MONGO ATLAS SEARCH

What is Mongo Atlas Search?

Atlas Search is a powerful search engine built into MongoDB Atlas that enables text-based searching of your data. Internally, it uses the Apache Lucene full-text search engine to provide efficient and fast searching. Atlas Search makes it easy to perform complex searches against large data sets without the need for an external search engine..

Why Do we Need it?

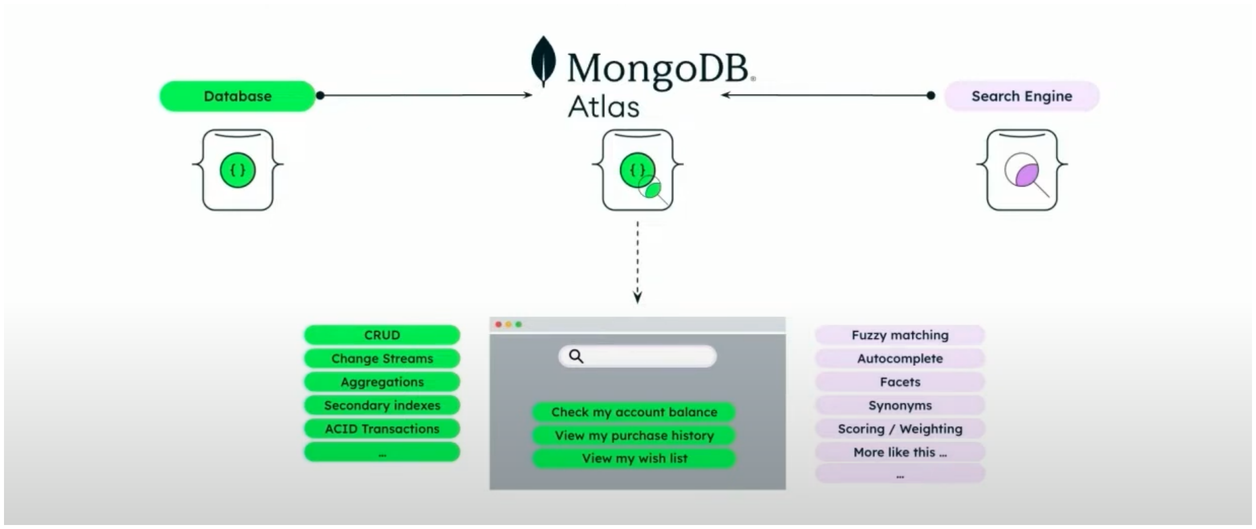
Before Atlas search, When we use transactional database with elastic search then that model have some disadvantages like.

- Diff API's Maintaining
- Sync Error
- Too much Infrastructure

Difference Between Atlas Search and Elastic Search

Atlas Search	**Elastic Search**
Flexibility to store various data types	Elasticsearch is not a good alternative for data storage compared to MongoDB.
It supports document-based queries.	Ease of use as it offers simple query syntax that is much easier to understand.
It perform searches quickly as it can analyze multiple records	It uses sharding when handling large datasets

Basic Infrastructure of How Atlas Work



Atlas Search Feature

- Facets
- Facets on sharded Clusters
- Query Analytics
- $lookup with $Search
- $unionWith with $search

What is Atlas Search Index?

An Atlas Search Index is a data structure that categorizes data in an easily searchable format. This is a mapping between terms and the documents that contain those terms. Atlas Search indexes allow you to find documents using specific identifiers more quickly. To query data in your Atlas cluster using Atlas Search, you need to configure an Atlas Search index.

How to Create Atlas Search Index?

You can create an Atlas search index for a single field or multiple fields. It's a good idea to index fields that you regularly use to sort or filter data so that documents with relevant data can be retrieved quickly when queried.

There are 4 Ways to create Atlas index

1- Using Atlas UI
2- Atlas search API
3- Atlas CLI

4- Programmarically

Steps to create Atlas search Index using Atlas CLI

Creating index in atlas cluster

1.Open Atlas Cluster
2.Click on Brose collections then click on Search
3. On top right corner click on Create Index.Here there are two way i.e using Visual Editor and JSON Editor. Here we use JSON Editor
Here,Default Index file look like this

```
{
"mappings": {
"dynamic": true
}
}
```

Here,

"mappings": This is the key that defines the mappings for the index.
"dynamic": This is set to true, which means that Atlas will dynamically detect and create field mappings as documents are indexed. If you set it to false, Atlas will not automatically create mappings for new fields, and any fields not explicitly defined in the mappings will be ignored.

