**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.

**CODE:**

using System;

using System.Threading.Tasks;

using Confluent.Kafka;

namespace KafkaChatApp

{

    class Program

    {

        static async Task Main(string[] args)

        {

            Console.WriteLine("Kafka Chat Application");

            Console.WriteLine("1. Producer (Send Message)");

            Console.WriteLine("2. Consumer (Receive Message)");

            Console.Write("Choose mode (1 or 2): ");

            var choice = Console.ReadLine();

            if (choice == "1")

            {

                await RunProducer();

            }

            else if (choice == "2")

            {

                RunConsumer();

            }

            else

            {

                Console.WriteLine("Invalid choice.");

            }

        }

        static async Task RunProducer()

        {

            var config = new ProducerConfig { BootstrapServers = "localhost:9092" };

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

            Console.WriteLine("Type your message and press Enter (empty to exit):");

            while (true)

            {

                Console.Write("You: ");

                var message = Console.ReadLine();

                if (string.IsNullOrWhiteSpace(message)) break;

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

            }

        }

        static void RunConsumer()

        {

            var config = new ConsumerConfig

            {

                BootstrapServers = "localhost:9092",

                GroupId = "chat-consumer-group",

                AutoOffsetReset = AutoOffsetReset.Earliest

            };

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

            consumer.Subscribe("chat-topic");

            Console.WriteLine("Waiting for messages (Ctrl+C to exit)...");

            while (true)

            {

                var consumeResult = consumer.Consume();

