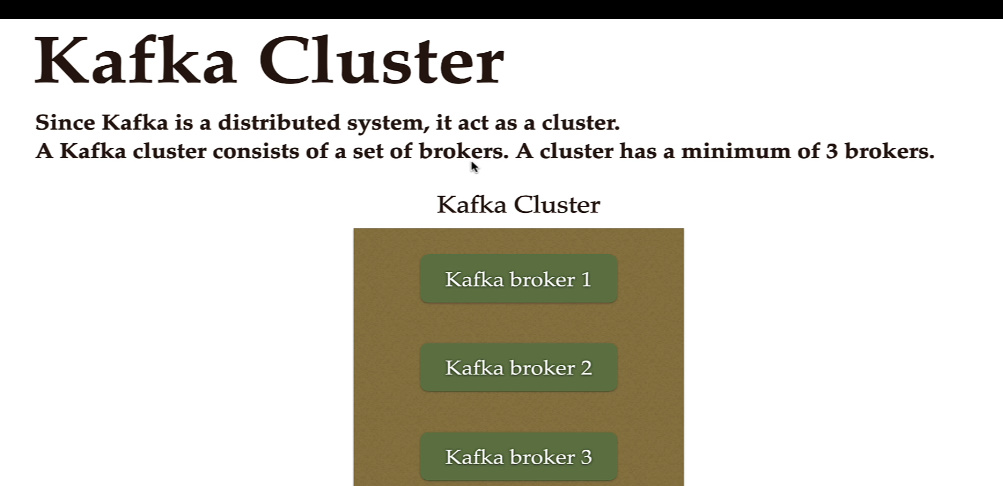


* Kafka este un distributed system, deci se comporta ca un Cluster
* Cluster are unul sau mai multe brokers, si daca unul cade, restul inca lucreaza
* Ar fi bine sa avem macar vreo 3 brokers
* Zookeper este un service. Acest service este cel care administreaza brokers la Kafka, caci el lucreaza cu Producer si Consumer
* Kafka ne ofera **event streaming.**
* **Event streaming** – practica de a colecta date reale de surse de eventuri, ca baze de date, senzori, cloud services etc. si forma de streams de eventuri si de a le stoca in mod durabil pentru o prelucrare mai tarzie in tehnologiile necesare.

**Core conepts**

* **Kafka cluster**



Kafka cluster contine mai multe Kafka brokers, pe diferite servere.

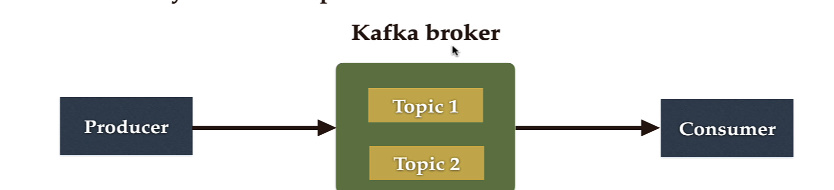
* **Kafka broker**



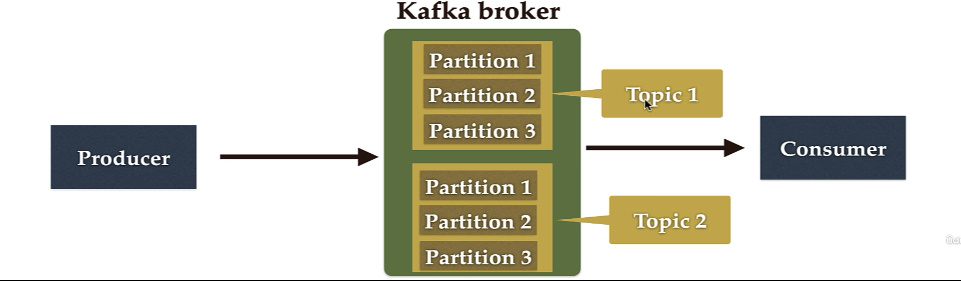
**- Kafka broker** – este numele oferit pentru un kafka server. Este doar un nume sugestiv, caci tot ce face Kafka este de a actiona ca un message broker intre producer si consumer

- Producer si consumer nu interactioneaza direct, dar folosesc kafka server ca un agent sau broker pentru a face schimb de mesaje