4. Click on Json Editor and press next
5. Now , you can create index file

There are the example for different search type

Autocomplete

The **autocomplete** operator performs a search for a word or phrase that contains a sequence of characters from an incomplete input string

Syntax:-

Index File using edge gram

```
{

"mappings": {

"dynamic": true,

"fields": {

"label": {
```

```
"foldDiacritics": true,

"maxGrams": 15,

"minGrams": 2,

"tokenization": "edgeGram",//change this as per your need

"type": "autocomplete"

}

}

}

}
```

Syntax for autocomplete:-

```
$search: {
"index": "<index name>", // optional, defaults to "default"
"autocomplete": {
"query": "<search-string>",
"path": "<field-to-search>",
"tokenOrder": "any|sequential",
"fuzzy": <options>,
}
```

Field	Type	Description	Necessity
query	string or array of strings	String or strings to search for. If there are multiple terms in a string, Atlas Search also looks for a match for each term in the string separately.	**yes**

fuzzy	object	Enable fuzzy search. Find strings which are similar to the search term or terms.	**no**
fuzzy.maxEdits	integer	Maximum number of single-character edits required to match the specified search term. Value can be 1 or 2. **Note:-** It’s default value is 2	**no**
fuzzy.prefixLength	integer	Number of characters at the beginning of each term in the result that must exactly match. **Note:-** It’s default value is 0	**no**
fuzzy.maxExpansions	integer	Maximum number of variations to generate and search for. This limit applies on a per-token basis.	50
tokenOrder	string	Order in which to search for tokens. Value can be one of the following:- **Any:-** ndicates tokens in the query can appear in any order in the documents. Results contain documents where the tokens appear sequentially and non-sequentially. **Sequential:-** Indicates tokens in the query must appear adjacent to each other or in the order specified in the query	no

		in the documents. Results contain only documents where the tokens appear sequentially.	

Query

```
db.meal.aggregate([
   {$search:
      {autocomplete:
      {
         query: "men with", path: "title"
         fuzzy: {"maxEdits": 1, "prefixLength": 1, "maxExpansions": 256}}
      }
   },
   {$limit: 4},
   {$project: {_id: 0,title: 1}
   }])
```

Result-:

```
{ title: 'The Blood of a Poet' }
{ title: 'The Private Life of Henry VIII.' }
{ title: 'The Private Life of Don Juan' }
{ title: 'The Prisoner of Shark Island' }
{ title: 'The Prince and the Pauper' }
{ title: 'The Prisoner of Zenda' }
{ title: 'Dance Program' }
{ title: 'The Pied Piper' }
{ title: 'Prelude to War' }
```

Synonym

Synonyms allow you to index and search your collection for words that have the same or nearly the same meaning

Syntax:-

Index file

```
{
"name": "<index-name>",
"analyzer": "<analyzer-for-index>",
"searchAnalyzer": "<analyzer-for-query>",
"mappings": {
"dynamic": <boolean>,
"fields": { <field-definition> }
},
"synonyms": [
{
"name": "<synonym-mapping-name>",
"source": {
"collection": "<source-collection-name>"
},
"analyzer": "<synonym-mapping-analyzer>"
}
]
}
```

Field	Type	Description	Necessity
analyzer	string	Name of the analyzer to use with this synonym mapping.	Required
name	string	Name of the synonym mapping. Name must be unique in the index definition.	Required

		Value can't be an empty string.	
source	document	Source collection for synonyms. The source option takes the collection field.	Required
source.collection	string	Name of the MongoDB collection that is in the same database as the Atlas Search index	Required

Note:-Synonym Collection Should be in a proper format:- to learn more view mappping link

Index file for static mapping

```
{
"mappings": {
"dynamic": false,
"fields": {
"title": {
"analyzer": "lucene.english",
"type": "string"
}
}
},
"synonyms": [
{
"analyzer": "lucene.english",
"name": "transportSynonyms",
"source": {
"collection": "transport_synonyms"
}
}
]
}
```

Query:-

```
db.automobile.aggregate([
  {
    $search: {
      index: "synonyms-tutorial",
      text: {
        path: "title",
        query: "automobile",
        synonyms: "transportSynonyms",
      },
    },
  },
  {
    $limit: 10,
  },
  {
    $project: {
      _id: 0,
      title: 1,
      score: { $meta: "searchScore" },
    },
  },
])
```

Result:-

```
{ title: 'Cars', score: 4.197734832763672 }
{ title: 'Planes, Trains & Automobiles', score: 3.8511905670166016 }
{ title: 'Car Wash', score: 3.39473032951355 }
{ title: 'Used Cars', score: 3.39473032951355 }
{ title: 'Blue Car', score: 3.39473032951355 }
{ title: 'Cars 2', score: 3.39473032951355 }
{ title: 'Stealing Cars', score: 3.39473032951355 }
{ title: 'Cop Car', score: 3.39473032951355 }
{ title: 'The Cars That Eat People', score: 2.8496146202087402 }
{ title: 'Khrustalyov, My Car!', score: 2.8496146202087402 }
```

GeoWithIn

The geoWithin operator supports querying geographic points within a given geometry. Only points are returned
You can query points within a:

- Circle
- Bounding box
- Polygon

Note:- When specifying the coordinates to search, longitude must be specified first and then the latitude. Longitude values can be between -180 and 180, both inclusive. Latitude values can be between -90 and 90, both inclusive. Coordinate values can be integers or doubles.

Syntax

Index for GeoWithIn and it work for all the shapes

```
{
"mappings": {
"fields": {
"address": {
"fields": {
"location": {
"type": "geo"
}
},
"type": "document"
}}}
```

Query Syntax

```
{
 "$search": {
   "index": <index name>, // optional, defaults to "default"
   "geoWithin": {
      "path": "<field-to-search>",
      "box | circle | geometry": <object>,
      "score": <score-options>
    }
  }
}
```

Field	Type	Description	Necessity
Box	Object	Object that specifies the bottom left and top right GeoJSON points of a box to search within	Conditional
Circle	object	Object that specifies the center point and the radius in meters to search within It Contain following attribute center - Center of the circle specified as a GeoJSON point. radius - Radius, which is a number, specified in meters. Value must be greater than or equal to 0.	Conditional
geometry	GeoJSON object	GeoJSON object that specifies the MultiPolygon or Polygon to search within. The polygon must be specified as a closed loop where the last position is the same as the first position.	Conditional
path	String or array of string	Indexed geo type field or fields to	Required

		search. See Path Construction.	

