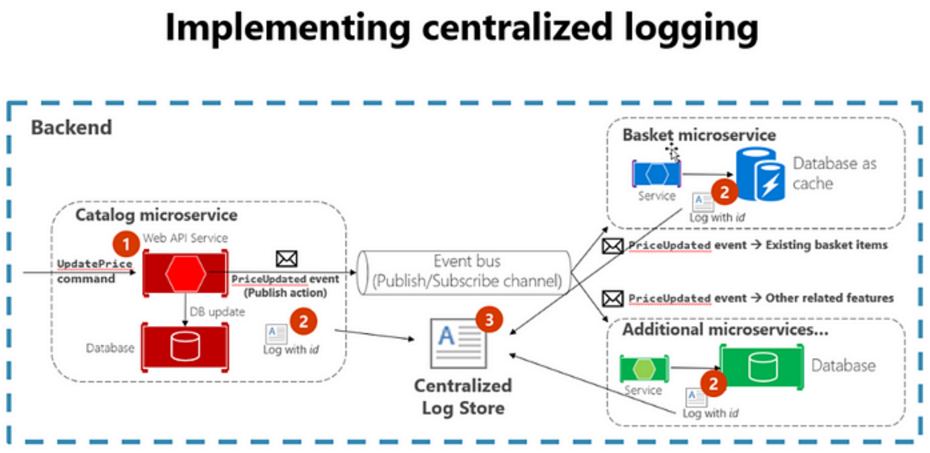
Microservices Observability with Distributed Logging using ElasticSearch and Kibana

In this article, we are going to Developing "Microservices Observability with Distributed Logging using ElasticSearch and Kibana".

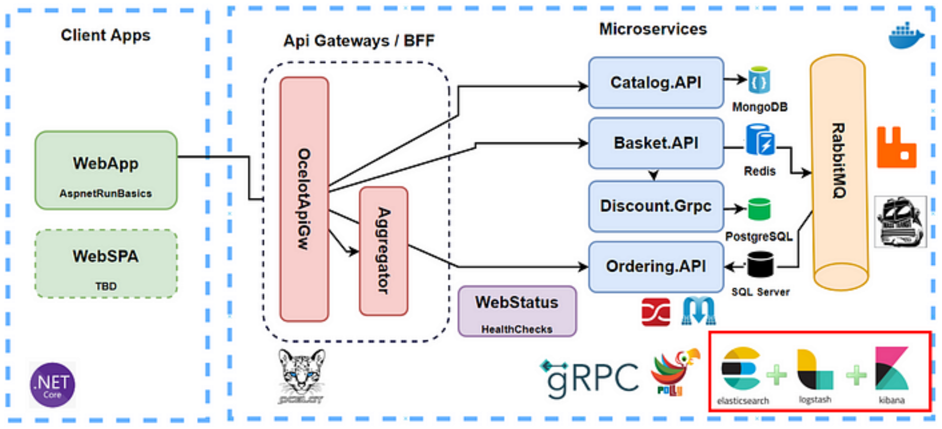

Implementing centralized logging
Backend
Catalog microservice
Web API Service
UpdatePrice command
PriceUpdated event (Publish action)
DB update
Database
Log with id
Event bus (Publish/Subscribe channel)
Centralized Log Store
Basket microservice
Service
Database as cache
Log with id
PriceUpdated event → Existing basket items
PriceUpdated event → Other related features
Additional microservices...
Service
Database
Log with id

Microservice architecture have become the new model for building modern cloud-native applications. And microservices-based applications are distributed systems.

How do you handle the complexity that comes with cloud and microservices? — **Observability**

Microservice should have a strategy for **monitoring** and **managing** the complex dependencies on microservices. That means we need to implement **microservices observability with using distributed logging features.**
Microservices Observability gives us greater operational insight and leading to understand incidents on our microservices architecture.

Let's check our big picture and see what we are going to build one by one.
As you can see that, we are in here and start to developing "Microservices Observability with Distributed Logging".



We are going to cover the;

- Start with Asp.Net logging basics and understand how to logging works on asp.net
- Applying **Elastic Stack** which includes **Elasticsearh** + **Logstach** + **Kibana**
- Use **SeriLog nuget package** for creating structured logs in .net microservices
- Use **docker-compose Kibana image** from docker hub and feed **Kibana** with **elastic stack**
- That means we Containerize **ElasticSearch** and **Kibana** on our docker environment with using Docker Compose

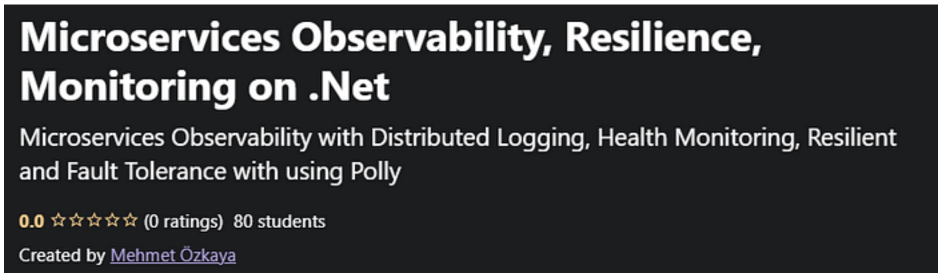
So in this article, we are going to Develop our "**Microservices Observability with Distributed Logging implementing ElasticStack** on Asp.Net Microservices". By the end of the article, we will have ElasticSearch and Kibana docker images and feeding this systems from our Asp.net microservices with creating structure logs using SeriLog nuget package.

Background

This is the introduction of the series. This will be the series of articles. You can follow the series with below links.

We will focus on microservices cross-cutting concerns on these article series.

Step by Step Development w/ Udemy Course


Microservices Observability, Resilience, Monitoring on .Net

Microservices Observability with Distributed Logging, Health Monitoring, Resilient and Fault Tolerance with using Polly

0.0 ☆☆☆☆☆ (0 ratings) 80 students

Created by Mehmet Özkaya

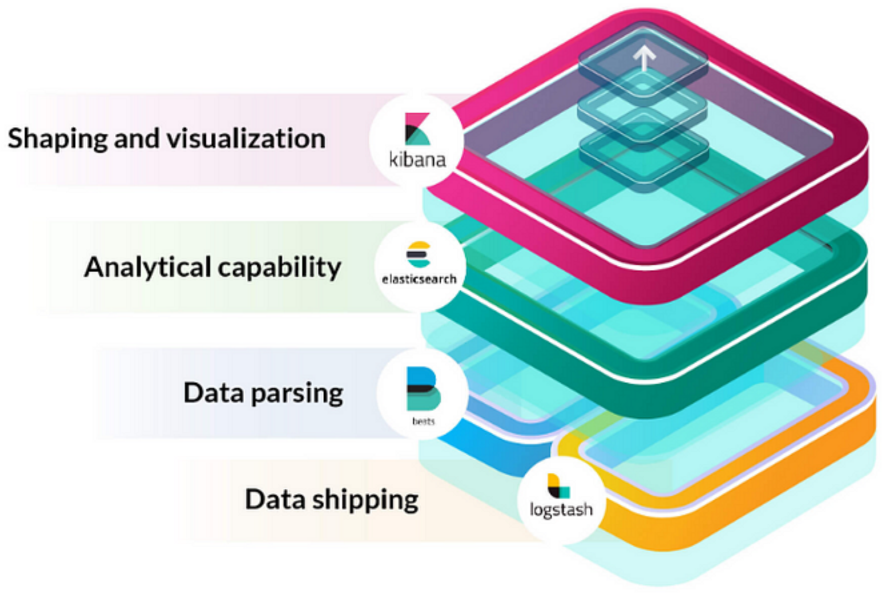
Get Udemy Course with discounted — Microservices Observability, Resilience, Monitoring on .Net.

