



AWS Kinesis

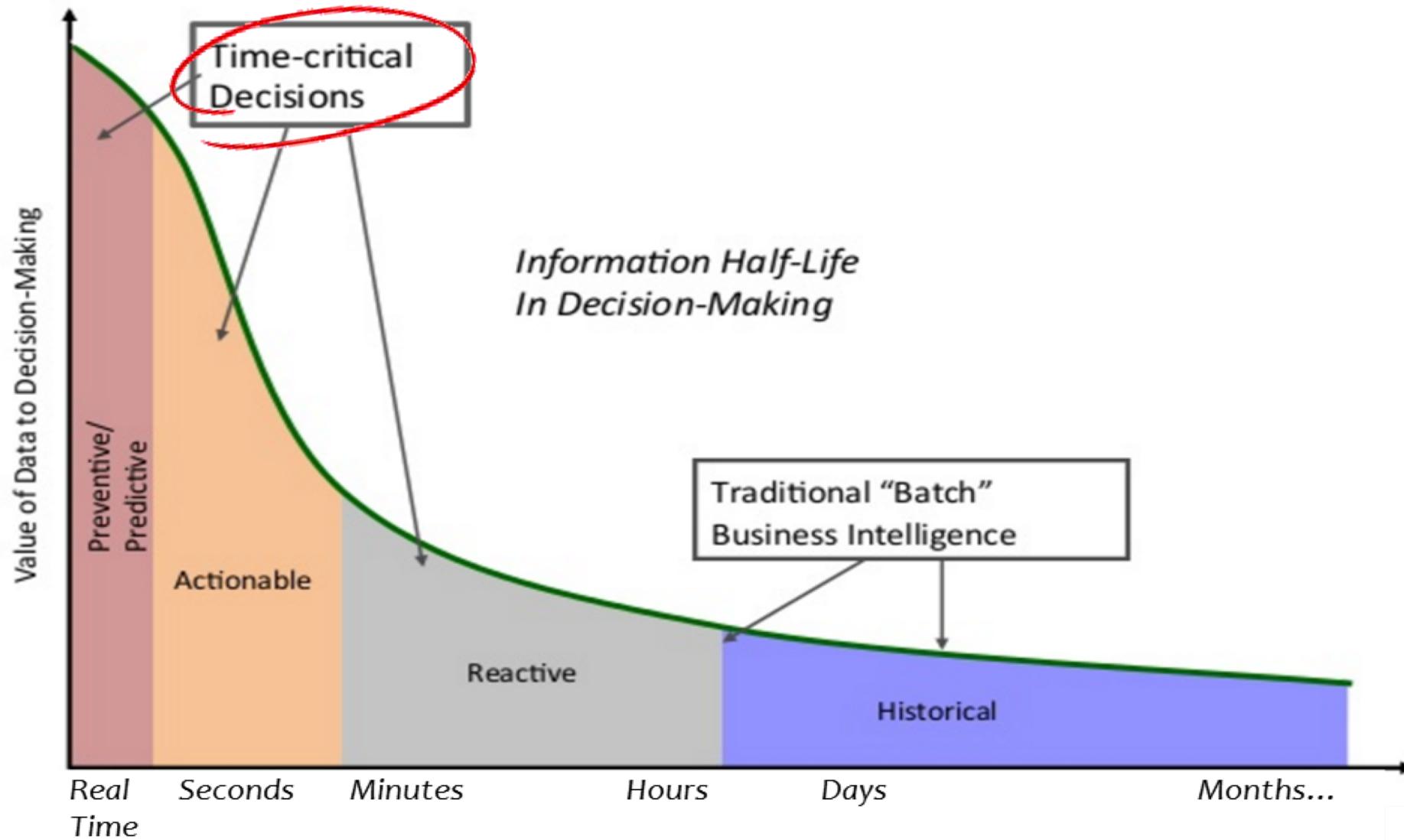
for streaming data ingestion, processing, and Analytics

목차

1. Amazon Kinesis Platform
2. Amazon Kinesis Streams
3. Amazon Kinesis Firehose
4. Amazon Kinesis Analytics

Amazon Kinesis Platform

데이터는 시간이 지날 수록 가치가 떨어집니다



시간에 따른 데이터 처리 방식



배치 처리



스트림 처리

- 유한 데이터를 처리하도록 설계
- 시간별, 일별, 주별, 월별 통계 레포트 등
ex) 일별 부정행위 리포트

- 연속 데이터를 처리하도록 설계
- 실시간 지표, 경고 등
ex) 실시간 부정 행위 탐지

스트리밍 데이터란 ?

- **실시간적인 움직임 :**
매우 낮은 latency (ms에서 수초)로 발생되고 처리
- **작고 높은 빈도로 발생됨 :**
초당 수백에서 수백만의 작은 데이터 레코드 (<5KB)
- **순차적임 :**
데이터는 시간별 혹은 활동별로 생성됨

