# Connecting DocumentDB with Azure Search using indexers

If you’re looking to implement great search experiences over your DocumentDB data, use Azure Search indexer for DocumentDB! In this article, we will show you how to integrate Azure DocumentDB with Azure Search without having to write any code to maintain indexing infrastructure!

To set this up, you have to [setup an Azure Search account](/documentation/articles/search-create-service-portal) (you don’t need to upgrade to standard search), and then call the [Azure Search REST API](https://msdn.microsoft.com/library/azure/dn798935.aspx) to create a DocumentDB **data source** and an **indexer** for that data source.

## Azure Search indexer concepts

Azure Search supports the creation and management of data sources (including DocumentDB) and indexers that operate against those data sources.

A **data source** specifies what data needs to be indexed, credentials to access the data, and policies to enable Azure Search to efficiently identify changes in the data (such as modified or deleted documents inside your collection). The data source is defined as an independent resource so that it can be used by multiple indexers.

An **indexer** describes how the data flows from your data source into a target search index. You should plan on creating one indexer for every target index and data source combination. While you can have multiple indexers writing into the same index, an indexer can only write into a single index. An indexer is used to:

* Perform a one-time copy of the data to populate an index.
* Sync an index with changes in the data source on a schedule. The schedule is part of the indexer definition.
* Invoke on-demand updates to an index as needed.

## Step 1: Create a data source

Issue a HTTP POST request to create a new data source in your Azure Search service, including the following request headers.

POST https://[Search service name].search.windows.net/datasources?api-version=[api-version]  
Content-Type: application/json  
api-key: [Search service admin key]

The api-version is required. Valid values include 2015-02-28 or a later version.

The body of the request contains the data source definition, which should include the following fields:

* **name**: The name of the data source.
* **type**: Use documentdb.
* **credentials**:
  + **connectionString**: Required. Specify the connection info to your Azure DocumentDB database in the following format: AccountEndpoint=<DocumentDB endpoint url>;AccountKey=<DocumentDB auth key>;Database=<DocumentDB database id>
* **container**:
  + **name**: Required. Specify the DocumentDB collection to be indexed.
  + **query**: Optional. You can specify a query to flatten an arbitrary JSON document into a flat schema that Azure Search can index.
* **dataChangeDetectionPolicy**: Optional. See [Data Change Detection Policy](#DataChangeDetectionPolicy) below.
* **dataDeletionDetectionPolicy**: Optional. See [Data Deletion Detection Policy](#DataDeletionDetectionPolicy) below.

### Capturing changed documents

The purpose of a data change detection policy is to efficiently identify changed data items. Currently, the only supported policy is the High Water Mark policy using the \_ts last-modified timestamp property provided by DocumentDB - which is specified as follows:

{  
 "@odata.type" : "#Microsoft.Azure.Search.HighWaterMarkChangeDetectionPolicy",  
 "highWaterMarkColumnName" : "\_ts"  
}

You will also need to add \_ts in the projection and WHERE clause for your query. For example:

SELECT s.id, s.Title, s.Abstract, s.\_ts FROM Sessions s WHERE s.\_ts > @HighWaterMark

### Capturing deleted documents

When rows are deleted from the source table, you should delete those rows from the search index as well. The purpose of a data deletion detection policy is to efficiently identify deleted data items. Currently, the only supported policy is the Soft Delete policy (deletion is marked with a flag of some sort), which is specified as follows:

{  
 "@odata.type" : "#Microsoft.Azure.Search.SoftDeleteColumnDeletionDetectionPolicy",  
 "softDeleteColumnName" : "the property that specifies whether a document was deleted",  
 "softDeleteMarkerValue" : "the value that identifies a document as deleted"  
}

[AZURE.NOTE] You will need to include the property in your SELECT clause if you are using a custom projection.

### Request body example

The following example creates a data source with a custom query and policy hints:

{  
 "name": "mydocdbdatasource",  
 "type": "documentdb",  
 "credentials": {  
 "connectionString": "AccountEndpoint=https://myDocDbEndpoint.documents.azure.com;AccountKey=myDocDbAuthKey;Database=myDocDbDatabaseId"  
 },  
 "container": {  
 "name": "myDocDbCollectionId",  
 "query": "SELECT s.id, s.Title, s.Abstract, s.\_ts FROM Sessions s WHERE s.\_ts > @HighWaterMark"  
 },  
 "dataChangeDetectionPolicy": {  
 "@odata.type": "#Microsoft.Azure.Search.HighWaterMarkChangeDetectionPolicy",  
 "highWaterMarkColumnName": "\_ts"  
 },  
 "dataDeletionDetectionPolicy": {  
 "@odata.type": "#Microsoft.Azure.Search.SoftDeleteColumnDeletionDetectionPolicy",  
 "softDeleteColumnName": "isDeleted",  
 "softDeleteMarkerValue": "true"  
 }  
}

### Response

You will receive an HTTP 201 Created response if the data source was successfully created.

## Step 2: Create an index

Create a target Azure Search index if you don’t have one already. You can do this from the [Azure Portal UI](/documentation/articles/search-get-started#test-service-operations) or by using the [Create Index API](https://msdn.microsoft.com/library/azure/dn798941.aspx).