* **Producer** – aplicatie care trimite mesage. Nu le trimite direct la consumer, ci doar catre kafka server
* **Consumer** – aplicatie care primeste si citeste mesajele de la Kafka server. El nu primeste mesajele direct de la producer, ci de la kafka server, caci kafka server le stocheaza, si consumer va cere aceste date de la kafka server
* **Kafka Topic –** este o structura(categorie de date) care stocheaza datele sau mai bine zis mesajele venit de la producer si ofera acces la consumer sa le citeasca. Topic e ca un tabel intr-o baza de date sau ca un folder in system. Fiecare topic are une nume.Topics sunt deobicei create pentru diferite categorii de mesaje. Un kafka broker poate avea oricate topics

****

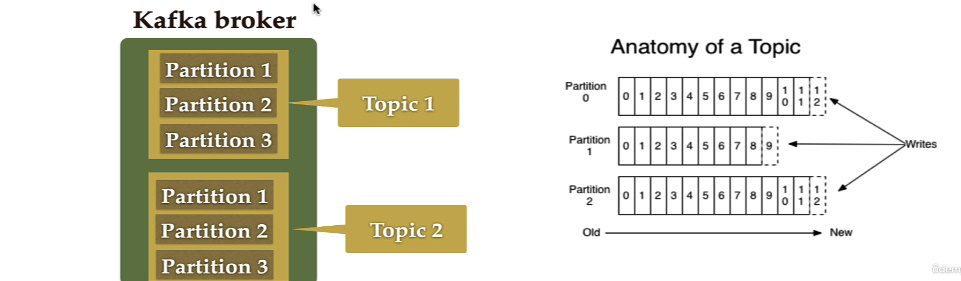
* **Kafka Partitions**



Kafka topics sunt divizate intr-un numar de partitii, care contin inregistrari intr-o unchangable sequence

Kafka Brokers stocheaza messagele intr-un topic. Dar, capacitatea datelor poate fi enorma si ar putea sa nu fie posibil sa fie stocate pe un singur PC. Dar, un topic poate fi partitionat in multiples parti si distribuit printre mai multe calculatoare, asa cum kafka este un distributed system si suporta asta.

* **Offset –** id atribuit unui mesaj din partitie. Odata ce offset este atribuit, el nu se va mai modifica niciodata. Primul mesaj primeste id 0, al 2 id 1 si tot asa. Fiecare mesaj inserat intr-o partitie va avea un id cu +1 mai mare ca cel precedent. Vedem ca 10 e scris vertical, nu orizontal.



Deci, id nu e altceva decat un fel de index

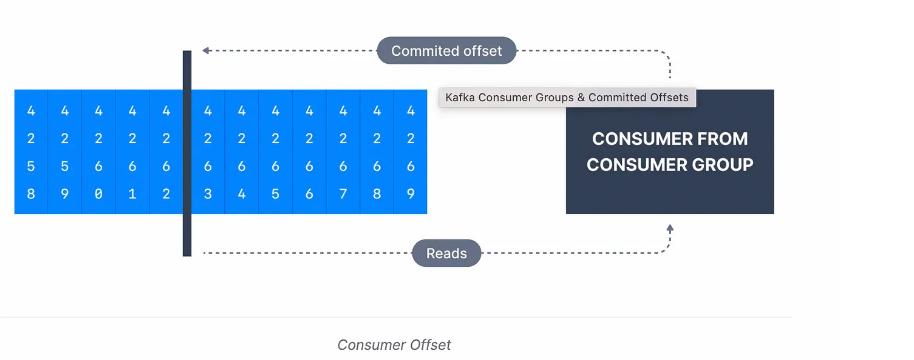
- Fiecare partition din topic are offsets ale sale, care apartin la mesaje.

- Avem 2 tipuri de offset:

**current(consumer) offset** – este offset pe care urmatorul consumer dintr-un group il va consuma. Atentie! Fiecare consumer group are propriile sale current si commited offset. Grupurile de consumers nu impartasesc current si commited offset, ci lucreaza complet independent!

**commited offset** – este offset care a fost ultimul consumat **cu succes** de catre vreun consumer din group si marcat ca consumat cu succes. Current offset nu garanteaza ca messages de pana la el au fost cu succes consumate, dar commited da.

