その構造体は決定しません。テーブル、オブジェクト、およびファイルのフィールドは、ナレッジストアをプログラムで作成する場合は Shaper スキルの出力によって決定され、ポータルを使用している場合はデータのインポート ウィザードによって決定されます。

- `tables` は、エンリッチメントされたコンテンツを Table Storage に投影します。分析ツールへの入力のために表形式のレポート構造が必要な場合や、データフレームとして他のデータストアにエクスポートする場合は、テーブルプロジェクトを定義します。同じプロジェクトグループ内の複数の `tables` を指定して、エンリッチメントされたドキュメントのサブセットまたは断面を取得することができます。同じプロジェクトグループ内では、テーブルのリレーションシップが保持されるため、すべてのテーブルを操作できます。

プロジェクトされたコンテンツは集計または正規化されません。次のスクリーンショットは、キー フレーズで並べ替えられたテーブルを示しており、隣接する列に親ドキュメントが示されています。インデックス作成中のデータインジェストとは対照的に、言語分析やコンテンツの集計はありません。複数形と大文字と小文字の違いは、一意のインスタンスと見なされます。

Content.metadata_storage_name	Content.KeyPhrases
Cognitive Services and Content Intelligence.pptx	Computer Vision
10-K-FY16.html	computing device
10-K-FY16.html	computing devices
MSFT_FY17_10K.docx	computing devices
10-K-FY16.html	Computing segment
Cognitive Services and Bots (spanish).pdf	confianza

- objects では、JSON ドキュメントを BLOB ストレージに投影します。 object の物理的表現は、エンリッチメントされたドキュメントを表す階層型の JSON 構造体です。
- files では、イメージファイルを BLOB ストレージに投影します。 file は、ドキュメントから抽出され、BLOB ストレージにそのまま転送されるイメージです。 "ファイル" という名前ですが、ファイルストレージではなく Blob Storage に表示されます。

## ナレッジストアの作成

ナレッジストアを作成するには、ポータルまたは API を使用します。

Azure Storage、スキルセット、インデクサーが必要になります。インデクサーには検索インデックスが必要なので、インデックス定義も指定する必要があります。

完成したナレッジストアへの最短ルートとしては、ポータルアプローチを採用してください。または、オブジェクトがどのように定義され、関連しているかをより深く理解するには、REST API を選択します。

Azure Portal

データのインポート ウィザードを使用して、[4つの手順で最初のナレッジストアを作成します。](#)

1. エンリッチするデータを含むデータソースを定義します。
2. スキルセットを定義します。スキルセットにより、エンリッチメントステップとナレッジストアが指定されます。
3. インデックススキーマを定義します。これは必要ない場合もありますが、インデクサーでは必要です。このウィザードではインデックスを推測できます。

4. ウィザードの完了。この最後のステップで、抽出、エンリッチメント、ナレッジストアの作成が行われます。

このウィザードを使用すると、いくつかのタスクを自動化できます。具体的には、整形とプロジェクトの両方 (Azure Storage 内の物理データ構造の定義) が作成されます。

## アプリに接続する

エンリッチされたコンテンツがストレージに存在するようになると、Azure Blob に接続する任意のツールまたはテクノロジを使用して、コンテンツを探索、分析、または使用できます。次の一覧が開始点です。

- エンリッチされたドキュメント構造とコンテンツを表示するための Azure portal の [Storage Explorer](#) またはストレージブラウザー (プレビュー)。これは、ナレッジストアのコンテンツを表示するためのベースライン ツールと考えてください。
- レポートと分析のための [Power BI](#)。
- さらに操作するための [Azure Data Factory](#)。

## コンテンツのライフサイクル

インデクサーとスキルセットを実行するたび、スキルセットまたは基になるソース データが変更された場合、ナレッジストアが更新されます。インデクサーによって取得された変更は、エンリッチメントプロセスを通じてナレッジストア内のプロジェクトに反映され、投影されたデータが元のデータ ソース内のコンテンツの現在の表現になります。

### ① 注意

プロジェクト内のデータを編集することができますが、ソース データ内のドキュメントが更新された場合、次のパイプライン呼び出しですべての編集が上書きされます。

## ソース データの変更

変更の追跡をサポートするデータ ソースの場合、インデクサーは新規および変更されたドキュメントを処理し、既に処理されている既存のドキュメントをバイパスします。タイムスタンプ情報はデータ ソースによって異なりますが、BLOB コンテナーでは、イ

