**Create an index**

Creating an index establishes the physical data structures on your search service. Once the index exists, [**load the index**](https://learn.microsoft.com/en-us/azure/search/search-what-is-data-import) as a separate task.

**Prerequisites**

* Write permissions as a [**Search Service Contributor**](https://learn.microsoft.com/en-us/azure/search/search-security-rbac) or an [admin API key](https://learn.microsoft.com/en-us/azure/search/search-security-api-keys) for key-based authentication.
* An understanding of the data you want to index. A search index is based on external content that you want to make searchable. Searchable content is stored as fields in an index. You should have a clear idea of which source fields you want to make searchable, retrievable, filterable, facetable, and sortable on Azure AI Search. See the [schema checklist](https://learn.microsoft.com/en-us/azure/search/search-how-to-create-search-index?tabs=portal#schema-checklist) for guidance.
* You must also have a unique field in source data that can be used as the [document key (or ID)](https://learn.microsoft.com/en-us/azure/search/search-how-to-create-search-index?tabs=portal#document-keys) in the index.
* A stable index location. Moving an existing index to a different search service isn't supported out-of-the-box. Revisit application requirements and make sure that your existing search service (capacity and region), are sufficient for your needs. If you're taking a dependency on Azure AI services or Azure OpenAI, [choose a region](https://learn.microsoft.com/en-us/azure/search/search-create-service-portal#checklist-for-choosing-a-region) that provides all of the necessary resources.
* Finally, all service tiers have [index limits](https://learn.microsoft.com/en-us/azure/search/search-limits-quotas-capacity#index-limits) on the number of objects that you can create. For example, if you're experimenting on the Free tier, you can only have three indexes at any given time. Within the index itself, there are [limits on vectors](https://learn.microsoft.com/en-us/azure/search/search-limits-quotas-capacity#vector-index-size-limits) and [index limits](https://learn.microsoft.com/en-us/azure/search/search-limits-quotas-capacity#index-limits) on the number of simple and complex fields.

**Document keys**

Search index creation has two requirements: an index must have a unique name on the search service, and it must have a document key. The boolean key attribute on a field can be set to true to indicate which field provides the document key.

A document key is the unique identifier of a search document, and a search document is a collection of fields that completely describes something. For example, if you're indexing a [movies data set](https://www.kaggle.com/datasets/harshitshankhdhar/imdb-dataset-of-top-1000-movies-and-tv-shows), a search document contains the title, genre, and duration of a single movie. Movie names are unique in this dataset, so you might use the movie name as the document key.

In Azure AI Search, a document key is a string, and it must originate from unique values in the data source that's providing the content to be indexed. As a general rule, a search service doesn't generate key values, but in some scenarios (such as the [Azure table indexer](https://learn.microsoft.com/en-us/azure/search/search-howto-indexing-azure-tables)) it synthesizes existing values to create a unique key for the documents being indexed. Another scenario is one-to-many indexing for chunked or partitioned data, in which case document keys are generated for each chunk.

During incremental indexing, where new and updated content is indexed, incoming documents with new keys are added, while incoming documents with existing keys are either merged or overwritten, depending on whether index fields are null or populated.

Important points about document keys include:

* The maximum length of values in a key field is 1,024 characters.
* Exactly one top-level field in each index must be chosen as the key field and it must be of type Edm.String.
* The default of the key attribute is false for simple fields and null for complex fields.

Key fields can be used to look up documents directly and update or delete specific documents. The values of key fields are handled in a case-sensitive manner when looking up or indexing documents. See [GET Document (REST)](https://learn.microsoft.com/en-us/rest/api/searchservice/documents/get) and [Index Documents (REST)](https://learn.microsoft.com/en-us/rest/api/searchservice/documents) for details.

**Schema checklist**

Use this checklist to assist the design decisions for your search index.

1. Review [naming conventions](https://learn.microsoft.com/en-us/rest/api/searchservice/naming-rules) so that index and field names conform to the naming rules.
2. Review [supported data types](https://learn.microsoft.com/en-us/rest/api/searchservice/supported-data-types). The data type affects how the field is used. For example, numeric content is filterable but not full text searchable. The most common data type is Edm.String for searchable text, which is tokenized and queried using the full text search engine. The most common data type for a vector field is Edm.Single but you can use other types as well.
3. Identify a [document key](https://learn.microsoft.com/en-us/azure/search/search-how-to-create-search-index?tabs=portal#document-keys). A document key is an index requirement. It's a single string field populated from a source data field that contains unique values. For example, if you're indexing from Blob Storage, the metadata storage path is often used as the document key because it uniquely identifies each blob in the container.
4. Identify the fields in your data source that contribute searchable content in the index.

Searchable nonvector content includes short or long strings that are queried using the full text search engine. If the content is verbose (small phrases or bigger chunks), experiment with different analyzers to see how the text is tokenized.

Searchable vector content can be images or text (in any language) that exists as a mathematical representation. You can use narrow data types or vector compression to make vector fields smaller.