If the current offset is 5 and the committed offset is 2, the next record that will be consumed by the consumer in the consumer group will be the record with offset 3

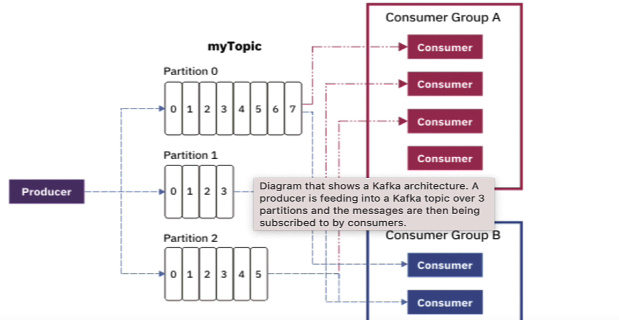


- asa, offsets se asigura ca mesajele dintr-o partition sa fie procesate in ordinea in care ele vin, indiferent de consumer group ce le acceseaza.

- offsets permit la consumers sa preia mesajul ce urmeaza, in ordine corecta, chiar de un consumer da fail. Cand un consumer preia un message, el da commit la offset si daca cade, urmatorul stie unde s-a oprit acela

- daca chiar se intampla ca vreun mesaj trebuie reprocesat, putem liber sa mutam offsets

* **Consumer Group** – grup de consumers care lucreaza impreuna pentru a procesa mesajele



**Instalare**

**STEP 1: DOWNLOAD AND INSTALL KAFKA AND CHANGE THE NAME TO BE SHORT, LIKE “Kaka:, OR AN ERROR WILL OCCUR**

https://dlcdn.apache.org/kafka/3.2.0/kafka\_2.13-3.2.0.tgz

**STEP 2: START THE KAFKA ENVIRONMENT TO MANAGE THE CLUSTER. ENTER IN CD IN KAFKA FOLDER**

**# Start the ZooKeeper service**

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

**# Start the Kafka broker service**

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

**STEP 3: CREATE A TOPIC TO STORE YOUR EVENTS**

.\bin\windows\kafka-topics.bat --create --topic topic\_demo --bootstrap-server localhost:9092

**STEP 4: WRITE SOME EVENTS INTO THE TOPIC**

.\bin\windows\kafka-console-producer.bat --topic topic\_demo --bootstrap-server localhost:9092

>hello world

>topic demo

**STEP 5: READ THE EVENTS**

.\bin\windows\kafka-console-consumer.bat --topic topic\_demo --from-beginning --bootstrap-server localhost:9092

hello world

topic demo

**Project**

* Cand cream project in spring, alegem si kafka ca dependency

**Configure Kafka producer and consumer**

1. **Cream consumer si producer:**

spring.kafka.bootstrap-servers=localhost:9092

asa adaugam un server, dar daca am avea mai multe, am folosi , si am pune pe toate, gen: server1,server2,server3.... si se va folosi unul disponibil din ele mereu, adica un message broker disponibil, caci asta si punem aici

spring.kafka.consumer.group-id=myGroup

asa cream un group de consumers, si i oferim un id, ce trebuie sa fie unic. Daca asa grup deja exista, nu e nimic.

spring.kafka.consumer.auto-offset-reset=earliest

aceasta proprietate spune ce se intampla daca un group de consumers incepe a prelua mesajele dintr-o partition, dar el niciodata nu dat commit la vreun offset, sau offset la care a dat commit a fost cumva sters.

- earliest – incepe de la offset 0

- latest – incepe de la urma cu cele noi. Daca avem 4 offsets(0,1,2,3), latest inseamna sa inceapa cu 4, adica cel nou care va fi

- none – nu incepe fara un commited offset si arunca exceptie

spring.kafka.consumer.key-deserializer=org.apache.kafka.common.serialization.StringDeserializer

spring.kafka.consumer.value-deserializer=org.apache.kafka.common.serialization.StringDeserializer

proprietatea data spune cum trebuie sa fie deserializat key la datele consumate de un consumer ce vin de la Kafka. Cand consumam un message de la un topic, fiecare mesaj e format din key : value. Key si value sunt serializate inainte de a fi trimise la consumer si trebuie deserializate ca sa le putem folosi in aplicatie. Deserializarea consta in a transforma serialized bytes inainpoi in datele originale cum erau ele. SpringDeserializer va converti key ce vine la consumer in string simplu

**key vs value:**

value este mesajul trimis la consumer, dar key este ceva optional, folosit de kafka pentru a determina in ce partition sa puna mesajul. The key is an optional piece of data associated with a message. It is used to determine the partition to which the message will be written within a topic. Kafka guarantees that all messages with the same key will go to the same partition. This is often used to ensure that related data is kept together to allow for ordered processing or grouping. The key can be of any type and is typically used for scenarios where you want to maintain some sort of relationship or grouping between messages.