Query

The following query uses the geoWithin operator with the circle field to search for properties within one mile radius of specified coordinates

```
db.listingsAndReviews.aggregate([
  "$search": {
    "geoWithin": {
      "circle": {
        "center": {
          "type": "Point",
          "coordinates": [-73.54, 45.54]
        },
        "radius": 1600
      },
      "path": "address.location"
    }
  }
},
{
  $limit: 3
},
{
  $project: {
    "_id": 0,
    "name": 1,
    "address": 1
  }
}])
```

Result

```
{
"name" : "Ligne verte - à 15 min de métro du centre ville.",
"address" : {
"street" : "Montréal, Québec, Canada",
"suburb" : "Hochelaga-Maisonneuve",
"government_area" : "Mercier-Hochelaga-Maisonneuve",
"market" : "Montreal",
"country" : "Canada",
"country_code" : "CA",
"location" : {
"type" : "Point",
"coordinates" : [ -73.54949, 45.54548 ],
"is_location_exact" : false
}
}
}
{
"name" : "Belle chambre à côté Metro Papineau",
"address" : {
"street" : "Montréal, QC, Canada",
"suburb" : "Gay Village",
"government_area" : "Ville-Marie",
"market" : "Montreal",
"country" : "Canada",
"country_code" : "CA",
"location" : {
"type" : "Point",
"coordinates" : [ -73.54985, 45.52797 ],
"is_location_exact" : false
}
}
}
```

Phrase

The phrase operator performs search for documents containing an ordered sequence of terms using the analyzer specified in the index configuration. If no analyzer is specified, the default standard analyzer is used.

Syntax:-

Index file

```
{
"mappings": {
"fields": {
"label": {
"analyzer": "lucene.standard",
"type": "string"
}
}
}
}
```

Query Syntax

```
{
$search: {
"index": <index name>, // optional, defaults to "default"
"phrase": {
"query": "<search-string>",
"path": "<field-to-search>",
"score": <options>,
"slop": <distance-number>
}
}
}
```

Field	Type	Description	Necessity
Query	String or array of string	String or strings to search for.	yes
path	string or array of strings	Indexed field or fields to search.	yes
slop	integer	Allowable distance between words in the query phrase	no

Query

The following Atlas Search example performs a search of the title field for the query string men women. The slop value of 5 in the query allows greater movement of the words and distance between the words men and women.

```
db.movies.aggregate([
  "$search": {
    "phrase": {
      "path": "title",
      "query": "men women" //["the man", "the moon"]
      "slop": 5
    }
  }
},
{
  $project: {
    "_id": 0,
    "title": 1,
    score: { $meta: "searchScore" }
  }
}
])
```

Result

```
{ "title" : "Men Without Women", "score" : 3.39743709564209 }

{ "title" : "Men Vs Women", "score" : 3.39743709564209 }
{ "title" : "Good Men, Good Women", "score" : 2.878715753555298 }

{ "title" : "The War Between Men and Women", "score" : 2.205303192138672 }

{ "title" : "Women Without Men", "score" : 1.983487844467163 }

{ "title" : "Women Vs Men", "score" : 1.983487844467163 }
```

Diacritic-Insensitive(Case Insensitive) Search

A diacritic-insensitive atlas search refers to a search functionality that treats diacritics as equivalent to their base characters when looking up information in an atlas database.

Example:- "Cafe" and "Café" would be considered the same term.

Syntax:-

Index file

The index definition specifies a string type for the genres and title fields. It also applies the custom analyzer named diacriticFolder on the title field.

```
{
"mappings": {
  "fields": {
    "genres": {
      "type": "string"
    },
    "title": {
      "analyzer": "diacriticFolder",
      "type": "string"
    }
  }
},
"analyzers": [{
  "charFilters": [],
  "name": "diacriticFolder",
  "tokenizer": {
    "type": "keyword"
  },
  "tokenFilters": [{
    "type": "icuFolding"
  }]
}]
}
```

Query

```
db.meal.aggregate([{
        '$search': {
          'index': 'diacritic-insensitive-tutorial',
          'compound': {
                'must': [{
                    'wildcard': {
                        'query': "alle*",
                        'path': "title",
                        'allowAnalyzedField': true
                    }
                }],
                'should': [{'text': {'query': 'Drama', 'path': 'genres'}}]
            }}},
        { '$project': { '_id': 0, 'title': 1 , 'genres': 1, 'score': {'$meta': 'searchScore'}}}])
```

Result:-

```
{
genres: [ 'Drama', 'Family', 'Sport' ],
title: 'Alley Cats Strike',
score: 1.2084882259368896
}
{
genres: [ 'Drama', 'Romance', 'Sci-Fi' ],
title: 'Allegro',
score: 1.179288625717163
}
{
genres: [ 'Animation', 'Comedy', 'Fantasy' ],
title: 'Allegro non troppo',
score: 1
}
{
genres: [ 'Comedy' ],
title: 'Allez, Eddy!',
score: 1
}
```

In Operator

The in operator performs a search for an array of number, date, boolean, or objectId values at the given path and returns documents where the value of the field equals any value in the specified array.

