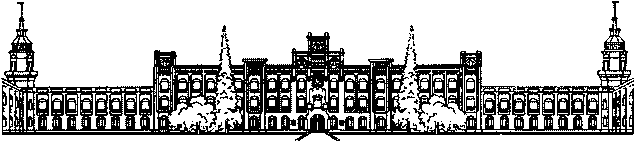
****

Міністерство освіти і науки України

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформаційних систем та технологій

Лабораторна робота №5

з дисципліни

Сучасні методи обробки масивів даних

|  |  |  |
| --- | --- | --- |
| Виконав(ла): |  | Перевірила: |
|  |  |  |
| студент(ка) групи ІМ-42мп: |  | ст. викладач |
| Бардін В. Д. |  | Тимофєєва Ю.С. |
|  |  |  |

Київ 2024

**1 КОД ЗАСТОСУНКУ**

public class Lab5Streams : IStreamTopologyBuilder  
{  
 private const string **InputTopic** = "int.k-connect.csv.plastic-pollution";  
  
 private static JsonSerializerSettings GetJsonSerializerSettings()  
 {  
 return new JsonSerializerSettings  
 {  
 Converters = { new NaIntConverter() },  
 ContractResolver = new DefaultContractResolver  
 {  
 NamingStrategy = new SnakeCaseNamingStrategy(),  
 }  
 };  
 }  
  
 public StreamBuilder BuildTopology(StreamBuilder streamBuilder)  
 {  
 JsonConvert.DefaultSettings = GetJsonSerializerSettings;  
  
 var records = streamBuilder.Stream(InputTopic, new StringSerDes(), new JsonSerDes<PlasticPollutionInfo>())  
 .Filter((\_, v, \_) => v is not null);  
  
 BuildVolunteersCountTopology(records);  
 BuildUkraineTotalCountTopology(records);  
  
 return streamBuilder;  
 }  
  
 private static void BuildVolunteersCountTopology(IKStream<string, PlasticPollutionInfo> records)  
 {  
 const int **minEventsThreshold** = 10;  
 var volunteersCountStream = records  
 .Filter((\_, v, \_) => v.NumEvents < **minEventsThreshold**)  
 .Map<string, int>((\_, v, \_) => KeyValuePair.Create("passed", v.Volunteers))  
 .GroupByKey<StringSerDes, Int32SerDes>()  
 .WindowedBy(TumblingWindowOptions.Of(TimeSpan.FromSeconds(10)))  
 .Aggregate<int, Int32SerDes>(  
 () => 0,  
 (\_, v, agg) => agg + v  
 )  
 .ToStream();  
  
 volunteersCountStream.Print(Printed<Windowed<string>, int>.ToOut());  
 }  
  
 private static void BuildUkraineTotalCountTopology(IKStream<string, PlasticPollutionInfo> records)  
 {  
 const string **ukraineEventsKey** = "Ukraine";  
 var totalCollectedInUkraine = records.MapValues<long>((\_, v, \_) => v.NumEvents)  
 .Filter((k, \_, \_) => k is **ukraineEventsKey**)  
 .GroupByKey()  
 .WindowedBy(TumblingWindowOptions.Of(TimeSpan.FromSeconds(10)))  
 .Aggregate<long, Int64SerDes>(  
 () => 0,  
 (\_, v, agg) => agg + v  
 )  
 .ToStream();  
  
 totalCollectedInUkraine.Print(Printed<Windowed<string>, long>.ToOut());  
 }  
}

**1.2 Код застосунку для збору метрики затримки споживача**

public class Lab5ConsumerLagStreams : IStreamTopologyBuilder  
{  
 private const string **InputTopic** = "int.streaming.plastic.pollution.consumer-lag";  
  
 private static JsonSerializerSettings GetJsonSerializerSettings()  
 {  
 return new JsonSerializerSettings  
 {  
 Converters = { new NaIntConverter() },  
 ContractResolver = new DefaultContractResolver  
 {  
 NamingStrategy = new SnakeCaseNamingStrategy(),  
 }  
 };  
 }  
  
 public StreamBuilder BuildTopology(StreamBuilder streamBuilder)  
 {  
 JsonConvert.DefaultSettings = GetJsonSerializerSettings;  
  
 var records = streamBuilder.Stream(InputTopic, new StringSerDes(), new JsonSerDes<ConsumerLagDescriptor>())  
 .Filter((\_, v, \_) => v is not null)  
 .GroupByKey()  
 .WindowedBy(Of(TimeSpan.FromSeconds(10)))  
 .Aggregate<ConsumerSumCountDescriptor, ConsumerSumCountDescriptorSerDes>(  
 () => new ConsumerSumCountDescriptor(),  
 (\_, v, agg) => new ConsumerSumCountDescriptor(  
 v.ConsumerGroup + v.Topic,  
 agg.Sum + v.Lag,  
 agg.Count + 1  
 )  
 )  
 .MapValues((r, \_) => new ConsumerAverageLagDescriptor(r.Consumer, Math.Floor(r.Sum \* 1.0 / r.Count)))  
 .Suppress(  
 SuppressedBuilder.UntilWindowClose<Windowed<string>, ConsumerAverageLagDescriptor>(  
 TimeSpan.FromMinutes(1),  
 StrictBufferConfig.Unbounded()  
 )  
 .WithValueSerdes(new ConsumerAverageLagDescriptorSerDes())  
 )  
 .ToStream();  
  
