# Lab 6 - Kafka Chat Application with C# and .NET

## Objective

In this lab, I learned how to integrate Apache Kafka with a C# console application using the Confluent.Kafka .NET client. The goal was to build a basic chat system where messages could be produced and consumed via Kafka topics.

## Prerequisites

• Java JDK 17 installed and JAVA\_HOME configured

• Apache Kafka and Zookeeper installed on Windows

• .NET SDK installed

• Visual Studio Code installed

• Confluent.Kafka NuGet package added to the project

## Setup Steps

1. Start Zookeeper:

.\zookeeper-server-start.bat ..\..\config\zookeeper.properties

2. Start Kafka server:

.\kafka-server-start.bat ..\..\config\server.properties

3. Create topic:

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

## Project Structure

KafkaChatApp/

├── Program.cs

├── Producer.cs

├── Consumer.cs

## Producer.cs

Code to produce messages to Kafka topic.

using System;

using System.Threading.Tasks;

class Program

{

    static async Task Main(string[] args)

    {

        if (args.Length == 0)

        {

            Console.WriteLine("Usage: dotnet run producer|consumer");

            return;

        }

        if (args[0] == "producer")

        {

            var p = new Producer();

            await p.Run();

        }

        else if (args[0] == "consumer")

        {

            var c = new Consumer();

            c.Run();

        }

        else

        {

            Console.WriteLine("Unknown command. Use 'producer' or 'consumer'.");

        }

    }

}

## Consumer.cs

Code to consume messages from Kafka topic.

using System;

using Confluent.Kafka;

public class Consumer

{

    public void Run()

    {

        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("Listening for messages (Ctrl+C to stop)...");

        try

        {

            while (true)

            {

                var cr = consumer.Consume();

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

            }

        }

        catch (OperationCanceledException) { }

        finally

        {

            consumer.Close();

        }

    }

}

## Program.cs

The entry point that decides whether to run the producer or consumer based on command-line arguments.

using System;

using System.Threading.Tasks;

class Program

{

    static async Task Main(string[] args)

    {

        if (args.Length == 0)

        {

            Console.WriteLine("Usage: dotnet run producer|consumer");

            return;

        }

        if (args[0] == "producer")

        {

            var p = new Producer();

            await p.Run();

        }

        else if (args[0] == "consumer")

        {

            var c = new Consumer();

            c.Run();

        }

        else

        {

            Console.WriteLine("Unknown command. Use 'producer' or 'consumer'.");

        }

    }

}

## Running the Application

To run the producer:

dotnet run -- producer

To run the consumer:

dotnet run -- consumer

## Conclusion

This lab gave me hands-on experience with message-based communication using Apache Kafka and C#. I learned how to build, run, and test producer and consumer applications in a distributed environment. Kafka's real-time message handling and the simplicity of Confluent's .NET library made integration efficient.

## Screenshots