インデクサーによって `lastmodified` の日付が確認され、取り込む必要がある BLOB が特定されます。

## スキルセットの変更

スキルセットに変更を加える場合は、[エンリッチメントされたドキュメントのキャッシュを有効にして](#)、可能な限り既存のエンリッチメントを再利用する必要があります。

増分キャッシュを使用しない場合、インデクサーは常に高いウォーター マークの順に逆戻りせずドキュメントを処理します。BLOB の場合、インデクサーは、インデクサーの設定やスキルセットに対する変更に関係なく、`lastModified` で並べ替えた BLOB を処理します。スキルセットを変更した場合、以前に処理されたドキュメントは、新しいスキルセットを反映するように更新されません。スキルセットの変更後に処理されたドキュメントでは新しいスキルセットが使用され、その結果、インデックス ドキュメントには古いスキルセットと新しいスキルセットが混在します。

増分キャッシュを使用する場合、スキルセットの更新後に、インデクサーはスキルセットの変更の影響を受けないエンリッチメントを再利用します。アップストリーム エンリッチメントは、変更されたスキルから独立して分離されたエンリッチメントと同様に、キャッシュからプルされます。

## 削除

インデクサーは、Azure Storage 内の構造とコンテンツを作成および更新しますが、それらを削除しません。インデクサーまたはスキルセットが削除された場合でも、プロジェクトは引き続き存在します。ストレージ アカウントの所有者は、不要になったプロジェクトを削除する必要があります。

## 次のステップ

ナレッジストアは、エンリッチメントされたドキュメントを永続化する手段として、スキルセットを設計する際に役立つほか、Azure Storage アカウントにアクセスする機能を備えた、あらゆるクライアント アプリケーションから利用する新しい構造やコンテンツを作成する際にも役立てることができます。

エンリッチメントされたドキュメントを作成する最も簡単なアプローチは、[ポータルを使用することですが](#)、Postman と REST API を使用する方法もあります。プログラムでオブジェクトがどのように作成され、参照されるのかについて分析情報が必要な場合には、後者の方が便利です。

[Postman と REST を使用してナレッジストアを作成する](#)

# Azure AI Search でのデータインポート

[アーティクル] • 2024/01/19

Azure AI Search では、クエリは、[検索インデックス](#)に読み込まれたユーザー所有のコンテンツに対して実行されます。この記事では、インデックスを作成する 2 つの基本的なワークフローについて説明します。プログラムでデータをインデックスにプッシュするワークフローと、[検索インデクサー](#)を使用してデータをプルするワークフローです。

どちらの方法でも、外部データ ソースからドキュメントが読み込まれます。空のインデックスを作成することもできますが、コンテンツを追加するまでクエリは実行できません。

## ① 注意

AI エンリッチメントがソリューションの要件である場合は、プル モデル (インデクサー) を使用してインデックスを読み込む必要があります。スキルセットはインデクサーにアタッチされ、独立して実行されません。

## インデックスにデータをプッシュする

プッシュ モデルは、API を使用して既存の検索インデックスにドキュメントをアップロードするアプローチです。ドキュメントは、1 バッチあたり最大 1,000 個、またはバッチあたり 16 MB (メガバイト) (どちらか早い方) に個別に、またはバッチでアップロードできます。

主な利点:

- データ ソースの種類に制限はありません。ペイロードは、インデックス スキーマにマップされる JSON ドキュメントで構成されている必要がありますが、データはどこからでもソース化できます。
- 実行頻度に制限はありません。インデックスには、必要に応じて何度でも変更をプッシュすることができます。待機時間の要件が低いアプリケーションの場合 (たとえば、インデックスを製品在庫の変動と同期させる必要がある場合など)、プッシュ モデルが唯一のオプションです。
- ドキュメントの接続性と安全な取得は、完全に制御下にあります。これに対し、インデクサー接続は、Azure AI Search で提供されるセキュリティフィーチャーを使用して認証されます。

