

Data Streaming 실습 with **Kafka** and **Flink**

연세대학교 원주캠퍼스 DB lab

개요

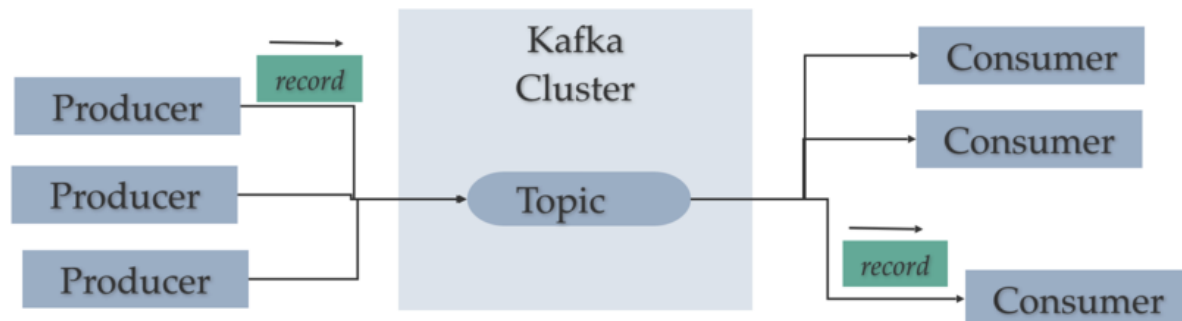
- Apache Kafka
 - 개념
 - 시연 I – AWS를 이용한 Kafka 클러스터(cluster) 구축
 - 시연 II – Kafka를 활용한 메시지 교환
- Flink
 - 개념
 - 시연 I – AWS를 이용한 Flink 클러스터 구축
 - 시연 II – Flink 및 Kafka 를 활용한 어플리케이션
 1. 트위터(Twitter)
 2. 단어 개수 세기(Word Count)
 3. 택시 이용 데이터 분석(Taxi ride)
 - SQL

Apache Kafka – 개념 (1/4) APACHE **kafka**[®] A distributed streaming platform

- Apache Kafka는 스트리밍 데이터를 사용하여 실시간 애플리케이션을 구축할 수 있게 해주는 오픈 소스 분산 메시징 시스템
- 기존의 비동기 메시지 패러다임인 발행-구독(Pub/Sub) 모델을 기반으로 하되, 상대적으로 높은 효율성과 확장성을 기반으로 하여 레코드(record)들을 스트리밍 가능
 - **발행-구독 모델**
발신 메시지는 특별한 수신자가 정해져 있지 않고, 해당 메시지의 구독을 신청한 수신자에게만 전해지게 하는 방식
 - **레코드**
키(key), 값(value) 그리고 타임스탬프(timestamp)로 구성되어 있는 정보 형식

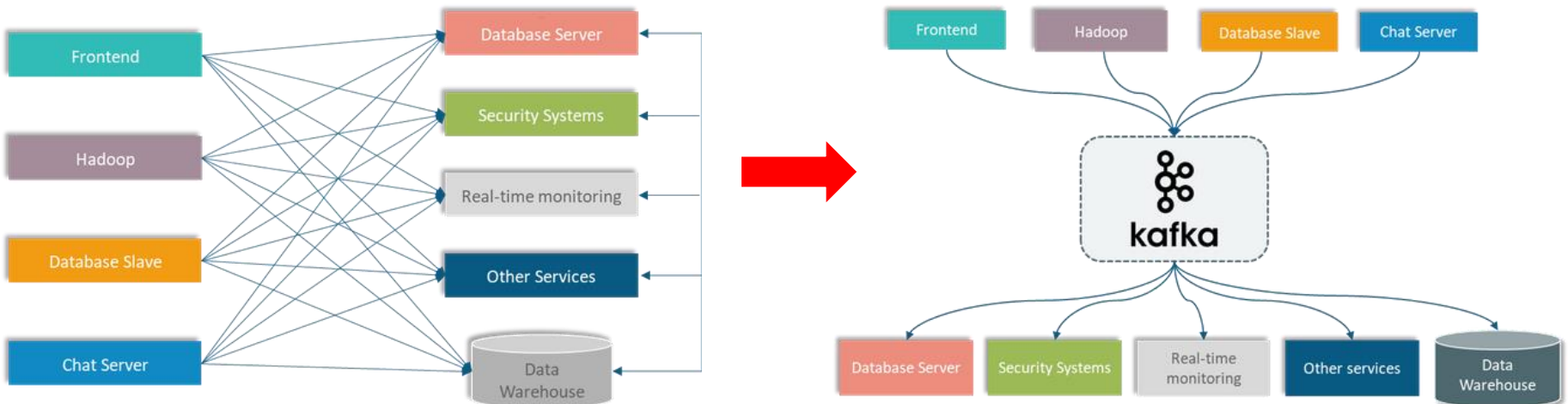
Apache Kafka – 개념 (2/4)

- Kafka 는 브로커(broker)라 불리는 하나 또는 그 이상의 서버들로 구성된 클러스터 형식으로 작동함
- Kafka 클러스터는 토픽(topics)이라 불리는 카테고리안에 스트림 형식의 레코드들을 저장하고 있음
- Kafka 에서 제공하는 핵심 API들
 - Producer API는 하나 이상의 토픽에 레코드 스트림을 생성(produce)하기 위해 사용됨
 - Consumer API는 하나 이상의 토픽을 구독하여 레코드 스트림을 소비(consume)함
 - 이 외에도 다양한 형식의 데이터를 지닌 외부 어플리케이션과 데이터 시스템에 접근하기 위해 Connector API를 제공함



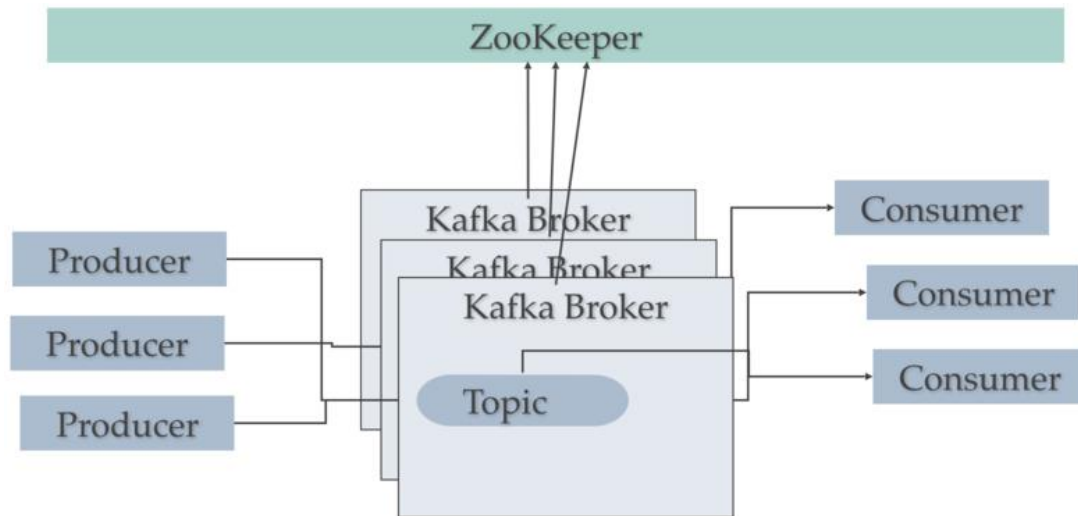
Apache Kafka – 개념 (3/4)

- 높은 확장성: Kafka 는 여러 개의 Consumer 및 Producer들을 낮은 지연(latency)으로 Kafka 클러스터에 연결할 수 있음
- 또한 토픽 자체를 여러 개의 세그먼트(segment)인 파티션(partition)으로 분리하여 하나 또는 그 이상의 데이터가 다양한 용도로 사용되어 질 수 있음
- 이러한 Kafka의 특성들 덕분에, 카프카는 복잡한 데이터 파이프라인들을 분리하고 데이터 형식을 통일하여 데이터의 높은 신뢰성과 처리량(throughput)을 가능하게 함
 - **데이터 파이프라인**
시스템 또는 서비스 간의 커뮤니케이션을 위해 연결을 형성하는 동작들의 집합



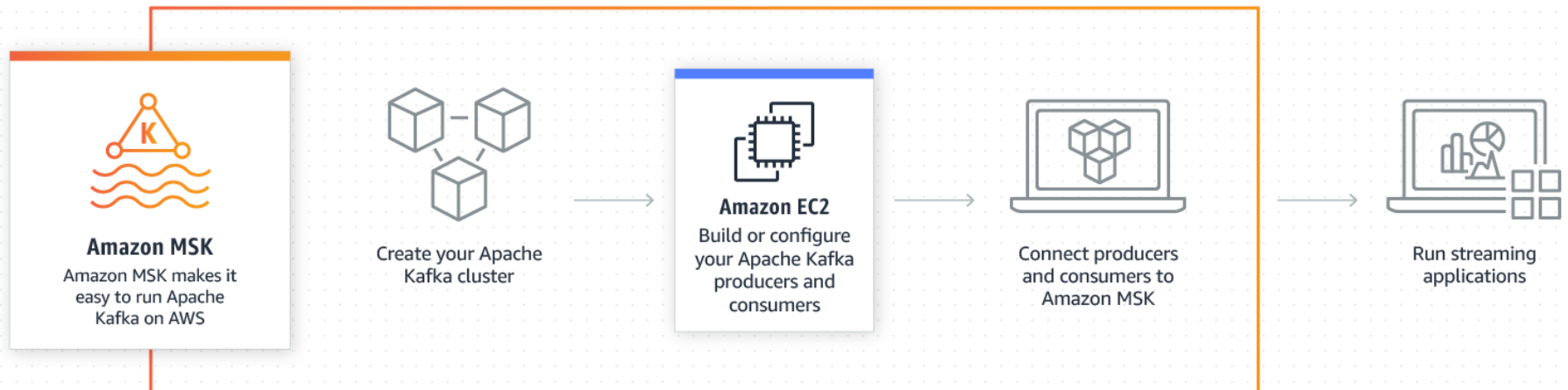
Apache Kafka – 개념 (4/4)

- Kafka 클러스터의 구동을 위해 Zookeeper가 필요함
- Zookeeper란 분산된 프로세스의 조직화(coordinate)를 위한 중앙화된 서비스
- Zookeeper는 Kafka와 직접 통신을 하면서 Kafka에서 얻을 수 있는 토픽, 브로커 등에 대한 메타데이터 정보들을 Zookeeper에 저장함
- 이 정보들을 이용해 Zookeeper는 Kafka 클러스터의 상태 관리와 조직화와 같은 그룹 서비스를 손쉽게 제공할 수 있음



Apache Kafka – 시연 I (1/2)