Syntax:-

The in operator has the following syntax:

```
{
 $search: {
   "index": <index name>, // optional, defaults to "default"
   "in": {
     "path": "<field-to-search>",
     "score": <options>,
     "value": <single-or-array-of-values-to-search>
   }
 }
}
```

Field	Type	Description	Necessity
value	boolean, objectId, number, or date	strings to array.	yes
Path	string	Indexed field to search. You can also specify a wildcard path to search. See path construction for more information	Yes
score	object	Score to assign to matching search term results. Use one of the following options to modify the score: **boost:** multiply the result score by the given number. **constant:** replace the result score with the given number. **function:** replace the result score using the function expression.	Optional

Index file

```
{
"mappings": {
"dynamic": true
}
}
```

Query

The following query uses the in operator to search the birthdate field, which contains a single value, for customers who were born on given dates.

```
db.customers.aggregate([{
"$search": {
"in": {
"path": "birthdate",
"value": [ISODate("1977-03-02T02:20:31.000+00:00"),
ISODate("1977-03-01T00:00:00.000+00:00"),
ISODate("1977-05-06T21:57:35.000+00:00")]
}
}
},
{
"$project": {
"_id": 0,
"name": 1,
"birthdate": 1
}
}
])
```

Result

```
[{
name: 'Elizabeth Ray',
birthdate: ISODate("1977-03-02T02:20:31.000Z")
},
{
name: 'Brad Cardenas',
birthdate: ISODate("1977-05-06T21:57:35.000Z")
}
]
```

Define index for different Field

Fields	**Indexing**
Array	{ "mappings": { "dynamic": true|false, "fields": { "<array-field-name>": { "type": "<array-element-data-type>" } } } }
Autocomplete	{ "mappings": { "dynamic": true|false, "fields": { "<field-name>": { "type": "autocomplete", "analyzer": "<lucene-analyzer>", "tokenization": "edgeGram|rightEdgeGram|nGram", "minGrams": <2>, "maxGrams": <15>, "foldDiacritics": true|false } } } }
boolean	{ "mappings": { "dynamic": false, "fields": { "<field-name>": { "type": "boolean" } } } }

Date type	{
	"mappings": {
	"dynamic": true|false,
	"fields": {
	"<field-name>": {
	"type": "date"
	}
	}
	}
	}

Datefacet	{
	"mappings": {
	"dynamic": true|false,
	"fields": {
	"<field-name>": {
	"type": "dateFacet"
	}
	}
	}
	}

Documenttype	{
	"mappings": {
	"dynamic": true|false,
	"fields": {
	"<field-name>": {
	"type": "document",
	"dynamic": true|false,
	"fields": {
	"<field-name>": {
	<field-mapping-definition>
	}
	}
	}
	}
	}
	}

EmbeddedDocuments	{ "mappings": { "dynamic": true|false, "fields": { "<field-name>": { "type": "embeddedDocuments", "dynamic": true|false, "fields": { "<field-name>": { <field-mapping-definition> } } } } } }
Geo	{ "mappings": { "dynamic": false, "fields": { "<field-name>": { "indexShapes": true|false, "type": "geo" } } } } //here in indexShape , true indicate shape and point and false indicate point
Number	{ "mappings": { "dynamic": true|false, "fields": { "<field-name>": { "type": "number", "representation": "int64|double",//for keep precision "indexIntegers": true|false,//indicates whether to index or omit indexing int32and int64type value "indexDoubles": true|false//indicates whether to index or omit indexing doubletype value } } } }

ObjectId	{
	"mappings": {
	"dynamic": true|false,
	"fields": {
	"<field-name>": {
	"type": "objectId"
	}
	}
	}
	}

String	{
	"mappings": {
	"dynamic": true|false,
	"fields": {
	"<field-name>": {
	"type": "string",
	"analyzer": "<atlas-search-analyzer>",
	"searchAnalyzer": "<atlas-search-analyzer>",
	"indexOptions": "docs|freqs|positions|offsets",
	"store": true|false,
	"ignoreAbove": <integer>,
	"multi": {<string-field-definition>},
	"norms": "include|omit"
	}
	}
	}
	}

Token	{
	"mappings": {
	"dynamic": true|false,
	"fields": {
	"<field-name>": {
	"type": "token",
	"normalizer": "lowercase | none"
	}
	}
	}
	}

Analyzers

Analyzers in Atlas Search are policies that combine a tokenizer, which extracts tokens from text, with filters that you define. They are used to create indexable terms that correct for differences in punctuation, capitalization, filler words, and more1.

There are two type of Analyzers:-
1-Predefined anallyzers
2-Custom analyzers

Predefined Analyzer

There are basically 5 type of analyser that are use in Mongo Atlas search.

