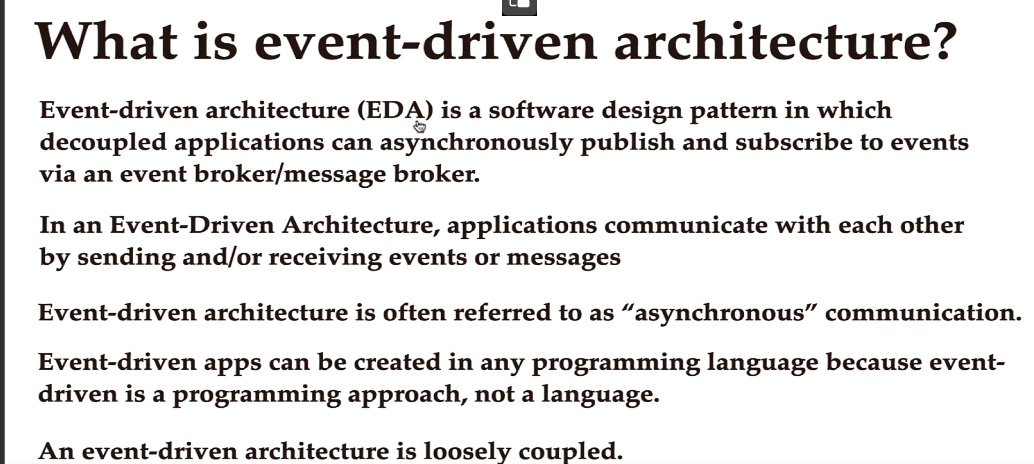
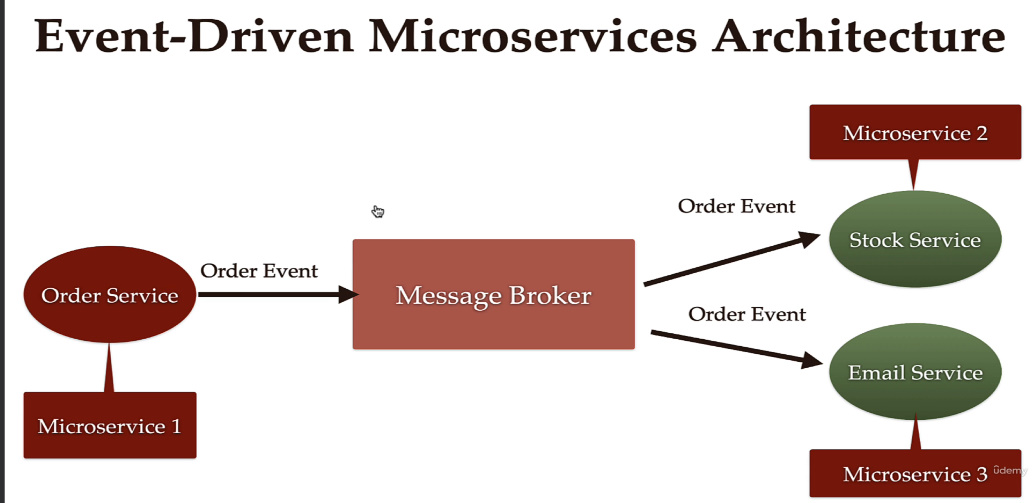
**Event-driven architecture**







**Event-Driver Architecture** – design pattern din software in care microserviciile(aplicatiie) comunica in mod asynch prin Message Broker

* In imagine, e clar ca Order Service va trimite un event(message) la Message Broker si nu va astepta niciun raspuns, il da pe mana la message broker.
* In plus, in event-driven arch producers nu cunosc nimic despre consumers si invers.