# Azure AI Search インデックスにデータをプッシュする方法

1つまたは複数のドキュメントをインデックスに読み込むには、次の API を使用します。

- ドキュメントの追加、更新、削除 (REST API)
- IndexDocumentsAsync (Azure SDK for .NET) または [SearchIndexingBufferedSender](#)
- IndexDocumentsBatch (Azure SDK for Python) または [SearchIndexingBufferedSender](#)
- IndexDocumentsBatch (Azure SDK for Java) または [SearchIndexingBufferedSender](#)
- IndexDocumentsBatch (Azure SDK for JavaScript) または [SearchIndexingBufferedSender](#)

Azure portal を使用したデータのプッシュはサポートされていません。

プッシュ API の概要については、以下を参照してください。

- クイックスタート: Azure SDK を使用したフルテキスト検索
- C# チュートリアル: プッシュ API を使用してインデックス作成を最適化する
- REST クイック スタート: PowerShell を使用して Azure AI Search インデックスを作成する

## インデックス作成アクション: upload、merge、mergeOrUpload、delete

インデックス作成アクションの種類をドキュメントごとに制御できます。つまり、ドキュメントを全部アップロードするか、既存のドキュメントコンテンツとマージするか、または削除するかを指定できます。

REST API と Azure SDK のどちらを使用する場合でも、データのインポートでは次のドキュメント操作がサポートされます。

- ドキュメントが新しい場合は挿入され、存在する場合は更新または置き換えられる "upsert" と同様にアップロードします。インデックスに必要な値がドキュメントにない場合、ドキュメントフィールドの値は null に設定されます。
- マージでは、既に存在するドキュメントが更新され、見つからないドキュメントが失敗します。マージは既存の値を置き換えます。そのため、`Collection(Edm.String)` 型のフィールドなど、複数の値を含むコレクションフィールドは必ず確認してください。たとえば、`tags` フィールドの値が `["budget"]` で始まり、値 `["economy", "pool"]` でマージを実行した場合、`tags` フィールドの

最終値は `["economy", "pool"]` になります。 `["budget", "economy", "pool"]` にはなりません。

- `mergeOrUpload`。ドキュメントが存在する場合は `merge` と同様な動作をし、ドキュメントが新しい場合は `upload` の動作をします。
- `delete`。インデックスから指定したドキュメントを削除します。個々のフィールドを削除する場合は、代わりに `merge` を使い、問題のフィールドを `null` に設定します。

## インデックスへのデータのプル

プレモデルでは、サポートされているデータソースに接続するインデクサーが使用され、データがインデックスに自動的にアップロードされます。Microsoft のインデクサーは、次のプラットフォームで利用できます。

- [Azure BLOB Storage](#)
- [Azure Table Storage](#)
- [Azure Data Lake Storage Gen2](#)
- [Azure Files \(プレビュー\)](#)
- [Azure Cosmos DB](#)
- [Azure SQL Database、SQL Managed Instance、および Azure VM 上の SQL Server](#)
- [Microsoft 365 での SharePoint \(プレビュー\)](#)

Microsoft パートナーによって開発およびメインされたサードパーティ製コネクタを使用できます。詳細とリンクについては、「[データソースギャラリー](#)」を参照してください。

インデクサーは、インデックスをデータソース(通常はテーブル、ビュー、または同等の構造体)に接続し、ソースフィールドをインデックスの同等のフィールドにマップします。実行中、行セットが自動的に JSON に変換され、指定したインデックスに読み込まれます。すべてのインデクサーはスケジュールをサポートしているため、データの更新頻度を指定できます。ほとんどのインデクサーは、変更の追跡を提供します(データソースでサポートされている場合)。インデクサーは、新しいドキュメントを認識するだけでなく、既存のドキュメントの変更と削除を追跡するため、インデックス内のデータをアクティブに管理する必要がありません。

## Azure AI Search インデックスにデータをプルする方法

インデクサーベースのインデックス作成には、次のツールと API を使用します。

- [Azure portal のデータのインポート ウィザード](#)

- REST API: [インデクサーの作成 \(REST\)](#)、[データソースの作成 \(REST\)](#)、[インデックスの作成 \(REST\)](#)
- Azure SDK for .NET: [SearchIndexer](#)、[SearchIndexerDataSourceConnection](#)、[SearchIndex](#)、
- Azure SDK for Python: [SearchIndexer](#)、[SearchIndexerDataSourceConnection](#)、[SearchIndex](#)、
- Azure SDK for Java: [SearchIndexer](#)、[SearchIndexerDataSourceConnection](#)、[SearchIndex](#)、
- Azure SDK for JavaScript: [SearchIndexer](#)、[SearchIndexerDataSourceConnection](#)、[SearchIndex](#)、