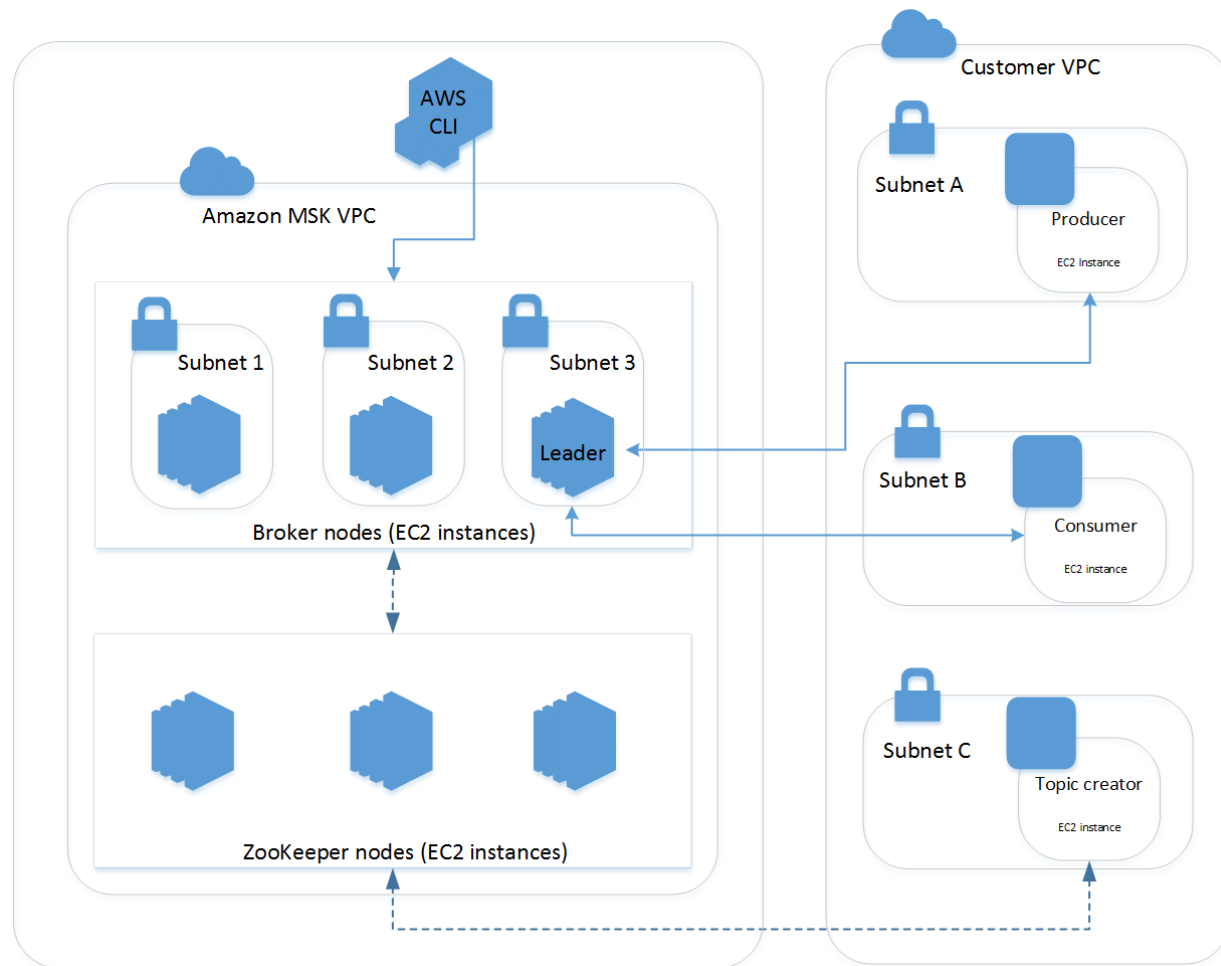
- AWS를 이용한 Kafka 클러스터 구축
- AWS에서 제공되는 서비스 중 하나인 Amazon MSK(Amazon Managed Streaming for Kafka)는 스트리밍 데이터를 사용하는 어플리케이션을 손쉽게 빌드 및 실행하기 위해 완전히 관리되는 서비스
- Amazon MSK를 이용하면 Kafka 인프라 구축 및 관리에 적은 시간을 소요하고 애플리케이션 개발에 더 많은 시간을 할애할 수 있음



Apache Kafka – 시연 I (2/2)

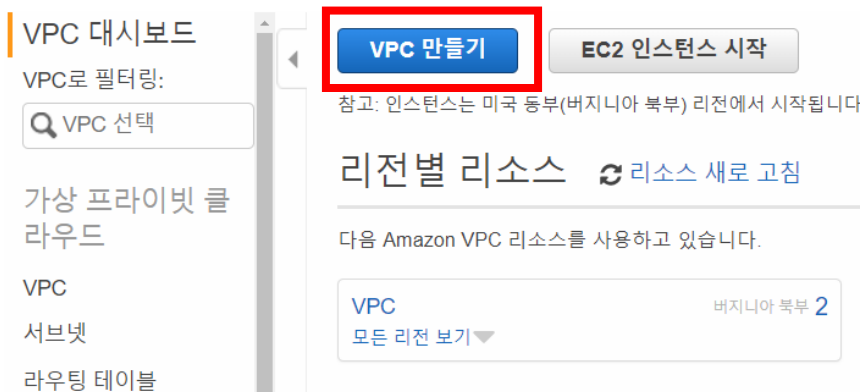
- AWS에서 사용자가 지정한 개수만큼, 미리 설계된 브로커 및 Zookeeper 노드(node)들을 사용자 정의 가상 네트워크(VPC, virtual private cloud)에서 생성할 수 있음
- AWS의 VPC 관리를 통해 지정된 Producer 및 Consumer들만 Amazon MSK에서 생성된 Kafka 클러스터에 접근할 수 있음

Amazon MSK의 동작 다이어그램



시연 I 내용 (1/3)

AWS 콘솔에서 VPC 대시보드로 이동 후 클러스터에 사용할 VPC 생성



생성된 VPC 모습

	Name	Vpc Id	상태	CIDR 블록	IPv6 CIDR	DHCP options set
<input type="checkbox"/>	AWSKafkaT...	vpc-07ed5aa6c4e1c1168	available	10.0.0.0/16	-	dopt-00652e7b

Fault Tolerance와 High Availability 를 위해 Amazon MSK 에서는 VPC 당 최소 3개의 서브넷이 필요함

태그 및 속성별 필터 또는 키워드별 검색						
	Name	서브넷 ID	상태	VPC	IPv4 CIDR	사용 가능한 IPv
<input type="checkbox"/>	AWSKafkaT...	subnet-0a398207ffd2a3565	available	vpc-07ed5aa6c4e1c1168 ...	10.0.0.0/24	251
<input type="checkbox"/>	AWSKafkaT...	subnet-0c7cf08aef96cef56	available	vpc-07ed5aa6c4e1c1168 ...	10.0.1.0/24	251
<input type="checkbox"/>	AWSKafkaT...	subnet-0fcc1e4c78377147b	available	vpc-07ed5aa6c4e1c1168 ...	10.0.2.0/24	251

시연 I 내용 (2/3)

- AWS CLI (Command Line interface) 또는 Amazon MSK 서비스 웹에서 Kafka 클러스터 생성

```
CLI> aws kafka create-cluster --cluster-name "AWSKafkaTutorialCluster" --broker-node-group-info file://brokernodegroupinfo.json --kafka-version "2.1.0" --number-of-broker-nodes 3 --enhanced-monitoring PER_TOPIC_PER_BROKER --region us-east-1
```

- Amazon MSK 웹

MSK > Clusters > Create cluster

Create Kafka cluster

General

Cluster name
You can't change it after you create the cluster.

AWSKafkaTutorialCluster

The name must be unique and can have a maximum of 64 characters.

VPC
Defines the virtual networking environment for this cluster. You can't change this setting after you create the cluster.

vpc-07ed5aa6c4e1c1168 (AWSKafkaTutorialVPC)

Apache Kafka version
Software version for deployed Kafka brokers.

☒ 2.1.0

☐ 1.1.1

이전 슬라이드에서
만들어 둔 VPC를 사용
(CLI에서는 json 파일에
VPC 및 서브넷 정보가
담겨있음)

시연 I 내용 (3/3)

- 최종적으로 생성된 Kafka 클러스터

Clusters (1)

[Delete](#)[Create cluster](#)

Use the CLI to retrieve cluster and broker connection details. [Learn more](#)

[<](#) [1](#) [>](#)

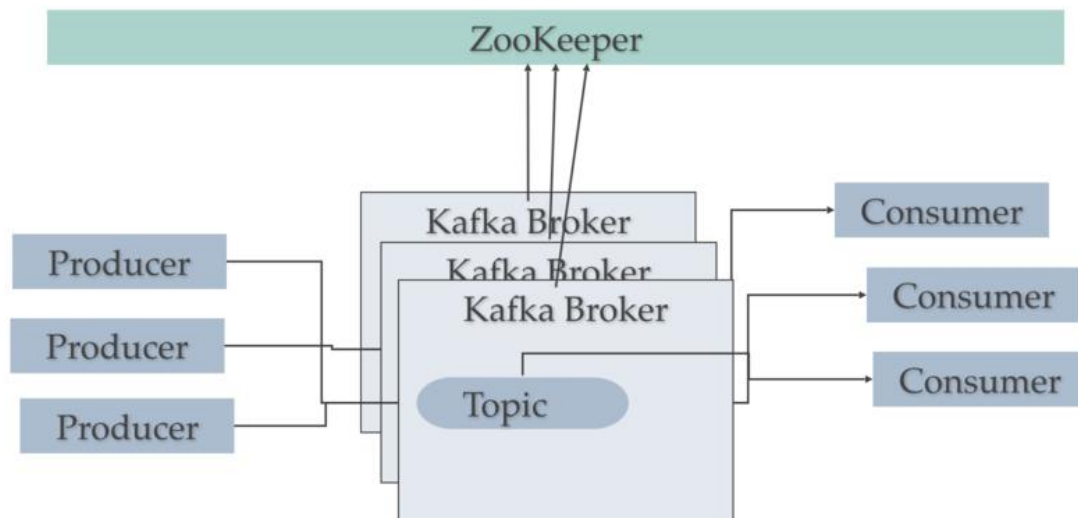
	Name	Status	Apache Kafka version	Brokers per Availability Zone	Availability Zones	Creation time
<input type="radio"/>	AWSKafkaTutorialCluster	Active	2.1.0	1	3	Wed, 06 Feb 2019 12:28:40 GMT

- AWS CLI 에서 명령어를 통해서도 생성됨(ACTIVE)을 확인 가능

```
"ClusterName": "AWSKafkaTutorialCluster",
"CreationTime": "2019-02-06T12:28:40.614Z",
"CurrentBrokerSoftwareInfo": {
  "KafkaVersion": "2.1.0"
},
"CurrentVersion": "K13V1IB3VIYZZH",
"EncryptionInfo": {
  "EncryptionAtRest": {
    "DataVolumeKMSKeyId": "arn:aws:kms:us-east-1:333329361832:key/58e89"
  }
},
"EnhancedMonitoring": "DEFAULT",
"NumberOfBrokerNodes": 3,
"State": "ACTIVE",
"ZookeeperConnectString": "10.0.2.159:2181,10.0.1.218:2181,10.0.0.44:2181"
```