Observam ca folosim doar deserialization, caci asta va trebui facut la consumer, sa deserializam mesajul

spring.kafka.producer.bootstrap-servers=localhost:9092

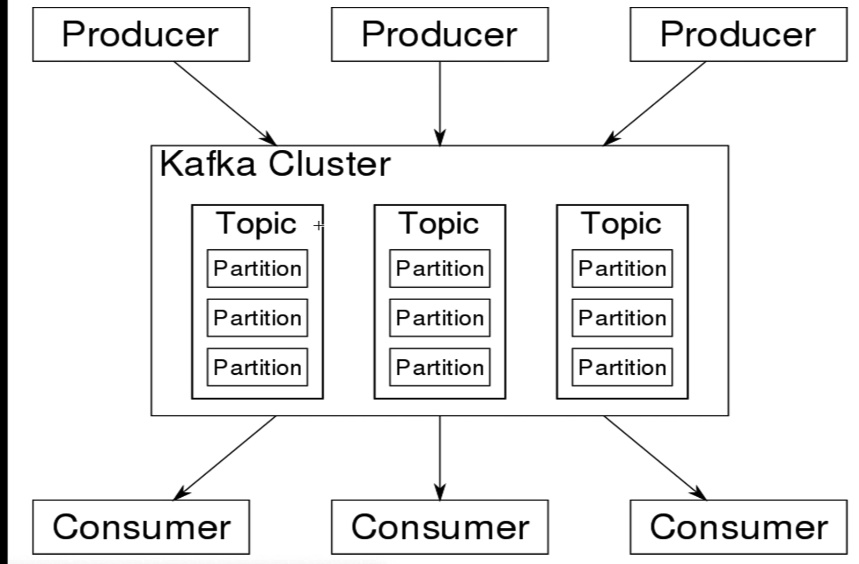
aici cream producer pentru acest server kafka.

spring.kafka.producer.key-serializer=org.apache.kafka.common.serialization.StringSerializer  
spring.kafka.producer.value-serializer=org.apache.kafka.common.serialization.StringSerializer

observam ca deja folosim serializer pentru producer, ca el sa creeze mesaje si sa le serializeze

1. **Cream topic**

In Kafka message broker putem crea oricate topics. In imaginea de mai jos, kafka ruleaza local si deci kafka cluster are doar un broker



Nu trebuie sa cream si partitiile manual, caci spring boot ne ofera deja solutie

@Configuration  
public class KafkaTopicConfig {  
   
 @Bean  
 public NewTopic createTopic1(){  
 return TopicBuilder.*name*("topic1")  
 .partitions(10)  
 .build();  
 }

Daca o asemenea partitie exista, se va crea doar o referinta la ea, daca nu, se va crea partitia.

1. **Cream producer**

@Service  
@RequiredArgsConstructor  
@Log4j  
public class KafkaProducer {  
 private final KafkaTemplate<String, String> kafkaTemplate;  
  
 public void sendMessage(String message){  
 kafkaTemplate.send("topic1",message);  
 *log*.info(String.*format*("Message sent: %s",message));  
 }  
}

Producer este un @Service. Kafka deja ne ofera un KafkaTemplate pe baza la ceea ce am creat in application.properties, si metoda send() are ca prim argument numele la topic dat in @Configuration si mesajul propriu zis de trimis.

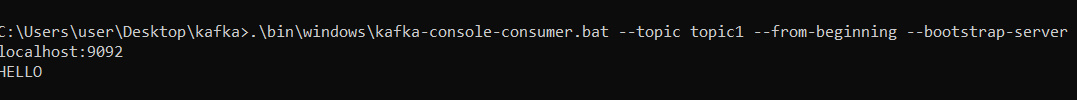
1. **Trimitem un mesaj folosind producer creat**

@RestController  
@RequestMapping("/api/b1/kafka")  
@RequiredArgsConstructor  
public class MessageController {  
 private final KafkaProducer kafkaProducer;  
  
