Inicio rápido

Este tutorial asume que está comenzando de nuevo y no tiene datos existentes de Kafka o ZooKeeper. Dado que los scripts de la consola Kafka son diferentes para las plataformas basadas en Unix y Windows, en las plataformas Windows, use en `bin\windows\` lugar de `bin/` y cambie la extensión del script a `.bat` .

Paso 1: descargue el código

Descargue la versión 2.5.0 y descomprímalo.

```
> tar -xzf kafka_2.12-2.5.0.tgz
> cd kafka_2.12-2.5.0
```

Paso 2: iniciar el servidor

Kafka usa ZooKeeper, por lo que primero debe iniciar un servidor ZooKeeper si aún no tiene uno. Puede usar el script de conveniencia empaquetado con kafka para obtener una instancia de ZooKeeper de un solo nodo rápida y sucia.

```
> bin/zookeeper-server-start.sh config/zookeeper.properties
[2013-04-22 15:01:37,495] INFO Reading configuration from: config/zookeeper.properties (org.apache.zookeeper.server.quorum.
...
```

Ahora inicie el servidor Kafka:

```
> bin/kafka-server-start.sh config/server.properties
[2013-04-22 15:01:47,028] INFO Verifying properties (kafka.utils.VerifiableProperties)
```

```
[2013-04-22 15:01:47,051] INFO Property socket.send.buffer.bytes is overridden to 1048576 (kafka.utils.VerifiableProperti
...
```

Paso 3: crea un tema

Creemos un tema llamado "prueba" con una sola partición y solo una réplica:

```
> bin/kafka-topics.sh --create --bootstrap-server localhost:9092 --replication-factor 1 --partitions 1 --topic test
```

Ahora podemos ver ese tema si ejecutamos el comando list topic:

```
> bin/kafka-topics.sh --list --bootstrap-server localhost:9092
test
```

Alternativamente, en lugar de crear temas manualmente, también puede configurar sus intermediarios para que creen automáticamente temas cuando se publique un tema inexistente.

Paso 4: envía algunos mensajes

Kafka viene con un cliente de línea de comando que tomará la entrada de un archivo o de la entrada estándar y la enviará como mensajes al clúster de Kafka. Por defecto, cada línea se enviará como un mensaje separado.

Ejecute el productor y luego escriba algunos mensajes en la consola para enviarlos al servidor.

```
> bin/kafka-console-producer.sh --bootstrap-server localhost:9092 --topic test
This is a message
This is another message
```

Paso 5: Comience un consumidor

Kafka también tiene un consumidor de línea de comando que volcará los mensajes a la salida estándar.

```
> bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic test --from-beginning
This is a message
This is another message
```

Si tiene cada uno de los comandos anteriores ejecutándose en una terminal diferente, ahora debería poder escribir mensajes en la terminal del productor y verlos aparecer en la terminal del consumidor.

Todas las herramientas de línea de comando tienen opciones adicionales; ejecutar el comando sin argumentos mostrará información de uso que los documenta con más detalle.

Paso 6: configurar un clúster de varios intermediarios

Hasta ahora hemos estado corriendo contra un solo corredor, pero eso no es divertido. Para Kafka, un único agente es solo un grupo de tamaño uno, por lo que nada cambia mucho más que iniciar algunas instancias más de agente. Pero solo para sentirlo, expandamos nuestro clúster a tres nodos (todavía todos en nuestra máquina local).

Primero creamos un archivo de configuración para cada uno de los corredores (en Windows use el `copy` comando en su lugar):

```
> cp config/server.properties config/server-1.properties
> cp config/server.properties config/server-2.properties
```

Ahora edite estos nuevos archivos y configure las siguientes propiedades:

```
config/server-1.properties:
    broker.id=1
    listeners=PLAINTEXT://:9093
    log.dirs=/tmp/kafka-logs-1

config/server-2.properties:
    broker.id=2
    listeners=PLAINTEXT://:9094
    log.dirs=/tmp/kafka-logs-2
```

La `broker.id` propiedad es el nombre único y permanente de cada nodo en el clúster. Tenemos que anular el puerto y el directorio de registro solo porque los estamos ejecutando todos en la misma máquina y queremos evitar que todos los intermediarios intenten registrarse en el mismo puerto o sobrescribir los datos de los demás.

Ya tenemos Zookeeper y nuestro nodo único iniciado, por lo que solo necesitamos iniciar los dos nuevos nodos:

```
> bin/kafka-server-start.sh config/server-1.properties &
...
> bin/kafka-server-start.sh config/server-2.properties &
```

```
4 4   ...
```

Ahora cree un nuevo tema con un factor de replicación de tres:

```
1   > bin/kafka-topics.sh --create --bootstrap-server localhost:9092 --replication-factor 3 --partitions 1 --topic my-replicate
```

Bien, pero ahora que tenemos un clúster, ¿cómo podemos saber qué agente está haciendo qué? Para ver eso, ejecute el comando "describir temas":

```
1   > bin/kafka-topics.sh --describe --bootstrap-server localhost:9092 --topic my-replicated-topic
2   Topic:my-replicated-topic   PartitionCount:1    ReplicationFactor:3 Configs:
3       Topic: my-replicated-topic  Partition: 0    Leader: 1   Replicas: 1,2,0 Isr: 1,2,0
```

Aquí hay una explicación de la salida. La primera línea ofrece un resumen de todas las particiones, cada línea adicional brinda información sobre una partición. Como solo tenemos una partición para este tema, solo hay una línea.

- "líder" es el nodo responsable de todas las lecturas y escrituras para la partición dada. Cada nodo será el líder de una parte de las particiones seleccionada al azar.
- "réplicas" es la lista de nodos que replican el registro de esta partición, independientemente de si son el líder o incluso si están actualmente vivos.
- "isr" es el conjunto de réplicas "sincronizadas". Este es el subconjunto de la lista de réplicas que actualmente está vivo y atrapado por el líder.

Tenga en cuenta que en mi ejemplo, el nodo 1 es el líder de la única partición del tema.

Podemos ejecutar el mismo comando en el tema original que creamos para ver dónde está:

```
1   > bin/kafka-topics.sh --describe --bootstrap-server localhost:9092 --topic test
2   Topic:test  PartitionCount:1    ReplicationFactor:1 Configs:
3       Topic: test Partition: 0    Leader: 0   Replicas: 0 Isr: 0
```

Por lo tanto, no hay sorpresa: el tema original no tiene réplicas y está en el servidor 0, el único servidor en nuestro clúster cuando lo creamos.

Publiquemos algunos mensajes sobre nuestro nuevo tema:

```
  1   > bin/kafka-console-producer.sh --bootstrap-server localhost:9092 --topic my-replicated-topic
  2   ...
  3   my test message 1
4 4   my test message 2
5 5   ^C
```

Ahora consumamos estos mensajes:

```
> bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --from-beginning --topic my-replicated-topic
...
my test message 1
my test message 2
^C
```

Ahora probemos la tolerancia a fallas. El corredor 1 estaba actuando como el líder, así que vamos a matarlo:

```
> ps aux | grep server-1.properties
7564 ttys002    0:15.91 /System/Library/Frameworks/JavaVM.framework/Versions/1.8/Home/bin/java...
> kill -9 7564
```

En uso de Windows:

```
> wmic process where "caption = 'java.exe' and commandline like '%server-1.properties%'" get processid
ProcessId
6016
> taskkill /pid 6016 /f
```

El liderazgo se ha cambiado a uno de los seguidores y el nodo 1 ya no está en el conjunto de réplicas sincronizadas:

```
> bin/kafka-topics.sh --describe --bootstrap-server localhost:9092 --topic my-replicated-topic
Topic:my-replicated-topic   PartitionCount:1    ReplicationFactor:3 Configs:
    Topic: my-replicated-topic  Partition: 0    Leader: 2   Replicas: 1,2,0 Isr: 2,0
```

Pero los mensajes todavía están disponibles para el consumo a pesar de que el líder que tomó las escrituras originalmente no está disponible:

```
> bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --from-beginning --topic my-replicated-topic
...
my test message 1
my test message 2
^C
```

Paso 7: use Kafka Connect para importar / exportar datos

Leer los datos de la consola y volver a escribirlos en la consola es un lugar conveniente para comenzar, pero es probable que desee utilizar datos de otras fuentes o exportar datos de Kafka a otros sistemas. Para muchos sistemas, en lugar de escribir un código de integración personalizado, puede usar Kafka Connect para importar o exportar datos.

Kafka Connect es una herramienta incluida con Kafka que importa y exporta datos a Kafka. Es una herramienta extensible que ejecuta *conectores* , que implementan la lógica personalizada para interactuar con un sistema externo. En este inicio rápido, veremos cómo ejecutar Kafka Connect con conectores simples que importan datos de un archivo a un tema de Kafka y exportan datos de un tema de Kafka a un archivo.

Primero, comenzaremos creando algunos datos semilla para probar con:

```
> echo -e "foo\nbar" > test.txt
```

O en Windows:

```
> echo foo> test.txt
> echo bar>> test.txt
```

A continuación, iniciaremos dos conectores que se ejecutan en modo *independiente* , lo que significa que se ejecutan en un único proceso local dedicado. Proporcionamos tres archivos de configuración como parámetros. El primero es siempre la configuración para el proceso Kafka Connect, que contiene una configuración común, como los corredores Kafka para conectarse y el formato de serialización de datos. Los archivos de configuración restantes especifican cada uno un conector para crear. Estos archivos incluyen un nombre de conector único, la clase de conector para crear instancias y cualquier otra configuración requerida por el conector.

```
> bin/connect-standalone.sh config/connect-standalone.properties config/connect-file-source.properties config/connect-file-
```

Estos archivos de configuración de muestra, incluidos con Kafka, utilizan la configuración de clúster local predeterminada que inició anteriormente y crean dos conectores: el primero es un conector de origen que lee las líneas de un archivo de entrada y produce cada uno para un tema de Kafka y el segundo es un conector de sumidero que lee mensajes de un tema de Kafka y produce cada uno como una línea en un archivo de salida.

Durante el inicio, verá una serie de mensajes de registro, incluidos algunos que indican que se están instanciando los conectores. Una vez que el proceso de Kafka Connect ha comenzado, el conector de origen debe comenzar a leer líneas `test.txt` y producirlas para el tema `connect-test` , y el conector de sumidero debe comenzar a leer mensajes del tema `connect-test` y escribirlos en el archivo `test.sink.txt` . Podemos verificar que los datos se hayan entregado a través de toda la tubería examinando el contenido del archivo de salida:

```
> more test.sink.txt
foo
bar
```

Tenga en cuenta que los datos se almacenan en el tema de Kafka `connect-test` , por lo que también podemos ejecutar un consumidor de consola para ver los datos en el tema (o usar un código de consumidor personalizado para procesarlo):

```
> bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic connect-test --from-beginning
{"schema":{"type":"string","optional":false},"payload":"foo"}
{"schema":{"type":"string","optional":false},"payload":"bar"}
...
```

Los conectores continúan procesando datos, por lo que podemos agregar datos al archivo y verlo moverse a través de la tubería:

```
> echo Another line>> test.txt
```

Debería ver que la línea aparece en la salida del consumidor de la consola y en el archivo receptor.

Paso 8: use Kafka Streams para procesar datos

Kafka Streams es una biblioteca cliente para crear aplicaciones y microservicios en tiempo real de misión crítica, donde los datos de entrada y / o salida se almacenan en grupos de Kafka. Kafka Streams combina la simplicidad de escribir e implementar aplicaciones estándar Java y Scala en el lado del cliente con los beneficios de la tecnología de clúster del lado del servidor de Kafka para hacer que estas aplicaciones sean altamente escalables, elásticas, tolerantes a fallas, distribuidas y mucho más. Este ejemplo de inicio rápido demostrará cómo ejecutar una aplicación de transmisión codificada en esta biblioteca.