Apache Kafka Nedir?

Sadece bir kaynak sisteminiz ve bir hedef sisteminiz olduğunda, sistemler arası veri transferi yapmanız gerektiğinde çözüm basittir.
Ancak birçok kaynak, birçok hedef sisteminiz olduğunda ve hepsinin birbirleriyle veri transferi yapması gerektiğinde işler karmaşık hale gelecek ve herbir sistemin birbiri ile entegrasyonu gerekecektir.
Bu entegrasyonlar;

Protokol seçimi, verilerin nasıl aktarılacağı (TCP, HTTP, FTP, JDBC, REST, SOAP vb), veri formatı, verilerin nasıl parse edileceği (JSON, CSV, XML, Avro, Thrift vb) gibi birçok

zorluğu beraberinde getirir. Bunlara ek olarak bir kaynak sisteme yapılan her entegrasyon o sistem üzerinde ek yük yaratacaktır.

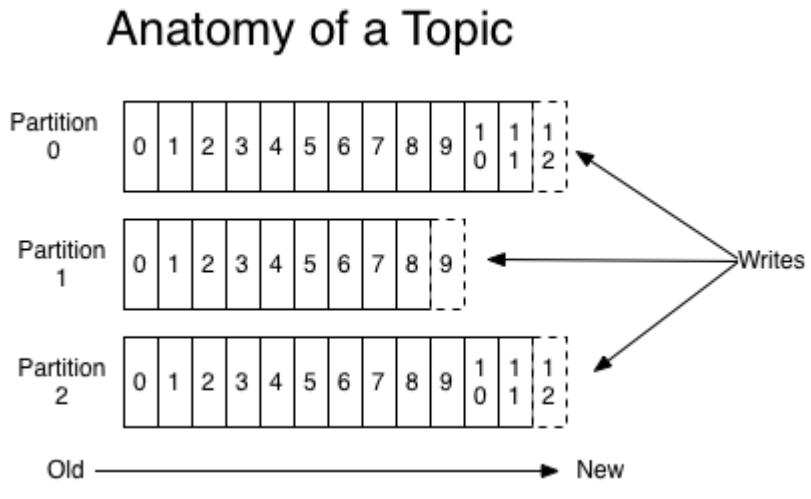
Apache Kafka bu sorunları aşmak için doğru çözüm olacaktır.
Sistemlerin birbirlerine bağımlılıklarını ortadan kaldırarak, üzerlerindeki yüklerini de düşürür.

Apache Kafka, LinkedIn tarafından geliştirilmiş, şu an Apache yönetiminde açık kaynak olarak çoğunlukla Confluent şirketi tarafından bakımı ve geliştirimi yapılan bir projedir.
Dağıtık (distrubuted) bir veri akış (streaming) platformudur.
Hataya dayanıklı, yatay olarak ölçeklenebilen, esnek bir mimariye sahiptir. Son derece yüksek performans ile bir sistemden diğer sisteme 10 ms'den az bir gecikme(latency) ile neredeyse gerçek zamanlı olarak veri transferini mümkün kılmaktadır.
Mesajlaşma sistemi (messaging system) olabilir, etkinlik takibi(activity tracking) için, uygulama loglarını toplamak için, sağladığı API ile stream processing amacıyla kullanılabilir.
Big Data entegrasyonları için kullanılabilir.
Gerçek zamanlı öneriler, kararlar ve bilgiler(insights) oluşturmak için kullanılabilir.

Apache Kafka ile ilgili temel kavramlar:

Topics, Partitions, Offsets

Topic kullanıcı tanımlı kategori ismidir.
Mesajların tutulduğu yerdir. Veritabınındaki tabloya benzer. Kafka içerisinde birçok topic olabilir, isim ile tanımlanırlar.
Bir veya birden fazla bölümden(partitions) meydana gelirler. Her partition'ın 0 dan başlayan numaraları vardır.
Resimde bir topic üzerinde 3 partitions (0, 1 ve 2) görüyoruz.
Mesajlar partition'lara sıralı olarak ve değiştirilemez olarak eklenirler ve artan şekilde bir kimlik değeri alırlar. Bu değere **offset** denir.
Bir mesajın partition ve offset değeri değişmez. Mesaj okuma işleminden sonra kaybolmaz, tekrar erişim mümkündür. Topiclerin bu partition özelliği sayesinde yazma ve okuma işlemleri paralel bir şekilde yapılabilir.
Topic'ler oluşturulurken kaç partition olacağı belirlenir, istenirse sonradan da değiştirilebilir.
Kafka'da mesajların sırası partition içinde garanti edilir. Yani partition 0 daki 4 nolu offset değerine sahip bir mesaj, partition 0 daki 5 no lu offset değerine sahip mesajdan önce yazılmıştır ve önce okunacaktır, Kafka bunu garanti eder. Partition 1 deki 8 no lu offset değerine sahip mesaj ise daha önce yazılmış olabilir. Partition'lar arası mesaj sırası için garanti verilmez.



Kafka'da veriler sadece sınırlı bir süre saklanırlar. Varsayılan olarak 1 hafta.
Sonrasında bu veriler ve offset değerleri silinecektir.
Topic'e eklenen yeni mesajlar bir kural (anahtar) belirtilmemişse rastgele bir partition'a atanır, her zaman partition'ın sonuna eklenir ve eklendikçe offset sürekli artmaya devam eder, 0 a hicbir zaman dönmez.

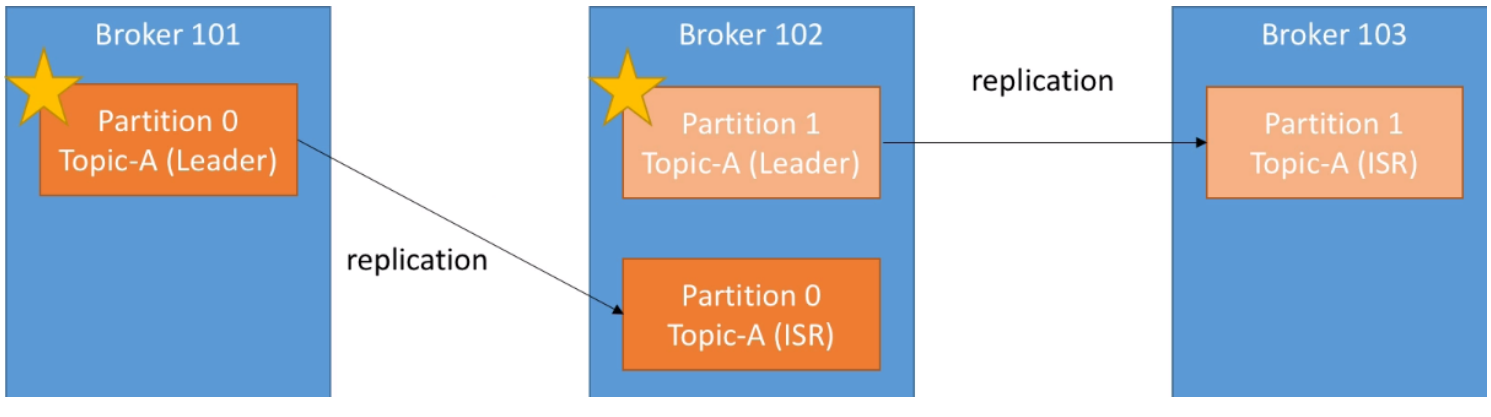
Brokers

Broker, topic ve partition'ları tutan sunuculardır.
Birçok broker birlikte kafka cluster'ı oluşturur. Bir broker'a bağlandığınızda, buna "**bootstrap broker**" denir ve tüm cluster'a bağlanmış olursunuz.
Her broker sadece belirli topic partitionlarını içerir. Yani tüm veriyi tutmaz, çünkü Kafka dağıtık bir yapıdadır.

3 broker'dan oluşan bir kafka cluster'a sahip olduğumuzu düşünelim.
İsimleri Broker 1, Broker 2 ve Broker 3 olsun.
3 partition'dan oluşan *transaction-events* isminde bir topic oluşturduğumuzda, herbir broker topic'in bir partition'ını tutacak şekilde dağılım yapılır.
transaction-events topic'i 5 partition'dan oluşsaydi 2 broker'a 2 partition, 1 broker a 1 partition tutacak şekilde dağılım yapılırdı.

Topic Replication

Kafka dağıtık(distrubuted) bir sistemdir, bir broker çökse bile veri kaybı olmaması ve işlemlerin devam ediyor olması gereklidir. Replikasyon bu işi yapar.
Replication Factor genellikle 2 yada 3 olarak belirlenir, 2 olarak belirlenmesi biraz risklidir.
Aşağıdaki resimdeki gibi 3 broker dan oluşan bir kafka cluster yapımız olduğunu düşünelim.
2 partition'dan oluşan Topic-A'i yaratırken, ayrıca replike olmasını istiyoruz. Bu nedenle replication factor bilgisini 2 olarak set ediyoruz.
Partition 0, broker 101'e atanıyor, partition 1 ise broker 102'ye atanıyor.
Replication factor 2 olarak set edildiği için partition'ların birer kopyası farklı bir broker üzerinde daha tutulacaktır. Resimdeki gibi bir dağılım oluşacaktır.
Broker 102 yi kaybettiğimizi düşünelim, bu durumda veriler 101 ve 103 te olduğu için kayıp olmayacak, işlemler devam edecektir.



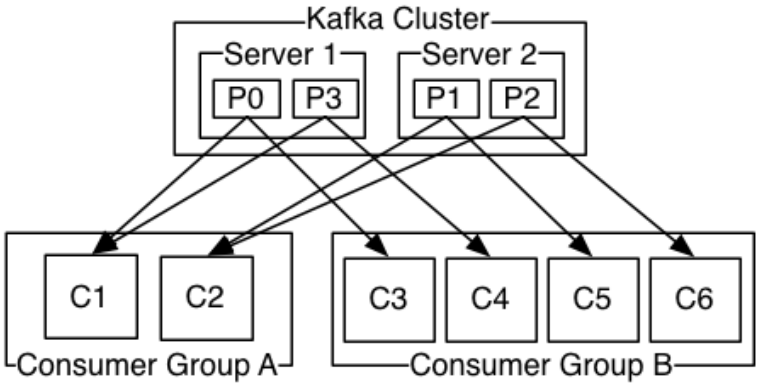
Kafka'da belirli bir zamanda bir partition için sadece bir lider(Leader) kavramı vardır. Lider olan broker veriyi alır ve sunar, diğer brokerlar pasif kopyalar olur, sadece verileri senkronize ederler. Yani her partition için bir lider, birçok ISR(in-sync replica) olur. Lider ve ISR'lara karar veren Zookeeper'dir.

Producer & Message Keys

Topic'lere veriyi yazarlar. Hangi broker ve partitiona yazacağını bilirler, cluster içerisindeki bir broker'a bağlanması yeterlidir.
Broker çökse bile otomatik olarak recover olur. Mesaj anahtarı (message key) belirtilmişse, partition'a yazma işlemi anahtar değerine göre yapılır. Aynı anahtar değerine sahip mesajlar, aynı partition'a yazılır.
Kafka partition bazında yazma ve okuma işlemini garanti ettiği için, sıranın önemli olduğu durumlarda kullanılabilir.
Mesaj anahtarı belirtilmediği durumlarda partition'lara yükü dengelemek için round robin ile yazacaktır.

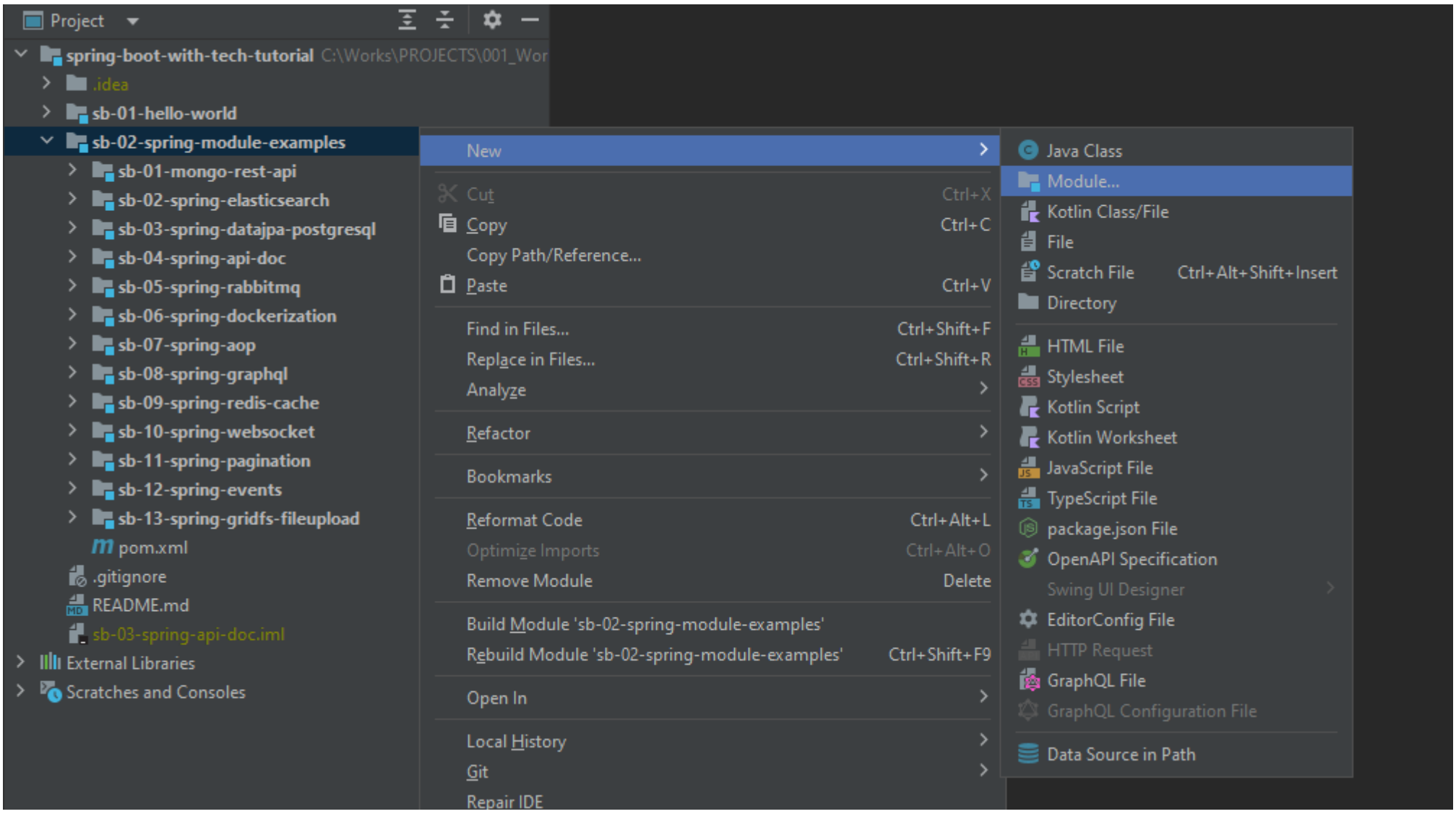
Consumer & Consumer Groups

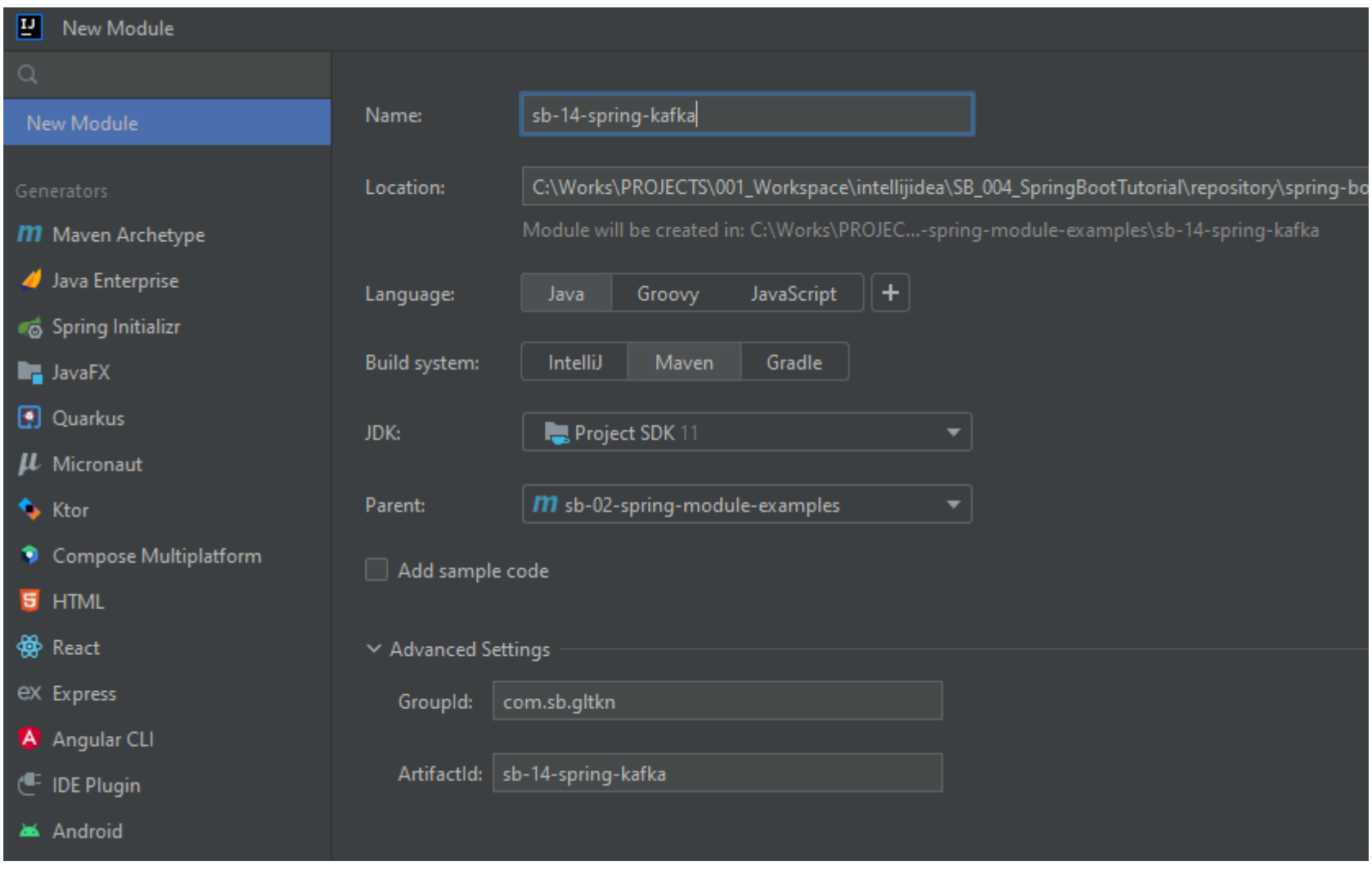
Topic'lerden veriyi okurlar. Hangi broker'dan okuyacağını bilir, broker çökse bile otomatik recover olur.
Ayni Partition içerisindeki mesajları sıralı bir şekilde okur. Birden fazla partition'dan okuyabilir.
Resimde de görüldüğü gibi her bir consumer, bir consumer group'a aittir. Bir partition'ı bir consumer group içerisinde sadece bir consumer okuyabilir.

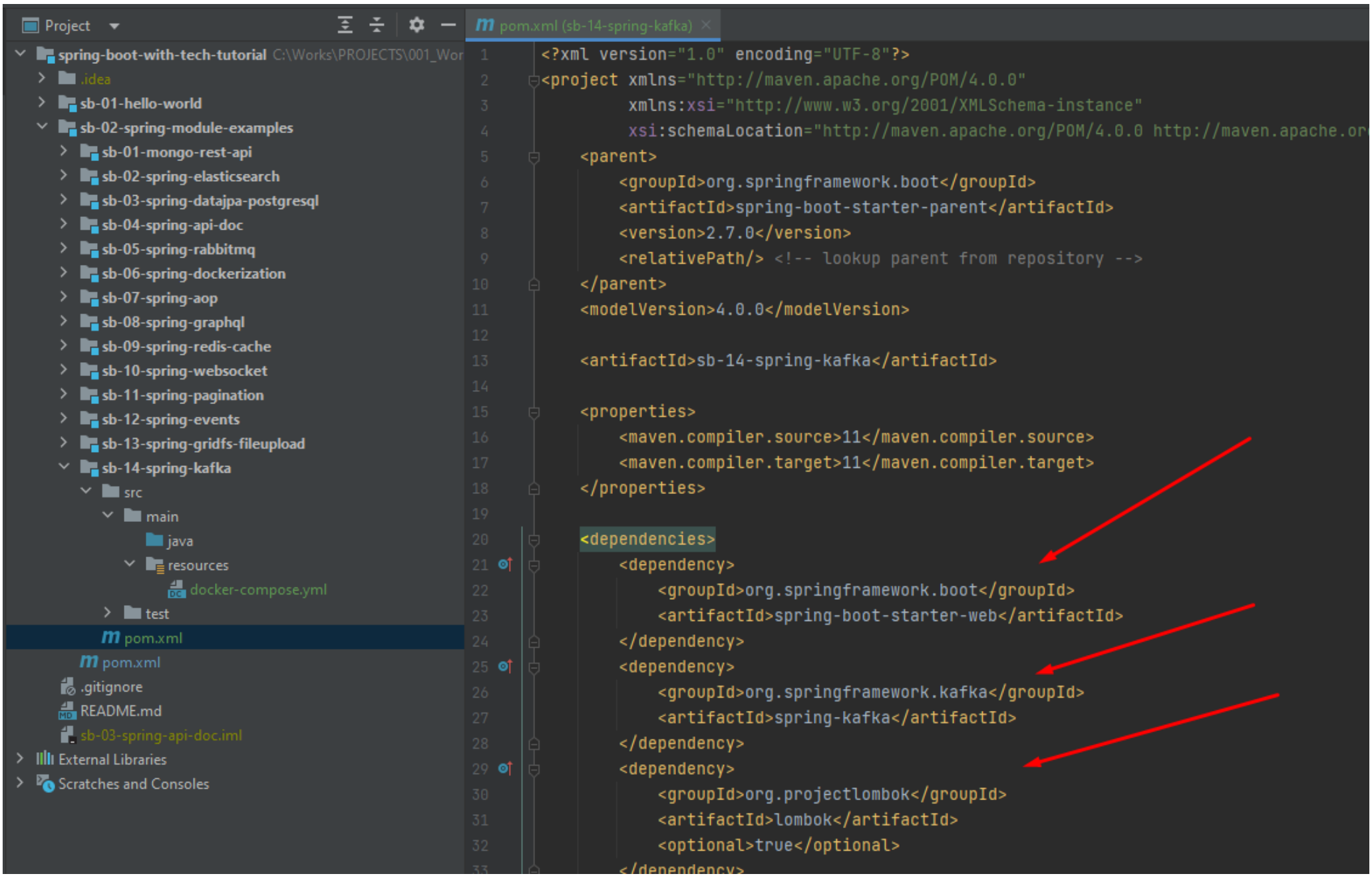


Consumer Groups

Modülümüzü oluşturalım.







Apache kafka için 2 bileşen kullanmalıyız. Birisi Zookeeper diğeri ise kafka'nın kendisidir. Bu iki bileşeni ayrı ayrı kurabiliriz. Ancak konfigürasyonları vs yapmamız gerekir. Docker'da kafka çalışmak için zookeeper'a bağımlılığımız bulunmaktadır.

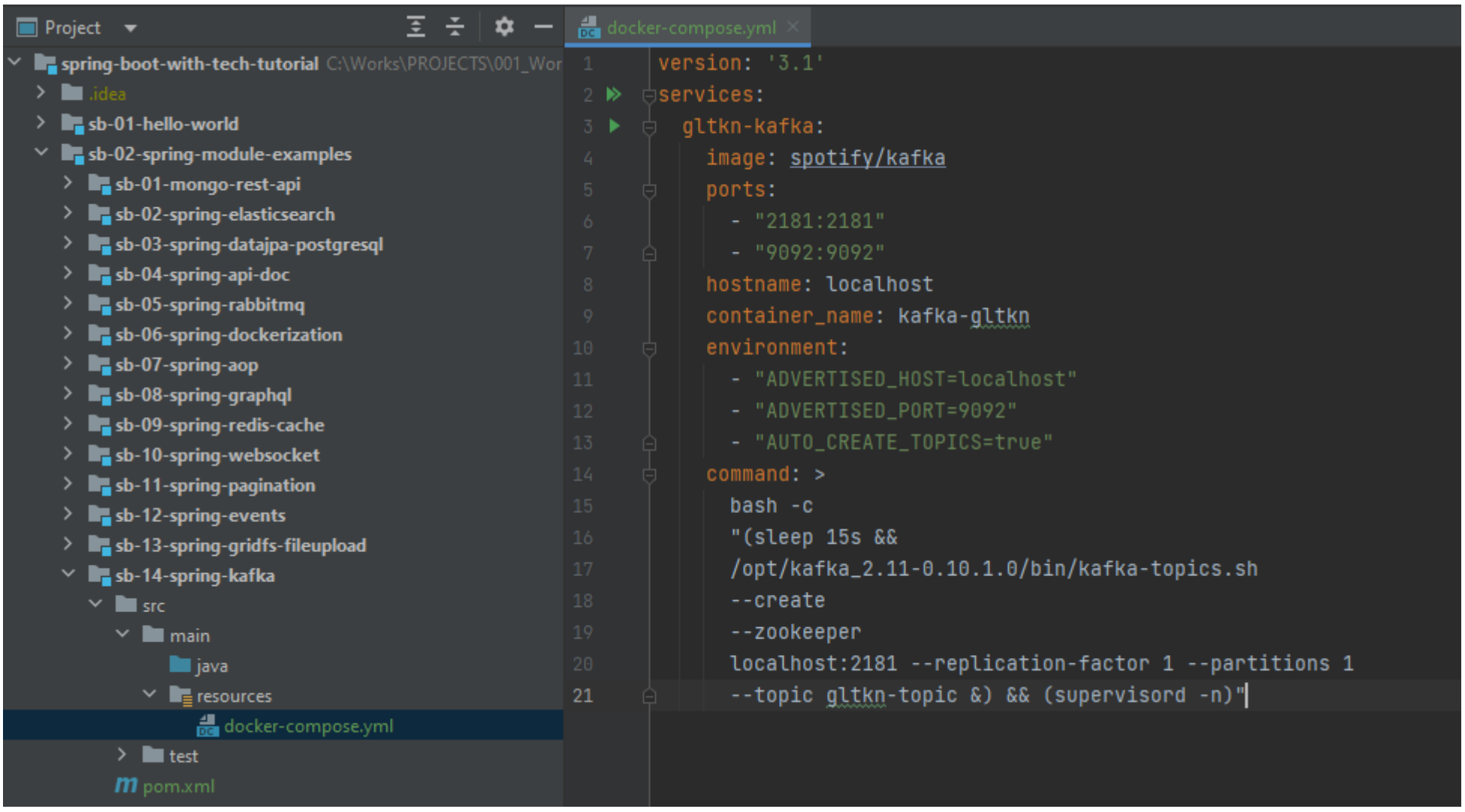
Alttaki linkte bulunan spotify'ın docker image'ı içerisinde hem zookeeper hem de kafka yapılandırılmış olarak sunulmaktadır. Zookeeper; kafka instance'ları arasındaki senkronizasyondan sorumludur.

