부록 A 카프카 스트림즈 구성

카프카 스트림즈는 고도로 구성가능하며 사용가능한 파라미터와 기본값이 변경되고 있다. 따라서 이 부록에 기술된 구성 속성은 다양한 구성 파라미터에 익숙해지기 위한 시작점으로 사용되어야 하며 최신 정보는 공식 문서를 참조하기 바란다.

구성 관리

이 책에서는 Properties 인스턴스를 생성하고 수동으로 다양한 구성 파라미터를 설정하여 카프카 스트림즈 애플리케이션을 구성하였다. 다음 코드는 이 전략의 예를 보여준다:

```
class App {
 public static void main(String[] args) {
 Topology topology = GreeterTopology.build();
 Properties config = new Properties();
 config.put(StreamsConfig.APPLICATION_ID_CONFIG, "dev-consumer");
 config.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, "kafka:9092");
 KafkaStreams streams = new KafkaStreams(topology, config);
 // ....
 }
}
```

그러나 애플리케이션을 운영 단계로 전달할 시점에는 애플리케이션에 값들을 직접 하드코딩하는 대신 파일로부터 구성을 적재하는 것을 고려해야 한다. 코드를 건드리지 않고 구성 변경을 할 수 있다는 것은 에러가 덜 발생한다는 것으로 구성 파일이 런타임 시 재정의된다면 (시스템 플래그를 통해) 여러 애플리케이션 배치를 관리하는 것은 매우 쉽다[1].

Java 애플리케이션 (카프카 스트림즈 포함)에 대한 전체적인 구성 관리 전략 논의는 이 책의 범위를 벗어나며 이 책의 소스 코드에서 Typesafe Config라는 라이브러리를 사용하여 단일 카프카 스트림즈 애플리케이션에 대해 여러 구성을 관리하는 방법을 볼 수 있다.

어떻게 구성을 관리하는지에 상관없이 카프카 스트림즈 또는 ksqlDB의 구성 파라미터에 제공한 값은 애플리케이션 성능에 영향을 미칠 수 있다. 따라서 이 절 대부분은 구성 파라미터 자체에 중점을 둘 것이다. 카프카 스트림즈 특정적인 구성을 살펴보자.

구성 속성

카프카 스트림즈 애플리케이션 구성 시 무엇보다도 2 개의 파라미터가 필요하다:

application.id

---

[1] 애플리케이션이 여러 카프카 클러스터 내 데이터를 처리해야 하거나 다른 환경에 배치되어야 하는 경우 (예, 스테이징, QA, 제품단계) 여러 배치를 가질 수 있다.

카프카 스트림즈 애플리케이션 고유 식별자. 많은 머신/프로세스에 워크로드를 분산하기 위해 동일 application.id를 갖는 여러 애플리케이션 인스턴스를 실행할 수 있다. 내부적으로 이 식별자는 다음을 포함하여 여러 용도로 사용된다:

- 컨슈머 그룹 ID
- 내부 토픽 이름의 접두사 (즉, 카프카 스트림즈에 의해 생성된 리파티션과 체인지로그 토픽)
- 카프카 스트림즈 라이브러리가 사용하는 기반 프로듀서와 컨슈머에 대한 기본 client.id 접두사
- 영구 상태 저장소 서브 디렉토리 (예제 6-1 참조).

bootstrap.servers

하나 이상의 카프카 브로커에 해당하는 호스트와 포트 쌍 목록. 카프카 클러스터와 연결 확립에 사용될 것이다.

또한 여러 선택적인 파라미터도 존재한다:

acceptable.recovery.lag

카프카 스트림즈 태스크가 태스크 재할당에 대해 warm과 ready로 고려되기 위해 입력 파티션에 가질 수 있는 최대 지연량 (즉, 읽지 않은 레코드 수). 스테이트풀 애플리케이션이 상태의 일부 또는 전부를 복구해야 할 때 복구 중 이에 작업이 할당되는 것을 원하지 않을 수 있다. 이 경우 애플리케이션 인스턴스가 그들의 지연이 이 임계치 미만인 경우에만 태스크 할당을 받을 수 있도록 이 파라미터를 설정할 수 있다.

cache.max.bytes.buffering

이는 카프카 스트림즈에서 레코드 캐시 크기를 제어한다. "애플리케이션의 출력 속도 제한"에서 보다 세부적으로 레코드 캐시에 대해 논의했다. 이를 0 보다 큰 값으로 설정함으로써 레코드 캐시를 활성화하며 애플리케이션의 출력 속도 제한에 영향을 미칠 수 있다.

