5.5 Create a knowledge mining solution with Azure AI Search

Unlock the hidden insights in your data with **Azure AI Search**. In this module, you'll learn how to implement a **knowledge mining solution** that extracts and enriches data, making it searchable and ready for deeper analysis.

Learning objectives

After completing this module, you'll be able to:

- Implement **indexing** with **Azure AI Search**
- Use AI skills to **enrich data in an index**
- Search an index to find relevant information
- Persist extracted information in a knowledge store

Introduction

Azure AI Search is a powerful cloud-based service that enables you to **extract, enrich, and explore information** from a wide variety of data sources. In this module, you'll learn how to build **intelligent search and knowledge mining solutions** using Azure AI Search.

We'll start by introducing the core concepts of Azure AI Search, including how to **connect to data sources and create indexes**. You'll discover how the indexing process works, and how AI skills can be used to enrich your data with insights such as **language detection, key phrase extraction, and image analysis**.

After learning how to implement an index, you'll explore how to **query and filter results** using full-text search.

Finally, you'll see how to **use the knowledge store to persist enriched data** for further analysis and integration with other systems.

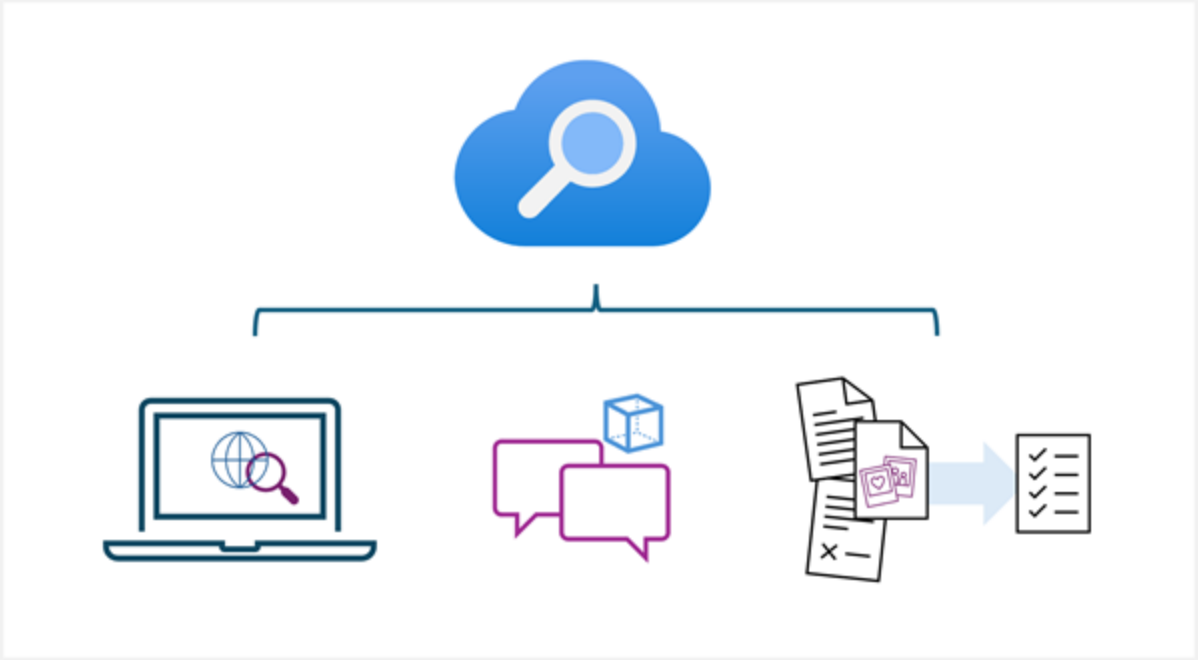
What is Azure AI Search?

Azure AI Search provides a cloud-based solution for **indexing and querying a wide range of data sources**, and **creating comprehensive and high-scale search solutions**. It provides the infrastructure and tools to create search solutions that **extract data from structured, semi-structured, and non-structured documents and other data sources**.

With Azure AI Search, you can:

- **Index documents and data** from a range of sources.
- Use AI skills to **enrich index data**.
- **Store extracted insights in a knowledge store** for analysis and integration.

Azure AI Search indexes contain insights extracted from your data; which can include **text inferred or read using OCR from images, entities and key phrases detection through text analytics**, and other derived information based on AI skills that are integrated into the indexing process.