Analyzer	Description
Standard	Uses the default analyzer for all Atlas Search indexes and queries.
Simple	Divides text into searchable terms wherever it finds a non-letter character
Whitespace	Divides text into searchable terms wherever it finds a whitespace character
Language	Provides a set of language-specific text analyzers.
Keyword	Indexes text fields as single terms

Standards Analyzer

1. The standardanalyzer is the default for all Atlas Search indexes and queries.
2. It divides text into terms based on word boundaries, which makes it language-neutral for most use cases.
3. It converts all terms to lower case and removes punctuation.
4. It provides grammar-based tokenization that recognizes email addresses, acronyms, Chinese-Japanese-Korean characters, alphanumerics, and more.

Index file

```
{
"mappings": {
"fields": {
"title": {
"type": "string",
"analyzer": "lucene.standard"
}
}
}
}
```

Query

The following query searches the title field for the term action and limits the output to two results.

```
db.movies.aggregate([
{
"$search": {
"text": {
"query": "action",
"path": "title"
}
}
},
{
"$limit": 2
},
{
"$project": {
"_id": 0,
"title": 1
}
}
])
```

Result

```
[
{
title: 'Action Jackson'
},
{
title: 'Class Action'
}
]
```

Note:-

For above query, Here is how other analyser create Searchable token.

Title	Standard Analyzer Tokens	Keyword Analyzer Tokens	Whitespace Analyzer Tokens
Action Jackson	action, jackson	Action Jackson	Action, Jackson
Class Action	class, action	Class Action	Class, Action

Simple Analyser

The simple analyzer divides text into searchable terms (tokens) wherever it finds a non-letter character, such as whitespace, punctuation, or one or more digits. It converts all text to lower case.

Index File

```
{
 "mappings": {
   "fields": {
     "title": {
       "type": "string",
       "analyzer": "lucene.simple"
     }
   }
 }
}
```

Query

The following query searches for the term lion in the title field and limits the output to five results

```
db.movies.aggregate([
  "$search": {
    "text": {
        "query": "lion",
        "path": "title"
    }
  }
 },
 {
  "$limit": 5
 },
 {
  "$project": {
    "_id": 0,
    "title": 1
  }
 }
])
```

Result

```
[
{ title: 'White Lion' },
{ title: 'The Lion King' },
{ title: 'The Lion King 1 1/2' },
{ title: 'The Lion King 1 1/2' },
{ title: 'Lion's Den' },
]
```

For above query, Here is how other analyser create tokens

Title	**Simple Analyzer Tokens**	**Standard Analyzer Tokens**	**Whitespace Analyzer Tokens**
White Lion	white, lion	white, lion	White, Lion
The Lion King	the, lion, king	the, lion, king	The, Lion, King
The Lion King 1 1/2	the, lion, king	the, lion, king, 1, 1, 2	The, Lion, King, 1, 1/2
Lion's Den	lion, s, den	lion's, den	Lion's, Den

Whitespace Analyser

The whitespace analyzer divides text into searchable terms (tokens) wherever it finds a whitespace character. It leaves all text in its original letter case

Index File

```
{
"mappings": {
"fields": {
"title": {
"type": "string",
"analyzer": "lucene.whitespace",
"searchAnalyzer": "lucene.whitespace"
}
}
}
}
```

Query

The following query searches for the term Lion's in the title field.

```
db.movies.aggregate([
{
"$search": {
"text": {
"query": "Lion's",
"path": "title"
}
}
},
{
"$project": {
"_id": 0,
"title": 1
}
}
])
```

Result

```
[
{ title: 'Lion's Den' },
{ title: 'The Lion's Mouth Opens' }
]
```

For above query, Here is how other analyser create tokens

Title	Whitespace Analyzer Tokens	Simple Analyzer Tokens	Keyword Analyzer Tokens
Lion's Den	Lion's, Den	lion, s, den	Lion's Den
The Lion's Mouth Opens	The, Lion's, Mouth, Opens	the, lion, s, mouth, opens	The Lion's Mouth Opens

Keyword Analyser

The keyword analyzer accepts a string or array of strings as a parameter and indexes them as a single term (token). Only exact matches on the field are returned. It leaves all text in its original letter case.

Index File

```
{
"mappings": {
"fields": {
"title": {
"type": "string",
"analyzer": "lucene.keyword"
}
}
}
}
```

Query

The following query searches for the phrase Class Action in the title field

```
db.movies.aggregate([
{
"$search": {
"text": {
"query": "Class Action",
"path": "title"
}
}
},
{
"$project": {
"_id": 0,
"title": 1
}
}
])
```

Result

```
[
{
title: 'Class Action'
}
]
```

For above query, Here is how other analyser create Token

Analyzer	**Output Tokens**	**Matches action**	**Matches Class Action**
Keyword Analyzer Tokens	Class Action	X	√
Standard Analyzer Tokens	class, action	√	√
Simple Analyzer Tokens	class, action	√	√

Language Analyzer

Use language-specific analyzers to create indexes for a particular language. Each language analyzer has built-in stop words and word divisions based on that language's usage patterns

Atlas Search offers the following language analyzers:

lucene.arabic	lucene.armenian	lucene.basque	lucene.bengali
lucene.brazilian	lucene.bulgarian	lucene.catalan	lucene.chinese
lucene.cjk 1	lucene.czech	lucene.danish	lucene.dutch
lucene.english	lucene.finnish	lucene.french	lucene.galician
lucene.german	lucene.greek	lucene.hindi	lucene.hungarian
lucene.indonesian	lucene.irish	lucene.italian	lucene.japanese
lucene.korean	lucene.kuromoji 2	lucene.latvian	lucene.lithuanian
lucene.morfologik 3	lucene.nori 4	lucene.norwegian	lucene.persian
lucene.polish	lucene.portuguese	lucene.romanian	lucene.russian
lucene.smartcn 5	lucene.sorani	lucene.spanish	lucene.swedish
lucene.thai	lucene.turkish	lucene.ukrainian	