Source Code

Get the Source Code from AspnetRun Microservices Github — Clone or fork this repository, if you like don't forget the star. If you find or ask anything you can directly open issue on repository.

What is Elastic Search ?

e have said that, we should have a strategy for monitoring and managing the complex dependencies on microservices. That means we need to implement microservices observability with using distributed logging features.
Elastic Search is one of the best tools for distributed logging on microservices architectures.



Official — ElasticSearch

Elasticsearch is a distributed, open source search and analytics engine for all types of data, including textual, numerical, geospatial, structured, and unstructured. Elasticsearch is built on Apache Lucene and was first released in 2010 by Elasticsearch N.V. (now known as Elastic)

So we can say that; **Elasticsearch** is the preferred full-text search search engine in processes such as content search, data analysis, queries and suggestions, especially due to its performance capabilities, powerful and flexible features. It is developed in Java and is based on Lucene.

In other words, Elasticsearch is an open source database that is well suited for indexing logs and analytical data.

Why is ElasticSearch so popular?

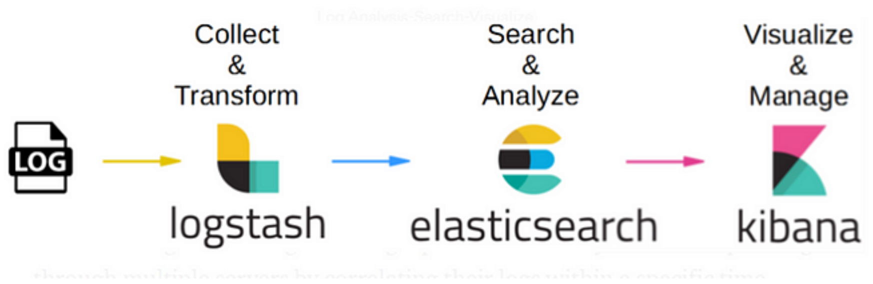
Besides being a requirement for almost every application, ElasticSearch solves a number of problems and does it really well:

- **Free and open source**
  Basic features are mostly free. If you need security and alert features with Kibana, then it is required to purchase commercial pack.
- **RESTful API**
  ElasticSearch has a RESTful API. Query results are returned in JSON format, which means working with results is easy. Querying and adding data through the RESTful API means it is easy to use any programming language to work with ElasticSearch.
- **Easy To Query**
  ElasticSearch has a built-in full-text search engine based on Apache Lucene. It is very easy to writing queries on Elastic Search, no required

advanced programming knowledge.
- **It's very fast**
  Elasticsearch is fast. Because Elasticsearch engine based on Lucene, it excels at full-text search.
  Elasticsearch is also a near real-time search platform, meaning the its property of high speed and high availability, it is also a popular tool to save big data today.
- **Scalable**
  It is easy to scale. it's open source and containerized on docker hub and that means it means it's easy work on docker containers.
- **Easy to Install**
  There is docker images on DockerHub for ElasticSearch and Kibana containers. So that means we can easly add these services into our docker-compose files and you are ready to start logging and searching.

What is Kibana ?

Kibana is an open source data visualization and user interface for Elasticsearch. We can think of Elasticsearch as a database and Kibana as the web user interface that we can use to create graphs, queries, indexes in Elasticsearch.
It provides search and visualization capabilities for data indexes in the ElasticSearch. You can query the data on Elasticsearch and create graphics.



Also you can see the image,

- Logstash is collecting logs and transform
- Elasticsearch is searching and analysing logs
- Kibana is visualize ana manage logs

Why logging with ElasticSearch and Kibana?

In microservices applications, logging is critical to implement in order to identify problems in distributed architecture. ElasticSearch makes any kind of logging easy, accessible and searchable.
When using ElasticSearch, It makes logging easily accessible
and searchable using a simple query language coupled with Kibana interface.
ElasticSearch's incredible speed and simple query language coupled with Kibana's interface.

