Apache Kafka para Iniciantes

E-book amostra do curso

Autor: Fábio José de Moraes
Revisão: Dagmar F. Napolitano

Kafka BR

20 agosto 2020

Conteúdo

Preparação

Antes de iniciar este módulo é importante garantir que seu ambiente: pc, notebook, workstation, esteja preparado para execução dos nossos laboratórios. Então, com base nessas orientações realize algumas preparações.

- **Prepare um computador com no mínimo 4GB RAM e 50GB de espaço no HD**
- **Tenha uma conexão banda-larga com a internet**
- **Verifique se você tem instalado o Java nas versões 1.8 ou 11**
- **Verifique se seu usuário no sistema operacional tem permissão para executar serviços de rede**
- **Verifique se seu usuário no sistema operacional tem permissão para realizar download de arquivos .zip**

Verificar versão Java

Windows:

- Abrir o prompt de comando
- Digitar este comando e teclar enter: `java -version`
- Se o resultado for algo do tipo:

```
openjdk version "1.8.0_242"
OpenJDK Runtime Environment (AdoptOpenJDK)(build 1.8.0_242-b08)
OpenJDK 64-Bit Server VM (AdoptOpenJDK)(build 25.242-b08, mixed mode)
```

Linux:

- Abrir o terminal
- Digitar este comando e teclar enter: `java -version`
- Se o resultado for algo do tipo, verifique a versão:

```
openjdk version "11.0.6"
OpenJDK Runtime Environment (build 11.0.6+10-post-Debian-1bpo91)
OpenJDK 64-Bit Server VM (build 11.0.6+10-post-Debian-1bpo91)
```

Instalar Java 1.8 ou Java 11

Java **1.8** para Linux.

Debian, Ubuntu, etc:

```
sudo apt-get install openjdk-8-jdk
```

Fedora, Oracle Linux, Red Hat Enterprise Linux, etc.


```
su -c "yum install java-1.8.0-openjdk-devel"
```

Java **1.8** para Windows.

- 32bits: http://bit.ly/java_1_8-x32
- 64bits: http://bit.ly/java_1_8-x64

Java **11** para Linux.

> procure no google: *how to install openjdk 11 DISTRO NAME install*

Java **11** para Windows.

- 64bits: http://bit.ly/java_11-x64

Adicionais

Se você é usuário do windows, recomendamos a instalação dos seguintes aplicativos:

- 7-Zip: https://www.7-zip.org/download.html
- Notepad++: https://notepad-plus-plus.org/downloads/v7.8.5/

Para Labs, recomendamos algum editor para você visualizar o código-fonte:

- VS Code: https://code.visualstudio.com/download
- Atom: https://atom.io/
- Notepad++: https://notepad-plus-plus.org/downloads/v7.8.5/

Conceitos Gerais

A conceituação geral é importante para que todas as pessoas tenham em mente diversos detalhes ligados à computação distribuída, que são utilizados com frequência quando falamos sobre Kafka.

Mensagens & Eventos

Hoje temos uma infinidade de termos, definições, padrões e *buzzwords*. Isso gera mal-entendidos sobre o que é **mensagem** e o que é **evento**. Saber quando estamos produzindo e/ou consumindo mensagens ou eventos é crucial para solucionar corretamente os problemas de negócio através da computação distribuída.

Bem, mas agora que vimos vínculos de transporte e formatos de dados, vamos estabelecer a Semântica da Comunicação. E aí que entram as mensagens e os eventos.

Mensagens são . . .

As mensagens definem uma semântica na comunicação onde há **intenção** nos dados produzidos, ou seja, existe a **expectativa** de que eles passem por algum tipo de processamento, produzindo um resultado em algum momento no futuro. Essas mensagens acontecem através de qualquer formato de dados, sobre qualquer vínculo de transporte.

> Mensagens são dados carregando a intenção de que algo aconteça no futuro.

Eventos são . . .

Os eventos definem uma semântica na comunicação onde os dados produzidos transmitem um **fato** ocorrido, ou seja, em nenhum momento existe a expectativa que ele seja processado ou exista algum resultado. Esses eventos acontecem através de qualquer formato de dados, sobre qualquer vínculo de transporte.

> Eventos são dados relatando um fato ocorrido em algum ponto no passado.

Idempotência

> Operações que podem ser aplicadas sem que o valor do resultado se altere após a aplicação inicial.

Wikipedia

Qualquer operação que seja executada uma ou cem vezes, manterá o resultado final mesmo após a aplicação inicial. Como exemplo, usaremos uma operação de débito no valor de `R$ 100,00`

identificada por `d598`, se esta operação for idempotente, mesmo que seja executada dez vezes, somente `R$ 100,00` serão debitados e não `R$ 1.000,00`.

Note que a idempotência é construída em torno de algo que possa ser validado, neste caso um **identificador**. No exemplo acima, se todas as dez operações de débito possuíssem identificadores diferentes não seria possível computar a idempotência, pois, semanticamente seriam coisas completamente diferentes.