**Avantajele la Event driven architecture**

* Flexibilitate
* Microservices sunt complet independente
* Scalability – putem adauga oricand alte noi microservices
* Availability – daca un microservice cade, nu le influenteaza nicidecum pe celelalte
* Daca nu e niciun consumer activ, message nu va fi pierdut nicaieri. Cand un consumer va fi activ, el va fi trimis lui. Deci, daca un user face un request, request service preia request, dar daca nu e niciun alt server sa il proceseze, acel request se pierde. Cu message broker, asa ceva nu mai e posibil

**Autocreating partitions**

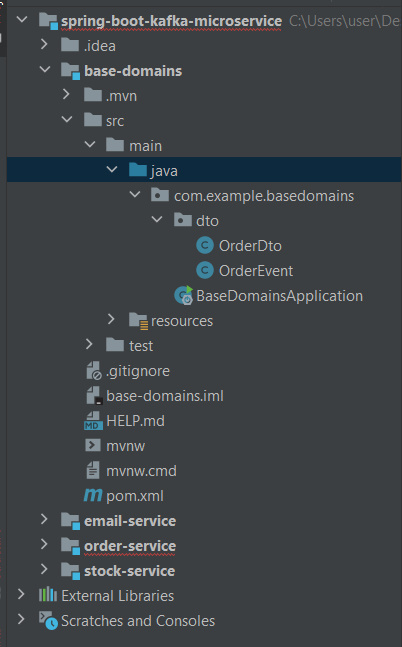
* Daca nu cream noi partitii in topic, Kafka automat va crea:

2023-08-09T23:30:32.501+03:00 INFO 10412 --- [ntainer#0-0-C-1] o.s.k.l.KafkaMessageListenerContainer : **stock: partitions assigned: [order\_topics-0]**

* Vedem in log ca numele partitiei create e numele la topic + “-0”, asa e conventia la kafka

**Project**

* Cand lucram cu microservices, de multe ori trebuie sa punem clase identice in mai multe services, ca DTO de ex. Asta nu e bine de loc
* Cea mai buna solutie este de a crea un microservice sau mai bine zise module separat ce sa contina doar clasele pe care mai multe microservicii le vor avea.
* De ex, fie base-domains acel microservice cee va contine doar clase comune si gata:



Acum, de ex, in Order-service vom avea nevoie de OrderEvent, dar daca facem asa

@Service  
@RequiredArgsConstructor  
public class OrderProducer {  
 private final NewTopic topic;  
 private KafkaTemplate<String, OrderEvent> template;  
}



vom primi eroare, caci OrderEvent nu e gasit.

Pentru a rezolva problema, base-domains ca dependenta in order-service si in celelalte si gata.

* Dar, apare intrebarea care e groupId si artifactId pentru base-domains. Pentru asta, intram in pom.xml de la base-domains si ne uitam aici:

<parent>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-parent</artifactId>  
 <version>3.1.2</version>  
 <relativePath/> <!-- lookup parent from repository -->  
</parent>  
<groupId>com.base-domains.project</groupId>  
<artifactId>base-domains</artifactId>  
<version>0.0.1-SNAPSHOT</version>  
<name>base-domains</name>  
<description>Demo project for Spring Boot</description>  
<properties>  
 <java.version>17</java.version>  
</properties>



Astea si sunt coordonatele la dependenta, deci in order-service vom pune asta:

<dependency>  
 <groupId>com.base-domains.project</groupId>  
 <artifactId>base-domains</artifactId>  
 <version>0.0.1-SNAPSHOT</version>  
</dependency>

si gata, putem folosi tot din base-domains

* Producer:
* @Service  
  @RequiredArgsConstructor  
  public class OrderProducer {  
   private final Logger logger = Logger.*getLogger*(OrderProducer.class.getName());  
   private final NewTopic topic;  
   private final KafkaTemplate<String, OrderEvent> kafkaTemplate;  
    
   public void sendMessage(OrderEvent orderEvent){  
   logger.info(String.*format*("Order event => %s",orderEvent));  
    
   Message<OrderEvent> message = MessageBuilder  
   .*withPayload*(orderEvent)  
   .setHeader(KafkaHeaders.*TOPIC*,topic.name())  
   .build();  
    
   kafkaTemplate.send(message);  
    
   }  
  }