Article

How To Integrate Elasticsearch In ASP.NET Core

By Pasang Tamang on Jan 10, 2023

Introduction

This article demonstrates how to integrate Elasticsearch in ASP.NET Core. This article also covers adding Elasticsearch middleware, configuring elastic search, and performing search operation with data stored in Elasticsearch index.

Pre-requisites

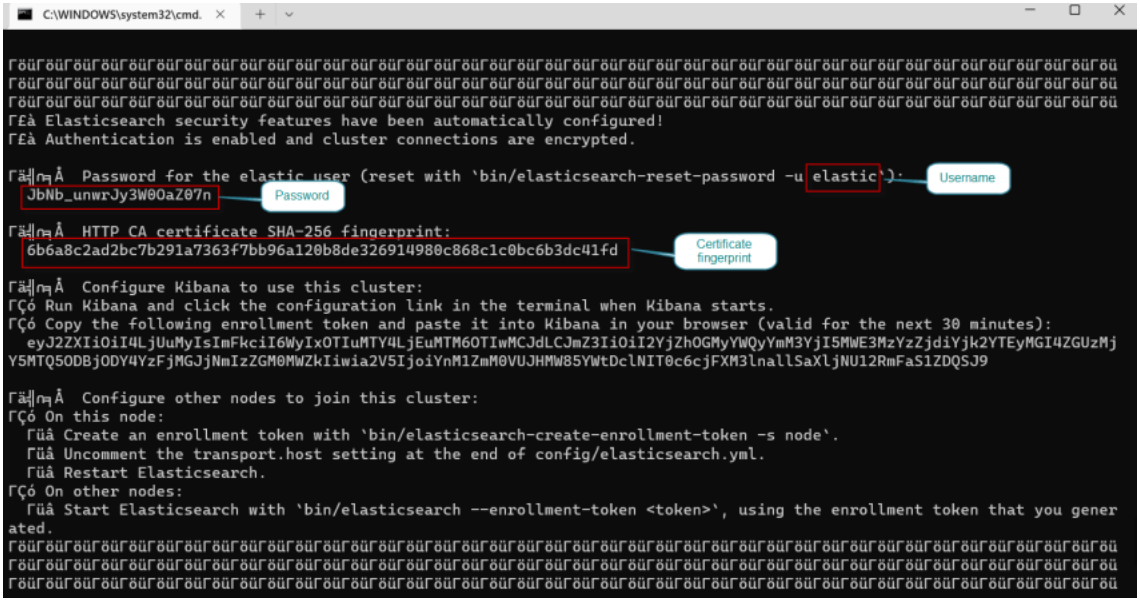
- Microsoft Visual Studio 2022
- Familiar with ASP.NET Core MVC

What is Elasticsearch?

Elasticsearch is one of the most popular free, open-source search databases built on Apache Lucene and developed in JAVA. It provides distributed full-text search engine with an HTTP web interface and schema free JSON documents. You can also say Elasticsearch is a server that processes JSON requests and give you back JSON data. It can be used to search and analyze huge volumes of data. Big names like Adobe, Vimeo, and Microsoft also uses Elasticsearch to provide better and smarter search experience to users.

Download and Install Elasticsearch

Elasticsearch is available in both cloud and local versions to use. Popular cloud services providers like Azure and AWS have pre-configured services for Elasticsearch. If you want to use Elasticsearch on your local machine then you can download it from here. Please make sure the updated JAVA Virtual Machine is also installed in your machine. Once you downloaded Elasticsearch zip file, extract it and run `\bin\elasticsearch.bat`. After running this file, you should be able to browse *http://localhost:9200*. You can use this as your Elasticsearch server. You will also get your username, password, and other necessary information that you have to use in ASP.NET Core middleware.



Integration in ASP.NET Core

For the demo purpose, I took a list of articles from C# Corner and converted them into JSON data. This data holds the article Id (auto-generated in code), title, link, author, link to author profile, and published date.

To integrate Elasticsearch in ASP.NET Core project you have to follow various steps which I will be discussing here step by step.

Package Installation

To integrate Elasticsearch in ASP.NET Core you have to install NEST package. You can install it from **NuGet Package Manager** or from **Package Manager Console**. You can use the below command in **Package Manager Console**

```
PM> Install-Package NEST -Version 7.17.5
```

As I mentioned about using NEST package, you might have a question about why NEST and not Elasticsearch.NET. Elasticsearch.NET is a low-level, dependency-free client that has no opinions about how you build and represent requests and responses. NEST is a high-level client that

comes with the advantage of having mapped all the requests and responses. It maps strongly typed query DSL with Elasticsearch query DSL. NEST is internally built on top of Elasticsearch.NET.

Adding Elasticsearch Middleware

After installing the NEST package successfully, the next step is to setup middleware. First, let's add a model class in the **Models** folder that holds matching properties with JSON data.

```
public class ArticleModel {
    public int Id {
        get;
        set;
    }
    public string Title {
        get;
        set;
    }
    public string Link {
        get;
        set;
    }
    public string Author {
        get;
        set;
    }
    public string AuthorLink {
        get;
        set;
    }
    public DateTime PublishedDate {
        get;
        set;
    }
}
```

C#
Copy

Next is to configure Elasticsearch instance URL to consume RESTFUL data. This URL can be put in the code also but a better recommendation is to configure in `appsettings.json` as the URL might change when the domain get changed. A default index is needed to store the documents.

```
{
    "Logging": {
        "LogLevel": {
            "Default": "Information",
            "Microsoft.AspNetCore": "Warning"
        }
    },
    "ElasticSettings": {
        "baseUrl": "https://localhost:9200/",
        "defaultIndex": "articles"
    },
    "AllowedHosts": "*"
}
```

C#
Copy

Now let's create a file `ElasticSearchExtension.cs` to manage connection with Elasticsearch instance. This file can be inside the **Extension** folder or any other location within the project.

```
public static class ElasticSearchExtension {
    public static void AddElasticSearch(this IServiceCollection services, IConfiguration configuration) {
        var baseUrl = configuration["ElasticSettings:baseUrl"];
        var index = configuration["ElasticSettings:defaultIndex"];
        var settings = new ConnectionSettings(new Uri(baseUrl ?? "")).PrettyJson().CertificateFingerprint("6b6a8c2ad2bc7b291a7
        settings.EnableApiVersioningHeader();
        AddDefaultMappings(settings);
        var client = new ElasticClient(settings);
        services.AddSingleton < IElasticClient > (client);
        CreateIndex(client, index);
    }
    private static void AddDefaultMappings(ConnectionSettings settings) {
        settings.DefaultMappingFor < ArticleModel > (m => m.Ignore(p => p.Link).Ignore(p => p.AuthorLink));
    }
    private static void CreateIndex(IElasticClient client, string indexName) {
        var createIndexResponse = client.Indices.Create(indexName, index => index.Map < ArticleModel > (x => x.AutoMap()));
    }
}
```

C#
Copy

In this code, in `AddElasticSearch` method you can see that basic options are configured. Username, password, and certificate footprint generated during Elasticsearch instance setup are used here. In `AddDefaultMappings` method, you can see Link and AuthorLink properties are ignored from the search. And `CreateIndex` is creating a mapping between the model class ArticleModel and the index from Elasticsearch instance.

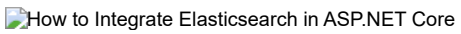
Now let's use this extension ElascitSearchExtension as middleware. For that open `Program.cs` in your project and use the extension like in this code

```
var builder = WebApplication.CreateBuilder(args);
builder.Services.AddElasticSearch(builder.Configuration);
```

C#
Copy

Adding Documents in Elasticsearch

Here I will discuss 2 ways to add documents in Elasticsearch, add a single item and bulk import from a JSON file. Let's take a scenario of an e-commerce site where you have thousands of products. Adding a single document can be used to add/update document when you add or update your product in your ecommerce site. And Bulk import can be used for rebuilding the indexing of all products.



This code will add new documents in the Elasticsearch index.

```
private readonly IElasticClient _elasticClient;
private readonly IWebHostEnvironment _hostingEnvironment;
public ArticleController(IElasticClient elasticClient, IWebHostEnvironment hostingEnvironment) {
    _elasticClient = elasticClient;
    _hostingEnvironment = hostingEnvironment;
}
```

C#
Copy

```
[HttpPost]
[ValidateAntiForgeryToken]
public async Task < IActionResult > Create(ArticleModel model) {
    try {
        var article = new ArticleModel() {
            Id = 1,
                Title = model.Title,
                Link = model.Link,
                Author = model.Author,
                AuthorLink = model.AuthorLink,
                PublishedDate = DateTime.Now
        };
        await _elasticClient.IndexDocumentAsync(article);
        model = new ArticleModel();
    } catch (Exception ex) {}
    return View(model);
}
```

C#
Copy

To update existing document you can use `UpdateAsync` method

```
await _elasticClient.UpdateAsync<ArticleModel>(article.Id, u => u
    .Index("articles")
    .Doc(article))
```

C#
Copy

To delete a document you can use `DeleteAsync` method

```
await _elasticClient.DeleteAsync<ArticleModel>(article);
```

C#
Copy

And bulk import can be done as in this code

```
[HttpPost]
[ValidateAntiForgeryToken]
public IActionResult Import() {
    try {
        var rootPath = _hostingEnvironment.ContentRootPath; //get the root path
        var fullPath = Path.Combine(rootPath, "articles.json"); //combine the root path with that of our json file inside myda
        var jsonData = System.IO.File.ReadAllText(fullPath); //read all the content inside the file
        var articleList = JsonConvert.DeserializeObject < List < ArticleModel >> (jsonData);
        if (articleList != null) {
            foreach(var article in articleList) {
                _elasticClient.IndexDocumentAsync(article);
```

```
            }
        }
    } catch (Exception ex) {}
    return RedirectToAction("Index");
}
```

C#
Copy

Searching Documents

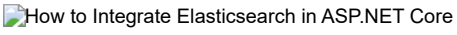
This code search documents from Elasticsearch instance based on the input keywords.

```
[HttpGet]
public ActionResult Index(string keyword) {
    var articleList = new List < ArticleModel > ();
    if (!string.IsNullOrEmpty(keyword)) {
        articleList = GetSearch(keyword).ToList();
    }
    return View(articleList.AsEnumerable());
}
public IList < ArticleModel > GetSearch(string keyword) {
    var result = _elasticClient.SearchAsync < ArticleModel > (s => s.Query(q => q.QueryString(d => d.Query('*' + keyword + '*'
    var finalResult = result;
    var finalContent = finalResult.Result.Documents.ToList();
    return finalContent;
}
```

C#
Copy

You can read more about the search here.

The output of the search looks like this in this screenshot



Conclusion

In this article, I discussed the basic introduction of Elasticsearch. I also discussed adding Elasticsearch middleware extension in ASP.NET Core, adding new documents, and searching documents. Elasticsearch has a lot of features that can not be covered in a single article. I hope this article will give you some idea about the use of Elasticsearch and put you in the right direction. You can download the source code from this article and give it a try.

Please feel free to ask in the comment section if you have any suggestions.

Thank you for reading the article.