Padrões de Entrega

Os padrões de entrega estão presentes tanto na produção, quanto no consumo de dados. Na produção, estes padrões garantem que o Kafka receberá os dados e tratará de processá-los para persistência. Já no consumo, eles garantem que os dados chegarão às instâncias que os requisitaram.

1 - At-least-once - Pelo-menos-uma-vez

Neste padrão "Pelo menos uma vez" os dados consumidos, em virtude de alguma situação, quando processados poderão ser consumidos novamente. No Kafka isso pode acontecer caso o *offset* consumido e processado não seja efetivado, ou seja, não aconteça o *commit*.

2 - Exactly-once - Exatamente-uma-vez

Neste padrão "Exatamente uma vez" os dados são consumidos, e processados apenas uma vez, ou seja, existirá uma garantia que os dados que já foram processados foram efetivados. Com Kafka podemos fazer isso através da **Transaction API**, que veremos em detalhes neste módulo.

Apache Kafka 101

Kafka é uma plataforma distribuída para *streaming*, como definido pelo seu site oficial, que escala horizontalmente e provê alta disponibilidade.

Kafka é . . .

- **Plataforma tolerante à falhas**: sua arquitetura e operação foi idealizada para ser tolerante às falhas na rede e na replicação de dados, bem como para a não perda de dados
- **Para alta vazão de dados (taxa de transferência)**: a forma como os dados são gravados no sistema de arquivos é desenhado para alta velocidade na escrita e na leitura, por isso não existem buscas ou filtros para consulta de dados.
- **O melhor dos dois mundos** - *tópicos* e *filas*: o melhor dos tópicos, porque vários destinos podem consumir os dados presentes no kafka. E o melhor das filas, porque ele garante que dado um registro, jamais será enviado para dois ou mais consumidores diferentes em um mesmo grupo de consumo. Isso quer dizer que, àquele registro será enviado apenas a um destino no grupo, assim como acontece com consumidores em ferramentas tradicionais para filas.
- **Garante a entrega de dados pelo-menos-uma-vez**: esse é o padrão de entrega *out-of-the-box* para consumidores no kafka. Há uma garantia de que os dados sempre chegarão, isso uma ou várias vezes.
- **Confiável, armazenamento persistente e durável**: uma vez produzidos nos kafka, os registros estarão lá por 7 dias na configuração padrão, ou *ad aeternum* se assim for desejado.
- **Código fonte aberto**: Kafka é uma ferramenta de código fonte aberto muito popular na comunidade, com cerca de 15 mil estrelas no Github. E inúmeras corporações, como Confluent, Red Hat, Microsoft e Google contribuem para sua evolução, além de milhares de outras que são suas usuárias.

Kafka não é . . .

- **Uma ferramenta que se preocupa com "quem"**: kafka não implementa mecanismos comumente encontrados em ferramentas tradicionais, como o gerenciamento de entrega por consumidor. Esse tipo de controle demanda um processamento adicional e degrada a vazão de dados. Então, temos o controle *offset* por grupos de consumo, que veremos em detalhe.
- **Uma fila**: não, kafka não é uma fila da forma como conhecemos onde existe exatamente um produtor e exatamente um consumidor realizando operações *push* e *pop*, respectivamente. Ou uma ordenação global dos registros, nem sua eliminação quando todos os consumidos já o receberam.
- **Uma ferramenta "inteligente"**: kafka deixa de lado controles que prejudicam a vazão de dados, deixando-os para o software consumidor e produtor. Só implementando o mínimo,

de forma muito inteligente, como veremos em “Funcionamento interno”.

- **Lento ou de execução pesada**: Kafka é escrito usando Scala e roda sobre a Java Virtual Machine e não requer uma infraestrutura cara ou proprietária.
- **Guiado por apenas uma empresa**: isso garante que essa ferramenta é guiada pela comunidade de código aberto e não por interesses corporativos.

Ecossistema Apache Kafka

Apache Kafka é um ecossistema de ferramentas rico, criado para atender diversos casos de uso e resolver problemas da computação distribuída de forma clara e objetiva. Nele encontramos a ferramenta certa para cada situação do nosso projeto de transformação digital.

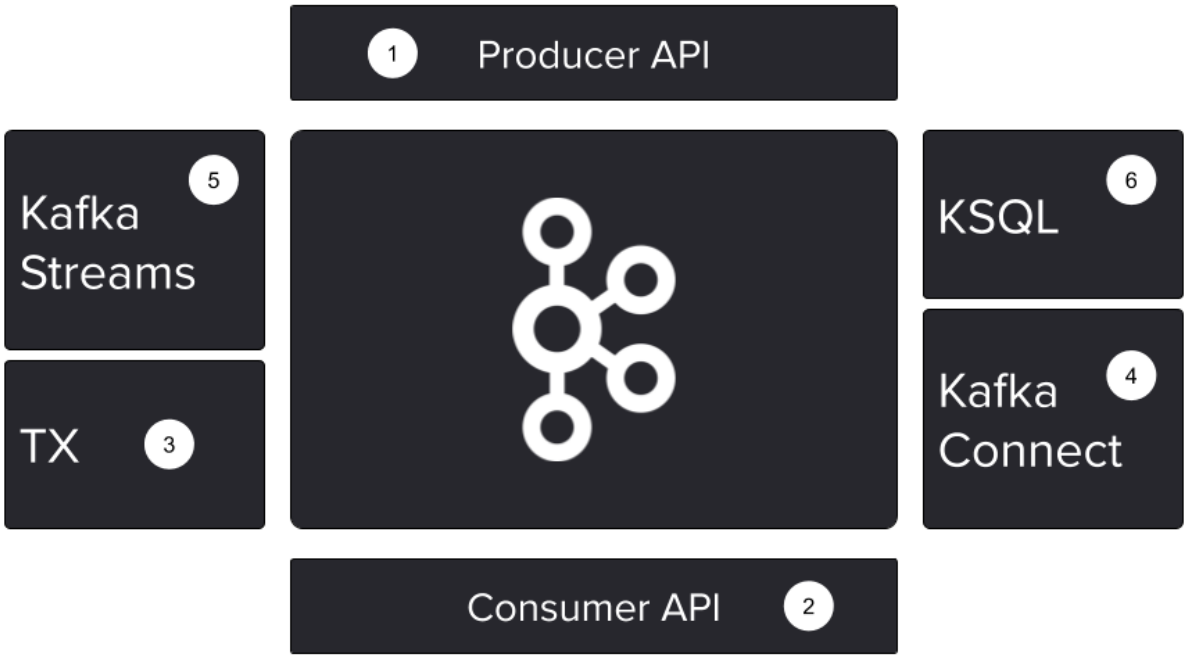

Figura 1: Ecossistema Apache Kafka

1. **Producer API**: API mais básica para produzir eventos no Kafka. Utilizada como base para todas as outras ferramentas.
2. **Consumer API**: API mais básica para consumir eventos do Kafka. Quem também é a base para muitas outras implementações.
3. **Transactions**: API para operar comunicação transacional, tanto na produção como no consumo de eventos. Ela também é base para Kafka Streams a Kafka Connect.
4. **Kafka Connect**: Integrar Kafka com origens de dados externas, como bancos de dados, arquivos, APIs, filas tradicionais, etc.
5. **Kafka Streams**: processamento de streams de eventos de forma declarativa no Java.
6. **KSQL**: processar streams de eventos com uma notação SQL like.

Lab 1: configurando Kafka CLI

- Crie o diretório `kafka-m1` no seu computador
  - Se você utiliza o **Windows**, crie o diretório `kafka-m1` na raiz do sistema, ou seja, na unidade `C:\`, ficando assim:

```
C:\kafka-m1
```

- Isso é necessário porque no windows existe uma limitação quanto ao número máximo de caracteres em um caminho de diretórios.
- Realize o download do arquivo `kafka_2.12-2.4.0.zip`, disponível através do link abaixo:
  - http://bit.ly/kafka_2_4_0
- Extraia o conteúdo do arquivo `kafka_2.12-2.4.0.zip` no diretório `kafka-m1`
  - **Linux**: `unzip kafka_2.12-2.4.0.zip`
  - **Windows**: *utilize o aplicativo 7-zip ou similar*
- Configure a variável de ambiente `KAFKA`.
  - **Linux**: `export KAFKA=/path/to/kafka-m1/kafka_2.12-2.4.0`
  - **Windows**: `set KAFKA=C:\kafka-m1\kafka_2.12-2.4.0`
- Configure a variável de ambiente `PATH`, assim você poderá executar os comandos Kafka a partir de qualquer local do seu computador.
  - **Linux**: `export PATH=$PATH:$KAFKA/bin`
  - **Windows**: `set PATH=%PATH%;%KAFKA%\bin\windows`

Meu primeiro tópico

Agora que temos um cluster funcional, vamos criar nosso primeiro tópico e descrever seus detalhes para entender como é sua operação.

> Veja no **Apêndice I** como subir um cluster Apache Kafka

- Abra um novo terminal **Linux**:
  - Digite o comando:

```
kafka-topics.sh --create \
  --bootstrap-server localhost:9092 \
  --replication-factor 3 \
  --partitions 7 \
  --topic kbr-meu.topico
```

- Abra o prompt de comando **Windows**:
  - Digite o comando:

```
kafka-topics.bat --create ^
--bootstrap-server localhost:9092 ^
--replication-factor 3 ^
--partitions 7 ^
--topic kbr-meu.topico
```

`kafka-topics.sh` ou `kafka-topics.bat`, são comandos para realizar operações relacionadas aos tópicos, como criar, listar, apagar ou detalhar suas configurações. Os argumentos utilizados foram:

- `--bootstrap-server` Ponto de contato com o cluster Kafka, que deverá ser `localhost:9092` ou aquele servidor de contingência que disponibilizamos.
- `--replication-factor` Número de réplicas deste novo tópico.
- `--partitions` Número de partições dentro deste novo tópico.