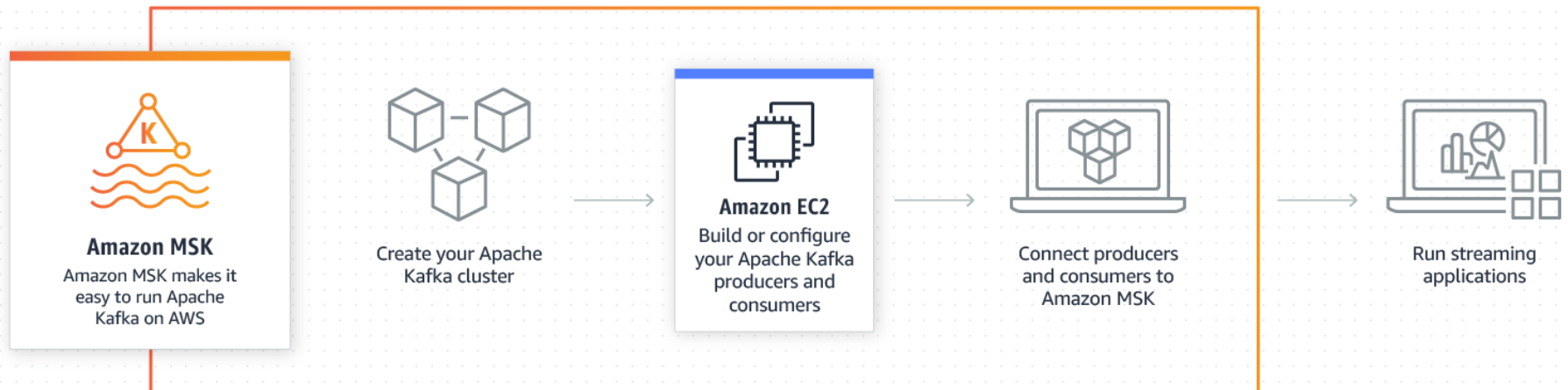
Apache Kafka – 시연 II (1/2)

- 구축된 Kafka 클러스터를 구성하는 브로커 노드가 Zookeeper에 접근하여 토픽 생성
- producer 역할을 맡은 노드가 생성된 토픽으로 레코드 스트림을 보냄
- consumer 역할을 맡은 노드는 Kafka 클러스터에 접근하여 원하는 토픽을 명시하고, 해당 토픽의 레코드를 실시간으로 스트리밍함



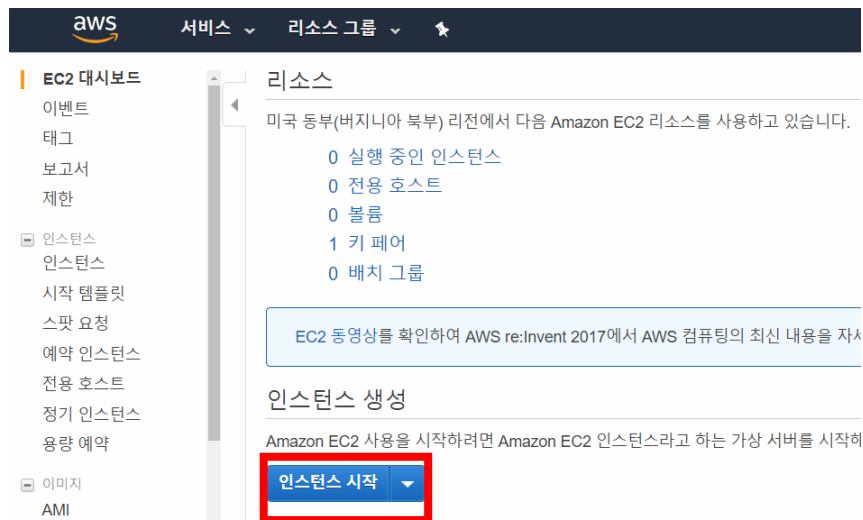
Apache Kafka – 시연 II (2/2)

- 이미 만들어진 Kafka 클러스터를 제외한, producer 및 consumer 역할을 맡을 노드들은 AWS에서 제공하는 Amazon EC2 서비스를 통해 생성할 수 있음
- Amazon EC2(Elastic Compute Cloud)는 안전하고 크기 조정이 가능한 컴퓨팅 파워를 클라우드에서 제공하는 웹 서비스
- Amazon EC2를 사용하여 사용자가 지정한 성능 및 용량을 지닌 서버 인스턴스(instance)를 몇 분 내로 빠르게 부팅하고 배포할 수 있음



시연 표 내용 (1/4)

- AWS 콘솔에서 EC2 대시보드로 이동 후 producer와 consumer 역할을 맡을 client machine (EC2 인스턴스) 생성



생성된 EC2 인스턴스 모습

Name	인스턴스 ID	인스턴스 유형	가용 영역	인스턴스 상태	상태 검사	경보 상태	퍼블릭 DNS(IPv4)	IPv4 퍼블릭 IP
AWSKafkaT...	i-08d0e04681d2f0d0f	t2.large	us-east-1a	running	초기화	없음	ec2-3-92-182-233.com...	3.92.182.233

인스턴스를 생성할 때 생성한 private key 파일을 가지고 독립 실행형 SSH 클라이언트를 통해 연결

```
$ ssh -i "MSKKeyPair.pem" ec2-user@ec2-3-92-182-233.compute-1.amazonaws.com
```

시연 표 내용 (2/4)

- 인스턴스와 연결에 성공한 모습

Last login: Wed Feb 6 12:43:12 2019 from 165.132.214.219

```
 _ | _ | _ )  
 _ | ( _ | /  
 _ | \ _ | _ |  
Amazon Linux 2 AMI
```

<https://aws.amazon.com/amazon-linux-2/>
3 package(s) needed for security, out of 3 available
Run "sudo yum update" to apply all updates.

- EC2 인스턴스와 Kafka 클러스터와의 연결을 위해
해당 보안 그룹에서 인바운드 규칙을 수정할 필요가 있음

인바운드 규칙 편집 ✕

유형 ⓘ	프로토콜 ⓘ	포트 범위 ⓘ	소스 ⓘ	설명 ⓘ	
모든 트래픽 ▾	모두	0 - 65535	사용자 지정 ▾ sg-03bf97719b70f2f26	예: SSH for Admin Desktop	✕
모든 트래픽 ▾	모두	0 - 65535	사용자 지정 ▾ sg-0b71151f419e1c60c	예: SSH for Admin Desktop	✕
모든 트래픽 ▾	모두	0 - 65535	사용자 지정 ▾ sg-08f975593d285b8d6	예: SSH for Admin Desktop	✕

규칙 추가

참고: 기존 규칙을 편집하면 편집된 규칙이 삭제되고 새 세부 정보로 새 규칙이 생성됩니다. 이렇게 하면 새 규칙이 생성될 때까지 해당 규칙에 의존하는 트래픽이 잠시 중단될 수 있습니다.

시연 II 내용 (3/4)

- 인스턴스의 설정이 모두 완료되었으므로 SSH 로 접속된 인스턴스에서 java와 kafka를 설치 및 다운로드

```
[ec2-user@ip-10-0-0-120 ~]$ java -version
openjdk version "1.8.0_191"
OpenJDK Runtime Environment (build 1.8.0_191-b12)
OpenJDK 64-Bit Server VM (build 25.191-b12, mixed mode)
[ec2-user@ip-10-0-0-120 ~]$ ls
kafka_2.11-2.1.0  kafka_2.11-2.1.0.tgz
```

- AWS CLI에서 Kafka 클러스터 정보를 통해 zookeeper 노드에 대한 주소를 획득

```
"EnhancedMonitoring": "DEFAULT",
"NumberOfBrokerNodes": 3,
"State": "ACTIVE",
"ZookeeperConnectionString": "10.0.2.159:2181,10.0.1.218:2181,10.0.0.44:2181"
```

- 다운받은 Kafka 를 이용해 원하는 토픽 생성

```
[ec2-user@ip-10-0-0-120 kafka_2.11-2.1.0]$ bin/kafka-topics.sh --create --zookeeper "10.0.2.159:2181,10.0.1.218:2181,10.0.0.44:2181" --replication-factor 3 --partitions 1 --topic AWSKafkaTutorialTopic
Created topic "AWSKafkaTutorialTopic".
```


시연 II 내용 (4/4)

- 토픽을 생성하는데 성공하였으니 해당 토픽에 대해 레코드를 produce 하고 consume 할 수 있음
- AWS CLI을 통해 Kafka 클러스터를 구성하는 브로커에 대한 정보를 알아냄

```
"BootstrapBrokerString": "10.0.1.108:9092,10.0.2.114:9092,10.0.0.90:9092"
```

- 해당 브로커 정보를 이용해 원하는 토픽에 메시지를 produce 함

```
^C[ec2-user@ip-10-0-0-120 kafka_2.11-2.1.0] bin/kafka-console-producer.sh --broker-list "10.0.1.108:9092,10.0.2.114:9092,10.0.0.90:9092" --topic AWSKafkaTutorialTopic
>hello
>nice to meet you
```

- produce 된 메시지를 consumer 콘솔을 통해 확인하는 모습

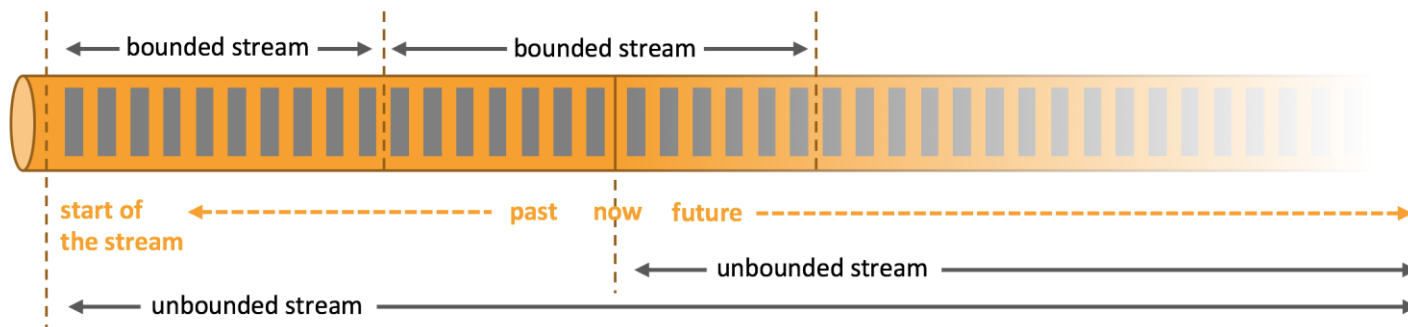
```
[ec2-user@ip-10-0-0-120 kafka_2.11-2.1.0]$ bin/kafka-console-consumer.sh --bootstrap-server "10.0.1.108:9092,10.0.2.114:9092,10.0.0.90:9092" --topic AWSKafkaTutorialTopic --from-beginning
hello
nice to meet you
```

- 만일 두개의 인스턴스가 존재한다면 실시간으로 메시지를 주고받을 수 있음

Apache Flink – 개념 (1/4)



- Apache Flink는 unbound 또는 bound 된 데이터 스트림에 기반한 state 처리를 제공하는 분산 프로세싱 엔진 및 프레임워크
 - 데이터는 unbound 또는 bound된 스트림 형태로 처리될 수 있음
 - Unbound된 스트림은 시작은 존재하지만 끝이 존재하지 않음
 - Bound된 스트림은 시작과 끝이 모두 존재함
- Flink는 Hadoop YARN, Apache Mesos와 같은 일반적인 자원 관리 클러스터와 손쉽게 통합되어 어플리케이션을 실행할 수 있음
- 뿐만 아니라, 스트리밍 어플리케이션들의 향상된 퍼포먼스를 위해 앱(App)을 여러 개의 태스크(task)로 나눠 병렬로 처리하거나 테라바이트급 이상의 어플리케이션 상태들을 간단히 관리 할 수 있음.