What is Serilog?

Serilog is a library for ASP.NET that makes logging easy. There are several Sinks available for Serilog, for example, File, SQL and Elasticsearch Sinks.

Adding ElasticSearch and Kibana image into Docker-Compose File for Multi-Container Docker Environment

we are going to add ElasticSearch and Kibana image into Docker-Compose File for Multi-Container Docker Environment.

Implementing Centralized Distributed Logging for Microservices

Elastic Stack (ELK) E-elasticsearch, L-logstash (log shipping), K-kibana
SeriLog -send logs to ElasticSearch
Kibana — query logs — data visualization
Enrich Logs

Before we start coding in .NET Core, it's important to first spin up the Elasticsearch and Kibana containers.
The easiest way to spin up these containers is to updating our docker-compose.yml file.

Before we start, we should check ElasticSearch and Kibana images from docker hub.

Docker compose setup ElasticSearch and Kibana

https://hub.docker.com/_/elasticsearch/
https://hub.docker.com/_/kibana

Now we can add ElasticSearch and Kibana image into our docker-compose.yml files

docker-compose.yml

```
elasticsearch:
 image: docker.elastic.co/elasticsearch/elasticsearch:7.9.2kibana:
 image: docker.elastic.co/kibana/kibana:7.9.2volumes:
 mongo_data:
 portainer_data:
 postgres_data:
 pgadmin_data:
 elasticsearch-data: — — — — — — → ADDED
```

We have added 2 image, 1 volume

docker-compose.override.yml

```
elasticsearch:
 container_name: elasticsearch
 environment:
 — xpack.monitoring.enabled=true
 — xpack.watcher.enabled=false
 — "ES_JAVA_OPTS=-Xms512m -Xmx512m"
 — discovery.type=single-node
 ports:
 — "9200:9200"
 volumes:
 — elasticsearch-data:/usr/share/elasticsearch/datakibana:
 container_name: kibana
 environment:
 — ELASTICSEARCH_URL=http://localhost:9200
 depends_on:
 — elasticsearch
 ports:
 — "5601:5601"
```

We have added elasticsearch and kibana
We set elasticsearch and kibana configuration as per environment variables.

Finally, we can create ElasticSearch and Kibana image for Implementing Centralized Distributed Logging for Microservices.

Open In Terminal

RUN with below command on that location;

```
docker-compose -f docker-compose.yml -f docker-compose.override.yml up -d
 docker-compose -f docker-compose.yml -f docker-compose.override.yml down
```

This will pull images for elasticsearch and kibana.

The first time you run the docker-compose command, it will download the images for ElasticSearch and Kibana from the docker registry, so it might take a few minutes depending on your connection speed.

Once you've run the docker-compose up command, check that ElasticSearch and Kibana are up and running.

RUN on DOCKER

Verify that Elasticsearch is up and running

Elasticsearch is up and running
http://localhost:9200

Check the indexes
http://localhost:9200/_aliases

So right now there is no any indexes defined in elastic search. I am not going to create indexes or data, because we have already generated logs and logs comes from different microservices, those are will be indexes and data for elasticsearh.

But its good to know, you can create data with sending POST request to elastic search. For example you can send set of json product data to this url and elastic search will cover data.
http://localhost:9200/products/_bulk
http://localhost:9200/products/_search
But these are not our topic for now we are focusing on distributed logging feature on microservices.

Verify that Kibana is up and running

Navigate to http://localhost:5601 to ensure Kibana is up and running

Kibana is up and running
http://localhost:5601

Normally it takes some time to run Kibana, we should wait. But If you get error , you should INCREASE DOCKER RAM.

DOCKER SETTINGS

RESOURCES
MEMORY
INCREASE 2 -> 4 GB

As you can see that, we have successfully added ElasticSearch and Kibana image into Docker-Compose File for Multi-Container Docker Environment.

Install and Configure SeriLog For ElasticSearch and Kibana Sink Integration

We are going to Install and Configure SeriLog For ElasticSearch and Kibana Sink Integration.

Let's develop Serilog in AspnetRunBasics Shopping Web Application and test it.

Go to
AspnetRunBasics

Install Packages

Install-Package Serilog.AspNetCore
Install-Package Serilog.Enrichers.Environment
Install-Package Serilog.Sinks.Elasticsearch

Check item group project file;
```
 <ItemGroup>
 <PackageReference Include="Microsoft.VisualStudio.Azure.Containers.Tools.Targets" Version="1.10.9" />
 <PackageReference Include="Serilog.AspNetCore" Version="3.4.0" />
 <PackageReference Include="Serilog.Enrichers.Environment" Version="2.1.3" />
 <PackageReference Include="Serilog.Sinks.Elasticsearch" Version="8.4.1" />
 </ItemGroup>
```

After the related packages are installed, we organize the logging section in our appsettings.json file according to the serilog section that we will use:

Change the Logging Settings on appsettings.json file

Existing
```
{
 "ApiSettings": {
 "GatewayAddress": "http://localhost:8010"
 },
 "Logging": {
 "LogLevel": {
 "Default": "Information",
 "Microsoft": "Warning",
 "Microsoft.Hosting.Lifetime": "Information",
 "AspnetRunBasics": "Debug"
 }
 },
 "AllowedHosts": "*"
 }
```

New One :
```
{
 "ApiSettings": {
 "GatewayAddress": "http://localhost:8010"
 },
 “Serilog”: {
 “MinimumLevel”: {
 “Default”: “Information”,
 “Override”: {
 “Microsoft”: “Information”,
 “System”: “Warning”
 }
 }
 },
 “ElasticConfiguration”: {
 “Uri”: “http://localhost:9200”
 },
 “AllowedHosts”: “*”
}
```

We have added the Serilog section and the section where our Elasticsearch endpoint is located.
Remove the Logging section in appsettings.json and replace it with the following configuration so that we can tell Serilog what the minimum log level should be, and what url to use for logging to Elasticsearch.

Configure SeriLog

After that, we will configure logging in **Program.cs** by adding the following details on Main method.

What we want to do is set up logging before creating the host. This way, if the host does not start, we can log any errors. After that we will add the **ConfigureLogging** and **ElasticsearchSinkOptions** methods to the Program.cs file.

AspnetRunBasics

```
Program.cs — UseSerilog — — ADDEDpublic static IHostBuilder CreateHostBuilder(string[] args) =>
 Host.CreateDefaultBuilder(args)
 .UseSerilog((context, configuration) =>
 {
 configuration
 .Enrich.FromLogContext()
 .Enrich.WithMachineName()
 .WriteTo.Console()
 .WriteTo.Elasticsearch(
 new ElasticsearchSinkOptions(new Uri(context.Configuration[“ElasticConfiguration:Uri”]))
 {
 IndexFormat = $”applogs-{Assembly.GetExecutingAssembly().GetName().Name.ToLower().Replace(“.”, “-”)}-{context.HostingEnvironment.EnvironmentName?.ToL
 AutoRegisterTemplate = true,
 NumberOfShards = 2,
 NumberOfReplicas = 1
 })
 .Enrich.WithProperty(“Environment”, context.HostingEnvironment.EnvironmentName)
 .ReadFrom.Configuration(context.Configuration);
 })
 .ConfigureWebHostDefaults(webBuilder =>
 {
 webBuilder.UseStartup<Startup>();
 });
```

We have configured Serilog according to writing Console and Elastic Search. When Configure for the Elastic Search we provide some sink options.

As you can see that, we have Install and Configured SeriLog For ElasticSearch and Kibana Sink Integration, next video we are going to test it.

Test SeriLog For ElasticSearch and Kibana Sink Integration in Shopping Web Microservices

We are going to Test SeriLog For ElasticSearch and Kibana Sink Integration in Shopping Web Microservices.