Depois de criar nosso tópico, vamos detalhar seus atributos para entender como ele está dentro do cluster e como foram distribuídos o líder-réplicas, ou *leader-followers* em inglês.

Linux

```
kafka-topics.sh --describe \
  --bootstrap-server localhost:9092 \
  --topic kbr-meu.topico
```

Windows

```
kafka-topics.bat --describe ^
--bootstrap-server localhost:9092 ^
--topic kbr-meu.topico
```

Exemplo de saída:

```
Topic: meu.topico    PartitionCount: 7    ReplicationFactor: 3
Configs: segment.bytes=1073741824
    Topic: meu.topico    Partition: 0    Leader: 1    Replicas: 1,2,3    Isr: 1,2,3
    Topic: meu.topico    Partition: 1    Leader: 2    Replicas: 2,3,1    Isr: 2,3,1
    Topic: meu.topico    Partition: 2    Leader: 3    Replicas: 3,1,2    Isr: 3,1,2
    Topic: meu.topico    Partition: 3    Leader: 1    Replicas: 1,3,2    Isr: 1,3,2
    Topic: meu.topico    Partition: 4    Leader: 2    Replicas: 2,1,3    Isr: 2,1,3
    Topic: meu.topico    Partition: 5    Leader: 3    Replicas: 3,2,1    Isr: 3,2,1
    Topic: meu.topico    Partition: 6    Leader: 1    Replicas: 1,2,3    Isr: 1,2,3
```

- Cada partição é listada, de `0` a `6`.
- `Topic`: nome do tópico.
- `PartitionCount`: número total de partições.
- `ReplicationFactor`: fator de replicação das partições deste tópico. Esse valor define o número de réplicas que o kafka tentará manter em sincronia.
- `Configs`: configurações informadas no argumento `--config` ao criar o tópico. Todas as outras são valores-padrão definidos na configuração do servidor e que foram herdadas por nosso tópico.

Anatomia dos Tópicos

Tópicos são divisões lógicas para armazenamento de dados pois são em suas partições, as divisões físicas, onde realmente os dados seguem persistentes e duráveis. Quanto ao número máximo de tópicos, virtualmente não existe uma limitação para criá-los no cluster kafka.

São as partições que persistem fisicamente os dados dentro de um tópico e proporcionam paralelismo em seu consumo.

- tecnicamente, as **partições** são comumente referenciadas como **unidades de paralelismo** para o consumo.

O número de partições por tópico é virtualmente ilimitado. E um número elevado de partições aumenta o paralelismo, mas pode causar lentidão na replicação e no re-balanceamento de consumidores ou no *failover*. Mas é comum observar instalações com duas ou cinco mil partições em produção rodando perfeitamente.

Kafka não permite busca por chave de registro, pois ele foi desenhado para alta vazão de dados, então o acesso acontece através de *offset* e da leitura em série. Os *offsets* são como um tipo de ponteiro que são mapeados para as posições físicas nos segmentos onde residem os dados.

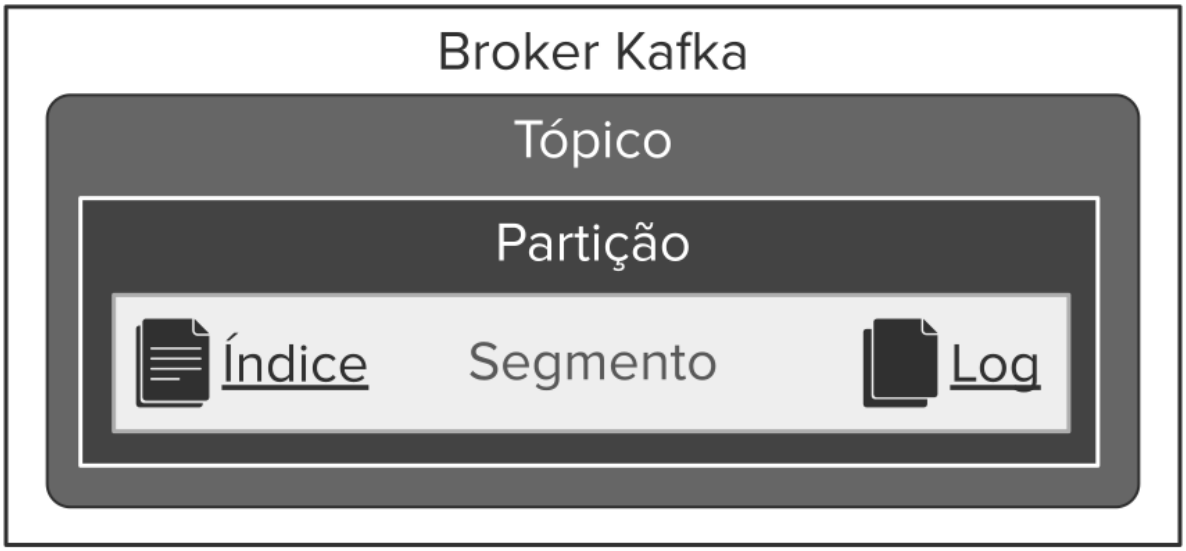

Figura 2: Segmento é composto pelos arquivos de índice e de log

Cada **segmento** é composto por tres arquivos principais:

- **índice para offset**: mapeamento entre *offset* e a posição física no arquivo de log.
- **índice para timestamp**: mapeamento entre carimbo data-hora e *offset*.
- **log**: arquivo físico onde estão os dados ou lotes de registros.

