Repository for both **NEST** and **Elasticsearch.Net**, the two official [elasticsearch](https://github.com/elastic/elasticsearch) .NET clients.

**Compatibility Matrix**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Elasticsearch** | **.NET clients** | **Supported** | **Build Status** | **Myget Feed** | **Nuget Feed** |
| 0.x | 0.x | ❌ | ➖ | ➖ | ➖ |
| 1.x | 1.x | ✅ |  |  |  |
| 2.x | 2.x | ✅ |  |  |  |
| 5.x | 5.x | ✅ |  |  |  |

**Upgrading**

Please consult the [current upgrading Elasticsearch guidelines](https://www.elastic.co/guide/en/elasticsearch/reference/current/setup-upgrade.html) to understand what you should consider when upgrading from an older version of Elasticsearch to a newer one.

**Upgrading from 1.x to 2.x**

Take a look at the [blog post for details around the evolution of NEST 2.x](https://www.elastic.co/blog/ga-release-of-nest-2-0-our-dot-net-client-for-elasticsearch), in addition to the list of breaking changes for [NEST](https://github.com/elastic/elasticsearch-net/blob/master/docs/2.0-breaking-changes/nest-breaking-changes.md) and [Elasticsearch.Net](https://github.com/elastic/elasticsearch-net/blob/master/docs/2.0-breaking-changes/elasticsearch-net-breaking-changes.md).

**Upgrading from 2.x to 5.x**

Take a look at the [blog post for the release of NEST 5.x](https://www.elastic.co/blog/nest-5-0-released), in addition to the list of breaking changes for [NEST](https://github.com/elastic/elasticsearch-net/blob/master/docs/5.0-breaking-changes/nest-breaking-changes.md) and [Elasticsearch.Net](https://github.com/elastic/elasticsearch-net/blob/master/docs/5.0-breaking-changes/elasticsearch-net-breaking-changes.md).

[**NEST**](https://github.com/elasticsearch/elasticsearch-net/tree/master/src/Nest#nest-)

NEST is the official high-level .NET client of [Elasticsearch](https://github.com/elasticsearch/elasticsearch). It aims to be a solid, strongly typed client with a very concise API.

* High-level client that internally uses the low-level **Elasticsearch.Net** client
* Maps requests and responses to strongly typed objects with both fluent interface and object initializer syntaxes to build them
* Comes with a very powerful query DSL that maps one-to-one with Elasticsearch
* Takes advantage of .NET features where they make sense (i.e., covariant T collection result types, type and index inference)
* All calls have async variants with support for cancellation

**Getting Started**

For a comprehensive, walkthrough-styled tutorial, check out the [NuSearch example repository](https://github.com/elastic/elasticsearch-net-example).

**Installing**

From the package manager console:

PM> Install-Package NEST

or by simply searching for NEST in the package manager UI.

**Connecting**

You can connect to your Elasticsearch cluster via a single node, or by specifying multiple nodes using a connection pool. Using a connection pool has a few advantages over a single node connection, such as load balancing and cluster fail over support.

**Connecting to a single node**

var node = new Uri("http://myserver:9200");

var settings = new ConnectionSettings(node);

var client = new ElasticClient(settings);

**Using a connection pool**

var nodes = new Uri[]

{

new Uri("http://myserver1:9200"),

new Uri("http://myserver2:9200"),

new Uri("http://myserver3:9200")

};

var pool = new StaticConnectionPool(nodes);

var settings = new ConnectionSettings(pool);

var client = new ElasticClient(settings);

**Indexing**

Indexing a document is as simple as:

var tweet = new Tweet

{

Id = 1,

User = "kimchy",

PostDate = new DateTime(2009, 11, 15),

Message = "Trying out NEST, so far so good?"

};

var response = client.Index(tweet, idx => idx.Index("mytweetindex")); //or specify index via settings.DefaultIndex("mytweetindex");

All the calls have async variants:

var response = client.IndexAsync(tweet, idx => idx.Index("mytweetindex")); // returns a Task<IndexResponse>

**Getting a document**

var response = client.Get<Tweet>(1, idx => idx.Index("mytweetindex")); // returns an IGetResponse mapped 1-to-1 with the Elasticsearch JSON response

var tweet = response.Source; // the original document

**Searching for documents**

NEST exposes a fluent interface and a [powerful query DSL](https://www.elastic.co/guide/en/elasticsearch/client/net-api/current/query-dsl.html)

var response = client.Search<Tweet>(s => s

.From(0)

.Size(10)

.Query(q =>

q.Term(t => t.User, "kimchy")

|| q.Match(mq => mq.Field(f => f.User).Query("nest"))

)

);

As well as an object initializer syntax if lambdas aren't your thing:

var request = new SearchRequest

{

From = 0,

Size = 10,

Query = new TermQuery { Field = "user", Value = "kimchy" }

|| new MatchQuery { Field = "description", Query = "nest" }

};

var response = client.Search<Tweet>(request);

**Falling back to Elasticsearch.Net**

NEST also includes and exposes the low-level [Elasticsearch.Net](https://github.com/elasticsearch/elasticsearch-net/tree/master/src/Elasticsearch.Net) client that you can fall back to incase anything is missing:

//.LowLevel is of type IElasticLowLevelClient

var response = client.LowLevel.SearchPost("myindex","elasticsearchprojects", new

{

from = 0,

size = 10,

fields = new [] {"id", "name"},

query = new {

term = new {

name = new {

value= "NEST",

boost = 2.0

}

}

}

});

[**Read the full documentation here**](https://www.elastic.co/guide/en/elasticsearch/client/net-api/current/index.html)

[**Elasticsearch.Net**](https://github.com/elastic/elasticsearch-net/blob/master/src/Elasticsearch.Net)

A low-level, dependency free, client that has no opinions how you build and represent your requests and responses.