インデクサー機能は、[Azure portal]、[REST API](#)、および[.NET SDK](#)で公開されています。

ポータルを使用する利点は、Azure AI Search では通常、ソースデータセットのメタデータを読み取ることでデフォルトのインデックススキーマを生成できることです。

## Search エクスプローラーを使用してデータのインポートを検証する

ドキュメントのアップロード時に事前チェックを実行する簡単な方法は、ポータルで[Search エクスプローラー](#)を使用することです。



エクスプローラーを使用すると、コードを記述することなくインデックスを照会できます。検索エクスペリエンスは、既定の設定([単純構文](#)、既定の[searchMode クエリパラメーター](#)など)に基づきます。結果は JSON で返されるため、ドキュメント全体を確認できます。

JSON ビューの検索エクスプローラーで実行できるクエリの例を次に示します。  
"HotelId" は hotels-sample-index のドキュメントキーです。フィルターには特定のドキュメントのドキュメント ID を指定します。

```
JSON
{
  "search": "*",
  "filter": "HotelId eq '50'"
}
```

REST を使っている場合は、この[参照クエリ](#)で同じ目的を達成できます。

## 関連項目

- インデクサーの概要
- ポータルのクイックスタート: インデックスの作成、読み込み、クエリ

# Azure AI Search のインデクサー

[アーティクル] • 2023/12/18

*Azure AI Search のインデクサー*は、クラウド データ ソースからテキスト データを抽出し、ソース データと検索インデックスの間のフィールド間マッピングを使用して検索インデックスを設定するクローラーです。この方法は、インデックスにデータを追加するコードを記述することなく、検索サービスがデータをプルするため、「プル モデル」と呼ばれることもあります。

インデクサーはスキルセットの実行と AI エンリッチメントも推進します。このエンリッチメントでは、インデックスにルーティングされるコンテンツの追加処理を統合するようにスキルを構成できます。画像ファイルの OCR、データ チャンクのテキスト分割スキル、複数の言語のテキスト翻訳など、いくつかの例があります。

インデクサーは、サポートされているデータ ソースを対象とします。インデクサー構成では、データ ソース (配信元) と検索インデックス (宛先) を指定します。Azure Blob Storage など、いくつかのソースには、そのコンテンツの種類に固有の追加の構成プロパティがあります。

インデクサーは、オンデマンドで実行することも、5 分ごとに実行される定期的なデータ更新スケジュールで実行することもできます。より頻繁に更新するには、Azure AI Search と外部データ ソースの両方のデータを同時に更新する "プッシュ モデル" が必要です。

検索サービスは、検索ユニットごとに 1 つのインデクサー ジョブを実行します。同時処理が必要な場合は、十分なレプリカがあることを確認してください。インデクサーはバックグラウンドで実行されないため、サービスに負荷がかかっている場合は、通常よりも多くのクエリ調整が検出される可能性があります。

## インデクサーのシナリオとユース ケース

インデクサーは、データ インジェストの唯一の手段として、または他の手法と組み合わせて使用できます。次の表に主なシナリオをまとめています。

〔〕 テーブルを展開する

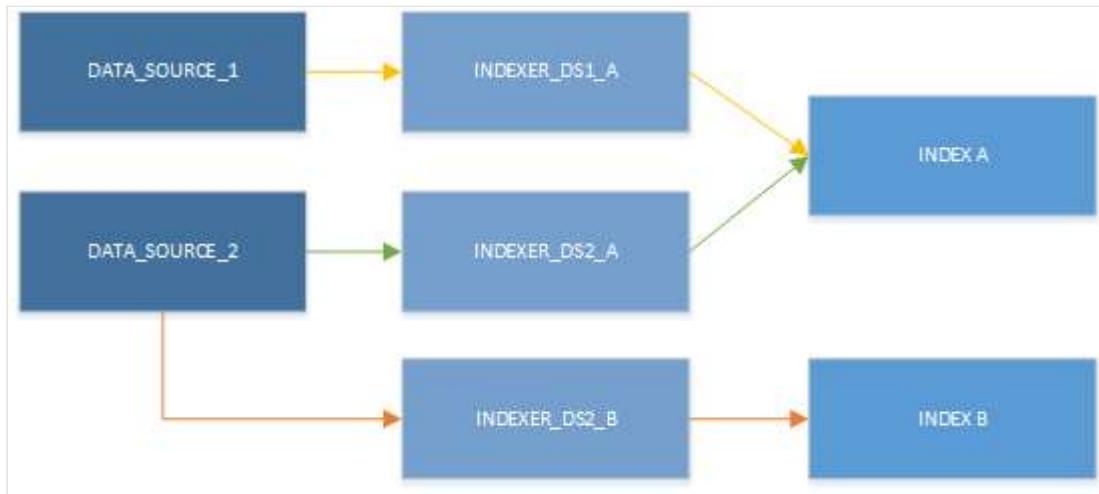
シナリオ	戦略
単一データ	このパターンは最も単純です。1 つのデータ ソースが検索インデックス用の唯一のコンテンツ プロバイダーです。サポートされているほとんどのデータソースで何らかの形式

シナリオ	戦略
ソース	の変更が検出されるため、ソースでコンテンツが追加または更新されたときに、後続のインデクサーの実行によって差分が取得されます。
複数のデータソース	インデクサーの指定で使用できるデータソースは1つだけですが、検索インデックス自体は複数のソースからのコンテンツを受け入れることができます。この場合、各インデクサーの実行によって、別のデータプロバイダーから新しいコンテンツが取り込まれます。各ソースは、完全なドキュメントの共有を投稿したり、各ドキュメントで選択したフィールドを設定したりできます。このシナリオの詳細については、 <a href="#">複数のデータソースからのインデックスのチュートリアル</a> を参照してください。
複数のインデクサー	実行時のパラメーター、スケジュール、またはフィールドマッピングを変更する必要がある場合、複数のデータソースは通常、複数のインデクサーとペアになります。
複数のインデクサー	Azure AI Search のクロスリージョンのスケールアウトがもう1つのシナリオです。同じ検索インデックスのコピーが異なるリージョンに存在する場合があります。検索インデックスのコンテンツを同期するには、同じデータソースからプルする複数のインデクサーを作成できます。この場合、各インデクサーのターゲットはリージョンごとに異なる検索インデックスです。非常に大きなデータセットの並列インデックスでも、各インデクサーがデータのサブセットをターゲットとするマルチインデクサー戦略が必要です。
コンテンツの変換	インデクサーは、スキルセットの実行とAIエンリッチメントを推進します。コンテンツの変換は、インデクサーにアタッチする <a href="#">スキルセット</a> で定義されます。スキルを使用して、 <a href="#">データチャunkとベクター化</a> を組み込むことができます。

ターゲットインデックスとデータソースの組み合わせごとにインデクサーを1つ作成するように設計する必要があります。複数のインデクサーが同じインデックスに書き込みできます。複数のインデクサーに同じデータソースを再利用できます。ただし、インデクサーが1回に利用できるデータソースは1つだけです。そして、書き込めるインデックスは1つだけです。次の図に示すように、1つのデータソースが1つのインデクサーに入力を提供し、1つのインデックスを設定します:



一度に使用できるインデクサーは1つだけですが、リソースはさまざまな組み合わせで使用できます。次の図の主なポイントは、データソースを複数のインデクサーと組み合わせることができ、複数のインデクサーが同じインデックスに書き込むことができる点です。



## サポートされるデータ ソース

インデクサーは、Azure および Azure 外部でデータストアをクロールします。

- [Azure Blob Storage](#)
- [Azure Cosmos DB](#)
- [Azure Data Lake Storage Gen2](#)
- [Azure SQL Database](#)
- [Azure Table Storage](#)
- [Azure SQL Managed Instance](#)
- [Azure Virtual Machines における SQL Server](#)
- [Azure Files \(プレビュ一段階\)](#)
- [Azure MySQL \(プレビュ一段階\)](#)
- [Microsoft 365 での SharePoint \(プレビュ一段階\)](#)
- [Azure Cosmos DB for MongoDB \(プレビュ一段階\)](#)
- [Azure Cosmos DB for Apache Gremlin \(プレビュ一段階\)](#)

