**Kafka Integration with C#:**

**Outline:**

* Introduction to Kafka
* Kafka Architecture
* Topics
* Partitions
* Brokers
* Kafka plug in .NET
* Installation of Kafka
* Basics of Zookeeper
* Demo

**Hands On:**

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 message in different client applications.

**Reference Links:**

1. <https://www.c-sharpcorner.com/article/apache-kafka-net-application/>
2. <https://www.c-sharpcorner.com/article/step-by-step-installation-and-configuration-guide-of-apache-kafka-on-windows-ope/>

**Demo output:**

A screenshot of a computer

AI-generated content may be incorrect.

**Kafka Server Running:**

Kafka-server-start.bat ../../config/server.properties

A screenshot of a computer

AI-generated content may be incorrect.

**Zookeeper Running:**

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

A computer screen with white text

AI-generated content may be incorrect.

**Topic Created:**

A computer screen with white text

AI-generated content may be incorrect.

**Creating Publisher in Command Prompt:**

A computer screen with white text

AI-generated content may be incorrect.

**Client Application:**

A screenshot of a computer

AI-generated content may be incorrect.

**A screenshot of a computer

AI-generated content may be incorrect.Implementation**

**A computer screen shot of a black screen

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Introduction to Kafka**

Apache Kafka is a powerful open-source platform designed for building real-time data pipelines and streaming applications. It was originally developed at LinkedIn to handle massive volumes of data efficiently and has since become one of the most popular tools for distributed messaging. Kafka allows you to send, store, and process data streams in a fault-tolerant and scalable way, making it ideal for use cases like activity tracking, log aggregation, and even building chat applications.

**Kafka Architecture**

Kafka works on a distributed system architecture where data is organized into topics, and these topics are partitioned across multiple brokers (servers). Producers send data to Kafka topics, and consumers read data from these topics. Kafka relies on Zookeeper to maintain cluster metadata and manage leader election for partitions. The architecture supports high throughput, scalability, and fault tolerance, which is why Kafka is often called the backbone of modern streaming systems.

**Topics**

A **topic** in Kafka acts like a logical channel or category where messages are stored and published. Producers send messages to a specific topic, and consumers subscribe to these topics to receive the data. Think of a topic as a folder where related messages are grouped. For example, in a chat application, there could be a topic called chat-topic where all chat messages flow.

**Partitions**

Each Kafka topic is split into smaller units called **partitions**. Partitions help distribute the data across different brokers in the cluster, allowing parallel processing and improving performance. Partitions also maintain an **offset** (a unique sequence number for each message) so consumers know exactly where they left off, even if they restart.

**Brokers**

A **broker** is essentially a Kafka server. It is responsible for storing data and serving client requests (producers and consumers). Kafka clusters are made up of multiple brokers, and each broker manages one or more partitions. If one broker fails, other brokers in the cluster can take over, ensuring no data is lost.

**Kafka Plugin in .NET**

Kafka can be integrated with .NET applications using libraries like Confluent.Kafka. This NuGet package provides easy-to-use APIs for producing and consuming messages from Kafka within C# projects. With this integration, developers can build real-time streaming systems or microservices in .NET that interact seamlessly with Kafka.

**Basics of Zookeeper**

Zookeeper is a centralized service that Kafka uses to manage cluster metadata and coordinate distributed brokers. It keeps track of brokers, topics, and partitions and ensures leader election happens properly when brokers go down. In Kafka versions 2.x, Zookeeper is mandatory, but newer versions of Kafka (KRaft mode) can work without it.

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

**Scenario:**

**You are building a microservice that requires secure login. You need to implement JWTbased authentication.**

**Steps:**

**1. Create a new ASP.NET Core Web API project.**

**2. Add a `User` model and a login endpoint.**

**3. Generate a JWT token upon successful login.**

**4. Secure an endpoint using `[Authorize]`.**

**Solution Code:**

**Install NuGet Packages:**

**dotnet add package Microsoft.AspNetCore.Authentication.JwtBearer**

appsettings.json:

{

"Jwt": {

"Key": "ThisIsASecretKeyForJwtToken",

"Issuer": "MyAuthServer",

"Audience": "MyApiUsers",

"DurationInMinutes": 60

}

}

Program.cs:

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();

AuthController.cs:

[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 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);

}

}

**Implementation**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A computer screen shot of a black screen

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A computer screen shot of a black screen

AI-generated content may be incorrect.**