So far we have finished to development part of configurations, but Kibana will not show any logs at the moment. We need to create an index for Kibana to show your logs.
As a standard, I will create a pattern with ‘*’. We can give a specific pattern here, but I found it appropriate to give a general pattern as an example.
Let’s continue the log process by showing how to create an example index pattern on Kibana.

Before we start, let me Verify Docker compose worked :

```
docker-compose -f docker-compose.yml -f docker-compose.override.yml up -d
```

Run Application
Set a Startup Project and Run
AspnetRunBasics

See logs are colorfully

```
[15:02:50 INF] Now listening on: http://localhost:5006
 [15:02:50 INF] Application started. Press Ctrl+C to shut down.
 [15:02:50 INF] Hosting environment: Development
 [15:02:50 INF] Content root path: C:\Users\ezozkme\source\repos\mutank\src\WebApps\AspnetRunBasics
 [15:02:51 INF] Request starting HTTP/1.1 GET http://localhost:5006/ — -
 [15:02:52 INF] Executing endpoint ‘/Index’
 [15:02:52 INF] Route matched with {page = “/Index”}. Executing page /Index
 [15:02:52 INF] Executing handler method AspnetRunBasics.Pages.IndexModel.OnGetAsync — ModelState is Valid
 [15:02:57 INF] Executed handler method OnGetAsync, returned result Microsoft.AspNetCore.Mvc.RazorPages.PageResult.
 [15:02:57 INF] Executed page /Index in 5374.8521ms
 [15:02:57 INF] Executed endpoint ‘/Index’
```

```
[15:02:57 INF] Request finished HTTP/1.1 GET http://localhost:5006/ — — — 200 — text/html;+charset=utf-8 5688.4302ms
[15:03:29 INF] Request starting HTTP/1.1 GET http://localhost:5006/Product — -
[15:03:29 INF] Executing endpoint '/Product'
```

When we go to the Kibana interface with the localhost: 5601 url on the browser, it will meet us with a screen as follows:

Elasticsearch is up and running
http://localhost:9200

Kibana is up and running
http://localhost:5601

http://localhost:5601/app/home#/

Create an Index Pattern in Kibana to Show Data

Kibana will not show the logs yet. Before we can view the logged data, we must specify a index pattern.
In order to do this, click the Explore by myself link on the default Kibana page
and then click the Discover link in the navigation section.

- Use Elasticsearch data
  Connect to your Elasticsearch index
- Index patterns
  Create Index Pattern

my index seems here
aspnetrunbasics-development-2021–02

Then, type in an index pattern. It will show the index pattern that was just created. You can type in the entire index, or use wildcards.

put this
aspnetrunbasics-*

On the next page, select the @timestamp field as the time filter field name and click the Create index pattern button.

- Next Step
  timestamp
  Create Index Pattern

See Fields

You can now view the logs by clicking the Discover link in the navigation pane.

- Go to Main Menu
  Discover

See Logs, as you can see that we can see logs on Kibana Dashboard.

Change LogInformation — add new customPropery

Now I am going to add new log into my application.
```
AspnetRunBasics.Services
CatalogServicepublic async Task<IEnumerable<CatalogModel>> GetCatalog()
 {
 _logger.LogInformation("Getting Catalog Products from url : {url} and custom property: {customPropery}", _client.BaseAddress, 6);
```

Now that we've logged a message, refresh the application on http://localhost:5000 again. Then search for the log message for customPropery text.

- Run Again
  See
  fields.customPropery — 6
- Open Kibana
  Index Pattern
  Refresh
- Now you can search by
  fields.customPropery — 6

Go to Discover
fields.customPropery : 6

Search

As you can see that, we have Tested SeriLog For ElasticSearch and Kibana Sink Integration in Shopping Web Microservices.
But this is only 1 microservices logs, but we have more that 10 microservices need to centralized log in order to trace log exceptions.

Get Udemy Course with discounted — Microservices Observability, Resilience, Monitoring on .Net.

For the next articles ->