Index file

```
{
"mappings": {
"fields": {
"subject": {
"fields": {
"fr": {
"analyzer": "lucene.french",
"type": "string"
}
},
"type": "document"
}
}
}
}
```

Query

The following query searches for the string carburant in the subject.fr field

```
db.cars.aggregate([
{
$search: {
"text": {
"query": "carburant",
"path": "subject.fr"
}
}
},
{
$project: {
"_id": 0,
"subject.fr": 1
}
}
])
```

Result

```
{ subject: { fr: "Le meilleur moment pour le faire c'est immédiatement après que vous aurez fait le plein de carburant." } }
```

The language analyzer will create the following token for the above result

meileu	moment	fair
est	imediat	aprè
fait	plein	carburant

Custom Analyzer

An Atlas Search analyzer prepares a set of documents to be indexed by performing a series of operations to transform, filter, and group sequences of characters. We can define a custom analyzer to suit your specific indexing needs

Custom Analyzer has the following syntax

```
"analyzers": [
{
"name": "<name>",
"charFilters": [ <list-of-character-filters> ],
"tokenizer": {
"type": "<tokenizer-type>"
},
"tokenFilters": [ <list-of-token-filters> ]
}
]
```

Attribute	Type	Description	Required?
name	string	Name of the custom analyzer. Names must be unique within an index	yes
tokenizer	object	Tokenizer to use to create tokens. See Usage for more information.	yes
tokenFilters	list of objects	Array containing zero or more token filters. See Usage for more information.	no

CharFilters

Character filters require a type field, and some take additional options as well.

There are 4 type of CharFilters

HTML

Syntax

```
"charFilters": [
{
"type": "<filter-type>",
"<additional-option>": <value>
}
]
```

The htmlStripcharacter filter strips out HTML constructs. Which means it’s filter out result that contain HTML tag.

Attributes

It has the following attributes:-

Name	Type	Description	Required?
type	string	Human-readable label that identifies this character filter type. Value must be htmlStrip.	yes
ignoredTags	array of strings	List that contains the HTML tags to exclude from filtering.	no

Query

The following query looks for occurrences of the string head in the text.en_US field of the minutes collection

```
db.minutes.aggregate([
"$search": {
"text": {
"query": "head",
"path": "text.en_US"
}
}
},
{
"$project": {
"_id": 1,
"text.en_US": 1
}
}
])
```

Result

```
[
{
_id: 2,
text: { en_US: "The head of the sales department spoke first." }
},
{
_id: 3,
text: {
en_US: "<body>We'll head out to the conference room by noon.</body>"
}
}
]
```

Atlas Search doesn't return the document with <head> tag because the string headis part of the HTML tag <head>.The document with _id: 3 contains HTML tags, but the string head is elsewhere so the document is a match.

Mapping

The mappingcharacter filter applies user-specified normalization mappings to characters. It is based on Lucene's MappingCharFilter.

Attributes

It has the following attributes:

Name	Type	Description	Required?
type	string	Human-readable label that identifies this character filter type. Value must be mapping.	yes
mappings	object	Object that contains a comma-separated list of mappings. A mapping indicates that one character or group of characters should be substituted for another, in the format <original> : <replacement>.	yes

Index File

The custom analyzer specifies the following:

- Tokenize the entire input as a single token using the keyword tokenizer.
- Remove instances of hyphen (-), dot (.), open parenthesis ((), close parenthesis ( )), and space characters in the phone field using the mappingcharacter filter.

```
{"mappings": {
"fields": {
"page_updated_by": {
"fields": {
"phone": {
"analyzer": "mappingAnalyzer",
"type": "string"
}
},
"type": "document"
}
}
},
"analyzers": [
{
"name": "mappingAnalyzer",
"charFilters": [
{
"mappings": {
"-": "",
".": "",
"(": "",
")": "",
" ": ""
},
"type": "mapping"
}
],
"tokenizer": {
"type": "keyword"
}
}
]
```

Query

```
db.minutes.aggregate([{
"$search": {
"text": {
"query": "1234567890",
"path": "page_updated_by.phone"
}
}
},
{
"$project": {
"_id": 1,
"page_updated_by.phone": 1,
"page_updated_by.last_name": 1
}
}
])
```

Result

```
[
{
_id: 1,
page_updated_by: { last_name: 'AUERBACH', phone: '(123)-456-7890' }
}
]
```

Note:- Atlas will create tokens like this

Query Term	Output Tokens
(123)-456-7890	1234567890
123-456-7890	1234567890

123.456.7890	1234567890

Tokenizers

A custom analyzer's tokenizer determines how Atlas Search splits up text into discrete chunks for indexing. Tokenizers require a type field, and some take additional options as well.

