ID:6363695

1. Create a Chat Application which uses Kafka as a streaming platform and consume the chat messages in the command prompt.

2. Create a Chat Application using C# Windows Application using Kafka and consume the

KafkaChatConsole

//program.cs

// KafkaProducerApp/Program.cs

using Confluent.Kafka;

Console.WriteLine("Kafka Producer Started. Type messages to send:");

var config = new ProducerConfig

{

BootstrapServers = "localhost:9092"

};

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

while (true)

{

Console.Write("You: ");

var message = Console.ReadLine();

if (string.IsNullOrWhiteSpace(message)) break;

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

Console.WriteLine($"Message sent to {result.TopicPartitionOffset}");

}

// KafkaConsumerApp/Program.cs

using Confluent.Kafka;

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

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");

try

{

while (true)

{

var cr = consumer.Consume();

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

}

}

catch (OperationCanceledException)

{

consumer.Close();

}

//Producer

Program.cs

// KafkaProducerApp/Program.cs

using Confluent.Kafka;

Console.WriteLine("Kafka Producer Started. Type messages to send:");

var config = new ProducerConfig

{

BootstrapServers = "localhost:9092"

};

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

while (true)

{

Console.Write("You: ");

var message = Console.ReadLine();

if (string.IsNullOrWhiteSpace(message)) break;

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

Console.WriteLine($"Message sent to {result.TopicPartitionOffset}");

}

// KafkaConsumerApp/Program.cs

using Confluent.Kafka;

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

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");

try

{

while (true)

{

var cr = consumer.Consume();

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

}

}

catch (OperationCanceledException)

{

consumer.Close();

}

//Consumer.cs

using System;

using Confluent.Kafka;

class Program

{

static void Main(string[] args)

{

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

var config = new ConsumerConfig

{

BootstrapServers = "localhost:9092", // Kafka server

GroupId = "chat-consumer-group", // Consumer group name

AutoOffsetReset = AutoOffsetReset.Earliest

};

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

consumer.Subscribe("chat-topic"); // Subscribe to the topic

try

{

while (true)

{

var consumeResult = consumer.Consume(); // Read from topic

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

}

}

catch (OperationCanceledException)

{

Console.WriteLine("Closing consumer...");

}

finally

{

consumer.Close();

}

}

}

OUTPUT :



Question 1: Implement JWT Authentication in ASP.NET Core Web API.

appsetting.json

{

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft.AspNetCore": "Warning"

}

},

"Jwt": {

"Key": "ThisIsASecureLongJwtKeyForDemoPurposes1234", // 44 characters

"Issuer": "JwtAuthAPI2",

"Audience": "JwtAuthAPI2User",

"DurationInMinutes": "60"

},

"AllowedHosts": "\*"

}

//program.cs

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.IdentityModel.Tokens;

using System.Text;

var builder = WebApplication.CreateBuilder(args);

// JWT config from appsettings.json

var jwtSettings = builder.Configuration.GetSection("Jwt");

var key = jwtSettings["Key"];

builder.Services.AddAuthentication(options =>

{

options.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;

options.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;

})

.AddJwtBearer(options =>

{

options.TokenValidationParameters = new TokenValidationParameters

{

ValidateIssuer = true,

ValidateAudience = true,

ValidateLifetime = true,

ValidateIssuerSigningKey = true,

ValidIssuer = jwtSettings["Issuer"],

ValidAudience = jwtSettings["Audience"],

IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(key))

};

});

builder.Services.AddControllers();

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen();

var app = builder.Build();

app.UseSwagger();

app.UseSwaggerUI();

app.UseAuthentication();

app.UseAuthorization();

app.MapControllers();

app.Run();

//login.model

namespace JwtAuthAPI2.Models

{

public class LoginModel

{

public string Username { get; set; }

public string Password { get; set; }

}

}

//AuthController.cs

using Microsoft.AspNetCore.Mvc;

using Microsoft.Extensions.Configuration;

using Microsoft.IdentityModel.Tokens;

using System;

using System.IdentityModel.Tokens.Jwt;

using System.Security.Claims;

using System.Text;

namespace JwtAuthAPI2.Controllers

{

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

[ApiController]

public class AuthController : ControllerBase

{

private readonly IConfiguration \_configuration;

public AuthController(IConfiguration configuration)

{

\_configuration = configuration;

}

[HttpPost("login")]

public IActionResult Login([FromBody] LoginModel model)

{

if (model.Username == "admin" && model.Password == "password")

{

var token = GenerateJwtToken(model.Username);

return Ok(new { token });

}

return Unauthorized("Invalid username or password");

}

private string GenerateJwtToken(string username)

{

// Read JWT settings from appsettings.json

var keyString = \_configuration["Jwt:Key"];

var issuer = \_configuration["Jwt:Issuer"];

var audience = \_configuration["Jwt:Audience"];

var duration = int.Parse(\_configuration["Jwt:DurationInMinutes"]);

// Ensure the key is at least 256 bits (32 bytes)

if (string.IsNullOrEmpty(keyString) || Encoding.UTF8.GetByteCount(keyString) < 32)

{

throw new Exception("JWT Key must be at least 32 characters (256 bits) long.");

}

var securityKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(keyString));

var credentials = new SigningCredentials(securityKey, SecurityAlgorithms.HmacSha256);

var claims = new[]

{

new Claim(ClaimTypes.Name, username),

new Claim(ClaimTypes.Role, "Admin")

};

var token = new JwtSecurityToken(

issuer,

audience,

claims,

expires: DateTime.Now.AddMinutes(duration),

signingCredentials: credentials);

return new JwtSecurityTokenHandler().WriteToken(token);

}

}

public class LoginModel

{

public string Username { get; set; }

public string Password { get; set; }

}

}

//SecureController.cs

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

namespace JwtAuthAPI2.Controllers

{

[ApiController]

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

public class SecureController : ControllerBase

{

[HttpGet("secret")]

[Authorize]

public IActionResult GetSecret()

{

return Ok("This is a protected API. You are authenticated!");

}

}

}

OUTPUT:

