Začíname s Apache Kafka

Apache Kafka je mnoho vecí:

- distribuovaná platforma pre streaming dát
- message broker ako prostredník pre výmenu správ medzi komponentami distribuovaného systému

Podrobnosti nájdeme v slajdoch.

Ingrediencie

- Docker — pre spustenie Kafky
- Conduktor — grafický nástroj pre Kafku
- Spring Boot — pre rýchle vytvorenie producentov a konzumentov
- IntelliJ IDEA — IDE pre kód v Jave

Spúšťame Kafku

Kafka pozostáva z dvoch vrstiev:

Apache Zookeper
: podvozok slúžiaci pre chod a konfiguráciu distribuovaných služieb. Nebudeme ho bližšie rozoberať, pretože je o úroveň nižšie, ako potrebujeme pre tento tutoriál.

samotná Kafka
: bežiaca nad Zookeeperom ako samotná platforma pre streaming

TIP	Kafku je najjednoduchšie spustiť pomocou nástroja **Docker**, resp. **Docker Compose**, čím si ušetríme množstvo konfigurácie.

Docker Compose

1. Stiahnime si infraštruktúrny súbor `docker-compose.yml`.
2. Spustime oba dockerovské kontajnery:

```
docker-compose up
```

3. Spustí sa Kafka v Dockeri a v logu uvidíme úspech:

```
kafka_1      | [2020-11-14 13:44:44,041] INFO [KafkaServer id=1] started
(kafka.server.KafkaServer)
```

Práca s Conduktorom

Stiahneme si Conduktor, ktorý v bezplatnej verzii podporuje pripojenie k jedinému clusteru Kafky s jedným brokerom, čo však pre tento tutoriál postačí.

Vytvoríme nový cluster Kafky (**New Kafka Cluster**) s nasledovnými parametrami:

Cluster Name

ľubovoľné meno, napríklad `local-docker`

BootStrap Servers

nastavme na `localhost:9092` ako jediný uzol, z ktorého sa štartuje cluster.

Konektivitu môžeme otestovať pomocou tlačidla **Test Kafka Connectivity**.

Producenti a konzumenti

Producentov a konzumentov vytvoríme pomocou Spring Bootu.

V IntelliJ IDEA vytvorme nový prázdny projekt (**File** | **New** | **Empty project**), do ktorého dodáme dva moduly (producentský a konzumentský).

Producentský modul

Vytvorenie modulu

Vytvorme nový modul (**File** | **New** | **Module**) typu **Spring Initializr**. Vyplňme názov skupiny (*Group*) a artefakt nazvime napríklad `producer`. V dialógovom okne následne vyberme z palety Spring Bootu súčasť **Spring for Apache Kafka** (sekcia **Messaging**).

NOTE	Alternatívne využime projekt https://start.spring.io/, kde si vieme vybrať súčasti, vygenerovať projekt v balíčku ZIP, stiahnuť a následne importovať.

Najdôležitejšou závislosťou je `spring-kafka`:

```
<dependency>
    <groupId>org.springframework.kafka</groupId>
    <artifactId>spring-kafka</artifactId>
</dependency>
```

Konfigurácia modulu

Náš broker Kafky beží na lokálnom stroji `localhost` na štandardnom porte 9092, na čo sa spolieha aj knižnica pre Spring Boot. V bežnom prípade nemusíme robiť žiadnu konfiguráciu.

Ak by to tak nebolo — napríklad kvôli inak nastavenému exponovanému portu v Dockeri — musíme to nakonfigurovať.

V súbore `src/main/resources` existuje konfiguračný súbor pre Spring Boot `application.properties`, do ktorého uveďme:

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

Kód modulu

Na komunikáciu s Kafkou sa využíva `KafkaTemplate`, ktorú si môžeme nechať autowirenúť / injectnúť do triedy aplikácie.

```
@Autowired
KafkaTemplate<String, String> kafka;
```

Trieda má dva generické parametre: pre kľúče (*keys*) a hodnoty (*values*). V tomto prípade budeme posielať reťazce.

Vďaka Spring Bootu je mnoho vecí už nakonfigurovaných a sústredíme sa len na to podstatné — odosielanie správ.