 @GetMapping("/publish")  
 public ResponseEntity<String> publish(@RequestParam("message") String message){  
 kafkaProducer.sendMessage(message);  
  
 return new ResponseEntity<>("Message sent", HttpStatus.*OK*);  
 }  
}

Dupa ce facem un get, scriem

.\bin\windows\kafka-console-consumer.bat --topic **topic1** --from-beginning --bootstrap-server localhost:9092

Si vedem mesajul HELLO trimis in link aici



1. **Cream Consumer**
2. @Service  
   @RequiredArgsConstructor  
   public class KafkaConsumer {  
    @KafkaListener(topics = "topic1", groupId = "myGroup")  
    public void consumer1(String message){  
    System.*out*.println("Consumed message by consumer1: "+message);  
    }  
    @KafkaListener(topics = "topic1", groupId = "myGroup")  
    public void consumer2(String message){  
    System.*out*.println("Consumed message by consumer2: "+message);  
    }  
   }

Si am creat un subscriber. Putem crea oricate subscribe cu @KafkaListener

Un group consumer deja l-am creat in application.properties, si acel id il dam la groupId

Desi am creat 2 consumers, unul din ele va prelua mesajul.

**JSON Serializer and Deserializer**

* Kafka defapt stocheaza si transporta byte[]
* Sunt cateva built-in serializers si deserializers, dar ele nu includ JSON
* Spring Kafka totusi a creat un JsonSerializer si JsonDeserializer pentru a converti obiectele din si in JSON, dar ele tot lucreaza cu byte[] intern

spring.kafka.consumer.key-deserializer=org.apache.kafka.common.serialization.StringDeserializer  
spring.kafka.consumer.value-deserializer=org.springframework.kafka.support.serializer.JsonDeserializer

spring.kafka.producer.value-serializer=org.apache.kafka.common.serialization.StringSerializer  
spring.kafka.producer.key-serializer=org.springframework.kafka.support.serializer.JsonSerializer

La key lasam StringDeserializer si StringSerializer, caci nu lucram noi cu key

Insa, la value am pus JsonSerializer, dar atentie, el nu e de la apache.kafka library, de la springframework.library. Kafka nu are asa ceva, dar Spring ne ofera

Mai adaugam si asta:

spring.kafka.consumer.properties.spring.json.trusted.packages=\*

asta va face ca orice clasa sa poata fi serializata si deserializata

Acum, vom crea un DTO:

@Getter  
@Setter  
@NoArgsConstructor  
@AllArgsConstructor  
@Builder  
@ToString  
public class User {  
 private int id;  
 private String firstName;  
 private String lastName;  
}

Acum un producer ce sa lucreze cu User

@Service  
@RequiredArgsConstructor  
public class KafkaProducerJson {  
 private final KafkaTemplate<String, User> kafkaTemplate;  
  
 public void sendUser(User user){  
 Message<User> message = MessageBuilder  
 .*withPayload*(user)  
 .setHeader(KafkaHeaders.*TOPIC*,"topic1")  
 .build();  
  
 kafkaTemplate.send(message);  
  
 System.*out*.println("User sent: "+user);  
 }  
}

desi, putem si fara Message, sa dam direct

@Service  
@RequiredArgsConstructor  
public class KafkaProducerJson {  
 private final KafkaTemplate<String, User> kafkaTemplate;  
  
 public void sendUser(User user){  
  
 kafkaTemplate.send("topic1",user);  
  
 System.*out*.println("User sent: "+user);  
 }  
}

Si acum RestController:

@RestController  
@RequestMapping("/api/v1/kafka")  
@RequiredArgsConstructor  
public class JsonMessageController {  
 private final KafkaProducerJson kafkaProducerJson;  
  
 @PostMapping  
 public ResponseEntity<String> publish(@RequestBody User user){  
 kafkaProducerJson.sendUser(user);  
  
 return ResponseEntity.*ok*("Success sent!");  
 }  
}

Si consumer:

@Service  
@RequiredArgsConstructor  
public class KafkaConsumer1 {  
 @KafkaListener(groupId = "myGroup",topics = "topic1")  
 public void getUser(User user){  
 System.*out*.println("Received user by Consumer1: "+user);  
 }  
}

Daca vom avea mai multi consumers in acest group consumer, tot unul doar se va ocupa de message.