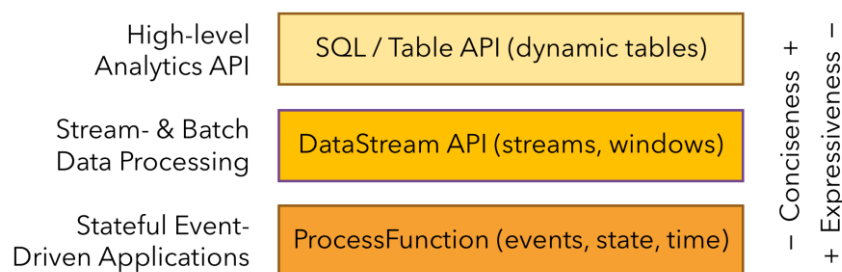


Apache Flink – 개념 (2/4)

- Flink와 같은 스트림 프로세싱 프레임워크에 의해 빌드 및 실행되는 어플리케이션은, 프레임워크가 스트림(stream), 상태(state) 그리고 시간(time)을 얼마나 잘 다루는지에 따라 성능이 결정됨
 - **스트림**
bound 또는 unbounded, 실시간 또는 기록된 스트림이 존재
 - **상태**
이벤트 발생 순간, 함수 작동의 중간 결과, 프로세스 처리상황 등의 상태 정보를 활용
 - **시간**
Event-time 또는 Processing-time Mode를 제공하여 어떻게 어플리케이션이 시간의 흐름을 측정할지 도움을 줌
 - Event-time: 이벤트가 실제로 발생한 시간을 기준으로 측정
 - Processing-time: 프로세스가 시작된 순간부터 시간을 측정

Apache Flink – 개념 (3/4)

- Flink는 추상화 레벨에 따른 다양한 API들을 제공하고 스트림 데이터의 일반적인 사용 사례에 대한 라이브러리들을 제공
- 전반적으로 Flink는 3개의 레벨 간 API들을 제공하며 각각의 API들은 사용 목적에 따라 간결성과 복잡성 간의 균형(trade-off)이 존재함



- **ProcessFunction (API)**
가장 복잡도가 높은 인터페이스로서, 하나 또는 두개의 스트림으로 부터 발생된 개별적 이벤트들 또는 윈도우 단위로 묶인 이벤트들을 처리하고, 시간과 상태에 대한 섬세한 조절을 가능케함
- **DataStream API**
맵리듀스(map-reduce), aggregate, 윈도우 처리, 레코드 변환, 쿼리를 통한 다수의 이벤트 생성 등 스트림 처리 작업에서 가장 주로 쓰이는 핵심 작업들을 수행할 수 있도록 해 줌
- **SQL/ Table API**
가장 간결성이 높은 인터페이스로서 -API의 이름 그대로- 주어진 스트림 데이터를 테이블 형식 데이터로 변환하거나 변환된 데이터를 통해 SQL 쿼리를 처리할 수 있게 해 줌

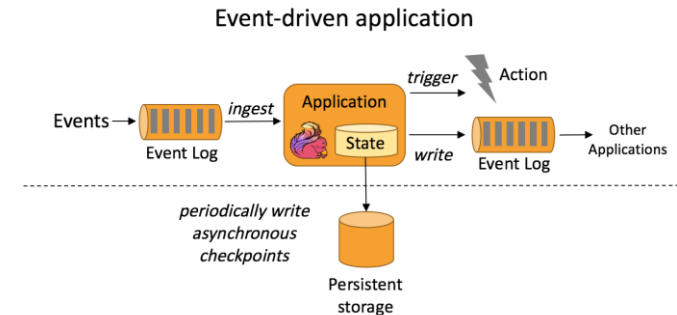
Apache Flink – 개념 (4/4)

- Apache Flink의 실제 사용 사례

1. 이벤트 기반의 어플리케이션

하나 이상의 이벤트 스트림을 어플리케이션에 주입하게 되면 계산 처리, 상태 업데이트, 또는 외부 행동을 함으로써 들어오는 이벤트에 대해 반응함

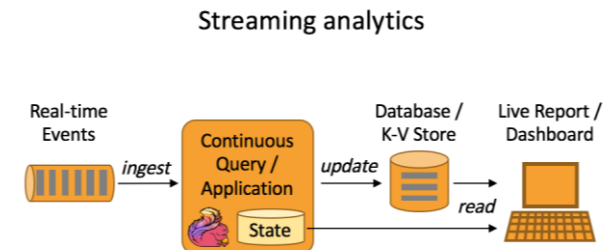
- 예) 사기 검출, 비정상 감지, 비즈니스 프로세스 모니터링, 소셜 네트워크 웹 앱



2. 데이터 분석 어플리케이션

실시간 이벤트를 받아들여서 쿼리, 스트림 작업을 통해 정보를 추출하고 결과를 저장소에 업데이트한 뒤, 대시보드 앱이 저장소에 접근해 데이터를 분석함

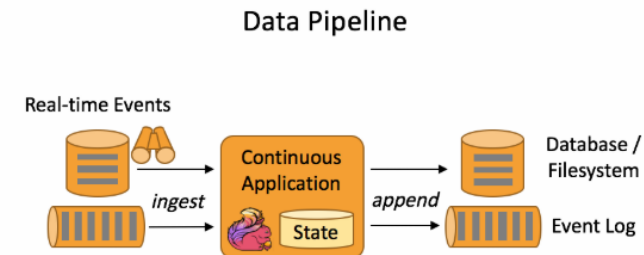
- 예) 대용량 그래프 분석, 네트워크 품질 모니터링, 소비 생활 패턴 분석



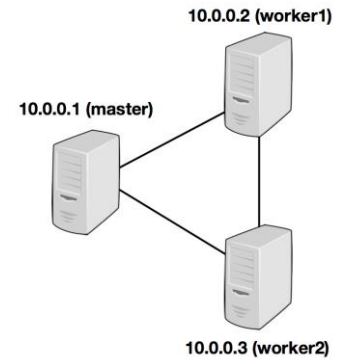
3. 데이터 파이프라인 어플리케이션

소스(source)로 부터 추출된 데이터를 변환하고 변환된 데이터를 저장소로 옮기는 작업을 주기적이 아닌 실시간으로 처리하여 적은 지연만으로 파이프라인 작업을 가동함

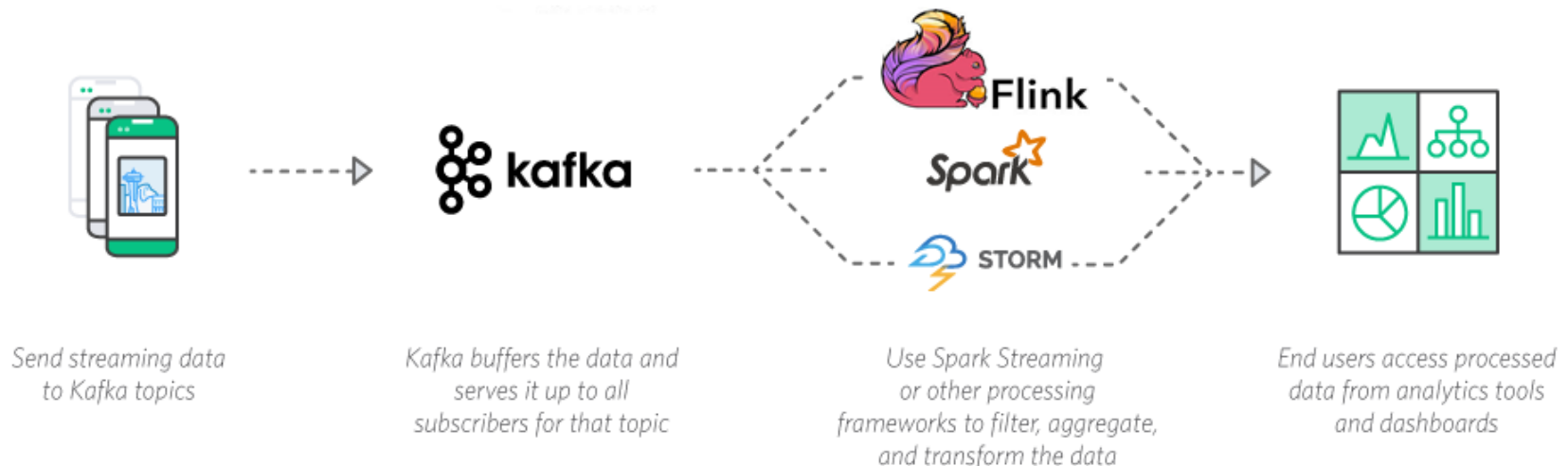
- 예) 실시간 검색 인덱스 빌드, 파일 시스템 모니터링



Apache Flink – 시연 I (1/2)



- Flink는 -마스터 (master)또는 워커(worker)라 불리는- 하나 또는 그 이상의 노드들을 이용해 클러스터를 형성하여 스트림 작업을 처리함
- 마스터 노드는 JobManager를 이용하여 스트림 이벤트를 처리하기 위한 job을 task 형태로 나누어 워커 노드에게 분배하고, 워커 노드는 TaskManager를 이용하여 분배 받은 task를 처리함
- Kafka 클러스터의 지정된 토픽에 저장되어 있는 메시지를 실시간으로 읽어와 Flink 클러스터에서 스트림 작업을 처리할 수 있음



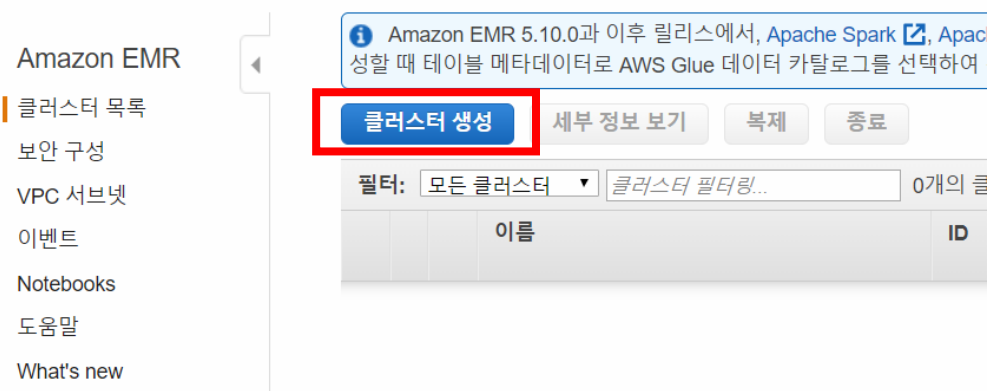
Apache Flink – 시연 I (2/2)



- AWS를 이용한 Flink 클러스터 구축
- AWS에서 제공하는 Amazon EMR은 AWS에서 Apache Hadoop 및 Apache Spark와 같은 빅 데이터 프레임워크 실행을 간소화하는 관리형 클러스터 플랫폼
- Amazon EMR의 중심 구성 요소는 클러스터이며 이 클러스터는 Amazon EC2 인스턴스의 집합
- Kafka 인프라 관리에 집중되어 있는 Amazon MSK와 달리 Amazon EMR은 빅데이터 처리를 위한 여러 구성 요소들을 패키징화 하여 클러스터 형태로 손쉽게 배포할 수 있음

시연 I 내용 (1/3)