Ukážka metódy, ktorá periodicky odosiela správy:

```
public void publish() {
    kafka.send("payment", "Paid some money " + Instant.now());
}
```

Metóda `send` potrebuje dva parametre: názov *topicu*, do ktorého sa pošle správa a samotná správa. Keďže Kafka považuje správy za polia bajtov, treba sa postarať o prevod, čo zabezpečí Spring Boot a jeho integrácia s Kafkou.

Spustenie producenta

Keďže aplikácia v Spring Boote má štandardnú metódu `main`, spustíme ju triviálne. Pripojí sa k brokeru a vypublikuje doň správu do topicu `payment`.

IMPORTANT	Broker je štandardne pripravený automaticky vytvárať topicy pri prvom publikovaní správy.

NOTE	V ukážkovom kóde na GitHube uvidíme periodické zasielanie správy.

Kontrola v Conduktore

V Conduktore môžeme zistiť, že vznikol topic `payment` a produkujú sa doňho správy: počet (*Count*) postupne narastá.

Konzumentský modul

Príprava a konfigurácia

Podobne ako pri konzumentovi vytvorme nový modul (**File** | **New** | **Module**) typu **Spring Initializr**. Vyplňme názov skupiny (*Group*) a artefakt nazvime napríklad `producer`. V dialógovom okne následne vyberme z palety Spring Bootu súčasť **Spring for Apache Kafka** (sekcia **Messaging**).

Ak používame Kafku na lokálnom stroji, teda na `localhost`-e a štandardnom porte 9092, nemusíme robiť nič. V opačnom prípade podobne nakonfigurujme cestu k bootstrapovému serveru v `application.properties` identickým spôsobom: do `application.properties` dajme

```
spring.kafka.consumer.bootstrap-servers=localhost:9092
```

Kód konzumenta

Kód konzumenta využíva anotáciu `@KafkaListener`:

```
@KafkaListener(topics = "payment", ①
               groupId = "bank-branch") ②
public void consume(String message) {
    logger.info("Received: {}", message);
}
```

① Uvedieme *topic*, z ktorého budeme konzumovať.