POST https://[Search service name].search.windows.net/indexes?api-version=[api-version]  
Content-Type: application/json  
api-key: [Search service admin key]

Ensure that the schema of your target index is compatible with the schema of the source JSON documents or the output of your custom query projection.

### Figure A: Mapping between JSON Data Types and Azure Search Data Types

|  |  |
| --- | --- |
| JSON DATA TYPE | COMPATIBLE TARGET INDEX FIELD TYPES |
| Bool | Edm.Boolean, Edm.String |
| Numbers that look like integers | Edm.Int32, Edm.Int64, Edm.String |
| Numbers that look like floating-points | Edm.Double, Edm.String |
| String | Edm.String |
| Arrays of primitive types e.g. “a”, “b”, “c” | Collection(Edm.String) |
| Strings that look like dates | Edm.DateTimeOffset, Edm.String |
| GeoJSON objects e.g. { “type”: “Point”, “coordinates”: [ long, lat ] } | Edm.GeographyPoint |
| Other JSON objects | N/A |

### Request body example

The following example creates an index with an id and description field:

{  
 "name": "mysearchindex",  
 "fields": [{  
 "name": "id",  
 "type": "Edm.String",  
 "key": true,  
 "searchable": false  
 }, {  
 "name": "description",  
 "type": "Edm.String",  
 "filterable": false,  
 "sortable": false,  
 "facetable": false,  
 "suggestions": true  
 }]  
 }

### Response

You will receive an HTTP 201 Created response if the index was successfully created.

## Step 3: Create an indexer

You can create a new indexer within an Azure Search service by using an HTTP POST request with the following headers.

POST https://[Search service name].search.windows.net/indexers?api-version=[api-version]  
Content-Type: application/json  
api-key: [Search service admin key]

The body of the request contains the indexer definition, which should include the following fields:

* **name**: Required. The name of the indexer.
* **dataSourceName**: Required. The name of an existing data source.
* **targetIndexName**: Required. The name of an existing index.
* **schedule**: Optional. See [Indexing Schedule](#IndexingSchedule) below.

### Running indexers on a schedule

An indexer can optionally specify a schedule. If a schedule is present, the indexer will run periodically as per schedule. Schedule has the following attributes:

* **interval**: Required. A duration value that specifies an interval or period for indexer runs. The smallest allowed interval is 5 minutes; the longest is one day. It must be formatted as an XSD “dayTimeDuration” value (a restricted subset of an [ISO 8601 duration](http://www.w3.org/TR/xmlschema11-2/#dayTimeDuration) value). The pattern for this is: P(nD)(T(nH)(nM)). Examples: PT15M for every 15 minutes, PT2H for every 2 hours.
* **startTime**: Required. An UTC datetime that specifies when the indexer should start running.

### Request body example

The following example creates an indexer that copies data from the collection referenced by the myDocDbDataSource data source to the mySearchIndex index on a schedule that starts on Jan 1, 2015 UTC and runs hourly.

{  
 "name" : "mysearchindexer",  
 "dataSourceName" : "mydocdbdatasource",  
 "targetIndexName" : "mysearchindex",  
 "schedule" : { "interval" : "PT1H", "startTime" : "2015-01-01T00:00:00Z" }  
}

### Response

You will receive an HTTP 201 Created response if the indexer was successfully created.

## Step 4: Run an indexer

In addition to running periodically on a schedule, an indexer can also be invoked on demand by issuing the following HTTP POST request:

POST https://[Search service name].search.windows.net/indexers/[indexer name]/run?api-version=[api-version]  
api-key: [Search service admin key]

### Response

You will receive an HTTP 202 Accepted response if the indexer was successfully invoked.

## Step 5: Get indexer status

You can issue a HTTP GET request to retrieve the current status and execution history of an indexer:

GET https://[Search service name].search.windows.net/indexers/[indexer name]/status?api-version=[api-version]  
api-key: [Search service admin key]

### Response

You will see a HTTP 200 OK response returned along with a response body that contains information about overall indexer health status, the last indexer invocation, as well as the history of recent indexer invocations (if present).

The response should look similar to the following:

{  
 "status":"running",  
 "lastResult": {  
 "status":"success",  
 "errorMessage":null,  
 "startTime":"2014-11-26T03:37:18.853Z",  
 "endTime":"2014-11-26T03:37:19.012Z",  
 "errors":[],  
 "itemsProcessed":11,  
 "itemsFailed":0,  
 "initialTrackingState":null,  
 "finalTrackingState":null  
 },  
 "executionHistory":[ {  
 "status":"success",  
 "errorMessage":null,  
 "startTime":"2014-11-26T03:37:18.853Z",  
 "endTime":"2014-11-26T03:37:19.012Z",  
 "errors":[],  
 "itemsProcessed":11,  
 "itemsFailed":0,  
 "initialTrackingState":null,  
 "finalTrackingState":null  
 }]  
}

Execution history contains up to the 50 most recent completed executions, which are sorted in reverse chronological order (so the latest execution comes first in the response).

## Next steps

Congratulations! You have just learned how to integrate Azure DocumentDB with Azure Search using the indexer for DocumentDB.

* To learn how more about Azure DocumentDB, see the [DocumentDB service page](/services/documentdb/).
* To learn how more about Azure Search, see the [Search service page](/services/search/).