default.deserialization.exception.handler

역직렬화 에러를 처리하는데 사용되는 클래스. 이를 "카프카 스트림즈에서 역직렬화 에러 처리하기"에서 팁으로 논의했다. 내장 옵션은 LogAndContinueExceptionHandler와 LogAndFailExceptionHandler를 포함한다. 전자는 보다 허용적인 것으로 역직렬화 에러 발생 시 카프카 스트림즈가 계속 레코드를 처리할 수 있도록 한다. 후자는 카프카 스트림즈가 예외를 기록하고 처리를 중지하도록 한다. 맞춤형 핸들러를 정의하기 위해 카프카 스트림즈 라이브러리에 포함되어 있는 DeserializationExceptionHandler 인터페이스를 구현할 수도 있다.

default.production.exception.handler

카프카에 데이터 생산과 관련된 에러를 처리하는데 사용되는 클래스. 예를 들어 레코드가 너무 큰 경우 기반 프로듀서가 어떤 방식으로 처리되어야 하는 예외 처리를 할 것이다. 기본으로 내장 DefaultProductionExceptionHandler 클래스가 사용되며 카프카 스트림즈가 실패 및 셧다운하도록 할 것이다. 다른 옵션은 AlwaysContinuePro ductionExceptionHandler 클래스를 사용하는 것으로 이는 카프카 스트림즈가 데이터 처리를 계속할 수 있도록 하며 자체 맞춤형 로직을 제공하려는 경우 ProductionExceptionHandler 인터페이스를 구현할 수 있도록 한다.

default.timestamp.extractor

레코드와 타임스탬프를 연관시키는데 사용되는 클래스. "타임스탬프 추출기"에서 이를 세부적으로 논의했다.

default.key.serde, default.value.serde

레코드 키와 값을 직렬화 및 역직렬화에 사용되는 기본 클래스. 이 책에서는 Serdes 클래스를 인라인으로 정의했다. 예를 들어

```
KStream<byte[], Tweet> stream =
 builder.stream(
 "tweets",
 Consumed.with(Serdes.ByteArray(), JsonSerdes.Tweet())); ①
```

① 키 Serdes (Serdes.ByteArray())와 값 Serdes (Json Serdes.Tweet()) 모두를 이 예에 정의되어 있다.

대신 이들 두 구성 파라미터를 사용하여 기본 Serdes 클래스를 사용하고 Consumed.with(...) 라인을 생략할 수 있다. 이는 카프카 스트림즈에 이들 파라미터가 지정한 기본 클래스를 사용하도록 할 것이다.

max.task.idle.ms

스트림 태스크가 파티션 버퍼 모두에 데이터가 포함될 때까지 기다려야 하는 최대 시간. 보다 큰 값은 지연을 증가시키지만 태스크가 여러 입력 파티션에서 읽을 때 순서가 뒤바뀐 데이터를 방지한다 (예, 조인)

max.warmup.replicas

(num.standbys 외에) 태스크 워밍업에 사용될 수 있는 복제본 수

metrics.recording.level

카프카 스트림즈가 포착하는 지표에 대한 세분화 수준. 기본값은 INFO 지만 애플리케이션에 대해 보다 많은 가시성을 원하는 경우 DEBUG로 재정의할 수 있다. 지표는 JMX를 통해 보고되며 "JMX 지표 추출"에서 이들 지표를 포착하는 방법을 논의했다.

num.standby.replicas

각 상태 저장소를 위해 생성할 복제본 수. 스테이트풀 태스크가 오프라인이 되도 카프카 스트림즈가 작업을 복제 상태를 갖고 있는 다른 애플리케이션 인스턴스에 재할당 할 수 있기 때문에 다운타임을 줄이는데 도움이 되며 따라서 처음부터 태스크 상태를 재구축하는 값비싼 연산을 방지한다. "대기 복제본"에서 이에 대해 논의했다.

num.stream.threads

"태스크와 스트림 쓰레드"에서 상기할 수 있듯이 스트림 쓰레드는 카프카 스트림즈 태스크를 실행시키는 것이다. 쓰레드 수를 증가시키는 것은 머신에서 사용가능한 CPU 자원을 최대한 활용하는 것을 도울 수 있으며 성능을 향상시킬 것이다. 예를 들어 카프카 스트림즈 애플리케이션이 8 개의 파티션을 갖는 단일 토픽에서 읽고 있는 경우 8 개의 태스크를 생성할 것이다. 원하는 경우 하나, 둘 또는 네 개 쓰레드 (숫자는 여러분에 달려있지만 태스크 수 이하여야 한다)에 대해 이들 태스크를 실행할 수 있지만 8 개 코어를 갖는 머신에서 실행한다면 쓰레드 수를 8 로 늘리는 것은 모든 사용가능 코어에 대해 작업을 병렬화할 수 있도록 할 것이다.

이 구성은 애플리케이션의 처리량을 최대화하는데 특히 중요하다.

processing.guarantee

지원되는 값

at_least_once

특정 실패 조건 (예, 프로듀서에게 소스 토픽에 메시지를 재전송하도록 하는 네트워킹 문제 또는 브로커 실패) 동안 레코드가 재전송될 수 있다. 그러나 메시지는 절대 손실되지 않는다. 이는 카프카 스트림즈의 기본 처리 보증으로 정확성이 중복 레코드 재처리에 의해 영향을 받지 않는 지연 민감 애플리케이션에 이상적이다.

exactly_once

트랜잭션 프로듀서와 격리 수준이 read_committed인 내장 컨슈머를 사용하여 레코드가 정확히 한 번 처리된다[2]. 애플리케이션이 이 처리 보증을 통해 컨슈머 지연에 더 취약할 수 있지만 (트랜잭션 쓰기에 대해 비교적 작은 성능 페널티가 존재) 애플리케이션의 정확성은 정확히 한번 처리를 필요로 한다. 이 선택사항은 브로커 버전이 0.11.0 이상이어야 한다.

exactly_once_beta

이 옵션은 스트림 애플리케이션에 대해 확장성 개선과 오버헤드 감소를 통해 정확히 한 번 처리 보증을 가능케 한다. 이 옵션은 브로커 버전이 2.5 이상이어야 한다[3].

---

[2] 격리 수준은 컨슈머가 실패한/중단된 트랜잭션에서 어떤 메시지도 읽지 않을 것임을 보장한다.

[3] Exactly_once는 입력 파티션 당 트랜잭션 프로듀서를 생성할 것이다. Exact_once_beta는 스트림 쓰

replication.factor

카프카 스티림이 생성한 내부 체인지로그 또는 리파티션 토픽의 복제 수. 모든 토폴로지가 내부 토픽 생성을 야기하지 않음을 상기하기 바란다. 그러나 레코드 키 재생성 또는 스테이트풀 집계를 수행한다면 (그리고 명시적으로 체인지 로깅을 비활성화했다면) 내부 토픽이 생성될 것이다 (토픽 이름 앞에 application.id가 붙을 것이다). 권고 값은 3으로 최대 2 대의 브로커 실패까지 허용할 수 있다.

rocksdb.config.setter

RocksDB 상태 저장소 구성에 사용되는 맞춤형 클래스. 스테이트풀 애플리케이션을 미세조정하는 사람에게는 이 구성이 매우 흥미로울 수 있다. 맞춤형 클래스를 사용한 RocksDB 구성은 일반적으로 필요치 않으며 공식 문서에서 관심 분야에 대한 예를 제공한다. 이는 보통 성능을 최적화하려고 할 때 처음 돌릴 손잡이는 아니지만 여러분이 애플리케이션에서 보다 많은 성능을 끄집어 낼 필요가 있고 RocksDB가 병목 현상임을 식별한 상황에 대한 옵션을 알아야 하기 때문에 이를 포함하였다.

state.dir

상태 저장소가 생성되는 디렉토리 이름 (절대경로, 예, /tmp/kafks-streams). "영구 저장소 디스크 구조"에서 이 구성을 논의했다. 기억해야 할 중요한 사실은 동일 application.id를 갖는 여러 애플리케이션 인스턴스들이 동일 상태 저장소 디렉토리에 절대로 쓰게 해서는 안된다는 것이다.

topology.optimization

코드가 더욱 효율적으로 실행되게 하기 위해 카프카 스트림즈가 내부 최적화 (예, 네트워크 트립을 줄일 수 있고 리파티션 토픽 수 줄이기 또는 소스 토픽에서 KTable이 생성될 때 불필요한 체인지로그 토픽 생성 방지)를 하길 원하는 경우 all(즉, StreamsConfig.OPTIMIZE)로 설정한다. 이 책 작성 시점에 토폴로지 최적화는 기본적으로 비활성화되어 있다.

upgrade.from

이 파라미터는 카프카 스트림즈를 업그레이드할 때 사용한다. "카프카 스트림즈 업그레이드"에서 이를 논의했다.

컨슈머 특정적인 구성

다음 구성 접두사를 사용하여 카프카 스트림즈가 사용하는 다양한 컨슈머를 구성할 수 있다.

main.consumer.

레드 당 트랜잭션 프로듀서를 생성할 것이다. 더 적은 프로듀서로 정확히 한 번 시맨틱을 얻을 수 있다는 것은 애플리케이션의 메모리 풋프린트와 브로커에 대한 네트워크 연결 수를 줄일 수 있다.

스트림 소스에 사용되는 기본 컨슈머 구성을 위해 이 접두사를 사용한다.

restore.consumer.

체인지로그 토픽으로부터 상태 저장소를 복구하는데 사용되는 복구 컨슈머를 구성하기 위해 이 접두사를 사용한다.

global.consumer.

GlobalKTable을 채우기 위해 사용되는 글로벌 컨슈머를 구성하기 위해 이 접두사를 사용한다.

부록 B ksqlDB 구성

ksqlDB는 대다수 카프카 스트림즈와 카프카 클라이언트 (즉, 프로듀서와 컨슈머) 구성을 수용한다. 권고 패턴은 모든 카프카 스트림즈와 카프카 클라이언트 구성에 ksql.streams 접두사를 붙이는 것이다. 예를 들어 카프카 스트림즈의 cache.max.bytes.buffering 파라미터를 사용하여 레코드 캐시를 구상하려면 server.properties 파일에 ksql.streams.cache.max.bytes.buffering 파라미터를 설정할 것이다. 또한 auto.offset.reset 카프카 컨슈머 구성을 구성하려면 이 또한 동일한 방식으로 접두사가 붙을 것이고 ksql.streams.auto.offset.reset가 될 것이다. 기술적으로 접두사는 선택사항이며 ksqlDB 창시자 (컨플루언트)의 권고사항이다.

표준 카프카 스트림즈 및 카프카 클라이언트 구성 외에 ksqlDB는 ksqlDB의 데이터 통합 기능을 사용하는 경우 (예, CREATE {SOURCE|SINK} CONNECT 문을 실행할 때마다) 카프카 커넥트 구성을 지정할 수 있도록 한다. "커넥트 워커 구성하기"에서 이를 이미 논의하였으며 더욱 자세한 사항은 그 절을 참조하기 바란다.

마지막으로 ksqlDB에 특정적인 몇몇 구성이 있다. 가장 중요한 구성 중 일부를 2 개의 범주로 분류하였다: 쿼리 구성과 서버 구성. 이 페이지는 ksqlDB 구성의 시작점으로 사용되어야 한다. ksqlDB의 전체 구성 목록은 공식 문서를 참고하기 바란다[4].

쿼리 구성

다음 구성은 ksqlDB가 실제 쿼리를 실행하는 방법의 다양한 측면을 제어한다:

ksql.service.id

이 필수 속성은 카프카 스트림즈의 application.id 파라미터와 비슷한 용도로 사용된다: 협업 애플리케이션 그룹을 식별하고 동일 식별자를 공유하는 모든 인스턴스에 작업이 분산될 수 있도록 한다. 카프카 스트림즈의 application.id와 달리 ksqlDB의 service.id와 컨슈머 그룹 ID 간 직접적인 연관은 없다.

기본: 기본값 없음

ksql.streams.bootstrap.servers

이 필수 속성은 카프카 클러스터와 초기 연결을 확립하는데 사용되는 카프카 브로커의 호스트/포트 쌍 목록을 지정한다.

기본: 기본값 없음

---

[4] 공식 문서는 https://docs.ksqldb.io에 있다. 이 부록에 포함된 구성 설명 일부는 이 문서에서 직접 가져온 것이다.

ksql.fail.on.deserialization.error

데이터 처리를 중지하거나 레코드를 역직렬화 할 수 없는 경우 true로 설정한다. 기본값은 false로 ksqlDB에 역직렬화 에러를 기록하고 처리를 계속하라는 것이다. 이는 카프카 스트림즈의 default.deserialization.exception.handler과 비슷해 보인다면 그것도 그렇기 때문이다. ksqlDB 구성은 내부적으로 적절한 예외 처리 클래스를 설정하는데 사용된다.

기본: false

ksql.fail.on.production.error

드문 경우로 카프카 토픽에 데이터를 생산할 때 문제에 직면할 수도 있다. 예를 들어 일시적인 네트워킹 문제는 프로듀서 예외를 야기한다. 프로듀서 예외 발생 시 데이터 처리를 계속하려면 (대부분의 경우 권고되지 않음) 이 파라미터를 false로 설정한다. 그렇지 않은 경우 기본값을 유지한다.

기본: true

ksql.schema.registry.url

쿼리가 컨플루언트 스키마 레지스트리를 필요로 하는 데이터 직렬화 포맷을 사용하는 경우 이 파라미터를 스키마 레지스트리 인스턴스가 실행 중인 URL로 설정한다.

기본: 기본값 없음

ksql.internal.topic.replicas

ksqlDB 서버가 사용하는 내부 토픽의 복제 수. 제품 환경에서는 2 이상의 값이 권고된다.

기본: 1

ksql.query.pull.enable.standby.reads

대기 복제본이 >=1[5] 일 때 이 파라미터는 (정상적으로 읽기 트래픽을 처리할) 활성 태스크가 죽었을 때 대기 복제본이 읽기 요청 제공에 사용될 수 있는 지 여부를 제어한다. True로 설정하는 경우 풀 쿼리에 대해 고가용성을 보장한다.

기본: false

ksql.query.pull.max.allowed.offset.lag

테이블에 대한 풀 쿼리에 의해 허용된 최대 지연량 (즉, 특정 파티션에 대해 읽지 못한 오프셋 수)을 제어한다. 이 구성은 ksql.lag.reporting.enable이 true로 설정된 경우에먼 활성화된다.

---

[5] 이는 카프카 스트림즈 구성에 의해 제어되며 ksql.streams.num.standby.replicas 파라미터를 사용하여 설정될 수 있다.

기본: Long.MAX_VALUE

서버 구성

다음 속성은 ksqlDB 서버 인스턴스 구성에 사용된다.

ksql.query.persistent.active.limit

상호대화식 모드에서 동시에 실행할 수 있는 영구 쿼리 최대 개수. 한계에 도달하면 새로운 쿼리를 시작하기 전에 실행 중인 영구 쿼리 중 하나를 종료해야 한다.

기본: Integer.MAX_VALUE

ksql.queries.file

헤드리스 모드로 ksqlDB를 실행하는 경우 ksqlDB 서버에서 실행할 쿼리를 포함하는 절대 파일 경로로 설정한다. 더욱 자세한 정보는 "배치 모드"를 참조하기 바란다.

기본: 기본값 없음

Listeners

ksqlDB 서버가 수신대기하는 REST API 엔드포인트

기본: 기본 리스너는 모든 IPv4 인터페이스에 바인딩하는 http://0.0.0.0:8088 이다. 모든 IPv6 인터페이스에 바인딩하기 위해서는 http://[::]:8088 로 설정한다. 단지 하나의 인터페이스에 바인딩하기 위해서 이를 특정 인터페이스로 업데이트한다.

ksql.internal.listener

노드 간 통신을 위해 바인딩할 주소. 이는 내부와 외부 엔드포인트에 대해 별도 주소를 바인딩하려는 경우 유용하다 (예를 들어 네트워크 수준에서 다른 엔드포인트를 별도로 보호하려는 경우).

기본: listenrs 속성에 정의된 첫번째 listener

ksql.advertised.listener

이 노드가 내부 통신을 위해 클러스터의 다른 ksqlDB 노드와 공유할 리스너. IaaS 환경에서 서버가 바인딩하는 주소와 달라야 할 수 있다.

기본: ksql.internal.listener 값 또는 설정되지 않은 경우 listeners 속성에 정의된 첫번째 listener

ksql.metrics.tags.custom

방출된 JMX 지표와 함께 포함되는 태그 목록. 콤마로 구분된 키-값 쌍의 문자열 포맷. 예를 들어, key1:value1,key2:value2.

기본: 기본값 없음

보안 구성

ksqlDB가 빠르게 진화하고 있고 보안이 사용가능한 가장 최신 정보를 참조하는 것이 중요한 분야로 이 책에서 보안 구성의 전체 범위를 포착하려 하지 않을 것이다. 대신 ksqlDB의 보안 기능 살펴보기에 대한 공식 문서를 참조하기 바란다.