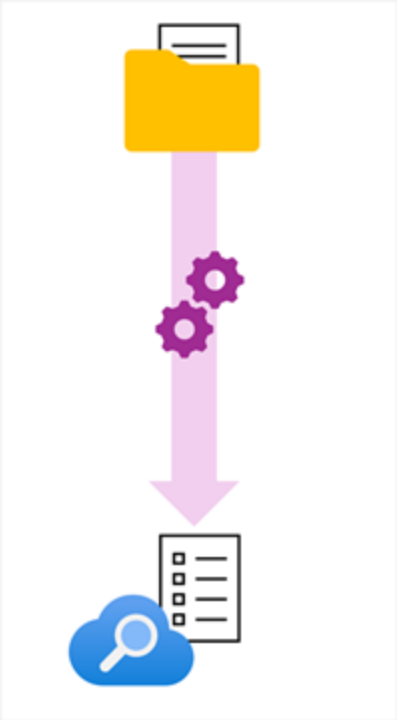
Azure AI search has many applications, including:

- Implementing an enterprise search solution to help employees or customers find information in websites or applications.
- **Supporting retrieval augmented generation (RAG) in generative AI applications** by using vector-based indexes for prompt grounding data.
- Creating **knowledge mining solutions** in which the indexing process is used to infer insights and extract granular data assets from documents to support data analytics.

In this module, we'll focus on Azure AI Search in knowledge mining scenarios.

Extract data with an indexer

At the heart of Azure AI Search solutions is the creation of an index. **An index contains your searchable content and is created and updated, unsurprisingly, by an indexer**.



The indexing process starts with a data source: the storage location of your original data artifacts; for example, an Azure blob store container full of documents, a database, or some other store.

The Indexer automates the extraction and indexing of data fields through an enrichment pipeline, in which it applies *document cracking to extract the contents of the source documents and applies incremental steps to create a hierarchical (JSON-based) document with the required fields for the index definition*.

The result is a populated index, which can be queried to return specified fields from documents that match the query criteria.

How documents are constructed during indexing

The indexing process works by creating a document for each indexed entity. *During indexing, an enrichment pipeline iteratively builds the documents that combine metadata from the data source with enriched fields extracted or generated by skills*. You can think of **each indexed document as a JSON**

structure, which initially consists of a document with the index fields you have mapped to fields extracted directly from the source data, like this:

- document
  - metadata_storage_name
  - metadata_author
  - content

When the documents in the data source contain images, you can configure the indexer to extract the image data and place each image in a **normalized_images** collection, like this:

- document
  - metadata_storage_name
  - metadata_author
  - content
  - normalized_images
    - image0
    - image1

Normalizing the image data in this way enables you to use the collection of images as an input for skills that extract information from image data.

Each skill adds fields to the document, so for example a skill that detects the language in which a document is written might store its output in a **language** field, like this:

- document
  - metadata_storage_name
  - metadata_author
  - content
  - normalized_images
    - image0
    - image1
  - language

The document is structured hierarchically, and the skills are applied to a specific context within the hierarchy, enabling you to run the skill for each item at a particular level of the document. For example, you could run an **optical character recognition (OCR)** skill for each image in the normalized images collection to extract any **text** they contain:

- document
  - metadata_storage_name

    - metadata_author
    - content
    - normalized_images
        - image0
            - Text
        - image1
            - Text
    - language

The output fields from each skill can be used as inputs for other skills later in the pipeline, which in turn store their outputs in the document structure. For example, we could use a merge skill to combine the original text content with the text extracted from each image to create a new **merged_content** field that contains all of the text in the document, including image text.

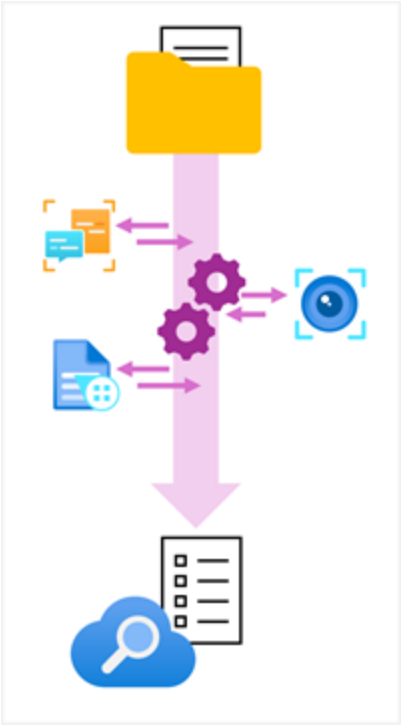
- document
    - metadata_storage_name
    - metadata_author
    - content
    - normalized_images
        - image0
            - Text
        - image1
            - Text
    - language
    - merged_content

The fields in the final document structure at the end of the pipeline are mapped to index fields by the indexer in one of two ways:

- **Fields extracted directly from the source data are all mapped to index fields**. These mappings can be **implicit** *(fields are automatically mapped to in fields with the same name in the index)* or **explicit** *(a mapping is defined to match a source field to an index field, often to rename the field to something more useful or to apply a function to the data value as it is mapped)*.
- Output fields from the skills in the skillset are explicitly mapped from their hierarchical location in the output to the target field in the index.