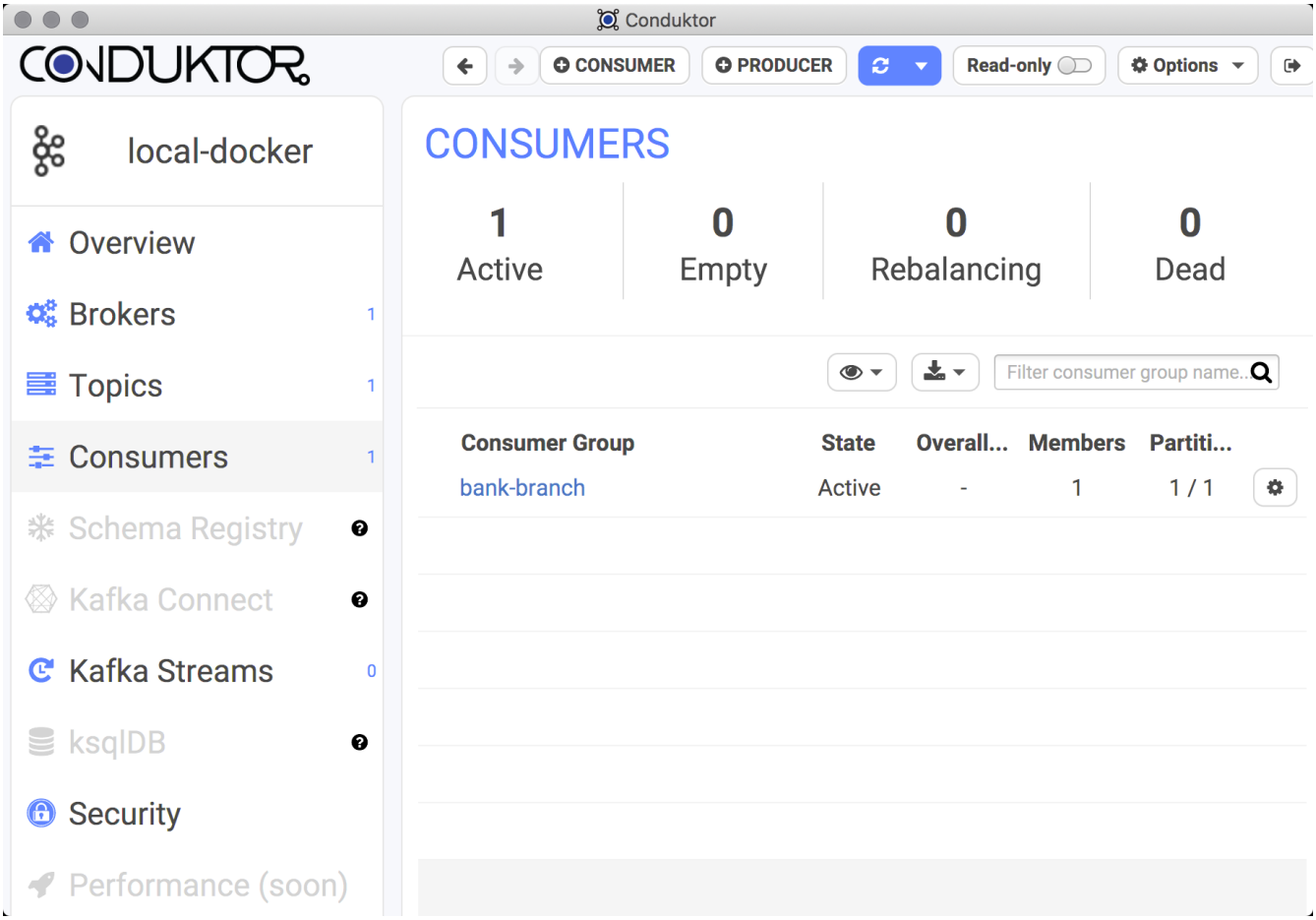
② Uvedieme identifikátor pre skupinu konzumentov (*consumer group*), do ktorej bude konzument patriť.

Spúšťame konzumenta

Konzumenta spustíme obvyklým spôsobom, cez spustenie hlavnej metódy. Sledujme, ako prijíma údaje z topicu.

Konzumenti v Conduktore

V Conduktore uvidíme, že v sekcii *Consumers* pribudla skupina (*Consumer Group*) s názvom `bank-branch`,ktorá má jediného člena.



Viacerí konzumenti a rebalance

Vytvorme teraz v IntelliJ IDEA ešte jednu konfiguráciu pre druhého konzumenta. Zvoľme z menu **Run | Edit Configurations**, vyberme z ľavého zoznamu v sekcii **Spring Boot** existujúcu konfiguráciu pre konzumenta a duplikujme ju. Nazvime ju odlišným spôsobom — napr. `Consumer #2`.

Spusťme teraz oboch konzumentov, najprv jedného a potom druhého.

Pri druhom konzumentovi uvidíme v jeho logu hlášku indikujúcu partíciu, na ktorú sa pripojil.

Log druhého konzumenta

```
2020-11-14 17:16:40.515  INFO 11741 --- [ntainer#0-0-C-1]
o.s.k.l.KafkaMessageListenerContainer    : bank-branch: partitions assigned: [payment-
0]
```

Zároveň uvidíme, že prvému konzumentovi bola odobratá partícia:

Log prvého konzumenta

```
2020-11-14 17:16:40.496  INFO 11739 --- [ntainer#0-0-C-1]
o.s.k.l.KafkaMessageListenerContainer    : bank-branch: partitions assigned: []
```

WARNING	Automatické vytváranie topicov cez Spring Boot pridelí takémuto topicu len jednu partíciu, čo znamená, že z nej môže konzumovať maximálne jeden člen skupiny konzumentov. Inak povedané, nevieme zabezpečiť distribuovanie práce.

Zvýšenie počtu partícií

V Kafke je možné dodatočne navyšovať počet partícií v topicu (nie však znižovať!). V Conduktore kliknime na topic `payment`, a cez tlačidlo **Advanced** a možnosť **Add Partition** dodajme rovno dve nové partície, t. j. nastavme tri partície.