Azure Cosmos DB for Cassandra はサポートされていません。

インデクサーは、テーブルやビューなどのフラット化された行セット、またはコンテナーまたはフォルダー内の項目を受け入れます。ほとんどの場合、行、レコード、または項目ごとに 1 つの検索ドキュメントが作成されます。

共有プライベートリンクを使用する場合、リモート データ ソースへのインデクサー接続は、標準のインターネット接続 (パブリック) または暗号化されたプライベート接続を使用して行うことができます。また、マネージド ID を使用して認証を行うように、接続を設定することもできます。セキュリティで保護された接続の詳細については、[Azure ネットワーク セキュリティ機能によって保護されたコンテンツへのインデクサーのアクセスと、マネージド ID を使用したデータ ソースへの接続](#)に関するページを参照してください。

# インデックス作成のステージ

最初の実行時に、インデックスが空の場合、テーブルまたはコンテナーで提供されるすべてのデータがインデクサーによって読み取られます。その後の実行では、通常、変更されたデータのみがインデクサーによって検出され取得されます。BLOB データの場合、変更の検出は自動で行われます。Azure SQL や Azure Cosmos DB などの他のデータソースで、変更の検出を有効にする必要があります。

受信したドキュメントごとに、インデクサーによって、ドキュメントの取得からインデックス付けのための最終的な検索エンジンの "ハンドオフ" までの、複数のステップが実装または調整されます。また、インデクサーを使用すると、スキルセットが定義されている場合に、[スキルセットの実行と出力](#)も促進されます。



## ステージ 1: ドキュメント解析

ドキュメント解析は、ファイルを開いてコンテンツを抽出するプロセスです。テキストベースのコンテンツは、サービスのファイル、テーブルの行、またはコンテナーやコレクションの項目から抽出できます。スキルセットと[画像スキル](#)を追加した場合、ドキュメント解析で画像を抽出し、画像処理のためにキューに登録することもできます。

データソースに応じて、インデックス付けが可能なコンテンツを抽出するために、インデクサーによってさまざまな操作が試行されます。

- ドキュメントが PDF などの画像が埋め込まれたファイルである場合、インデクサーはテキスト、画像、メタデータを抽出します。インデクサーは、[Azure Blob Storage](#)、[Azure Data Lake Storage Gen2](#)、[SharePoint](#) からファイルを開くことができます。
- ドキュメントが [Azure SQL](#) のレコードの場合は、インデクサーによって各レコードの各フィールドからバイナリ以外のコンテンツが抽出されます。
- ドキュメントが [Azure Cosmos DB](#) 内のレコードの場合は、インデクサーによって Azure Cosmos DB ドキュメントのフィールドとサブフィールドからバイナリ以外のコンテンツが抽出されます。

## ステージ 2: フィールド マッピング

インデクサーによって、ソース フィールドからテキストが抽出され、インデックスまたはナレッジ ストアの送信先フィールドにそれが送信されます。フィールド名とデータ型が一致すると、パスは明確になります。ただし、出力には異なる名前または型が必要な場合があります。その場合は、フィールドをマップする方法をインデクサーに指示する必要があります。

[フィールド マッピングを指定する](#)には、インデクサー定義に、ソース フィールドと宛先フィールドを入力します。

フィールド マッピングは、ドキュメント解析の後、変換前に、インデクサーがソース ドキュメントから読み取るときに行われます。フィールド マッピングを定義するときに、ソース フィールドの値は変更されずにそのまま送信先フィールドに送信されます。

## ステージ 3: スキルセットの実行

スキルセットの実行は、組み込みまたはカスタムの AI 処理を呼び出す省略可能なステップです。スキルセットでは、コンテンツがバイナリの場合、光学式文字認識 (OCR) またはその他の形式の画像分析を追加できます。スキルセットで自然言語処理を追加することもできます。たとえば、テキスト翻訳やキー フレーズ抽出を追加できます。

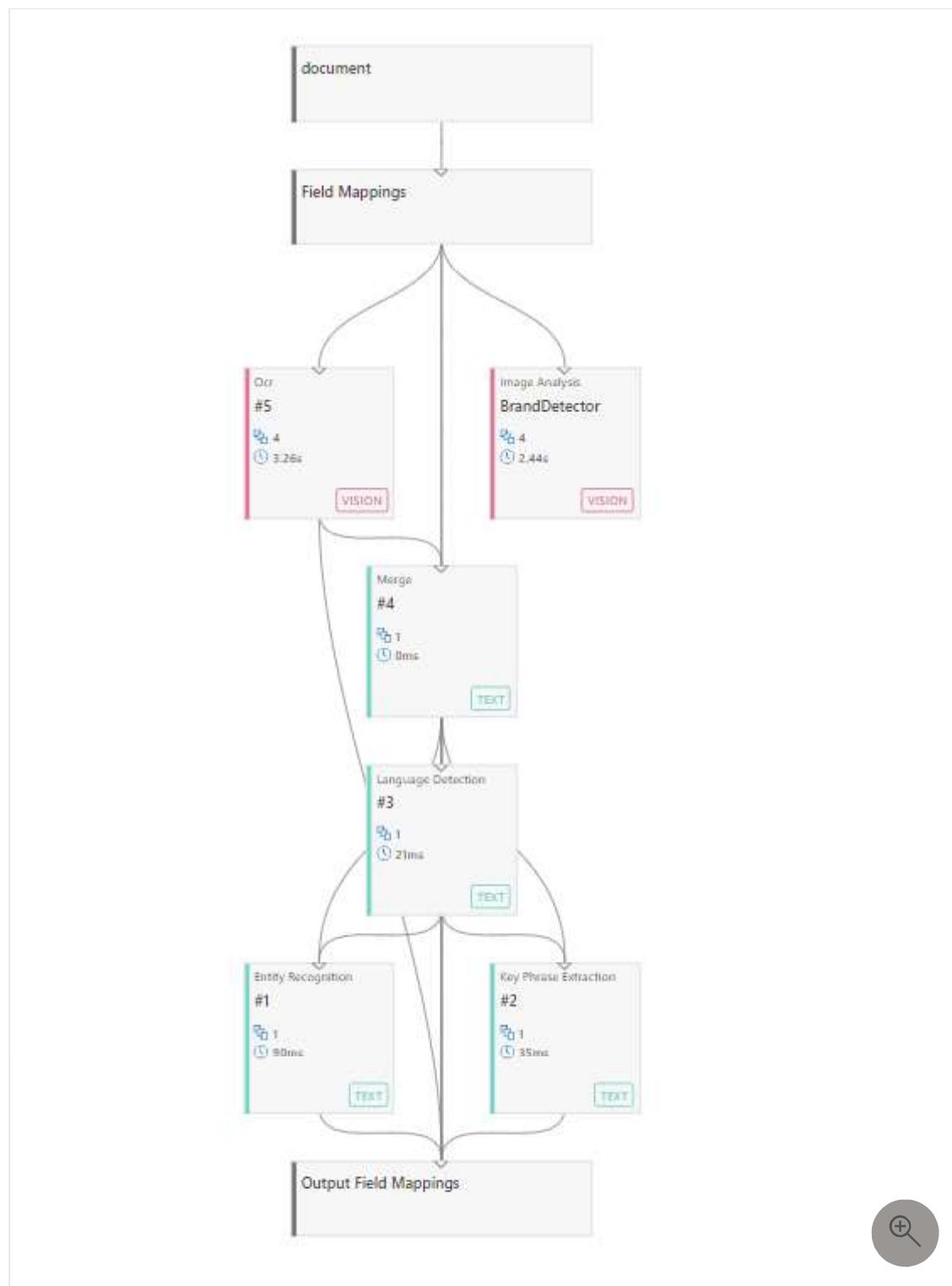
変換が何であれ、スキルセットの実行は、エンリッチメントが発生する場所です。インデクサーがパイプラインの場合、[スキルセット](#)を "パイプライン内のパイプライン" として考えることができます。

## ステージ 4: 出力フィールドマッピング

スキルセットを含める場合は、インデクサー定義で[出力フィールド マッピングを指定する](#)必要があります。スキルセットの出力は、エンリッチされたドキュメントと呼ばれるツリー構造として内部的に示されます。出力フィールド マッピングを使用すると、このツリーの部分を選択してインデックス内のフィールドにマップすることができます。

