# WEB API HANDS ON

Superset ID: 6363229

# 1. Kafka Setup

1. Start Zookeeper

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

Output:

A screen shot of a computer

AI-generated content may be incorrect.

1. Start Kafka Server

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

Output:

A screen shot of a computer

AI-generated content may be incorrect.

1. Create Kafka Topic

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

Output:

A screen shot of a computer

AI-generated content may be incorrect.

# 2. C# Console Producer (Chat Sender)

using System;  
using Confluent.Kafka;  
  
class Program  
{  
 public static async Task Main()  
 {  
 var config = new ProducerConfig { BootstrapServers = "localhost:9092" };  
 using var producer = new ProducerBuilder<Null, string>(config).Build();  
  
 Console.WriteLine("Enter messages to send to Kafka (chat-topic):");  
 while (true)  
 {  
 var msg = Console.ReadLine();  
 if (msg == "exit") break;  
 await producer.ProduceAsync("chat-topic", new Message<Null, string> { Value = msg });  
 }  
 }  
}

Output:

A black screen with white text

AI-generated content may be incorrect.

# 3. C# Console Consumer (Chat Receiver)

using System;  
using Confluent.Kafka;  
  
class Program  
{  
 public static void Main()  
 {  
 var config = new ConsumerConfig  
 {  
 BootstrapServers = "localhost:9092",  
 GroupId = "chat-group",  
 AutoOffsetReset = AutoOffsetReset.Earliest  
 };  
  
 using var consumer = new ConsumerBuilder<Ignore, string>(config).Build();  
 consumer.Subscribe("chat-topic");  
  
 Console.WriteLine("Receiving messages from Kafka (chat-topic):");  
 while (true)  
 {  
 var cr = consumer.Consume();  
 Console.WriteLine($"Received: {cr.Message.Value}");  
 }  
 }  
}

Output:

A black rectangle with white text

AI-generated content may be incorrect.

**MICROSERVICES HANDS ON  
  
Question1: Implement JWT Authentication in ASP.NET Core Web API**

**LoginModel.cs**

namespace Microservices\_Hands\_On.Models

{

public class LoginModel

{

public string Username { get; set; }

public string Password { get; set; }

}

}

**AuthController.cs**

using Microservices\_Hands\_On.Models;

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) =>

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("ThisIsASecretKeyForJwtToken123456"));

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;

namespace Microservices\_Hands\_On

{

public class Program

{

public static void Main(string[] args)

{

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddControllers();

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"]))

};

options.Events = new JwtBearerEvents

{

OnAuthenticationFailed = context =>

{

if (context.Exception.GetType() == typeof(SecurityTokenExpiredException))

{

context.Response.Headers.Add("Token-Expired", "true");

}

return Task.CompletedTask;

}

};

});

builder.Services.AddAuthorization();

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen();

var app = builder.Build();

if (app.Environment.IsDevelopment())

{

app.UseSwagger();

app.UseSwaggerUI();

}

app.UseHttpsRedirection();

app.UseAuthorization();

app.MapControllers();

app.Run();

}

}

}

**Token Genertation:**