Konzumentov nemusíme zastavovať, pretože rovno uvidíme, že nastal **rebalance** a podľa okolností jeden konzument vyfasoval dve partície a druhý konzument jednu partíciu.

TIP	Pohrajte sa so zastavovaním konzumentov a sledujte, ako konzumujú správy. Za štandardných okolností bude pri dvoch konzumentoch rovnomerné rozdelenie správ, približne na striedačku.

Architektonické vzorce

Dva najvšeobecnejšie architektonické vzorce sú *point to point* a *publish-subscribe*.

point to point
: vo všeobecnosti sa vytvorí front (*queue*), z ktorého konzumenti vyberajú správy, pričom o ne súperia.

publish-subscribe
: konzumenti sa prihlásia na odber správ z témy (*topic*) a každá správa sa doručí každému prihlásenému konzumentovi.

Point to Point

Topic potrebujeme rozbiť do toľkých partícií, koľko plánujeme konzumentov v jednej skupine konzumentov *consumer group*.

Pre tri partície v *topicu* podporujeme najviac 3 aktívnych konzumentov v spoločnej skupine *consumer group*.

Pre 2 konzumentoch v spoločnej skupine bude 1 konzument prijímať správy z jednej partície a 1 konzument z dvoch partícií.

Publish-subscribe

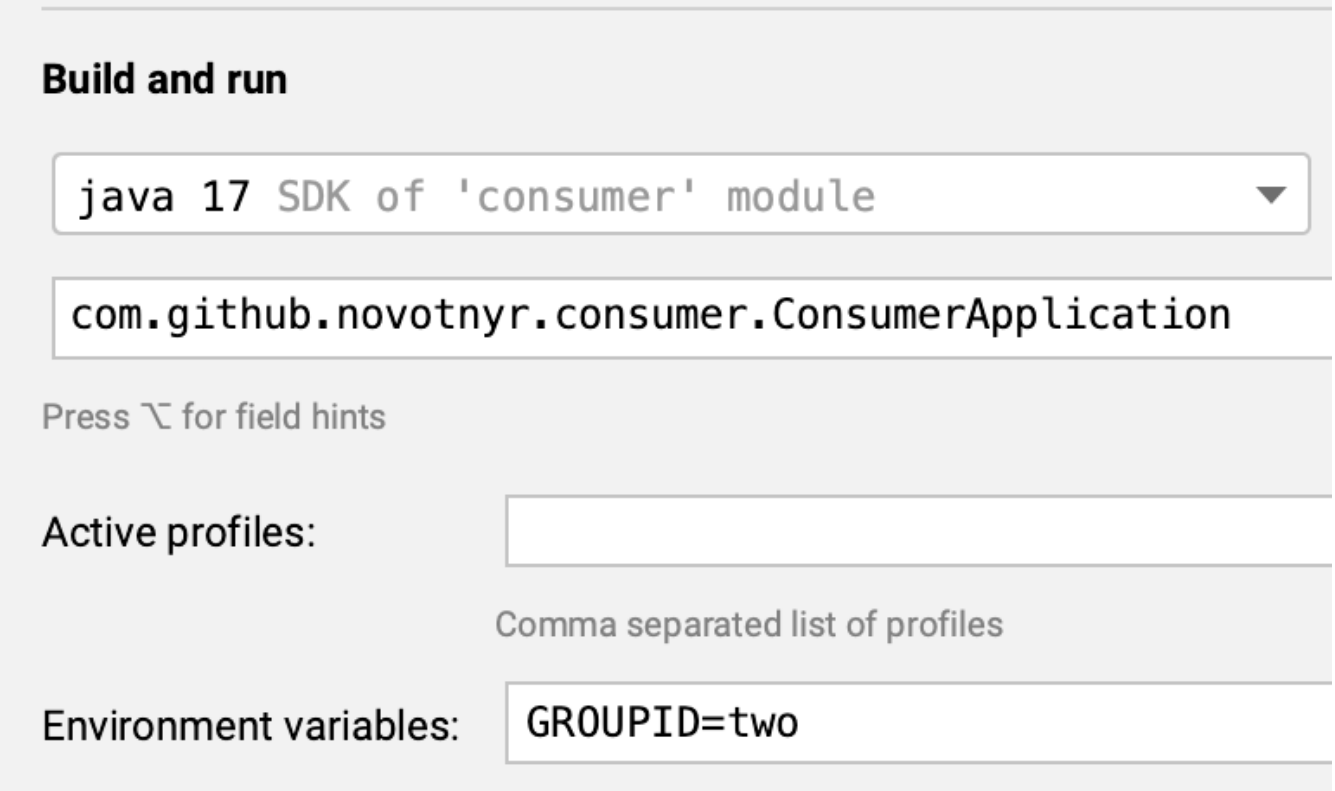
Vysielanie správ pre viacerých konzumentov zabezpečíme cez viacero skupín *consumer group*.

Dve skupiny konzumentov čítajú kópie správ z topicu. Prirodzene, konzumenti v skupinách sa prerozdelia podľa počtu partícií.

V Spring Boote môžeme názov skupiny konzumentov určiť v atribúte `groupId`:

```
@KafkaListener(topics = "car", groupId = "${groupId:car-group1}") ①
```

① V atribúte používame premennú `${…}` s implicitnou hodnotou. Spring zapojí konzumenta do skupiny podľa vlastnosti (*property*) `groupId` a ak taká vlastnosť neexistuje, použije implicitný názov skupiny konzumentov `car-group1`.



Všimnime si, ako v konfigurácii nastavíme názov druhej skupiny na `two`, pričom použijeme premennú prostredia `GROUPID`, čo sa namapuje na vlastnosť `groupId`. Toto mapovanie zariadi Spring Boot.

Zlyhávajúci konzumenti

Konzument môže vyhodiť výnimku a tým reprezentovať zlyhanie.

Štandardné správanie v Spring Kafke 2.8 a 2.9 zoberie typickú výnimku a:

- pokúsi sa o 9 opakovaní
- s odstupom 0 sekúnd
- ak klient zlyhá, pokračuje sa ďalším offsetom

Uvidíme napríklad:

```
2022-11-15 22:36:27.783 ERROR 18687 --- [ntainer#0-0-C-1]
o.s.k.l.KafkaMessageListenerContainer    : Error handler threw an exception

org.springframework.kafka.KafkaException: Seek to current after exception; nested
exception is org.springframework.kafka.listener.ListenerExecutionFailedException:
Listener method 'public void
com.github.novotnyr.kafka.KafkaConsumer.consume(org.apache.kafka.clients.consumer.Cons
umerRecord<java.lang.String, java.lang.String>)' threw exception; nested exception is
java.lang.IllegalArgumentException; nested exception is
java.lang.IllegalArgumentException
```

V aplikačnom kontexte vieme definovať bean s vlastným nastavením pre opakovanie:

```
@Bean
public DefaultErrorHandler meh() {
    return new DefaultErrorHandler(new FixedBackOff(5000L, 2L));
}
```

Voliteľne môžeme nastaviť:

recoverer

zavolá sa, ak sa potrebujeme zotaviť z chyby, po vyčerpaní možností

not retryable exception

výnimky, ktoré sa považujú za nezotaviteľné, teda správa sa rovno „zahodí“ a zavolá sa *recoverer*.

```
@Bean
public DefaultErrorHandler meh() {
    ConsumerRecordRecoverer recoverer = (record, e) -> System.err.println(e);③
    var errorHandler = new DefaultErrorHandler(recoverer, new FixedBackOff(5000L,
2L));②
    errorHandler.addNotRetryableExceptions(IllegalArgumentException.class);①
    return errorHandler;
}
```

① Výnimku `IllegalArgumentException` považujeme za nezotaviteľnú.

② Obsluha chýb sa pokúsi zotaviť z výnimky opakovaním čítania správy z offsetu, každý 5 sekúnd, najviac 2 pokusy.

③ Ak sa pokusy vyčerpajú alebo ak je výnimka nezotaviteľná, dostaneme správu na štandardný výstup.