 records.To(  
 "consumer-average-lag",  
 new TimeWindowedSerDes<string>(new StringSerDes(), 10\_000),  
 new JsonSerDes<ConsumerAverageLagDescriptor>()  
 );  
  
 return streamBuilder;  
 }  
}  
  
internal sealed record ConsumerSumCountDescriptor(string Consumer = "", long Sum = 0, int Count = 0);  
  
internal sealed record ConsumerAverageLagDescriptor(string Consumer, double Lag);  
  
internal sealed record ConsumerLagDescriptor(string ConsumerGroup, string Topic, long Lag);  
  
internal sealed class ConsumerSumCountDescriptorSerDes : AbstractSerDes<ConsumerSumCountDescriptor>  
{  
 public override byte[] Serialize(ConsumerSumCountDescriptor data, SerializationContext context)  
 {  
 var json = JsonConvert.SerializeObject(data);  
 return System.Text.Encoding.UTF8.GetBytes(json);  
 }  
  
 public override ConsumerSumCountDescriptor Deserialize(byte[] data, SerializationContext context)  
 {  
 return JsonConvert.DeserializeObject<ConsumerSumCountDescriptor>(System.Text.Encoding.UTF8.GetString(data))!;  
 }  
}  
  
internal sealed class ConsumerAverageLagDescriptorSerDes : AbstractSerDes<ConsumerAverageLagDescriptor>  
{  
 public override byte[] Serialize(ConsumerAverageLagDescriptor data, SerializationContext context)  
 {  
 var json = JsonConvert.SerializeObject(data);  
 return System.Text.Encoding.UTF8.GetBytes(json);  
 }  
  
 public override ConsumerAverageLagDescriptor Deserialize(byte[] data, SerializationContext context)  
 {  
 return JsonConvert.DeserializeObject<ConsumerAverageLagDescriptor>(System.Text.Encoding.UTF8.GetString(data))!;  
 }  
}

**Код на стороні споживача, для передачі метрики у кафку**  
var cts = new CancellationTokenSource();  
var lagChannel = Channel.CreateUnbounded<ConsumerLagDescriptor>();  
var lagExporter = new ConsumerLagExporter(lagChannel.Reader);  
\_ = Task.Run(() => lagExporter.StartAsync(cts.Token));

internal sealed class ConsumerLagExporter  
{  
 private readonly ChannelReader<ConsumerLagDescriptor> \_channelReader;  
 private readonly IProducer<string, string> \_producer;  
   
 private const int **ExportIntervalMs** = 100;  
 private const string **ExportTopic** = "int.streaming.plastic.pollution.consumer-lag";  
   
 public ConsumerLagExporter(ChannelReader<ConsumerLagDescriptor> channelReader)  
 {  
 \_channelReader = channelReader;  
 var producerConfig= new ProducerConfig  
 {  
 BootstrapServers = "PLAINTEXT://localhost:19092,PLAINTEXT://localhost:29092,PLAINTEXT://localhost:39092",  
 Acks = Acks.Leader  
 };  
  
 \_producer = new ProducerBuilder<string, string>(producerConfig).Build();  
 }  
  
 public async Task StartAsync(CancellationToken ct)  
 {  
 while (await \_channelReader.WaitToReadAsync(ct))  
 {  
 while (\_channelReader.TryRead(out var lagDescriptor))  
 {  
 try  
 {  
 var message = new Message<string, string>  
 {  
 Key = $"{lagDescriptor.ConsumerGroup.ToLowerInvariant()}-{lagDescriptor.Topic}",  
 Value = JsonSerializer.Serialize(lagDescriptor)  
 };  
  
 await \_producer.ProduceAsync(ExportTopic, message, ct);  
 }  
 catch (ProduceException<string, int> e)  
 {  
 Console.WriteLine($"❌ Kafka Produce Error: {e.Error.Reason}");  
 }  
   
 Console.WriteLine($"✅ Exported Lag: {lagDescriptor}");  
 await Task.Delay(TimeSpan.FromMilliseconds(ExportIntervalMs), ct);  
 Thread.Yield();  
 }  
 }  
 }  
}

internal sealed class KafkaConsumer  
{  
 ...

public void Consume()  
 {  
 ...  
  
 try  
 {  
 while (true)  
 {  
 try  
 {  
 // ... same as before ...  
  
 ExportConsumerLag(consumeResult);  
 }  
 catch (ConsumeException e)  
 {  
 Console.WriteLine($"❌ Kafka Consume Error: {e.Error.Reason}");  
 }  
 }  
 }  
 ...

}  
  
 private void ExportConsumerLag(ConsumeResult<string, string> consumeResult)  
 {  
 var watermarkOffsets = \_consumer.GetWatermarkOffsets(consumeResult.TopicPartition);  
 var committedOffset = consumeResult.Offset;  
  
 var lag = watermarkOffsets.High - committedOffset;  
   
 var lagDescriptor = new ConsumerLagDescriptor(  
 \_consumerGroup,  
 consumeResult.Topic,  
 lag  
 );  
  
 \_lagChannelWriter.TryWrite(lagDescriptor);  
 }  
}

**2 СКРІНШОТИ З РЕЗУЛЬТАТАМИ РОБОТИ**

**A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.**

**3 ВИСНОВКИ**

У цій роботі було використано Kafka Streams для обробки вхідного потоку даних та їх агрегації за допомогою використання операцій зі збереженням стану.