Kafka ao criar arquivos de segmento, os nomeia com o *offset* inicial neles armazenados. Vamos ilustrar isso com um exemplo:

- Imagine os *offsets* de `0` a `10`. Eles estarão presentes nos arquivos:
  - `00000000000000000000.index`
  - `00000000000000000000.log`
- Já os *offsets* de `11` em diante, estarão nos arquivos:
  - `00000000000000000011.index`
  - `00000000000000000011.log`

A criação de segmentos é ditada por duas configurações feitas nos tópicos:

- `segment.bytes`: tamanho máximo em bytes para cada segmento.
- `segment.ms`: tempo para que o kafka crie um novo segmento mesmo que o atual ainda não tenha atingido seu tamanho máximo. Essa configuração existe para que aconteça a compactação ou deleção, pois enquanto o segmento estiver aberto não é possível processá-lo

Compactação

A compactação de log, *log compaction* em inglês, é um procedimento operacional no Kafka que tem o objetivo de reduzir o tamanho dos dados persistidos. Ele é ativado quando se utiliza o valor `compact` na configuração `cleanup.policy`.

Primeiro vejamos a produção de registros dentro do kafka e sua estruturação.

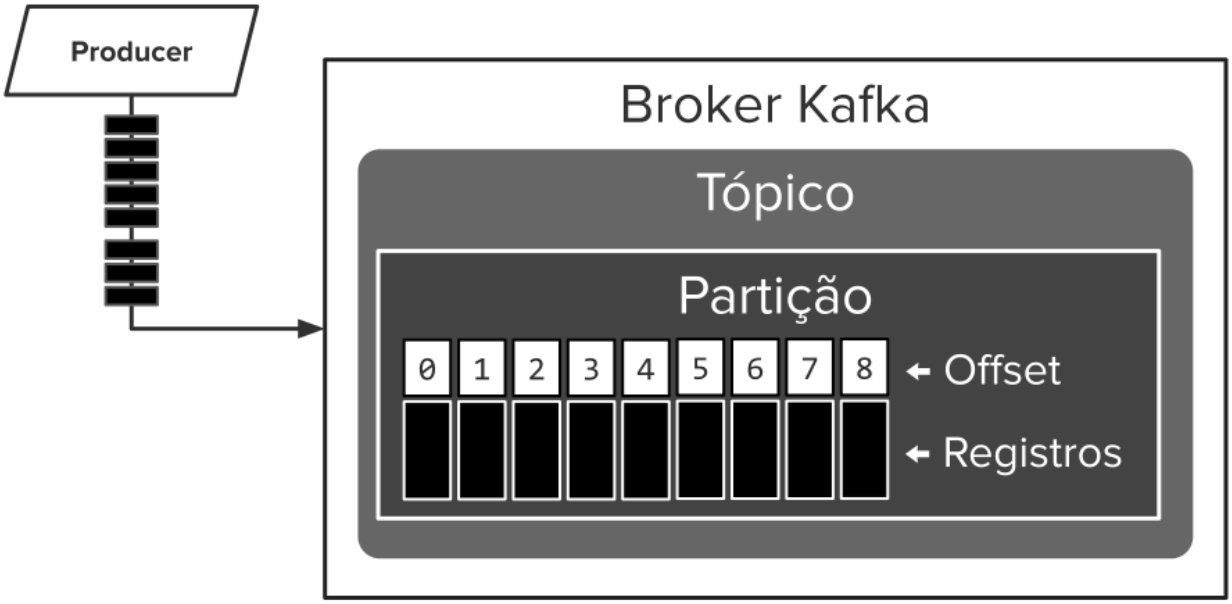

Figura 3: Produção de registros recebendo offsets nas partições

Registros são produzidos, seguindo para as partições do tópico e recebendo um *offset* toda vez que são escritos no final do segmento ativo. Pensemos que os *Offsets* são uma espécie de índice dos registros.

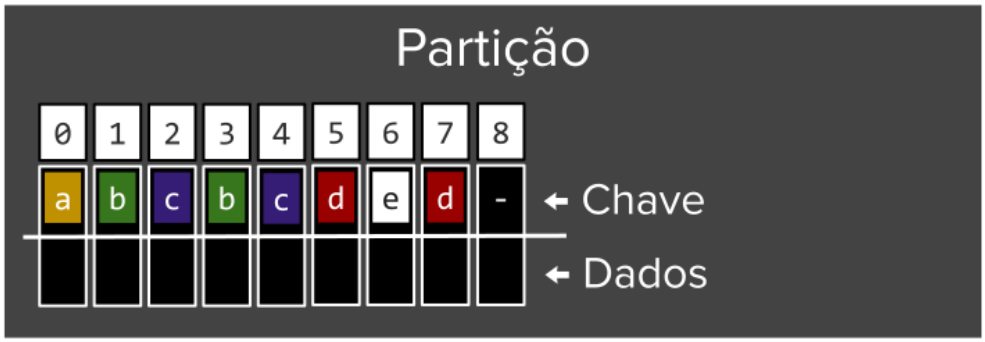

Figura 4: Registros possuem chaves, como: a, b, c, d, e

Cada registro pode ser produzido com uma chave, key em inglês, que é utilizada para determinar em qual partição ele será persistido. E kafka garante, na configuração padrão, que registros com a mesma chave sigam sempre para a mesma partição.

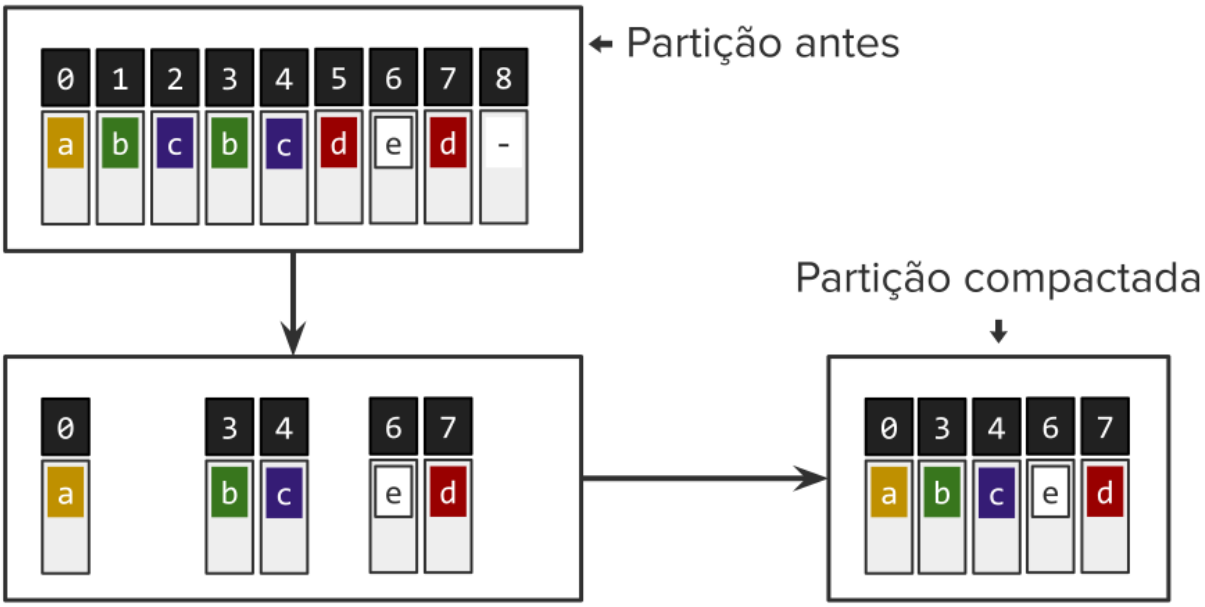

Figura 5: Partição após compactação

- os registros nos *offsets* 1 e 2 foram eliminados, mas nenhuma reordenação ou novos *offsets* foram atribuídos.

Manter os *offsets* é importante, caso contrário todos os segmentos deveriam ser completamente re-gerados degradando o desempenho geral dos brokers e sua taxa de transferência, *throughput* em inglês.

Principais configurações

Todas as configurações de um tópico possuem valores-padrão definidas no broker, através do arquivo `config/server.properties`. E quando forem criados herdarão essas configurações, desde que não sejam sobrescritas através do argumento `--config` do comando `kafka-topics.sh` (ou `kafka-topics.bat` para windows). Ou que por ventura forem criados automaticamente.

cleanup.policy

Define como será o processo de limpeza dos tópicos.

Para manter os dados sanitizados nos tópicos, o Kafka implementa sua limpeza, seja pela eliminação de registro ou pela remoção das repetições, a compactação.

- valor padrão: `delete`
- valores possíveis
  - `delete`: após o tempo ou tamanho máximos serem atingidos, os segmentos são limpos e eliminados fisicamente do armazenamento.
  - `compact`: serão mantidos somente os dados mais recentes sob uma determinada chave, já os antigos, são apagados através da compactação de log. Com essa configuração, qualquer registro que não contenha chave, será rejeitado para persistência no tópico, respondendo erros para o *producer* e não persistindo-o (KIP-135).
  - `delete,compact`: além da compactação de log, também será possível reduzir o tamanho do espaço ocupado através da eliminação dos segmentos antigos.
- configurações relacionadas
  - `min.compaction.lag.ms` quando valor é `compact`
  - `max.compaction.lag.ms` quando valor é `compact`
  - `retention.bytes` quando o valor é `delete`
  - `retention.ms` quando o valor é `delete`
  - `min.cleanable.dirty.ratio` quando o valor é `compact`
  - `segment.bytes`
  - `segment.ms`
- configuração no broker p/ valor padrão: `log.cleanup.policy`

A limpeza que apaga os dados, delete, é realizada sobre os segmentos, ou seja, não é uma política aplicada individualmente aos registros. Kafka só processa segmentos inativos quando o carimbo data-hora mais antigo neles existente, expirar.

Podemos habilitar ambas as políticas e configurá-las, deste modo:

```
cleanup.policy=compact,delete
```

max.message.bytes

Tamanho máximo permitido para uma mensagem.

Caso o lote de registros do producer seja maior que o valor desta configuração, sua requisição será rejeitada. O erro recebido será algo similar a este:

```
The request included a message larger than the max message size the server will accept.
```

Ou dependendo da linguagem ou biblioteca de conexão com o Kafka, a seguinte mensagem:

```
Message size too large
```

Todas elas levarão a entender que a mensagem é maior que o valor definido nesta configuração.

> Sempre devemos entender que mensagem trata-se na verdade de um lote, batch em inglês, de registros.

- valor padrão: `1000012` (~976 Kb)
- configurações relacionadas
  - `batch.size`, uma configuração no producer
- configuração no broker p/ valor padrão: `message.max.bytes`

Lab 2: Criando tópicos

Agora vamos exercitar a criação de tópicos com configurações variadas e entender seu comportamento, com os resultados dos comandos.

Experimento a

Vamos criar o tópico `kbr-lote300kb`, que restringe o tamanho máximo do lote de registros através da configuração `max.message.bytes`.

Linux:

```
kafka-topics.sh --create \
  --bootstrap-server localhost:9092 \
  --replication-factor 3 \
  --partitions 7 \
  --topic kbr-lote300kb \
  --config max.message.bytes=307200
```

Windows:

```
kafka-topics.bat --create ^
--bootstrap-server localhost:9092 ^
--replication-factor 3 ^
--partitions 7 ^
--topic kbr-lote300kb ^
--config max.message.bytes=307200
```

- `307200` é o equivalente a `300KB`

Produza lotes de registros **maiores** que o limite. Ao tentar enviar lotes maiores que o valor definido pela configuração `max.message.bytes`, serão retornados erros e nenhum registro será persistido no tópico.

Erros como este. Mas não se engane, estamos falando do lote, não de um único registro.

```
MESSAGE_TOO_LARGE
```

Linux:

```
kafka-producer-perf-test.sh \
  --topic kbr-lote300kb \
  --num-records 7 \
  --record-size 51200 \
  --throughput -1 \
  --producer-props \
    acks=1 \
    bootstrap.servers=localhost:9092 \
    batch.size=409600
```

Windows:

```
kafka-producer-perf-test.bat ^
--topic kbr-lote300kb ^
--num-records 7 ^
--record-size 51200 ^
--throughput -1 ^
--producer-props ^
acks=1 ^
bootstrap.servers=localhost:9092 ^
batch.size=409600
```

- `51200` é o equivalente a `50KB`
- `409600` é o equivalente a `50KB * 8`

Apêndice I

Iniciando um cluster kafka

Notas sobre Kafka no Windows

No windows um cluster kafka não irá executar muito bem, isso em função das limitações existentes no próprio windows. Existe uma issue que busca adaptar o Apache Kafka para rodar em servidores windows, mas ainda não foi aceita:

- https://github.com/apache/kafka/pull/3283

Download

- Crie o diretório `kafka-m1` no seu computador
- Realize o download do arquivo `kafka_2.12-2.4.0.zip`, disponível através do link abaixo:

Download Apache Kafka 2.4.0

- Extraia o conteúdo do arquivo `kafka_2.12-2.4.0.zip` no diretório `kafka-m1`
  - **Linux**: `unzip kafka_2.12-2.4.0.zip`
- Configure a variável de ambiente `KAFKA`.
  - **Linux**: `export KAFKA=/path/to/kafka-m1/kafka_2.12-2.4.0`

Iniciar o zookeeper

- Abra um novo terminal **Linux**:
- Execute o comando:

```
zookeeper-server-start.sh $KAFKA/config/zookeeper.properties
```

- Abra o prompt de comando **Windows**:
- Execute o comando:

```
zookeeper-server-start.bat %KAFKA%\config\zookeeper.properties
```