Syntax

```
"tokenizer": {
"type": "<tokenizer-type>",
"<additional-option>": "<value>"
}
```

There are many different tokenizer some of them are as follow:-

edgeGram

The edgeGram tokenizer tokenizes input from the left side, or "edge", of a text input into n-grams of given sizes

Note:- you can't use a custom analyzer with edgeGram tokenizer in the analyzer field for synonym or autocomplete field mapping definitions.

Attributes

It has the following attributes:

Name	Type	Description	Required?
type	string	Human-readable label that identifies this tokenizer type. Value must be edgeGram.	yes
minGram	integer	Number of characters to include in the shortest token created.	yes

maxGram	integer	Number of characters to include in the longest token created.	yes

Index File

```
{
"mappings": {
"dynamic": true,
"fields": {
"message": {
"analyzer": "edgegramExample",
"type": "string"
}
}
},
"analyzers": [
{
"charFilters": [],
"name": "edgegramExample",
"tokenFilters": [],
"tokenizer": {
"maxGram": 7,
"minGram": 2,
"type": "edgeGram"
}
}
]
}
```

Suppose there is a text message ‘**try to sign-in**’
Then Atlas create tokens as

Tokenizer	Token Outputs

standard	try, to, sign, in
edgeGram	tr, try, try{SPACE}, try t, try to, try to{SPACE}

Keyword

The keyword tokenizer tokenizes the entire input as a single token.

Note:- Atlas Search doesn't index string fields that exceed 32766 characters using the keyword tokenizer.

Attributes

It has the following attributes:

Name	Type	Description	Required ?
type	string	Human-readable label that identifies this tokenizer type. Value must be keyword.	yes

Index File

```
{
"mappings": {
"dynamic": true,
"fields": {
"message": {
"analyzer": "keywordExample",
"type": "string"
}
}
},
"analyzers": [
{
"charFilters": [],
"name": "keywordExample",
"tokenFilters": [],
"tokenizer": {
"type": "keyword"
}
}
]
}
```

Suppose there is a text message '**try to sign-in**'
Then Atlas create tokens as

Tokenizer	**Token Outputs**
standard	try, to, sign, in
keyword	try to sign-in

nGram

The nGram tokenizer tokenizes into text chunks, or "n-grams", of given sizes. You can't use a custom analyzer with nGram tokenizer in the analyzer field for synonym or autocomplete field mapping definitions

Attributes

It has the following attributes:

Name	**Type**	**Description**	**Required**
type	string	Human-readable label that identifies this tokenizer type. Value must be nGram.	yes

minGram	integer	Number of characters to include in the shortest token created.	yes
maxGram	integer	Number of characters to include in the longest token created.	yes

Index File

```
{
"mappings": {
"dynamic": true,
"fields": {
"title": {
"analyzer": "ngramExample",
"type": "string"
}
}
},
"analyzers": [
{
"charFilters": [],
"name": "ngramExample",
"tokenFilters": [],
"tokenizer": {
"maxGram": 6,
"minGram": 4,
"type": "nGram"
}
}
]
}
```

For the sentence ‘The team's weekly meeting’ the atlas create following tokens:-

Tokenizer	Token Outputs
standard	The, team's, weekly, meeting
edgeGram	The{SPACE}, The t, The te
nGram	The{SPACE}, The t, The te, he t, ... , week, weekl, weekly, eekl, ..., eetin, eeting, etin, eting, ting

Standard

The standard tokenizer tokenizes based on word break.

Attributes

It has the following attributes:

Name	Type	Description	Required	Default
type	string	Human-readable label that identifies this tokenizer type. Value must be standard.	yes	
maxTokenLength	integer	Maximum length for a single token. Tokens greater than this length are split at maxTokenLength into multiple tokens.	no	255

Index file

```
{
 "mappings": {
   "dynamic": true,
   "fields": {
    "message": {
      "analyzer": "standardExample",
      "type": "string"
    }
  }
 },
 "analyzers": [
  {
    "charFilters": [],
    "name": "standardExample",
    "tokenFilters": [],
    "tokenizer": {
      "type": "standard"
    }
  }
 ]
}
```

For the string ‘**write down your signature or phone №**’, Atlas create Token as:-

Tokenizer	Token Outputs
standard	write, down, your, signature, or, phone
keyword	write down your signature or phone №

uaxUrlEmail

The uaxUrlEmail tokenizer tokenizes URLs and email addresses. Although uaxUrlEmail tokenizer tokenizes based on word break rules.

Note:- It is recommend using uaxUrlEmail tokenizer only when the indexed field value includes URLs and email addresses

Attributes

It has the following attributes:

Name	Type	Description	Required?	Default
type	string	Human-readable label that identifies this tokenizer type. Value must be uaxUrlEmail.	yes	
maxTokenLength	int	Maximum number of characters in one token.	no	255

Index file

```
{
 "mappings": {
   "fields": {
     "page_updated_by": {
       "fields": {
         "email": {
           "analyzer": "basicEmailAddressAnalyzer",
           "type": "string"
         }
       },
       "type": "document"
     }
   }
 },
 "analyzers": [
   {
     "name": "basicEmailAddressAnalyzer",
     "tokenizer": {
       "type": "uaxUrlEmail"
     }
   }
 ]
}
```

For the string ‘**write down your signature or phone №**’, Atlas create Token as:-

Tokenizer	Token Outputs
standard	lewinsky, example.com
uaxUrlEmail	lewinsky@example.com

whitespace

The whitespace tokenizer tokenizes based on occurrences of whitespace between words.