**Atentie! Fiecare group de consumers se ocupa doar de un tip de obiect! De ex, daca mai cream un consumer pentru myGroup din topic1, dar care ia ca argument nu User, ci String, vom avea eroare!!!**

* Pentru a externaliza numele la topics, doar cream o propriety in application.properties si o folosim cu @Value. Cand vom folosi @KafkaLitener, vom folosi la topic=”${}” si va merge. Asa facem si u group id

[Intro to Apache Kafka with Spring | Baeldung](https://www.baeldung.com/spring-kafka)

**@KafkaHandler**

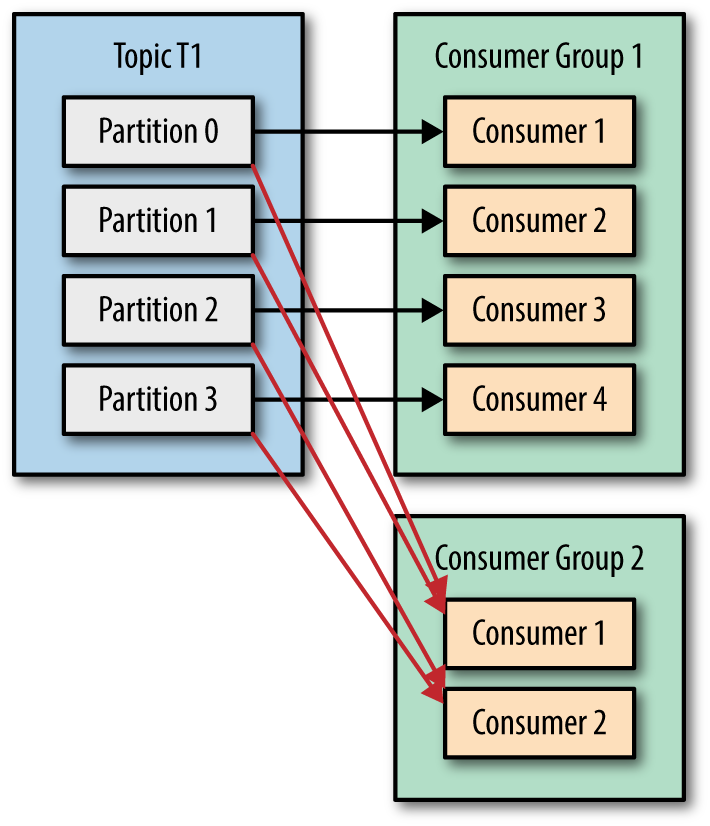
* Apare problema cand avem producers ce produc obiecte diferite, de ex unul producer User, altul UserDto, altul String
* Daca pur si simplu vom crea 3 @Service, adica 3 consumers, si fiecare se va ocupa de a consuma un mesaj, vom primi exceptie, caci se ia un consumer disponibil la alegere, si de ex daca producer producer un Useer, s-a putea lua Consumer pentru UserDTO si vom avea exceptie
* Solutia este de a anota o clasa cu @KafkaListener si fiecare metoda sa fie un consumer ce sa se ocupe de a consuma un obiect diferit, si sa aiba metoda @KafkaHandler. Asa, cream **un consumer** ce poate consuma mai multe obiecte si gata.
* @Service  
  @KafkaListener(id = "listener1",groupId = "myGroup1", topics = "topic2")  
  public class KafkaHandlerService {  
    
   @KafkaHandler  
   public void getUser(@Payload User user){  
   System.*out*.println("Received user by Consumer1: "+user);  
   }  
    
   @KafkaHandler  
   public void getMessage(@Payload String string){  
   System.*out*.println("Consumed by string consumer: "+string);  
   }  
   @KafkaHandler(isDefault = true)  
   public void getObject(@Payload Object string){  
   System.*out*.println("Consumed by string consumer: "+string);  
   }  
  }

@Payload il folosim doar cand avem mai multe argumente, pentru a arata care anume va fi obiectul propriu zis din mesaj, deci mai sus nu e necesar

isDefault = true – se pune cand vrem sa cream un handler default pentru obiectele negasite

**Consumer Group**

* Daca un group are mai multe consumers, fiecare consumer va citi o partitie din topic.
* De ex, daca sunt 4 partitii si 4 consumers :



Vedem ca fiecare partition e citita doar de un consumer per grup! Un consumer poate citit mai multe partitii, daca e necesar.