* Low-level client that provides a one-to-one mapping with the Elasticsearch REST API
* No dependencies
* Almost completely generated from the official REST API spec which makes it easy to keep up to date
* Comes with an integration test suite that can be generated from the YAML test definitions that the Elasticsearch core team uses to test their REST API
* Has no opinions on how you create or consume requests and responses
* Load balancing and cluster failover support
* All calls have async variants

**Installing**

From the package manager console:

PM> Install-Package Elasticsearch.Net

or by searching for Elastcsearch.Net in the package manager UI.

**Connecting**

Connecting using the low-level client is very similar to how you would connect using NEST. In fact, the connection constructs that NEST use are actually Elasticsearch.Net constructs. Thus, single node connections and connection pooling still apply when using Elasticsearch.Net.

var node = new Uri("http://myserver:9200");

var config = new ConnectionConfiguration(node);

var client = new ElasticLowLevelClient(config);

Note the main difference here is that we are instantiating an ElasticLowLevelClient rather than ElasticClient, and ConnectionConfiguration instead of ConnectionSettings.

**Calling an API endpoint**

Elasticsearch.Net is generated from the the [official REST specification](https://github.com/elastic/elasticsearch/tree/master/rest-api-spec), and thus maps to all Elasticsearch API endpoints.

client.GetSource("myindex","mytype","1",qs=>qs

.Routing("routingvalue")

);

will execute a GET to /myindex/mytype/1/\_source?routing=routingvalue. All the methods and arguments are fully documented based on the documentation of the specification.

As you can see, Elasticsearch.Net also strongly types the query string parameters that it knows exist on an endpoint with full Intellisense documentation. However, unknown query string parameters can still be added:

client.GetSource("myindex","mytype","1",qs=>qs

.Routing("routingvalue")

.Add("key","value")

);

The query string parameter is always optional.

**Providing a request body**

You can specify a request body directly with a string:

var myJson = @"{ ""hello"" : ""world"" }";

client.Index("myindex","mytype","1", myJson);

This will execute a POST to /myindex/mytype/1 with the provided string myJson passed verbatim as the request body.

Alternatively, you can specify an anonymous object:

var myJson = new { hello = "world" };

client.Index("myindex","mytype","1", myJson);

This will execute the same request, but this time myJson will be serialized by the registered ISerializer.

**Contributing**

[Pull requests](https://github.com/elastic/elasticsearch-net/pulls) and [issues](https://github.com/elastic/elasticsearch-net/issues) are very much welcomed and appreciated. If you'd like to report a bug or submit a feature/bug fix then please read our [contributing guide](https://github.com/elastic/elasticsearch-net/blob/master/contributing.md) first!

**Generating documentation from tests**

[All Elasticsearch.Net and NEST documentation on elastic.co](https://www.elastic.co/guide/en/elasticsearch/client/net-api/index.html) is generated from code within the [Tests project](https://github.com/elastic/elasticsearch-net/blob/master/src/Tests) using [Roslyn](https://github.com/dotnet/roslyn); multi-line comments serve as the main bodies of text, intermixed with code samples that test the documented components. The intention is to reduce the likelihood of documentation becoming outdated as the source changes.

Text within multi-line comments conforms to [asciidoc](http://asciidoc.org/), a lightweight markdown style text format well suited to technical documentation. To generate the asciidoc files from the test files, you need to run the [DocGenerator](https://github.com/elastic/elasticsearch-net/blob/master/src/CodeGeneration/DocGenerator) console application which will output the documentation files in the docs output directory. To verify that the generated asciidoc files can generate the documentation for the website, [clone the elastic docs repo](https://github.com/elastic/docs) and follow the instructions there for building documentation locally. as an example, suppose I have cloned the elastic docs to c:\source\elastic-docs, then to verify the generated asciidoc files for NEST are valid would be as following (using Cygwin on Windows)

cd /cygdrive/c/source/elastic-docs

./build\_docs.pl --doc /cygdrive/c/source/elasticsearch-net-master/docs/index.asciidoc

the result of running this for a successful build will be

Building HTML from /cygdrive/c/source/elasticsearch-net-master/docs/index.asciidoc

Done

See: /cygdrive/c/source/elasticsearch-docs/html\_docs/index.html

[Pull Requests](https://github.com/elastic/elasticsearch-net/pulls) are most welcome for areas of documentation that need improving.

**Blog posts**

Starting this section (2016) to list blogposts from our users that might be super helpful in your journey to learn Elasticsearch from a .NET perspective

* Read how buildclassifieds are using [ElasticSearch with ServiceStack (.NET)](http://buildclassifieds.com/2016/01/22/elasticsearch-and-servicestack/) in helping build a Classifieds Site.

**Many thanks to:**

* [Q42](https://q42.nl/) for supporting the development of NEST
* [redgate](http://www.red-gate.com/) for supplying @Mpdreamz with an ANTS Memory Profiler 8 & ANTS Performance Profiler 8 licenses
* [jetBrains](http://www.jetbrains.com/) for supplying @Mpdreamz with a dotTrace profiler and Resharper license
* [CodeBetter](http://codebetter.com/) for hosting the continuous integration for NEST
* Everyone who has been awesome enough to contribute back to NEST (You're listed automatically on the [documentation page](https://github.com/elastic/elasticsearch-net/graphs/contributors))

**Copyright and License**

This software is Copyright (c) 2014-2015 by Elasticsearch BV.

This is free software, licensed under: [The Apache License Version 2.0](https://github.com/elasticsearch/elasticsearch-net/blob/develop/license.txt).