https://hub.docker.com/r/spotify/kafka/

https://github.com/spotify/docker-kafka

Docker image'ımızda kafka 9092 portundan çalışmaktadır.
Zookeeper ise 2181 portundan çalışmaktadır.

Aşağıdaki command'da belirtilen **sleep 15s** bilgisi; sunucu başladıktan 15 saniye sonra devamındaki scripti çalıştırıyor.



Uygulamamızı spring-boot ile yazdık.

Şimdi docker-compose'u çalıştıralım.

```
EmreGltkn@DESKTOP-DN9PH1A MINGW64 ~
$ docker-machine ls
NAME      ACTIVE   DRIVER       STATE     URL                         SWARM   DOCKER      ERRORS
default   -        virtualbox   Running   tcp://192.168.99.102:2376           v19.03.12

EmreGltkn@DESKTOP-DN9PH1A MINGW64 ~
$ docker ps
error during connect: This error may indicate that the docker daemon is not running.: Get "http://%2F%2F.%2F
r_engine: Sistem belirtilen dosyayı bulamıyor.

EmreGltkn@DESKTOP-DN9PH1A MINGW64 ~
$ docker-machine env default
export DOCKER_TLS_VERIFY="1"
export DOCKER_HOST="tcp://192.168.99.102:2376"
export DOCKER_CERT_PATH="C:\Users\EmreGltkn\.docker\machine\machines\default"
export DOCKER_MACHINE_NAME="default"
export COMPOSE_CONVERT_WINDOWS_PATHS="true"
# Run this command to configure your shell:
# eval $("C:\Users\EmreGltkn\bin\docker-machine.exe" env default)

EmreGltkn@DESKTOP-DN9PH1A MINGW64 ~
$ eval "$(docker-machine env default)"
```

docker-compose -f /c/Works/PROJECTS/001_Workspace/intellijidea/SB_004_SpringBootTutorial/repository/spring-boot-with-tech-tutorial/sb-02-spring-module-examples/sb-14-spring-kafka/src/main/resources/docker-compose.yml up -d

```
EmreGltkn@DESKTOP-DN9PH1A MINGW64 ~
$ docker-compose -f /c/Works/PROJECTS/001_Workspace/intellijidea/SB_004_SpringBootTutorial/repository/spring-boot-with-tech-tutorial/sb-02-spring-m
ng-kafka/src/main/resources/docker-compose.yml up -d
Found orphan containers (resources_sboot-rabbitmq_1, resources_db_1, resources_mongo_1, resources_spboot-rabbitmq_1, resources_elasticsearch_1, res
s project. If you removed or renamed this service in your compose file, you can run this command with the --remove-orphans flag to clean it up.
Pulling gltkn-kafka (spotify/kafka:)...
latest: Pulling from spotify/kafka
```

```
EmreGltkn@DESKTOP-DN9PH1A MINGW64 ~
$ docker ps
CONTAINER ID   IMAGE           COMMAND                  CREATED         STATUS         PORTS                                                NAMES
0a73c0abfdda   spotify/kafka   "bash -c '(sleep 15s…"   4 minutes ago   Up 4 minutes   0.0.0.0:2181->2181/tcp, 0.0.0.0:9092->9092/tcp   kafka-gltkn
```

```
EmreGltkn@DESKTOP-DN9PH1A MINGW64 ~
$ docker-machine ip
192.168.99.102
```

Kafka'ya bağlanmak için alttaki tool'u kullanacağız.

https://www.kafkatool.com/download.html