Quando você observar resultados similares a estes, o zookeeper já está pronto:

```
[2020-03-22 20:50:26,772] INFO binding to port 0.0.0.0/0.0.0.0:2181
    (org.apache.zookeeper.server.NIOServerCnxnFactory)
[2020-03-22 20:50:26,802] INFO zookeeper.snapshotSizeFactor = 0.33
    (org.apache.zookeeper.server.ZKDatabase)
[2020-03-22 20:50:26,805] INFO Snapshotting: 0x0 to /tmp/zookeeper/version-2/snapshot.0
    (org.apache.zookeeper.server.persistence.FileTxnSnapLog)
[2020-03-22 20:50:26,808] INFO Snapshotting: 0x0 to /tmp/zookeeper/version-2/snapshot.0
    (org.apache.zookeeper.server.persistence.FileTxnSnapLog)
[2020-03-22 20:50:26,839] INFO Using checkIntervalMs=60000 maxPerMinute=10000
    (org.apache.zookeeper.server.ContainerManager)
```

Iniciar o Kafka Broker #1

- Abra um novo terminal **Linux**
- Digite o comando

```
kafka-server-start.sh $KAFKA/config/server.properties \
  --override broker.id=1 \
  --override log.dirs=/tmp/broker1-logs \
  --override listeners=PLAINTEXT://:9092 \
  --override zookeeper.connect=localhost:2181 \
  --override zookeeper.connection.timeout.ms=10000
```

- Abra o prompt de comando **Windows**:
- Execute o comando:

```
kafka-server-start.bat %KAFKA%\config\server.properties ^
--override broker.id=1 ^
--override log.dirs=/tmp/broker1-logs ^
--override listeners=PLAINTEXT://:9092 ^
--override zookeeper.connect=localhost:2181 ^
--override zookeeper.connection.timeout.ms=10000
```

Iniciar o Kafka Broker #2

- Abra um novo terminal **Linux**:
- Digite o comando:

```
kafka-server-start.sh $KAFKA/config/server.properties \
  --override broker.id=2 \
  --override log.dirs=/tmp/broker2-logs \
  --override listeners=PLAINTEXT://:9093 \
  --override zookeeper.connect=localhost:2181 \
  --override zookeeper.connection.timeout.ms=10000
```

- Abra o prompt de comando **Windows**:
- Execute o comando:

```
kafka-server-start.bat %KAFKA%\config\server.properties ^
--override broker.id=2 ^
--override log.dirs=/tmp/broker2-logs ^
--override listeners=PLAINTEXT://:9093 ^
--override zookeeper.connect=localhost:2181 ^
--override zookeeper.connection.timeout.ms=10000
```

Iniciar o Kafka Broker #3

- Abra um novo terminal **Linux**:
- Digite o comando:

```
kafka-server-start.sh $KAFKA/config/server.properties \
  --override broker.id=3 \
  --override log.dirs=/tmp/broker3-logs \
  --override listeners=PLAINTEXT://:9094 \
  --override zookeeper.connect=localhost:2181 \
  --override zookeeper.connection.timeout.ms=10000
```

- Abra o prompt de comando **Windows**:
- Execute o comando:

```
kafka-server-start.bat %KAFKA%\config\server.properties ^
--override broker.id=3 ^
--override log.dirs=/tmp/broker3-logs ^
--override listeners=PLAINTEXT://:9094 ^
--override zookeeper.connect=localhost:2181 ^
--override zookeeper.connection.timeout.ms=10000
```

Saber se o broker está funcionando

Ao iniciar os brokers, todos eles deverão apresentar a seguinte mensagem, com exceção do id que deverá ser diferente em cada um deles:

```
[2020-03-22 20:54:33,577] INFO [KafkaServer id=3] started (kafka.server.KafkaServer)
```

Problemas comuns no Windows

Para usuários do windows, alguns problemas podem surgir com relação à instalação do Java.

```
Error: missing `server' JVM at `C:\Program Files (x86)\Java\jre1.8.0_151\bin\server\jvm.dll'.
Please install or use the JRE or JDK that contains these missing components.
```

Se você receber o erro acima ao executar qualquer um dos comandos, sua instalação Java trata-se somente da JRE. Mas Kafka requer a JDK, que é a versão completa da Java Virtual Machine.

Solução: revisar a instalação java e utilizar os links disponibilizados na preparação deste módulo.

Docker

Se você tem Docker e Docker Compose instalados na sua máquina, também poderá iniciar o Kafka de forma muito simples.

Cluster c/ Três Brokers

- Baixar o arquivo: Docker Compose Cluster
- Abra um novo terminal Linux e vá até ao diretório onde você baixou o arquivo:
- Digite o comando:

```
docker-compose up
```

Como se trata de um cluster, as portas para os brokers e zookeeper são as seguintes:

- Utilize como bootstrap server `localhost:29092,localhost:39092`
- Zookeeper: `localhost:12181`

Single Broker

- Baixar o arquivo: Docker Compose Single Broker
- Abra um novo terminal Linux e vá até ao diretório onde você baixou o arquivo:
- Digite o comando:

```
docker-compose up
```