AWS 콘솔에서 Amazon EMR 대시보드로 이동 후 Flink를 위한 클러스터 생성



생성된 클러스터 확인, 또한 세부 정보에서 Flink를 바로 사용 가능함을 확인 할 수 있음

	이름	ID	상태
<input type="checkbox"/>	FlinkTutorialCluster	j-Q2T1VOHNYBPZ	시작

구성 세부 정보

릴리스 레이블: emr-5.20.0

Hadoop 배포: Amazon 2.8.5

애플리케이션: Hive 2.3.4, Pig 0.17.0, Hue 4.3.0, Flink 1.6.2

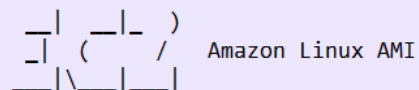
로그 URI: s3://aws-logs-333329361832-us-east-1/elasticmapreduce/

EMRFS 일관된 보기: 비활성화

사용자 지정 AMI ID: --

시연 I 내용 (2/3)

- EMR 클러스터의 마스터 노드에 SSH 클라이언트로 접근



<https://aws.amazon.com/amazon-linux-ami/2018.03-release-notes/>
11 package(s) needed for security, out of 15 available
Run "sudo yum update" to apply all updates.

```
EEEEEEEEEEEEEEEEEEEE MMMMMMM      MMMMMMM RRRRRRRRRRRRRR
E::::::::::::::::::::E M::::::::M      M::::::::M R:::::::::R
EE::::::::EEEEEEEEEE E M::::::::M      M::::::::M R::::RRRRR::::R
  E::::E      EEEEE M:::::::::M      M:::::::::M RR::::R      R::::R
  E::::E      M::::::::M::M      M:::M:::::M      R:::R      R:::R
  E::::EEEEEEEEEE M::::M M:::M M:::M M::::M      R::RRRRR::::R
  E::::::::::::E M::::M M:::M::M M::::M      R:::::::::RR
  E::::EEEEEEEEEE M::::M M::::M M::::M      R::RRRRR::::R
  E::::E      M::::M M:::M M::::M      R:::R      R::::R
  E::::E      EEEEE M::::M      MMM      M::::M      R:::R      R::::R
EE::::::::EEEEEEEEEE E M::::M      M::::M      R:::R      R::::R
E::::::::::::E M::::M      M::::M RR::::R      R::::R
EEEEEEEEEEEEEEEEEEEE MMMMMMM      MMMMMMM RRRRRRR      RRRRRR
```

- 접속한 마스터 노드에서 Flink 클러스터 세션 시작

```
[hadoop@ip-10-0-0-248 ~]$ flink-yarn-session
```

시연 I 내용 (3/3)

- Dynamic port-forwarding 및 브라우저 Proxy 설정 후 (AWS 에서 설명서 제공) 리소스 관리자에 접속하여 Flink에서 제공하는 UI를 확인 가능



Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed
1	0	1	0

Cluster Nodes Metrics

Active Nodes	Decommissioning Nodes
2	0

Scheduler Metrics

Scheduler Type	Scheduling Resource Type
Capacity Scheduler	[MEMORY]

Show 20 entries

ID	User	Name	Application Type	Queue	Applicat Priority
----	------	------	------------------	-------	-------------------

application 1549460062456 0001	hadoop	Flink session cluster	Apache Flink	default	0
--------------------------------	--------	-----------------------	--------------	---------	---

Showing 1 to 1 of 1 entries

Apache Flink Dashboard

- Overview
- Running Jobs
- Completed Jobs
- Task Managers
- Job Manager
- Submit new Job

Overview

Version: 1.6.2

Commit: <unknown>



0
Task Managers



0
Task Slots



0
Available Task Slots

Apache Flink – 시연 II

- Apache Flink에서 제공하는 여러 API들을 적용한 예시 데이터 스트림 어플리케이션들

1. Twitter

2. Word Count

3. Taxi Ride (data Analysis)

Apache Flink – Twitter



- 트윗 스트림 데이터 생성 어플리케이션
- 트위터(developer.twitter.com)에서 제공한 토큰 및 키(key)를 획득하여 Flink의 twitter connector API에 적용하면 실시간으로 올라오는 트윗 정보를 json 포맷의 데이터로 얻을 수 있음
- 이 어플리케이션은 얻어온 데이터에서 트윗의 내용만 따로 추출한 뒤에, producer 로써 Kafka 클러스터에 트윗 내용을 스트리밍함

Twitter 시연 내용 (1/3)

- EMR 클러스터의 마스터 노드에서 카프카를 다운받은 후 트위터 관련 레코드를 전달할 토픽(topic) 생성

```
[hadoop@ip-10-0-0-248 kafka_2.11-2.1.0]$ bin/kafka-topics.sh --create --zookeeper "10.0.2.159:2181,10.0.1.218:2181,10.0.0.44:2181" --replication-factor 3 --partitions 1 --topic twitterText  
Created topic "twitterText".
```

- 트위터 개발자 사이트에서 트윗 정보를 실시간으로 받아올 토큰과 키를 생성

Keys and tokens

Keys, secret keys and access tokens management.

Consumer API keys

blZVdN [REDACTED] (API key)

9c3dSVLA [REDACTED] (API secret key)

Regenerate

Access token & access token secret

38 [REDACTED]-sVF0A [REDACTED] (Access token)

c1uWi8 [REDACTED] (Access token secret)


Read and write (Access level)

Revoke

Regenerate

Twitter 시연 내용 (2/3)

- 획득한 키와 토큰 그리고 Kafka 클러스터 정보를 사전에 하드 코딩한 뒤 빌드하여 놓은 jar 파일을 Flink UI 를 통해 업로드 함
- 실행하기를 원하는 어플리케이션 class 를 명시하고 submit

 Apache Flink Dashboard

Overview

Running Jobs

Completed Jobs

Task Managers

Job Manager

Submit new Job

Submit new Job

Uploaded Jars

	Name	Upload Time	Entry Class
<input checked="" type="checkbox"/>	flink-training-exercises.jar	2019-02-06, 23:35:11	

com.dataartisans.flinktraining.examples.datastream_java.connectors.TwitterToKafkaAsString

Parallelism

Show Plan

Program Arguments

Submit

Savepoint Path

☐ Allow Non Restored State

Twitter 시연 내용 (3/3)

- EMR 마스터
노드에서 Kafka
클러스터에
consumer로 접근하여
실시간으로 트윗
되는 내용을 확인
가능
- 또한 현재 실행되고
있는 Job을 Flink
UI에서도 확인 가능

```
[hadoop@ip-10-0-0-248 kafka_2.11-2.1.0]$ ./bin/kafka-console-consumer.sh --bootstrap-server 10
.0.1.108:9092,10.0.2.114:9092,10.0.0.90:9092 --topic twitterText --from-beginning_
선착순 326분께 보내드리고 품 제출은 11시 이후부터 인정합니다!!...
게하 재미있어 다인시날은 다 재미있지만...아무튼 재미있어
RT @btscloudpiece: 미래의 나에게 주는 덕질 자금
고마워해라 미래의 나야 https://t.co/dmLd4rjb8y
@FF14_dangoman 크으 넘 잘어울려요~~^!~!~!~!!!! (쪼압!) 역시 선물드리길르잘했습니다 다음에드
입고 싶은 옷이 잇으시다면 루를루를-멘션맨션 주세요~~~!~!~!~!!!!^^)9
RT @avocadomato: 아 미친ㅋㅋㅋㅋ Die With Me 라는 천원짜리 앱이 등장했는데 배터리가 5% 이하일
때만 접속 가능한 채팅방에서 다른 사람들이랑 죽기 직전(ㅋㅋ) 의 대화를 나누는 거래 광고 문구 개
웃김 "당신의 오프라인 평화를...
RT @mcrchive: https://t.co/J0vgGuKDOH
RT @yl58126824: 우리구오즈~~축하축하

#PromiseByJimin50M
#TaehyungIsMyScenery40M

둘다 너무나 이쁘고

사랑해~

#Top50FansBTS
#방탄과아미는하나
@BTS_twt https://t...
@HJW928 오와 아침일찍부터 도서관을,,,, 잘 다녀오시구 잘자요 쌤 ♡
@sso_gold 할 완전 먹음직한데요? 저 머핀종이? 까지 큐티 완벽..
RT @VERNON_FLOWER_: 기 받아가요(구움때)
1.티켓팅할때 단 한번도 광탈 없었다(3000번대가 제일 늦은거였음)
2.취켓팅으로 스텐딩 딱 한번, 2층 좌석 정중앙 앞열or중간열 여러번 갔다음
3.티켓팅 시간 놓쳐서 허탈한채로 티켓팅 들...
@eom_7 야호~
```

Running Jobs

Start Time	End Time	Duration	Job Name	Job ID	Tasks	Status
2019-02-06, 23:51:22	2019-02-06, 23:59:27	8m 5s	Twitter Streaming Example	ba92d52e969bf7334d4170bff686c002	10001000000	RUNNING

Flink 프로그램 과 데이터 흐름

- 개념적으로 'stream'은 데이터의 흐름이고, 'transformation'은 하나 이상의 stream을 입력으로 받아서, 하나 이상의 stream을 결과로 생산하는 것
- Flink 프로그램은 stream과 transformation이 포함된 streaming 데이터 흐름에 적용됨
- 각 stream은 하나 이상의 source로 시작해서, 하나 이상의 sink에서 끝남
- 데이터의 흐름은 방향이 있는 임의의 비순환 그래프(DAGs)와 비슷함 (오른쪽 그림)

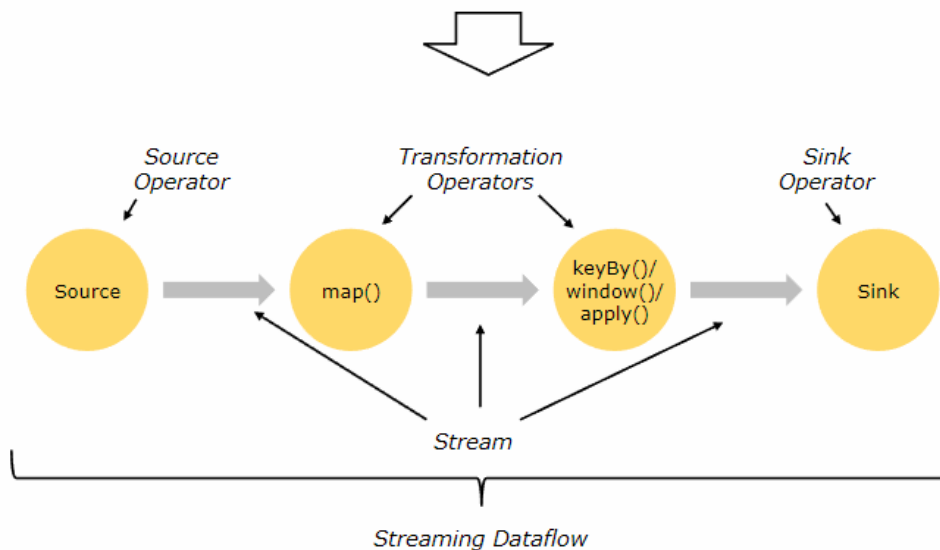
```
DataStream<String> lines = env.addSource(  
    new FlinkKafkaConsumer<> (...));  
DataStream<Event> events = lines.map((line) -> parse(line));  
DataStream<Statistics> stats = events  
    .keyBy("id")  
    .timeWindow(Time.seconds(10))  
    .apply(new MyWindowAggregationFunction());  
stats.addSink(new RollingSink(path));
```

Source

Transformation

Transformation

Sink



Twitter App, Main 함수 코드

// 스트림 어플리케이션 실행 환경

```
StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
```

// 웹 인터페이스에서 parameters 확인 가능하도록 설정

```
env.getConfig().setGlobalJobParameters(params);
```

```
env.setParallelism(params.getInt("parallelism", 1));
```

// 스트림 데이터 병렬 처리 정도

```
DataStream<String> streamSource;
```

// 트윗 정보를 받아올 데이터 스트림 형식 변수

```
Properties props = new Properties();
```

// Twitter connector API를 위한 토큰과 키 설정

```
props.setProperty(TwitterSource.CONSUMER_KEY, "b1ZVd ... g5S");
```

```
props.setProperty(TwitterSource.CONSUMER_SECRET, "9c3dSV ... 7cXVxEq");
```

```
props.setProperty(TwitterSource.TOKEN, "38... 892-s... NctnE");
```

```
props.setProperty(TwitterSource.TOKEN_SECRET, "c1uW.. 5Mr");
```

// 데이터 스트림 변수에 스트림 Source(Twitter connector API) 추가

```
streamSource = env.addSource(new TwitterSource(props));
```

```
DataStream<String> tweets = streamSource
```

// 한국 트윗 정보만 선택하고 선택한 정보(json 형식) 중에서 오직 text 값만 필터링함 (Map 함수)

```
.flatMap(new SelectKoreanAndFilterOnlyText());
```

// 최종적으로 얻어온 데이터 값을 미리 설정한 Kafka connector API를 통해 Kafka 클러스터 넘겨줌

```
tweets.addSink(new FlinkKafkaProducer<>(  
    LOCAL_KAFKA_BROKER,  
    CLEANSED_RIDES_TOPIC,  
    new SimpleStringSchema()));
```

Source

Transformation

Sink

// 프로그램을 실행시킴

```
env.execute("Twitter Streaming Example");
```

Twitter App, Map 함수 코드

```
// 한국 트윗 정보만 선택하고 선택한 정보(json 형식) 중에서 오직 text 값만 필터링함 (Map 함수)
public static class SelectKoreanAndFilterOnlyText implements FlatMapFunction<String, String> {
    private static final long serialVersionUID = 1L;

    private transient ObjectMapper jsonParser;

    /**
     * 들어온 JSON 텍스트에서 원하는 언어를 선택 (ko: 한글, en: 영어).
     * value : 들어온 JSON 텍스트, out : 변환을 거쳐 필터링 된 값 (map 함수의 최종 출력)
     */
    @Override
    public void flatMap(String value, Collector<String> out) throws Exception {
        if (jsonParser == null) {
            jsonParser = new ObjectMapper();
        }
        JsonNode jsonNode = jsonParser.readValue(value, JsonNode.class);

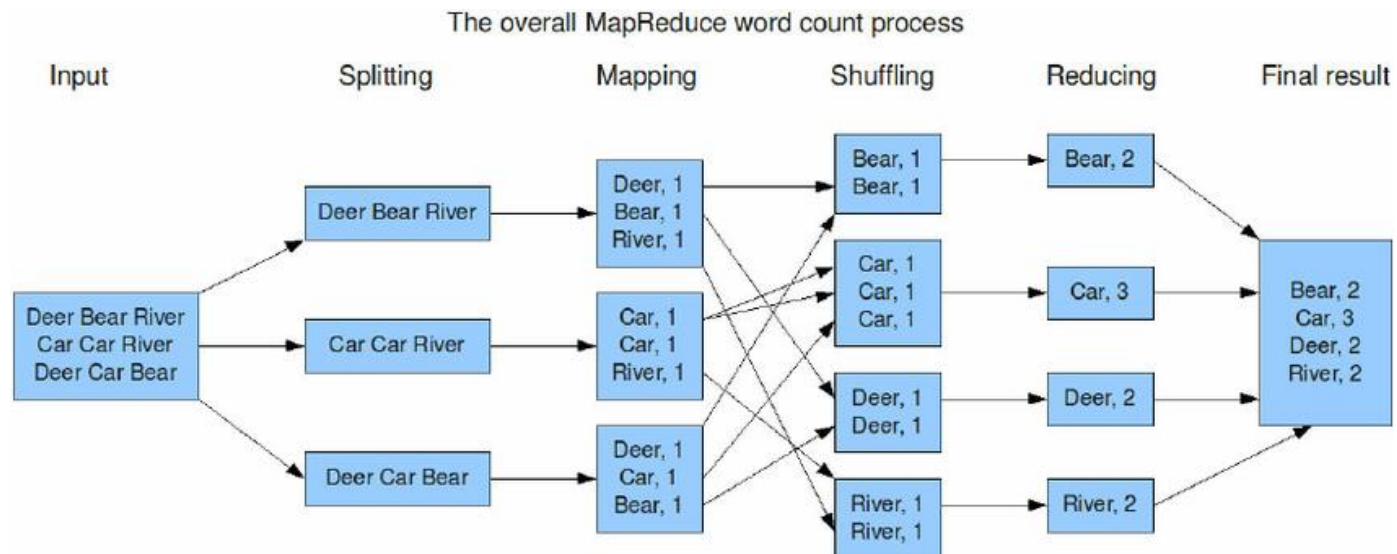
        boolean isKorean = jsonNode.has("user") && jsonNode.get("user").
            has("lang") && jsonNode.get("user").
            get("lang").asText().equals("ko");

        boolean hasText = jsonNode.has("text");

        if (isKorean && hasText) {
            // 트윗의 메시지를 collect 함수를 통해 최종 출력
            out.collect(jsonNode.get("text").asText());
        }
    }
}
```

Apache Flink – Word Count

- Word Count, 단어 수를 세는 어플리케이션
- 들어오는 문장 데이터를 적절하게 단어 단위로 나누어 맵리듀스 방식을 사용하여 실시간으로 각 단어의 빈도수를 계산함
- 사용할 문장 데이터는 이전 시연에서의 트윗 스트림 데이터를 Kafka 클러스터의 토픽에 접근하여 얻어올 수 있음



Word Count에서의 Sliding Window

- 최근 15초 동안 트윗된 문장들을 단어 단위로 나누어 5초 간격으로 단어 개수를 계산
- 15초는 Sliding Window의 크기 그리고 5초는 Window의 sliding step




(Happy, 2)
(New, 4)
(Year, 5) ...

(Happy, 5)
(New, 2)
(Pig, 4) ...

WordCount 시연 내용 (1/2)

- Kafka 클러스터 정보를 사전에 하드 코딩한 뒤 빌드하여 놓은 jar 파일을 Flink UI 를 통해 업로드 함
- 실행하기를 원하는 어플리케이션 class 를 명시하고 submit

 **Apache Flink Dashboard**

[Overview](#)
[Running Jobs](#)
[Completed Jobs](#)
[Task Managers](#)
[Job Manager](#)
[Submit new Job](#)

Submit new Job

Uploaded Jars

	Name	Upload Time	Entry Class	
<input checked="" type="checkbox"/>	flink-training-exercises.jar	2019-02-07, 0:21:32		×

com.dataartisans.flinktraining.examples.datastream_java.connectors.WordCountFromKafka

Parallelism

Show Plan

Program Arguments

Submit

Savepoint Path

☐ Allow Non Restored State

WordCount 시연 내용 (2/2)

- 세어지고 있는 단어 수 현황을 해당 태스크를 처리하고 있는 노드의 Task Manager에서 stdout 탭 으로 확인할 수 있음
- 또한 현재 실행되고 있는 Job을 Flink UI의 Running Jobs 세션에서도 확인 가능

The screenshot displays the Apache Flink Dashboard interface. On the left is a dark sidebar with navigation links: Overview, Running Jobs, Completed Jobs, Task Managers, Job Manager, and Submit new Job. The main panel is titled 'Task Manager' and shows the 'Stdout' tab selected. The output lists words and their counts, such as '(평범한,1)', '(https://t.co/mfpb7k5qxf,1)', '(아아,1)', etc. The top right corner indicates the 'Last Heartbeat: 2019-02-07, 0:26:15'.

Running Jobs

Start Time	End Time	Duration	Job Name	Job ID	Tasks	Status
2019-02-06, 23:51:22	2019-02-07, 0:26:53	35m 30s	Twitter Streaming Example	ba92d52e969bf7334d4170bff686c002	1 0 0 0 1 0 0 0 0 0	RUNNING
2019-02-07, 0:24:15	2019-02-07, 0:26:53	2m 37s	Streaming WordCount	f28e5caffd7e562bbe4ff5d8ea067e2f	2 0 0 0 2 0 0 0 0 0	RUNNING

WordCount App, Main 함수 코드

// 스트림 어플리케이션 실행 환경

```
final StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
```

...

// Kafka consumer 설정

```
Properties kafkaProps = new Properties();
```

...

```
kafkaProps.setProperty("bootstrap.servers", LOCAL_KAFKA_BROKER);
```

// Kafka connector API 와 미리 설정한 Properties 변수를 이용해 Kafka consumer 변수 생성

```
FlinkKafkaConsumer<String> consumer = new FlinkKafkaConsumer<>(  
    "twitterText",  
    new SimpleStringSchema(),  
    kafkaProps);
```

// 데이터 스트림 변수에 스트림 Source(Kafka connector API) 추가

```
DataStream<String> text = env.addSource(consumer);
```

```
DataStream<Tuple2<String, Integer>> counts =
```

```
    // 문장을 2-Tuple 형식으로 나눔: (단어, 1)
```

```
    text.flatMap(new Tokenizer())
```

```
    // '0' 번째 index 필드(단어)로 tuple을 그룹화 하고, '1' 번째 index 필드(개수)는 합침
```

```
    .keyBy(0).window(SlidingProcessingTimeWindows.of(Time.seconds(15), Time.seconds(5)))
```

```
    .sum(1);
```

// 최종 결과를 출력 (standard output)

```
counts.print();
```

// 프로그램을 실행시킴

```
env.execute("Streaming WordCount");
```

Source

Transformation

Sink

Apache Flink – Taxi Ride



- Taxi Ride, 뉴욕 택시 & 리무진 조합 (<https://www1.nyc.gov/site/tlc/index.page>)에서 제공하는 2009년부터 2015년 까지 뉴욕에서의 택시 이용 데이터셋을 이용하여 스트림 분석을 수행 하는 어플리케이션
- Flink API를 적용하면 과거에 생성된 데이터들을 이용하여 실시간으로 데이터가 생성되게끔 가상의 스트림 환경(Kafka 클러스터)을 만들 수 있음
- 가상의 스트림 데이터를 이용해 최근 15분 동안 택시가 출발하거나 도착한 장소의 빈도수가 많은 곳을 5분 간격으로 필터링하여 뉴욕에서 실시간으로 인기있는 장소를 분석함 (Sliding Window 개념)

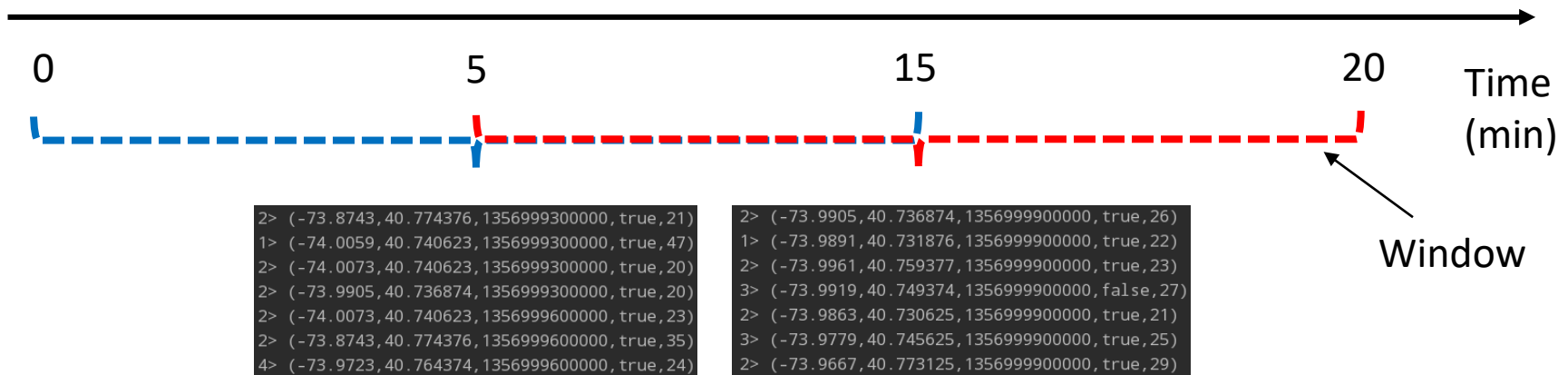
Taxi Ride 에서의 Sliding Window

- 최근 15분 동안 택시가 출발하거나 도착한 장소의 빈도수가 많은 곳을 5분 간격으로 필터링 뉴욕에서 실시간으로 인기있는 장소를 분석
- 15 분은 Sliding Window의 크기 그리고 5분은 Window의 sliding step

```
122,START,2013-01-01 00:01:00,1970-01-01 00:00:00,-73.924957,40.706707,
126,START,2013-01-01 00:01:00,1970-01-01 00:00:00,-73.928864,40.704819,
166,START,2013-01-01 00:01:00,1970-01-01 00:00:00,-73.942841,40.797031,
117,START,2013-01-01 00:01:00,1970-01-01 00:00:00,-73.95208,40.79169,
139,START,2013-01-01 00:01:00,1970-01-01 00:00:00,-73.952904,40.823254,
148,START,2013-01-01 00:01:00,1970-01-01 00:00:00,-73.957573,40.722225,
141,START,2013-01-01 00:01:00,1970-01-01 00:00:00,-73.95768,40.714569,
144,START,2013-01-01 00:01:00,1970-01-01 00:00:00,-73.959694,40.776882,
125,START,2013-01-01 00:01:00,1970-01-01 00:00:00,-73.964813,40.764027,
159,START,2013-01-01 00:01:00,1970-01-01 00:00:00,-73.968193,40.762428,
170,START,2013-01-01 00:01:00,1970-01-01 00:00:00,-73.971138,40.75898,
142,START,2013-01-01 00:01:00,1970-01-01 00:00:00,-73.973145,40.752827,
169,START,2013-01-01 00:01:00,1970-01-01 00:00:00,-73.973473,40.792583,
```

```
111096,START,2013-01-01 03:49:00,1970-01-01 00:00:00,-73.982262,40.740223,
111146,START,2013-01-01 03:49:00,1970-01-01 00:00:00,-73.98233,40.771408,-
111131,START,2013-01-01 03:49:00,1970-01-01 00:00:00,-73.982346,40.773136,
111248,START,2013-01-01 03:49:00,1970-01-01 00:00:00,-73.98246,40.765053,-
111175,START,2013-01-01 03:49:00,1970-01-01 00:00:00,-73.982521,40.764538,
111140,START,2013-01-01 03:49:00,1970-01-01 00:00:00,-73.982643,40.766621,
111136,START,2013-01-01 03:49:00,1970-01-01 00:00:00,-73.98288,40.722752,-
111217,START,2013-01-01 03:49:00,1970-01-01 00:00:00,-73.983475,40.738888,
111253,START,2013-01-01 03:49:00,1970-01-01 00:00:00,-73.983528,40.750118,
111262,START,2013-01-01 03:49:00,1970-01-01 00:00:00,-73.983658,40.686565,
111219,START,2013-01-01 03:49:00,1970-01-01 00:00:00,-73.983719,40.738098,
```

```
106369,END,2013-01-01 03:49:07,2013-01-01 03:36:44,-73.984116,40.737148,
111306,START,2013-01-01 03:49:08,1970-01-01 00:00:00,-73.982132,40.77784,
111305,START,2013-01-01 03:49:08,1970-01-01 00:00:00,-73.982971,40.69968,
111307,START,2013-01-01 03:49:08,1970-01-01 00:00:00,-74.00827,40.705391,
102611,END,2013-01-01 03:49:08,2013-01-01 03:27:46,-73.982239,40.777954,
103902,END,2013-01-01 03:49:08,2013-01-01 03:30:56,-73.976273,40.748432,
109100,END,2013-01-01 03:49:08,2013-01-01 03:43:48,-73.952141,40.775433,
111312,START,2013-01-01 03:49:09,1970-01-01 00:00:00,-73.960289,40.71979,
111313,START,2013-01-01 03:49:09,1970-01-01 00:00:00,-73.979393,40.66593,
111311,START,2013-01-01 03:49:09,1970-01-01 00:00:00,-73.982101,40.76882,
111310,START,2013-01-01 03:49:09,1970-01-01 00:00:00,-73.98484,40.732204,
111308,START,2013-01-01 03:49:09,1970-01-01 00:00:00,-73.993805,40.74660,
111309,START,2013-01-01 03:49:09,1970-01-01 00:00:00,-73.997307,40.71420,
```



Taxi Ride 시연 내용 (1/4)

- 택시 이용 레코드가 발행될 토픽(cleansedRides)을 EMR 마스터 노드에서 생성

```
[hadoop@ip-10-0-0-248 kafka_2.11-2.1.0]$ ./bin/kafka-topics.sh --list --zookeeper "10.0.2.159:2181,10.0.1.218:2181,10.0.0.44:2181"
AWSKafkaTutorialTopic
__consumer_offsets
cleansedRides
twitterText
```

- 가상 스트림 데이터를 위한 택시 이용 데이터를 HDFS(Hadoop Distributed File System)에 업로드

```
[hadoop@ip-10-0-0-125 kafka_2.11-2.1.0]$ hdfs dfs -ls /
Found 5 items
drwxr-xr-x   - hdfs   hadoop           0 2019-02-07 04:20 /apps
-rw-r--r--   1 hadoop hadoop      84135506 2019-02-07 04:23 /nycTaxiRides.gz
drwxrwxrwt   - hdfs   hadoop           0 2019-02-07 04:20 /tmp
drwxr-xr-x   - hdfs   hadoop           0 2019-02-07 04:19 /user
drwxr-xr-x   - hdfs   hadoop           0 2019-02-07 04:19 /var
```

Taxi Ride 시연 내용 (2/4)

- EMR 마스터 노드에서 Flink 명령어로 어플리케이션 실행
- 총 두가지 어플리케이션을 실행
 1. 택시 이용 데이터 셋을 가상 스트림 데이터로 Kafka 클러스터의 생성된 토픽에 발행하는 어플리케이션 (RideCleansingToKafka)
(업로드한 *.gz input 파일 명시 필요)

com.dataartisans.flinktraining.examples.datastream_java.connectors.RideCleansingToKafka

Parallelism

Show Plan

--input "hdfs:///nycTaxiRides.gz"

Submit

Savepoint Path

☐ Allow Non Restored State

2. 해당 토픽에 접근하여 택시 이용 레코드를 소비하여 인기 장소를 분석하는 어플리케이션 (PopularPlacesFromKafka)

com.dataartisans.flinktraining.examples.datastream_java.connectors.PopularPlacesFromKafka

Parallelism

Show Plan

Program Arguments

Submit

Savepoint Path

☐ Allow Non Restored State

Taxi Ride 시연 내용 (3/4)

- Kafka 클러스터의 cleansedRides 토픽을 확인함으로써 첫번째 어플리케이션의 동작 확인
- 택시 이용 이벤트의 스키마(Schema)는 다음과 같음
 - 이용 Id, 택시 출발 또는 도착, 시작 시간, 도착 시간(출발일 경우 1970-..), 시작 위치의 경도/위도, 목적지 위치의 경도/위도, 승객 수, 택시 Id, 운전자 Id

```
[hadoop@ip-10-0-0-125 kafka_2.11-2.1.0]$ bin/kafka-console-consumer.sh --bootstrap-server 10.0.1.25:9092,10.0.2.19:9092,10.0.0.138:9092 --topic cleansedRides --from-beginning
```

```
7377,START,2013-01-01 00:20:00,1970-01-01 00:00:00,-73.98866,40.722633,-73.99004,40.729404,1,2013006135,2013006131
7112,START,2013-01-01 00:19:34,1970-01-01 00:00:00,-73.99086,40.724506,-73.95045,40.78016,1,2013005988,2013005984
7382,START,2013-01-01 00:20:00,1970-01-01 00:00:00,-73.96592,40.758488,-73.92247,40.70634,3,2013006138,2013006134
7566,START,2013-01-01 00:20:22,1970-01-01 00:00:00,-73.98985,40.72957,-73.992516,40.742844,1,2013006236,2013006232
2764,END,2013-01-01 00:10:16,2013-01-01 00:20:15,-73.96024,40.715557,-73.93652,40.697453,2,2013002688,2013002685
5474,END,2013-01-01 00:16:03,2013-01-01 00:20:03,-73.999054,40.728065,-73.997696,40.72175,1,2013004933,2013004930
7330,START,2013-01-01 00:20:00,1970-01-01 00:00:00,-73.98836,40.734596,-73.91704,40.769817,1,2013003700,2013003696
4542,END,2013-01-01 00:14:08,2013-01-01 00:20:08,-73.99749,40.72578,-74.00683,40.71985,1,2013004224,2013004221
7124,START,2013-01-01 00:19:36,1970-01-01 00:00:00,-73.95722,40.774555,-73.95821,40.769627,2,2013005997,2013005993
4447,END,2013-01-01 00:14:00,2013-01-01 00:20:00,-74.00887,40.71089,-74.001335,40.729256,5,2013000636,2013000633
5636,END,2013-01-01 00:16:43,2013-01-01 00:19:47,-73.992676,40.724342,-73.989426,40.718742,2,2013005046,2013005043
3856,END,2013-01-01 00:13:00,2013-01-01 00:20:00,-73.957085,40.766354,-73.9687,40.75467,6,2013002456,2013002453
7185,START,2013-01-01 00:19:52,1970-01-01 00:00:00,-73.97753,40.725983,-73.989334,40.74764,2,2013003123,2013003120
4182,END,2013-01-01 00:13:32,2013-01-01 00:19:29,-73.99107,40.73968,-73.975815,40.73316,1,2013001366,2013001363
7183,START,2013-01-01 00:19:52,1970-01-01 00:00:00,-73.980675,40.733932,-73.991714,40.749027,3,2013006033,2013006029
7512,START,2013-01-01 00:20:08,1970-01-01 00:00:00,-73.97719,40.745293,-73.95612,40.771873,1,2013000228,2013000228
7424,START,2013-01-01 00:20:00,1970-01-01 00:00:00,-73.952,40.7161,-73.992325,40.696938,1,2013006163,2013006159
```

Taxi Ride 시연 내용 (4/4)

- 최근 15분 동안 택시의 목적지 또는 탑승지에서 가장 인기 많은 장소를 5분 간격으로 측정한 결과는, 태스크를 처리하고 있는 노드의 Task Manager에서 stdout탭 으로 확인할 수 있음
- Output 포맷
 - (인기장소 경도/위도, 시간, 출발지 또는 목적지/ 그 장소가 몇 번 선택되었는지)
- 또한 Flink UI의 Overview 섹션에서 두개의 job이 Flink에 의해 동작하고 있는 것을 확인 가능

Task Manager

Last Heartbeat: 2019-02-07, 14:27:57

Metrics

Logs

Stdout

Task Manager Output

```
(-74.0059,40.740623,1356999300000,true,25)
(-74.0073,40.740623,1356999300000,true,20)
(-73.9807,40.754375,1356999300000,true,22)
(-73.9905,40.736874,1356999300000,true,20)
(-74.0059,40.740623,1356999300000,true,47)
(-73.9821,40.768124,1356999300000,true,27)
(-73.8743,40.774376,1356999300000,true,21)
(-74.0045,40.741875,1356999300000,true,22)
(-73.9779,40.745625,1356999300000,true,23)
(-73.9835,40.744373,1356999300000,true,24)
(-74.0059,40.740623,1356999300000,false,27)
(-73.9821,40.771873,1356999300000,true,35)
(-73.9779,40.745625,1356999600000,true,25)
(-73.9891,40.731876,1356999600000,true,23)
(-74.0073,40.740623,1356999600000,true,23)
```

Running Jobs

Start Time	End Time	Duration	Job Name	Job ID	Tasks	Status
2019-02-07, 14:13:10	2019-02-07, 14:25:24	12m 14s	Taxi Ride Cleansing	251072bdd3e3b3f0a6e2b9052250a426	<div><div>1</div><div>0</div><div>0</div><div>0</div><div>1</div><div>0</div><div>0</div><div>0</div><div>0</div><div>0</div></div>	<div>RUNNING</div>
2019-02-07, 14:18:51	2019-02-07, 14:25:24	6m 33s	Popular Places from Kafka	99d8d20f660ea58a941385811056133f	<div><div>2</div><div>0</div><div>0</div><div>0</div><div>2</div><div>0</div><div>0</div><div>0</div><div>0</div><div>0</div></div>	<div>RUNNING</div>

Taxi Ride App - RideCleansingToKafka, Main 함수 코드 (스트림 가상 생성기)

```
ParameterTool params = ParameterTool.fromArgs(args);  
// 사용자 실행 명령에서 사용할 택시 이용데이터의 HDFS의 주소를 입력받음  
String input = params.getRequired("input");  
  
final int maxEventDelay = 60; // 이벤트 간격마다 60초 이내로 랜덤하게 딜레이 생성  
final int servingSpeedFactor = 300; // 10분 간격 이벤트가 현실에서는 2초 간격으로 스트림 되도록 설정  
  
// 스트림 어플리케이션 실행 환경  
StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();  
env.setStreamTimeCharacteristic(TimeCharacteristic.EventTime);  
  
// 스트림 데이터 생성기 시작  
DataStream<TaxiRide> rides = env.addSource(new TaxiRideSource(input, maxEventDelay, servingSpeedFactor));  
  
DataStream<TaxiRide> filteredRides = rides  
    // 택시 이용 데이터 중 뉴욕 내에서 진행되지 않은 정보는 필터링하여 스트림 하지않음  
    .filter(new NYCFilter());  
  
// 필터링 된 스트림 데이터를 Kafka connector API를 이용하여 Kafka 클러스터에 produce 함  
filteredRides.addSink(new FlinkKafkaProducer<TaxiRide>(  
    LOCAL_KAFKA_BROKER,  
    CLEANSSED_RIDES_TOPIC,  
    new TaxiRideSchema()));  
  
// 데이터 (cleansing) 파이프 라인을 실행  
env.execute("Taxi Ride Cleansing");
```

Source

Transformation

Sink

[Github에서 전체 코드 확인](#)

Taxi Ride App - PopularPlacesFromKafka, Main 함수 코드 (인기 장소 분석)

// 스트림 어플리케이션 실행 환경

```
StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();  
env.setStreamTimeCharacteristic(TimeCharacteristic.EventTime);
```

...

// Kafka connector API를 이용해 Kafka consumer를 생성

```
FlinkKafkaConsumer<TaxiRide> consumer = new FlinkKafkaConsumer<>(  
    "cleansedRides", new TaxiRideSchema(), kafkaProps);
```

...

```
DataStream<TaxiRide> rides = env.addSource(consumer); // Kafka consumer를 통해 데이터 스트림 획득
```

// 인기 장소를 찾기

```
DataStream<Tuple5<Float, Float, Long, Boolean, Integer>> popularPlaces = rides  
    // 데이터 스트림 중 장소 id 와 이벤트 type (출발 또는 도착)만을 이용해 tuple 형성  
    .map(new GridCellMatcher())  
    // 장소 id와 이벤트 type으로 그룹화 함  
    .keyBy(0, 1)  
    // sliding window 생성  
    .timeWindow(Time.minutes(15), Time.minutes(5))  
    // window 안의 택시 이용 횟수를 셈  
    .apply(new RideCounter())  
    // 방문 횟수가 미리 설정한 임계점 보다 높은 값을 가진 장소가 인기 장소  
    .filter((Tuple4<Integer, Long, Boolean, Integer> count) -> (count.f3 >= popThreshold))  
    // 장소 id를 실제 경도 위도 값으로 전환  
    .map(new GridToCoordinates());
```

Source

Transformation

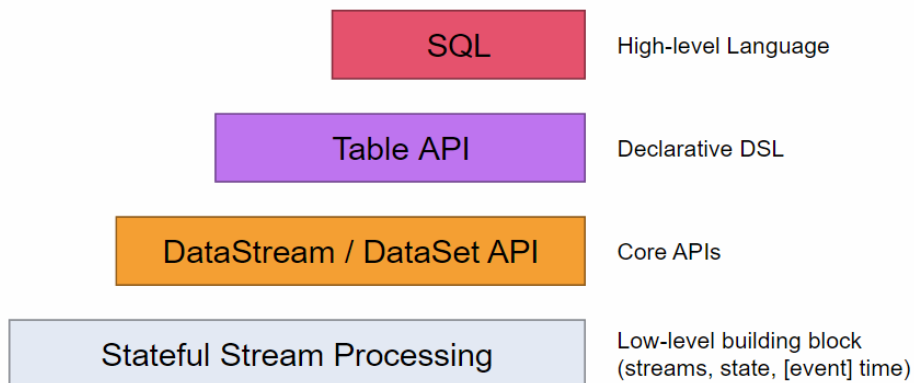
Sink

```
popularPlaces.print();
```

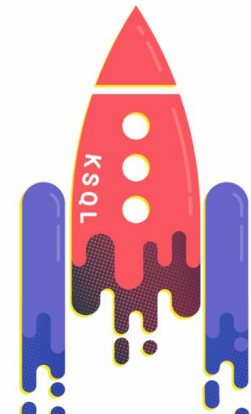
```
env.execute("Popular Places from Kafka"); // 프로그램을 실행시킴
```


Apache Flink – SQL

- Flink는 Table API 와 SQL 를 지원함
- 이를 이용하면 스트림 데이터를 테이블에 등록하고 selection, filter 그리고 join과 같은 관계형 연산자로부터 쿼리를 수행할 수 있음
- 종종 낮은 레벨의 DataStream API 에서 작성된 어플리케이션은, 더욱 직관적으로, 보다 높은 레벨인 Table API 및 SQL을 통해서 비슷한 동작을 가진 어플리케이션을 개발 가능함
- Confluent, Inc에서 제공하는 SQL 엔진인 KSQL 역시 Flink의 SQL과 비슷한 역할을 수행가능



 confluent



Taxi Ride App - PopularPlacesSql, Main 함수 코드 (인기 장소 분석)

```
.....  
// 택시 이용 데이터 스트림을 테이블 형식으로 등록  
tEnv.registerTableSource("TaxiRides", new TaxiRideTableSource(input, maxEventDelay, servingSpeedFactor));
```

```
// 테이블 형식에 적용할 사용자 함수를 등록함  
tEnv.registerFunction("isInNYC", new GeoUtils.IsInNYC());  
tEnv.registerFunction("toCellId", new GeoUtils.ToCellId());  
tEnv.registerFunction("toCoords", new GeoUtils.ToCoords());
```

```
Table results = tEnv.sqlQuery(  
    "SELECT " + "toCoords(cell), wstart, wend, isStart, popCnt " +  
    "FROM " + "(SELECT cell, isStart, " +  
    "HOP_START(eventTime, INTERVAL '5' MINUTE, INTERVAL '15' MINUTE) AS wstart, " +  
    "HOP_END(eventTime, INTERVAL '5' MINUTE, INTERVAL '15' MINUTE) AS wend, " +  
    "COUNT(isStart) AS popCnt " + "FROM " + "(SELECT " + "eventTime, " + "isStart, " +  
    "CASE WHEN isStart THEN toCellId(startLon, startLat)" +  
    "ELSE toCellId(endLon, endLat) END AS cell " +  
    "FROM TaxiRides " +  
    "WHERE isInNYC(startLon, startLat) AND isInNYC(endLon, endLat)) " +  
    "GROUP BY cell, isStart, HOP(eventTime, INTERVAL '5' MINUTE, INTERVAL '15' MINUTE)) " +  
    "WHERE popCnt > 20"  
);
```

```
// 쿼리의 결과인 테이블 형식을 다시 스트림 형식으로 바꾸고 그것을 출력  
tEnv.toAppendStream(results, Row.class).print();
```

```
// 설정한 쿼리를 실행  
env.execute();
```

Source
Transformation
Sink

하위 레벨 API(DataStream API)와 상위 레벨 API(SQL) 비교

- Taxi Ride 어플리케이션 개발시 Table API / SQL 을 이용하면
코드 내용이 더욱 직관적으로 바뀔 수 있음

```
DataStream<Tuple5<Float, Float, Long, Boolean, Integer>> popularPlaces = rides
    .map(new GridCellMatcher())
    .keyBy(0, 1)
    .timeWindow(Time.minutes(15), Time.minutes(5))
    .apply(new RideCounter())
    .filter((Tuple4<Integer, Long, Boolean, Integer> count) -> (count.f3 >= popThreshold))
    .map(new GridToCoordinates());
```



```
Table results = tEnv.sqlQuery(
    "SELECT " + "toCoords(cell), wstart, wend, isStart, popCnt " +
    "FROM " + "(SELECT cell, isStart, " +
    "HOP_START(eventTime, INTERVAL '5' MINUTE, INTERVAL '15' MINUTE) AS wstart, " +
    "HOP_END(eventTime, INTERVAL '5' MINUTE, INTERVAL '15' MINUTE) AS wend, " +
    "COUNT(isStart) AS popCnt " + "FROM " + "(SELECT " + "eventTime, " + "isStart, " +
    "CASE WHEN isStart THEN toCellId(startLon, startLat)" +
    "ELSE toCellId(endLon, endLat) END AS cell " +
    "FROM TaxiRides " +
    "WHERE isInNYC(startLon, startLat) AND isInNYC(endLon, endLat)) " +
    "GROUP BY cell, isStart, HOP(eventTime, INTERVAL '5' MINUTE, INTERVAL '15' MINUTE)) " +
    "WHERE popCnt > 20"
);
```

감사합니다