名前が類似しているにもかかわらず、出力フィールド マッピングとフィールド マッピングは、異なるソースから関連付けを構築します。フィールド マッピングでは、ソース フィールドの内容を検索インデックスの宛先フィールドに関連付けます。出力フィールド マッピングでは、内部のエンリッチされたドキュメント (スキル出力) の内容をインデックス内の宛先フィールドに関連付けます。省略可能と見なされるフィールド マッピングとは異なり、インデックスに存在する必要がある変換されたすべてのコンテンツには、出力フィールド マッピングが必要になります。

次の図は、インデクサーのステージ（ドキュメント解析、フィールド マッピング、スキルセットの実行、出力フィールド マッピング）のサンプルインデクサーのデバッグ セッション表現を示しています。



## の基本的なワークフロー

インデクサーで実行できる機能は、データソースごとに異なります。そのためインデクサーとデータソースの構成には、インデクサーの種類ごとに異なる点があります。しかし基本的な成り立ちと要件は、すべてのインデクサーに共通です。以降、すべてのインデクサーに共通の手順について取り上げます。

## 手順 1: データ ソースを作成する

インデクサーには、接続文字列と必要に応じて資格情報を提供する、"データ ソース" オブジェクトが必要です。データ ソースは独立したオブジェクトです。複数のインデクサーは、同じデータ ソース オブジェクトを使用して、一度に複数のインデックスを読み込むことができます。

次のいずれかの方法を使用してデータ ソースを作成できます。

- Azure portal を使用し、検索サービス ページの [データ ソース] タブで [データ ソースの追加] を選択して、データ ソースの定義を指定します。
- Azure portal を使用して、[データのインポート] ウィザードでデータ ソースを出力します。
- REST API を使用して、[データ ソースの作成] を呼び出します。
- Azure SDK for .NET を使用して、`SearchIndexerDataSourceConnection` クラスを呼び出します

## 手順 2: インデックスを作成する

インデクサーは、データ インジェストに関連したいくつかのタスクを自動化しますが通常、そこにはインデックスの作成は含まれていません。前提条件として、お使いの外部データ ソース内のすべてのソース フィールドの対応するターゲット フィールドが含まれた定義済みインデックスが必要になります。各フィールドでは、名前とデータ型が一致する必要があります。そうでない場合は、[フィールド マッピングを定義](#)して関連付けを確立できます。

詳細については、「[インデックスの作成](#)」を参照してください。

## 手順 3: インデクサーを作成して実行 (またはスケジュール) する

インデクサーの定義は、インデクサーを一意に識別するプロパティ、使用するデータ ソースとインデックスを指定するプロパティ、実行時の動作に影響する他の構成オプションを提供するプロパティ (インデクサーをオンデマンドで実行するか、スケジュールに基づき実行するかなど) で構成されます。

データ アクセスまたはスキルセットの検証に関するエラーまたは警告は、インデクサーの実行中に発生します。インデクサーの実行が開始されるまで、データ ソース、インデックス、スキルセットなどの依存オブジェクトは、検索サービスでパッシブになります。

詳細については、「[インデクサーの作成](#)」を参照してください

インデクサーの初回実行の後、[オンデマンドで再実行](#)することや、[スケジュールを設定](#)することができます。

インデクサーの状態は、[ポータル](#)か [Get Indexer Status API](#) を使用して監視できます。また、[インデックスのクエリを実行](#)し、期待した結果が得られるかどうかを確認する必要があります。

インデクサーには専用の処理リソースがありません。これに基づき、インデクサーの状態が(キュー内の他のジョブに応じて)実行される前にアイドル状態として表示され、実行時間が予測できない場合があります。インデクサーのパフォーマンスは、ドキュメントサイズ、ドキュメントの複雑さ、画像分析などのその他の要因によっても定義されます。

## 次のステップ<sup>°</sup>

インデクサーの概要がわかったので、次のステップは、インデクサーのプロパティとパラメーター、スケジュール、およびインデクサーの監視について確認することです。また、[サポートされているデータ ソース](#)の一覧に戻り、特定のソースの詳細を確認することもできます。

- [インデクサーを作成する](#)
- [インデクサーのリセットと実行](#)
- [インデクサーをスケジュールする](#)
- [フィールド マッピングを定義する](#)
- [インデクサーの状態を監視する](#)