Enrich extracted data with AI skills

The enrichment pipeline that is orchestrated by an indexer uses a skillset of AI skills to create AI-enriched fields. **The indexer applies each skill in order, refining the index document at each step.**



Built-in skills

Azure AI Search provides a collection of built-in skills that you can include in a skillset for your indexer. **Built-in skills include functionality from Azure AI services such as Azure AI Vision and Azure AI Language**, enabling you to apply enrichments such as:

- Detecting the **language** that text is written in.
- Detecting and extracting **places, locations, and other entities in the text**.
- Determining and extracting **key phrases** within a body of text.
- **Translating** text.
- Identifying and extracting (or removing) **personally identifiable information (PII)** within the text.
- Extracting **text from images**.
- Generating **captions and tags** to describe images.

To use the built-in skills, your indexer must have access to an Azure AI services resource. You can use a restricted Azure AI search resource that is included in Azure AI Search (and which is limited to indexing 20 or fewer documents) or you can attach an Azure AI services resource in your Azure subscription (which must be in the same region as your Azure AI Search resource).

Custom skills

You can further extend the enrichment capabilities of your index by creating custom skills. As the name suggests, custom skills perform custom logic on input data from your index document to return new field values that can be incorporated into the index. Often, **custom skills are "wrappers" around services that are specifically designed to extract data from documents**. For example, you could implement a custom skill as an Azure Function, and use it to pass data from your index document to an **Azure AI Document Intelligence** model, which can extract fields from a form.

> Tip: To learn mode about using custom skills with Azure AI Search, see Add a custom skill to an Azure AI Search enrichment pipeline in the Azure AI Search documentation.

Search an index

The index is the searchable result of the indexing process. It consists of *a collection of JSON documents, with fields that contain the values extracted during indexing*. Client applications can query the index to retrieve, filter, and sort information.

Each index field can be configured with the following attributes:

- **key**: Fields that define a unique key for index records.
- **searchable**: Fields that can be queried using full-text search.

- **filterable**: Fields that can be included in filter expressions to return only documents that **match specified constraints**.
- **sortable**: Fields that can be used to **order** the results.
- **facetable**: Fields that can be used to determine values for facets (user interface elements used to filter the results based on a list of **known field values**).
- **retrievable**: Fields that can be included in search results (by default, all fields are retrievable unless this attribute is explicitly removed).

Full-text search

While you could retrieve index entries based on simple field value matching, most search solutions use full-text search semantics to query an index.

Full-text search describes search solutions that parse text-based document contents to find query terms. Full-text search queries in Azure AI Search are based on the Lucene query syntax, which provides a rich set of query operations for searching, filtering, and sorting data in indexes. Azure AI Search supports two variants of the Lucene syntax:

- **Simple** - An intuitive syntax that makes it easy to perform basic searches that match literal query terms submitted by a user.
- **Full** - An extended syntax that supports complex filtering, regular expressions, and other more sophisticated queries.

Client applications submit queries to Azure AI Search by specifying a search expression along with other parameters that determine how the expression is evaluated and the results returned. Some common parameters submitted with a query include:

- **search** - A search expression that includes the terms to be found.
- **queryType** - The **Lucene syntax** to be evaluated (simple or full).
- **searchFields** - The index fields to be searched.
- **select** - The fields to be included in the results.
- **searchMode** - Criteria for including results based on multiple search terms. For example, suppose you search an index of travel-related documents for comfortable hotel. A searchMode value of Any returns documents that contain "comfortable", "hotel", or both; while a searchMode value of All restricts results to documents that contain both "comfortable" and "hotel".

Query processing consists of four stages:

1. **Query parsing**. The search expression is *evaluated and reconstructed as a tree of appropriate subqueries*. Subqueries might include **term queries** (finding specific individual words in the

search expression - for example hotel), **phrase queries** (finding multi-term phrases specified in quotation marks in the search expression - for example, "free parking"), and **prefix queries** (finding terms with a specified prefix - for example air*, which would match airway, air-conditioning, and airport).

2. **Lexical analysis** - The query terms are analyzed and refined based on linguistic rules. For example, *text is converted to lower case and nonessential stopwords (such as "the", "a", "is", and so on) are removed*. Then words are *converted to their root form* (for example, "comfortable" might be simplified to "comfort") and *composite words are split into their constituent terms*.
3. **Document retrieval** - The *query terms are matched against the indexed terms*, and the set of matching documents is identified.
4. **Scoring** - A *relevance score* is assigned to each result based on a term frequency/inverse document frequency (TF/IDF) calculation.

> Tip: For more information about querying an index, and details about simple and full syntax, see Query types and composition in Azure AI Search in the Azure AI Search documentation.

It's common in a search solution for users to want to refine query results by filtering and sorting based on field values. Azure AI Search supports both of these capabilities through the search query API.

Filtering results

You can apply filters to queries in two ways:

- By including filter criteria in a simple search expression.
- By providing an OData filter expression as a **$filter** parameter with a *full* syntax search expression.

You can apply a filter to any *filterable* field in the index.

For example, suppose you want to find documents containing the text *London* that have an **author** field value of *Reviewer*.

You can achieve this result by submitting the following simple search expression:

```
search=London+author='Reviewer'
queryType=Simple
```

Alternatively, you can use an OData filter in a **$filter** parameter with a **full Lucene search expression** like this:

```
search=London
$filter=author eq 'Reviewer'
```

```
queryType=Full
```

> Note: OData **$filter** expressions are case-sensitive!

Filtering with facets

Facets are a useful way to present users with filtering criteria based on field values in a result set. They work best when a field has a small number of discrete values that can be displayed as links or options in the user interface.

To use facets, you must specify **facetable** fields for which you want to retrieve the possible values in an initial query. For example, you could use the following parameters to return all of the possible values for the author field:

```
search=*
facet=author
```

The results from this query include a collection of discrete facet values that you can display in the user interface for the user to select. Then in a subsequent query, you can use the selected facet value to filter the results:

```
search=*
$filter=author eq 'selected-facet-value-here'
```

Sorting results

By default, results are sorted based on the relevancy score assigned by the query process, with the highest scoring matches listed first. However, you can override this sort order by including an OData **orderby** parameter that specifies one or more **sortable** fields and a **sort order (asc or desc)**.

For example, to sort the results so that the most recently modified documents are listed first, you could use the following parameter values:

```
search=*
$orderby=last_modified desc
```

> Tip: For more information about using filters, see Filters in Azure AI Search in the Azure AI Search documentation.

Persist extracted information in a knowledge store

While the index might be considered the primary output from an indexing process, the enriched data it contains might also be useful in other ways. For example:

- Since the index is essentially a collection of JSON objects, each representing an indexed record, it might be useful to *export the objects as JSON files for integration into a data orchestration process for* **extract, transform, and load (ETL) operations**.
- You may want to *normalize the index records into* **a relational schema of tables** for analysis and reporting.
- Having *extracted embedded images from documents during the indexing process*, you might want to save those images as files.

Azure AI Search supports these scenarios by enabling you to define a **knowledge** store in the skillset that encapsulates your enrichment pipeline. The knowledge store consists of *projections of the enriched data*, which can be *JSON objects, tables, or image files*. When an indexer runs the pipeline to create or update an index, the projections are generated and persisted in the knowledge store.

> Tip: To learn more about using a knowledge store, see Knowledge store in Azure AI Search in the Azure AI Search documentation.

Exercise - Create a knowledge mining solution

It's time to out what you've learned into practice!

In this exercise, you use Azure AI Search to extract and enrich information from documents into a searchable index and a knowledge store.

Create an knowledge mining solution

In this exercise, you use AI Search to index a set of documents maintained by Margie's Travel, a fictional travel agency. The indexing process involves using AI skills to extract key information to make them searchable, and generating a knowledge store containing data assets for further analysis.

Module assessment

1. Which component of an Azure AI Search solution is scheduled to extract and enrich data to populate an index? **Indexer**.
2. Which service supports built-in AI skills in Azure AI Search? **Azure AI Services**
3. Which kind of projection results in a relational data schema for extracted fields? **Table**

Summary

In this module, you've learned how Azure AI Search enables you to build **intelligent search and knowledge mining** solutions by **indexing and enriching data** from various sources. You *explored the indexing process, the use of AI skills for data enrichment, and how to persist enriched data in a knowledge store for further analysis and integration*.

With these skills, you're now equipped to design and implement solutions that unlock valuable insights from your data using Azure AI Search.

> Tip: To learn more about Azure AI Search, see the Azure AI Search documentation.