[Attributes set on fields](https://learn.microsoft.com/en-us/azure/search/search-what-is-an-index#index-attributes), such as retrievable or filterable, determine both search behaviors and the physical representation of your index on the search service. Determining how fields should be attributed is an iterative process for many developers. To speed up iterations, start with sample data so that you can drop and rebuild easily.

1. Identify which source fields can be used as filters. Numeric content and short text fields, particularly those with repeating values, are good choices. When working with filters, remember:
   * Filters can be used in vector and nonvector queries, but the filter itself is applied to alphanumeric (nonvector) fields in your index.
   * Filterable fields can optionally be used in faceted navigation.
   * Filterable fields are returned in arbitrary order and don't undergo relevance scoring, so consider making them sortable as well.
2. For vector fields, specify a vector search configuration and the algorithms used for creating navigation paths and filling the embedding space. For more information, see [Add vector fields](https://learn.microsoft.com/en-us/azure/search/vector-search-how-to-create-index).

Vector fields have extra properties that nonvector fields don't have, such as which algorithms to use and vector compression.

Vector fields omit attributes that aren't useful on vector data, such as sorting, filtering, and faceting.

1. For nonvector fields, determine whether to use the default analyzer ("analyzer": null) or a different analyzer. [Analyzers](https://learn.microsoft.com/en-us/azure/search/search-analyzers) are used to tokenize text fields during indexing and query execution.

For multi-lingual strings, consider a [language analyzer](https://learn.microsoft.com/en-us/azure/search/index-add-language-analyzers).

For hyphenated strings or special characters, consider [specialized analyzers](https://learn.microsoft.com/en-us/azure/search/index-add-custom-analyzers#built-in-analyzers). One example is [keyword](https://lucene.apache.org/core/6_6_1/analyzers-common/org/apache/lucene/analysis/core/KeywordAnalyzer.html) that treats the entire contents of a field as a single token. This behavior is useful for data like zip codes, IDs, and some product names. For more information, see [Partial term search and patterns with special characters](https://learn.microsoft.com/en-us/azure/search/search-query-partial-matching).

**Note :** Full text search is conducted over terms that are tokenized during indexing. If your queries fail to return the results you expect, [**test for tokenization**](https://learn.microsoft.com/en-us/rest/api/searchservice/indexes/analyze) to verify the string you're searching for actually exists. You can try different analyzers on strings to see how tokens are produced for various analyzers.

**Configure field definitions**

The fields collection defines the structure of a search document. All fields have a name, data type, and attributes.

Setting a field as searchable, filterable, sortable, or facetable has an effect on index size and query performance. Don't set those attributes on fields that aren't meant to be referenced in query expressions.

If a field isn't set to be searchable, filterable, sortable, or facetable, the field can't be referenced in any query expression. This is desirable for fields that aren't used in queries, but are needed in search results.

The REST APIs have default attribution based on [data types](https://learn.microsoft.com/en-us/rest/api/searchservice/supported-data-types), which is also used by the [Import wizards](https://learn.microsoft.com/en-us/azure/search/search-import-data-portal) in the Azure portal. The Azure SDKs don't have defaults, but they have field subclasses that incorporate properties and behaviors, such as [SearchableField](https://learn.microsoft.com/en-us/dotnet/api/azure.search.documents.indexes.models.searchablefield) for strings and [SimpleField](https://learn.microsoft.com/en-us/dotnet/api/azure.search.documents.indexes.models.simplefield) for primitives.

Default field attributions for the REST APIs are summarized in the following table.

A screenshot of a white sheet

Description automatically generated

String fields can also be optionally associated with [analyzers](https://learn.microsoft.com/en-us/azure/search/search-analyzers) and [synonym maps](https://learn.microsoft.com/en-us/azure/search/search-synonyms). Fields of type Edm.String that are filterable, sortable, or facetable can be at most 32 kilobytes in length. This is because values of such fields are treated as a single search term, and the maximum length of a term in Azure AI Search is 32 kilobytes. If you need to store more text than this in a single string field, you should explicitly set filterable, sortable, and facetable to false in your index definition.

Vector fields must be associated with [dimensions and vector profiles](https://learn.microsoft.com/en-us/azure/search/vector-search-how-to-create-index). Retrievable default is true if you add the vector field using the [Import and vectorize wizard](https://learn.microsoft.com/en-us/azure/search/search-get-started-portal-import-vectors) in the Azure portal, otherwise it's false if you use the REST API.

Field attributes are described in the following table.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**Create an index**

When you're ready to create the index, use a search client that can send the request. You can use the Azure portal or REST APIs for early development and proof-of-concept testing, otherwise it's common to use the Azure SDKs.

During development, plan on frequent rebuilds. Because physical structures are created in the service, [dropping and re-creating indexes](https://learn.microsoft.com/en-us/azure/search/search-howto-reindex) is necessary for many modifications. You might consider working with a subset of your data to make rebuilds go faster.

**Using Azure Portal (Indexes Menu)**

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**Using Azure Portal (Import Data)**

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**Using Azure Portal (Storage Account)**

A screenshot of a computer

Description automatically generated

A computer screen shot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated