.net core with kafka

.

.

src : https://github.com/psmon/CQRSAkka

: (,)

```
docker network create --driver=bridge --subnet=172.19.0.0/16 devnet

docker network inspect devnet
```

overlay ,

ip .

-

docker-compose.yml

```
version: '3.5'
services:
  zookeeper:
    image: 'bitnami/zookeeper:latest'
    ports:
    - '2181:2181'
    networks:
      devnet:
        ipv4_address: 172.19.0.20
    environment:
    - ALLOW_ANONYMOUS_LOGIN=yes
  kafka:
    hostname: kafka
    image: 'bitnami/kafka:latest'
    ports:
    - '9092:9092'
    networks:
      devnet:
        ipv4_address: 172.19.0.21
    environment:
    - KAFKA_ADVERTISED_HOST_NAME=kafka
    - KAFKA_ZOOKEEPER_CONNECT=172.19.0.20:2181
    - ALLOW_PLAINTEXT_LISTENER=yes

networks:
  devnet:
    external:
      name: devnet
```

,

docker-compose up .

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Threading.Tasks;
using Confluent.Kafka;
using Confluent.Kafka.Serialization;

namespace DDDSample.Adapters.kafka
{
    public class KafkaProduce
    {
        private readonly Dictionary<string, object> config;
        private readonly Producer<Null, string> producer;
        private readonly String topic;

        public KafkaProduce(string server,string _topic)
        {
            config = new Dictionary<string, object>
            {
                { "bootstrap.servers", server },
                { "group.id","kafka_consumer" }
            };

            topic = _topic;


            producer = new Producer<Null, string>(config, null, new StringSerializer(Encoding.UTF8));
        }

        public void Produce(string data)
        {
            producer.ProduceAsync(topic, null, data).Wait();
            producer.Flush(100);
        }

        public async Task ProduceAsync(string data)
        {
            await producer.ProduceAsync(topic, null, data);
        }

        public void Flush(int milisecondTimeOut)
        {
            producer.Flush(milisecondTimeOut);
        }

    }
}
```

? ? .

.

- : ,
- : ,

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
```

```
using System.Threading;
using System.Threading.Tasks;
using Akka.Actor;
using Confluent.Kafka;
using Confluent.Kafka.Serialization;

namespace DDDSample.Adapters.kafka
{
    public class KafkaConsumer
    {
        private string server;
        private string topic;

        private CancellationToken ct;
        CancellationTokenSource tokenSource2;

        public Boolean HasMessage { get; set; }

        public KafkaConsumer(string _server,string _topic)
        {
            server = _server;
            topic = _topic;
            tokenSource2 = new CancellationTokenSource();
            ct = tokenSource2.Token;
        }

        public void Stop()
        {
            tokenSource2.Cancel();
        }

        public Task CreateConsumer(IActorRef consumeAoctor)
        {
            var config = new Dictionary<string, object>
                {
                    {"group.id","kafka_consumer" },
                    {"bootstrap.servers", server },
                    { "enable.auto.commit", "false" }
                };

            Console.WriteLine("kafka StartConsumer ");

            var task = new Task(() => {

                // Were we already canceled?
                ct.ThrowIfCancellationRequested();

                using (var consumer = new Consumer<Null, string>(config, null, new StringDeserializer(Encoding.
UTF8)))
                {
                    consumer.Subscribe(topic);
                    consumer.OnMessage += (_, msg) => {
                        //message(msg.Value);
                        Console.WriteLine(string.Format("kafka msg {0} === {1}",msg.Offset.Value,msg.Value));
                        if (consumeAoctor != null) consumeAoctor.Tell(new KafkaMessage(msg.Topic, msg.Value));
                        HasMessage = true;
                    };

                    while (true)
                    {
                        if (ct.IsCancellationRequested)
                        {
                            // Clean up here, then...
                            ct.ThrowIfCancellationRequested();
                        }
                        consumer.Poll(100);
                        //consumer.CommitAsync();
                    }
                }

            }, tokenSource2.Token);
```

```
            return task;
        }
    }
}
```

, ?

 . .

,

 AkkaStream Flow .

,

    .

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Threading.Tasks;
using Akka.Actor;
using Akka.TestKit;
using Akka.TestKit.NUnit3;
using DDDSample.Adapters.kafka;
using NUnit.Framework;

namespace DDDSampleTest.Kafka
{
    public class KafkaConsumerTest : TestKit
    {
        KafkaProduce kafkaProduce;
        KafkaConsumer kafkaConsumer;
        TestProbe probe;

        [SetUp]
        public void Setup()
        {
            kafkaConsumer = new KafkaConsumer("kafka:9092", "test_consumer");
            probe = this.CreateTestProbe();
            kafkaConsumer.CreateConsumer(probe).Start();

            kafkaProduce = new KafkaProduce("kafka:9092", "test_consumer");
        }

        [Test]
        public void ProduceAndConsumerTest()
        {
            kafkaProduce.Produce("SomeMessage");

            Within(TimeSpan.FromSeconds(3), () => {

                AwaitCondition(() => probe.HasMessages);

                probe.ExpectMsg<KafkaMessage>(TimeSpan.FromSeconds(0));

                KafkaMessage lastMessage = probe.LastMessage as KafkaMessage;

                Assert.AreEqual("SomeMessage", lastMessage.message);

            });
        }
    }
}
```

, .

,

, ?

.

" " IT . - : MessageDeliveryReliability

, , .