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

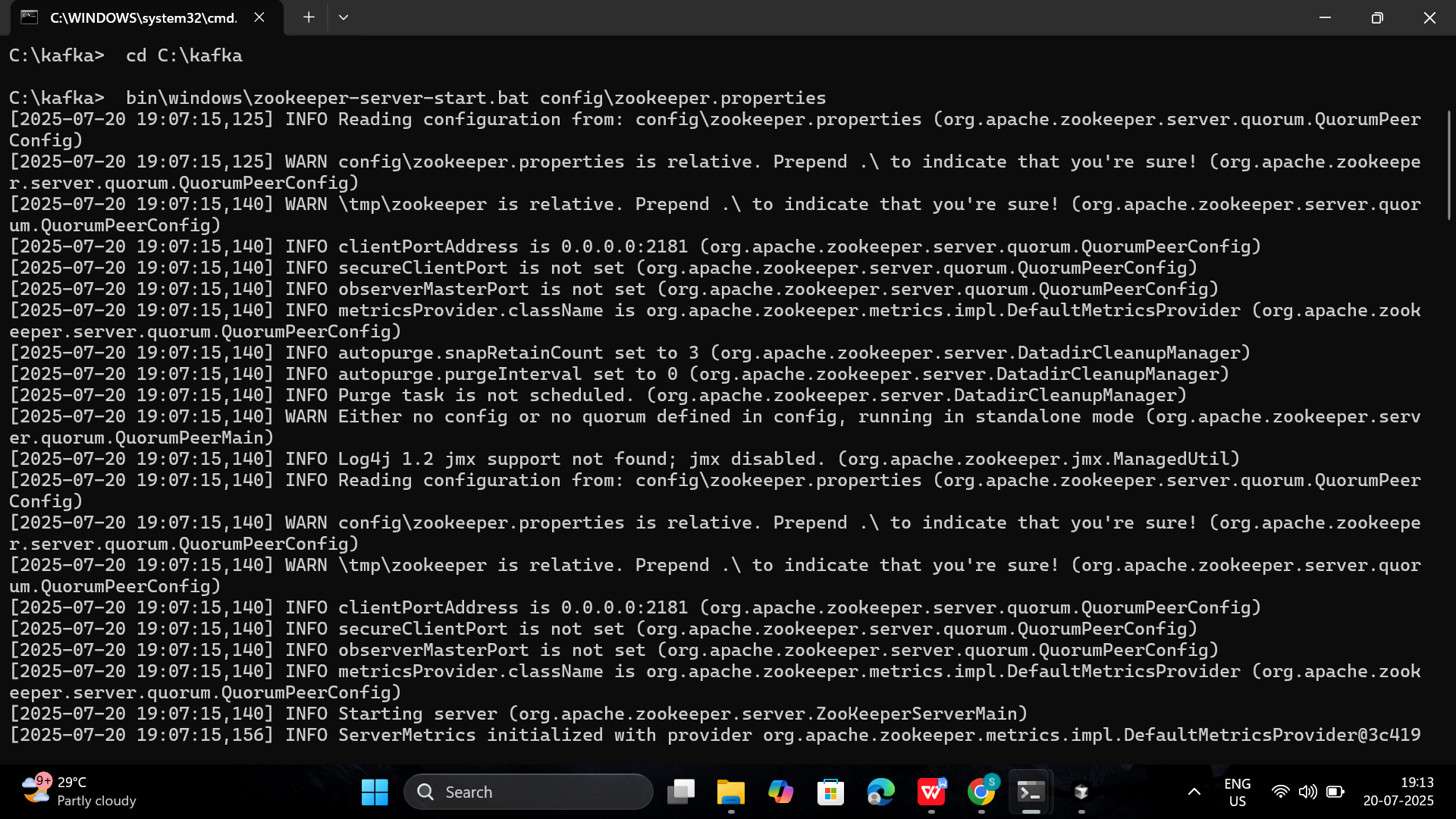
            }

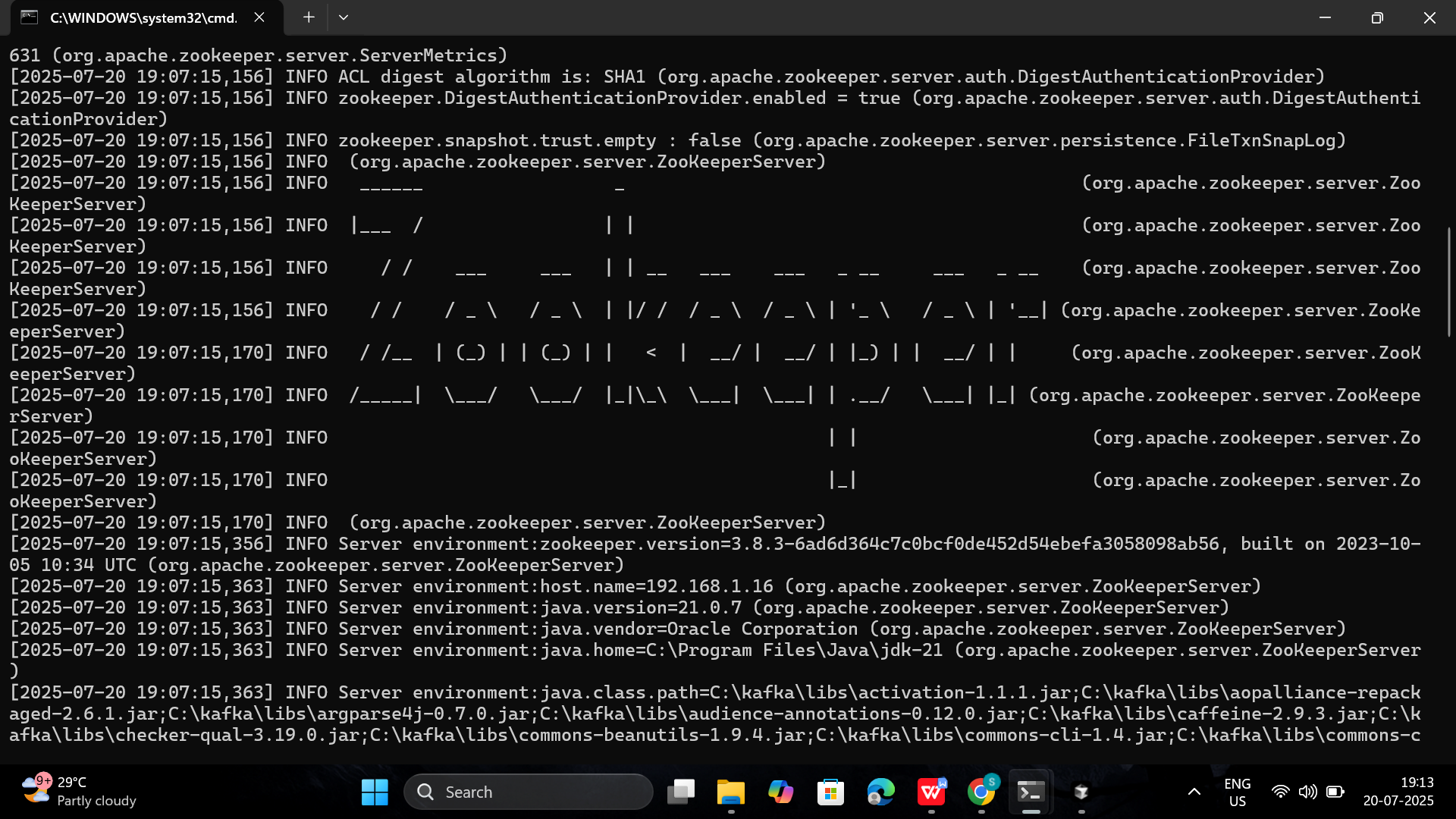
        }

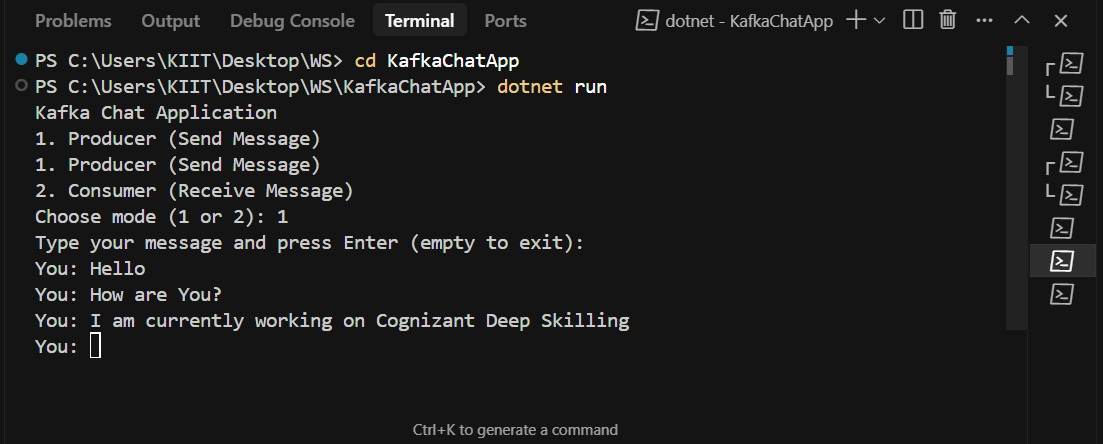
    }

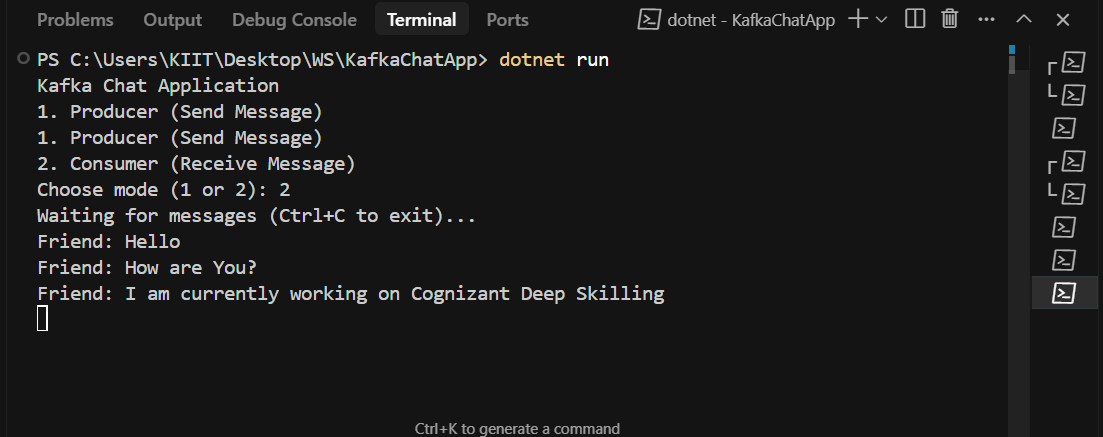
}

**Output:**









Hands-On Exercises: Authentication and Authorization in ASP.NET Core Web API Microservices

This document contains 4 hands-on exercises focusing on Authentication and Authorization in ASP.NET Core Web API microservices, with an emphasis on implementing JWT (JSON Web Tokens) authentication. Each exercise includes a scenario, step-by-step instructions, and complete solution code.

# 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 JWT- based 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);

}

}

**Program.cs**

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.IdentityModel.Tokens;

using System.Text;

var builder = WebApplication.CreateBuilder(args);

*// Add services to the container.*

*// Learn more about configuring OpenAPI at https://aka.ms/aspnet/openapi*

builder.Services.AddOpenApi();

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();

*// Configure the HTTP request pipeline.*

if (app.Environment.IsDevelopment())

{

    app.MapOpenApi();

}

app.UseHttpsRedirection();

app.UseAuthentication();

app.UseAuthorization();

app.MapControllers();

app.MapGet("/ping", () => "pong");

app.Run();

**AuthController.cs**

using Microsoft.AspNetCore.Mvc;

using Microsoft.IdentityModel.Tokens;

using System.IdentityModel.Tokens.Jwt;

using System.Security.Claims;

using System.Text;

using JwtAuthDemo.Models;

namespace JwtAuthDemo.Controllers

{

    [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)

        {

*// For demo: accept any username/password. Replace with real validation.*

            return !string.IsNullOrEmpty(model.Username) && !string.IsNullOrEmpty(model.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);

        }

    }

}

**SecureController.cs**

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

namespace JwtAuthDemo.Controllers

{

    [ApiController]

    [Route("api/[controller]")]

    public class SecureController : ControllerBase

    {

        [Authorize]

        [HttpGet("secret")]

        public IActionResult GetSecret()

        {

            return Ok("This is a protected endpoint!");

        }

    }

}

**Output:**