Attributes

It has the following attributes:

Name	Type	Description	Required?	Default
type	string	Human-readable label that identifies this tokenizer type. Value must be whitespace.	yes	
maxTokenLength	integer	Maximum length for a single token. Tokens greater than this length are split at maxTokenLength into multiple tokens.	no	255

Index File

```
{
 "mappings": {
   "dynamic": true,
   "fields": {
      "message": {
        "analyzer": "whitespaceExample",
        "type": "string"
      }
    }
  },
  "analyzers": [
    {
      "charFilters": [],
      "name": "whitespaceExample",
      "tokenFilters": [],
      "tokenizer": {
        "type": "whitespace"
      }
    }
  ]
}
```

For the string ‘do not forget to SIGN-IN’, Atlas create Token as:-

Tokenizer	Token Outputs
standard	do, not, forget, to, sign, in
whitespace	do, not, forget, to, SIGN-IN

Token Filtering

A token filter performs operations such as the following:

- Stemming, which reduces related words, such as "talking", "talked", and "talks" to their root word "talk".
- Redaction, the removal of sensitive information from public documents.

Token Filters always require a type field, and some take additional options as well. It has the following syntax:

```
"tokenFilters": [
{
"type": "<token-filter-type>",
"<additional-option>": <value>
}
]
```

A basic example of Token filtering using englishPossessive
The englishPossessivetoken filter removes possessives (trailing 's) from words. It has the following attributes:

Name	**Type**	**Description**	**Required?**	**Default**
type	string	Human-readable label that identifies this token filter type. Value must be asciiFolding.	yes	
originalTokens	string	String that specifies whether to include or omit the original tokens in the output of the token filter. Value can be one of the following: ● include - include the original tokens with the converted tokens in the output of the token filter. We recommend this value if you want to support queries on both the original tokens as well as the converted forms. ● omit - omit the original tokens and include only the converted tokens in the output of the token filter. Use this value if you want to query only on the converted forms of the original tokens.	no	omit

The custom analyzer specifies the following:

1. Apply the standard tokenizer to create tokens (search terms) based on word break rules.
2. Apply the englishPossessive token filter to remove possessives (trailing's) from the tokens.

Index file

```
{
 "mappings": {
   "fields": {
     "title": {
       "type": "string",
       "analyzer": "englishPossessiveStemmer"
     }
   }
 },
 "analyzers": [
   {
     "name": "englishPossessiveStemmer",
     "charFilters": [],
     "tokenizer": {
       "type": "standard"
     },
     "tokenFilters": [
       {
         "type": "englishPossessive"
       }
     ]
   }
 ]
}
```

Query

```
db.minutes.aggregate([
 {
  "$search": {
    "index": "default",
    "text": {
      "query": "team",
      "path": "title"
    }
  }
 },
 {
  "$project": {
    "_id": 1,
    "title": 1
  }
 }
])
```

Result

```
[
{
_id: 1,
title: 'The team's weekly meeting'
},
{
_id: 2,
title: 'The check-in with sales team'
}
]
```

For above query output token will look like

Document ID	Output Tokens
"_id": 1	The, team, weekly, meeting
"_id": 2	The, check, in, with, sales, team

Some other token Filters that are available also are.

Name	Description	Link
asciiFolding	The asciiFoldingtoken filter converts alphabetic, numeric, and symbolic unicode characters that are not in the Basic Latin Unicode block to their ASCIIequivalents, if available	Link
daitchMokotoffSoundex	The daitchMokotoffSoundextoken filter creates tokens for words that sound the same based on the Daitch-Mokotoff Soundex phonetic algorithm. This filter can generate multiple encodings for each input, where each encoded token is a 6 digit number.	Link
EdgeGram	The edgeGramtoken filter tokenizes input from the left side, or "edge", of a text input into n-grams of configured sizes.	Link
englishPossessive	The englishPossessivetoken filter removes possessives (trailing's) from words	Link
flattenGraph	The flattenGraphtoken filter transforms a token filter graph into a flat form suitable for indexing. If you use the wordDelimiterGraph token filter, use this filter after the wordDelimiterGraph token filter	Link
icuFolding	The icuFoldingtoken filter applies character folding from Unicode Technical Report such as accent removal, case folding, canonical duplicates folding, and many others detailed in the report	Link
Kstremming	The kStemmingtoken filter combines algorithmic stemming with a built-in dictionary for the english language to stem words. It expects lowercase text and doesn't modify uppercase text.	Link
Length	The lengthtoken filter removes tokens that are too short or too long	Link
Lowercase	The lowercasetoken filter normalizes token text to lowercase.	Link

nGram	The nGramtoken filter tokenizes input into n-grams of configured sizes. You can't use the nGram token filter in synonym or autocomplete mapping definitions	Link
PorterStreamming	TheporterStemmingtoken filter uses the porter stemming algorithm to remove the common morphological and inflectional suffixes from words in English. It expects lowercase text and doesn't work as expected for uppercase text	Link
regex	The regextoken filter applies a regular expression to each token, replacing matches with a specified string	Link
Reverse	The reversetoken filter reverses each string token	Link
Stopword	The stopwordtoken filter removes tokens that correspond to the specified stop words. This token filter doesn't analyze the specified stop words	Link