WebApi\_Handson

Kafka Integration with C#

Outline

1. Introduction to Kafka

Apache Kafka is a distributed event streaming platform capable of handling trillions of events a day.

Originally developed by LinkedIn and now part of the Apache Software Foundation.

Used for building real-time data pipelines and streaming applications.

2. Kafka Architecture

Producer: Publishes data to topics.

Consumer: Subscribes to topics and processes the feed.

Broker: Kafka server that stores data and serves clients.

Cluster: Group of brokers working together.

Zookeeper: Manages the Kafka brokers and helps with leader election.

3. Topics

Logical channels to which producers publish and from which consumers read.

Topics are split into partitions.

4. Partitions

Topics are divided into multiple partitions for scalability and parallelism.

Each message within a partition has a unique offset.

5. Brokers

Kafka brokers receive messages from producers, assign offsets to them, and commit them to storage on disk.

A Kafka cluster is made of multiple brokers.

6. Kafka Plugin in .NET

Use Confluent.Kafka client library for .NET.

Supports producer and consumer APIs.

Compatible with Apache Kafka 0.10+.

7. Installation of Kafka

Kafka requires Java and Zookeeper to run.

Download Kafka binaries from the [Apache Kafka website](https://kafka.apache.org/downloads" \t "_new).

Extract and configure using server.properties.

8. Basics of Zookeeper

Keeps track of Kafka brokers.

Handles leader election and configuration management.

Required for Kafka to run in older versions (<2.8).

Output

Zookeeper Running:

> zookeeper-server-start.bat ../../config/zookeeper.properties

Kafka Server Running:

> kafka-server-start.bat ../../config/server.properties

Topic Created:

> kafka-topics.bat --create --topic chat-topic --bootstrap-server localhost:9092 --partitions 1 --replication-factor 1

Publisher in Command Prompt:

> Chat Message Sent: "Hello, Kafka!"

Consumer Output:

> New Chat Message Received: "Hello, Kafka!" from User1

Client Application Output

User1 (Sender):  
TextBox: Hey, are you online?  
Button Click → Message sent to Kafka.

User2 (Receiver):  
ListBox:

csharp

CopyEdit

[10:10 AM] User1: Hey, are you online?

Kafkachatproducer

using System;

using Confluent.Kafka;

class KafkaChatProducer

{

public static async Task Main()

{

var config = new ProducerConfig

{

BootstrapServers = "localhost:9092"

};

using var producer = new ProducerBuilder<Null, string>(config).Build();

Console.WriteLine("Kafka Chat Producer Started.");

Console.WriteLine("Type messages to send to chat-topic...");

while (true)

{

var message = Console.ReadLine();

if (string.IsNullOrWhiteSpace(message)) continue;

await producer.ProduceAsync("chat-topic", new Message<Null, string> { Value = message });

Console.WriteLine($"Sent: {message}");

}

}

}

Kafkachatconsumer

using System;

using Confluent.Kafka;

class KafkaChatConsumer

{

public static void Main()

{

var config = new ConsumerConfig

{

BootstrapServers = "localhost:9092",

GroupId = "chat-consumer-group",

AutoOffsetReset = AutoOffsetReset.Earliest

};

using var consumer = new ConsumerBuilder<Ignore, string>(config).Build();

consumer.Subscribe("chat-topic");

Console.WriteLine("Kafka Chat Consumer Started. Listening for messages...");

try

{

while (true)

{

var cr = consumer.Consume();

Console.WriteLine($"Received: {cr.Message.Value}");

}

}

catch (OperationCanceledException)

{

consumer.Close();

}

}

}

Web API Microservices

1. Create a new ASP.NET Core Web API Project

dotnet new webapi -n JwtAuthDemo

cd JwtAuthDemo

1. Add NuGet Packages

dotnet add package Microsoft.AspNetCore.Authentication.JwtBearer

1. Configure JWT in appsettings.json

{

"Jwt": {

"Key": "ThisIsASecretKeyForJwtToken",

"Issuer": "MyAuthServer",

"Audience": "MyApiUsers",

"DurationInMinutes": 60

},

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft.AspNetCore": "Warning"

}

},

"AllowedHosts": "\*"

}

4. Create Models

// Models/LoginModel.cs

public class LoginModel

{

public string Username { get; set; }

public string Password { get; set; }

}

Program.cs

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.IdentityModel.Tokens;

using System.Text;

var builder = WebApplication.CreateBuilder(args);

// JWT Authentication

builder.Services.AddAuthentication("Bearer")

.AddJwtBearer("Bearer", options =>

{

options.TokenValidationParameters = new TokenValidationParameters

{

ValidateIssuer = true,

ValidateAudience = true,

ValidateLifetime = true,

ValidateIssuerSigningKey = true,

ValidIssuer = builder.Configuration["Jwt:Issuer"],

ValidAudience = builder.Configuration["Jwt:Audience"],

IssuerSigningKey = new SymmetricSecurityKey(

Encoding.UTF8.GetBytes(builder.Configuration["Jwt:Key"]))

};

});

builder.Services.AddAuthorization();

builder.Services.AddControllers();

var app = builder.Build();

app.UseAuthentication();

app.UseAuthorization();

app.MapControllers();

app.Run();

Auth Controller

using Microsoft.AspNetCore.Mvc;

using Microsoft.IdentityModel.Tokens;

using System.IdentityModel.Tokens.Jwt;

using System.Security.Claims;

using System.Text;

[ApiController]

[Route("api/[controller]")]

public class AuthController : ControllerBase

{

[HttpPost("login")]

public IActionResult Login([FromBody] LoginModel model)

{

if (IsValidUser(model))

{

var token = GenerateJwtToken(model.Username);

return Ok(new { Token = token });

}

return Unauthorized();

}

private bool IsValidUser(LoginModel model)

{

// Simulate user validation (replace with real DB call)

return model.Username == "admin" && model.Password == "password";

}

private string GenerateJwtToken(string username)

{

var claims = new[]

{

new Claim(ClaimTypes.Name, username)

};

var key = new SymmetricSecurityKey(

Encoding.UTF8.GetBytes("ThisIsASecretKeyForJwtToken"));

var creds = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);

var token = new JwtSecurityToken(

issuer: "MyAuthServer",

audience: "MyApiUsers",

claims: claims,

expires: DateTime.Now.AddMinutes(60),

signingCredentials: creds);

return new JwtSecurityTokenHandler().WriteToken(token);

}

}

Protected Endpoint

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

[ApiController]

[Route("api/[controller]")]

public class SecureController : ControllerBase

{

[HttpGet]

[Authorize]

public IActionResult GetSecret()

{

return Ok("You are authorized to view this secret message!");

}

}

Login to Get Token

POST https://localhost:{port}/api/auth/login

Body (JSON):

json

Copy

Edit

{

"username": "admin",

"password": "password"

}

Response:

json

Copy

Edit

{

"token": "eyJhbGciOiJIUzI1NiIsInR5cCI..."

}

Authorization: Bearer <your-token>

Response:

json

Copy

Edit

"You are authorized to view this secret message!"