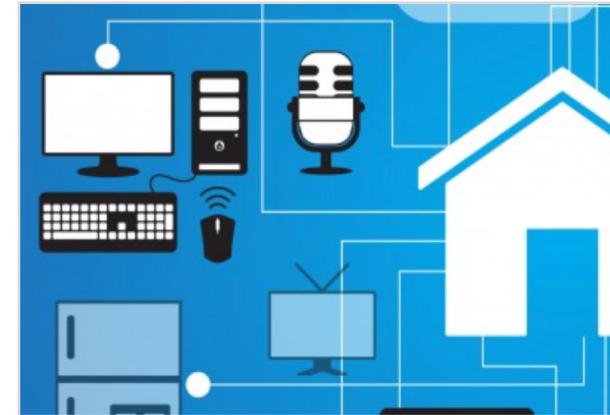
각 산업부문 스트리밍 데이터 시나리오

시나리오	수집-변환-로드	지표 추출	데이터분석
마케팅/광고	게시자, 입찰자의 데이터 수집	적용 범위, 산출량 및 전환과 같은 광고 통계	광고에 대한 사용자 클릭수 , 최적화 된 입찰 / 구매 엔진
IoT	센서, 장치 원격 측정 데이터 처리	운영 메트릭 및 대시 보드	장치 운영 정보 및 경고
게이밍	온라인 데이터 수집 (예 : 상위 10 위)	대규모 멀티 플레이어 온라인 게임 (MMOG) 라이브 대시 보드	리더 보드 생성, 플레이어 - 기술 match
금융	시장 / 금융 거래 주문 데이터의 수집	금융 시장 데이터 측정	부정사용 모니터링 및 가치평가 시장데이터 감사
Consumer Online	클릭 스트림 분석	노출 수 및 페이지 뷰와 같은 측정 항목	추천엔진 및 사전적 고객응대

스트리밍 솔루션들이 필요로 하는 기능 요건은?



① 수집과 처리 분리



② 복수 스트림 동시 수집 가능 및 버퍼링, 알림

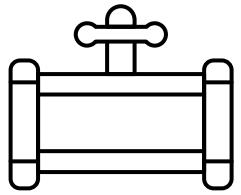


③ 데이터 순서 유지, 확장 가능한 구조



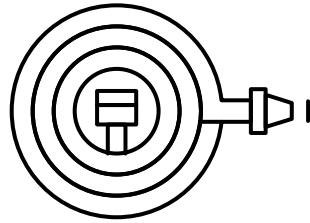
④ 병렬 소비

Amazon Kinesis



Kinesis Data Streams

스트리밍 데이터를 처리 및 분석하는 사용자 지정 애플리케이션을 개발



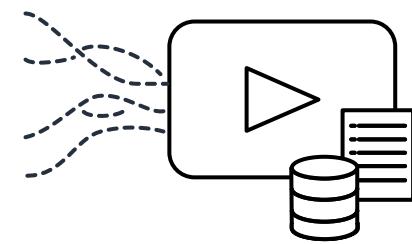
Kinesis Data Firehose

스트리밍 데이터를 S3, Redshift, Elasticsearch로 손쉽게 로드



Kinesis Data Analytics

표준 SQL을 통해 스트리밍 데이터를 간편하게 처리 및 분석

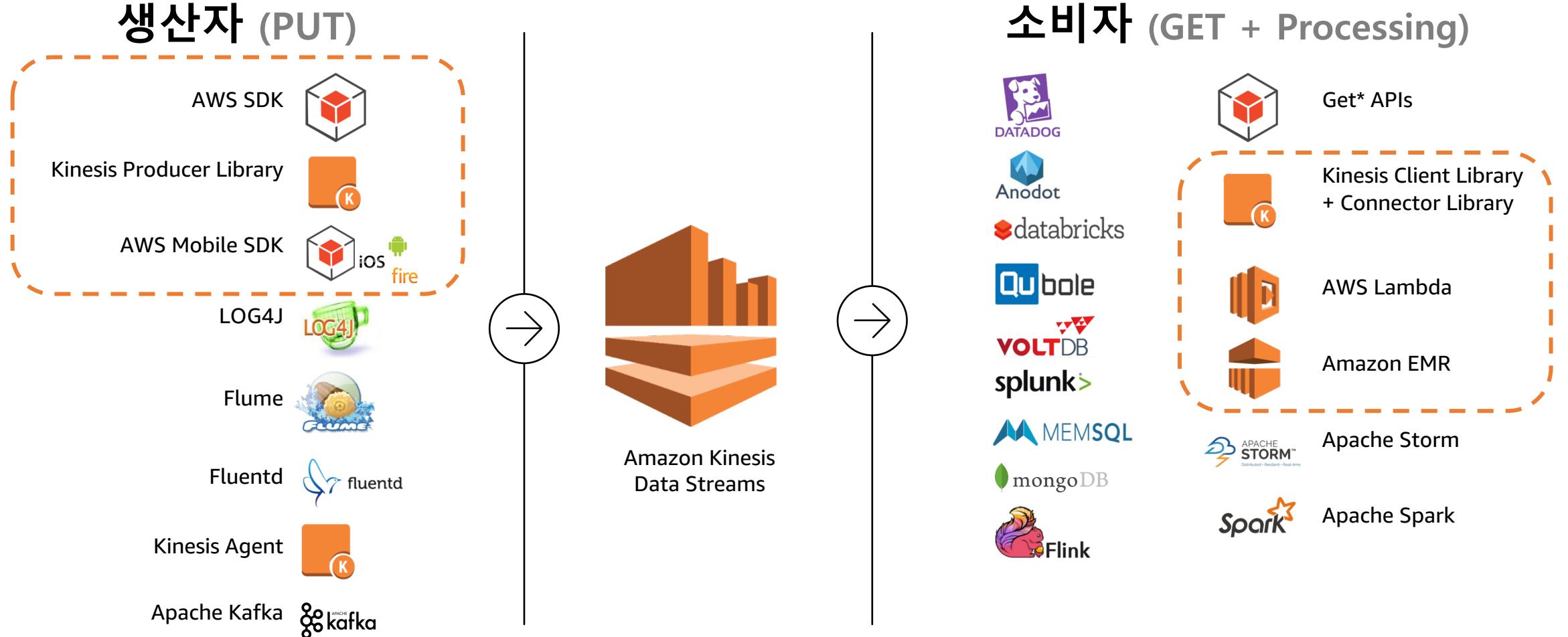


Kinesis Video Streams

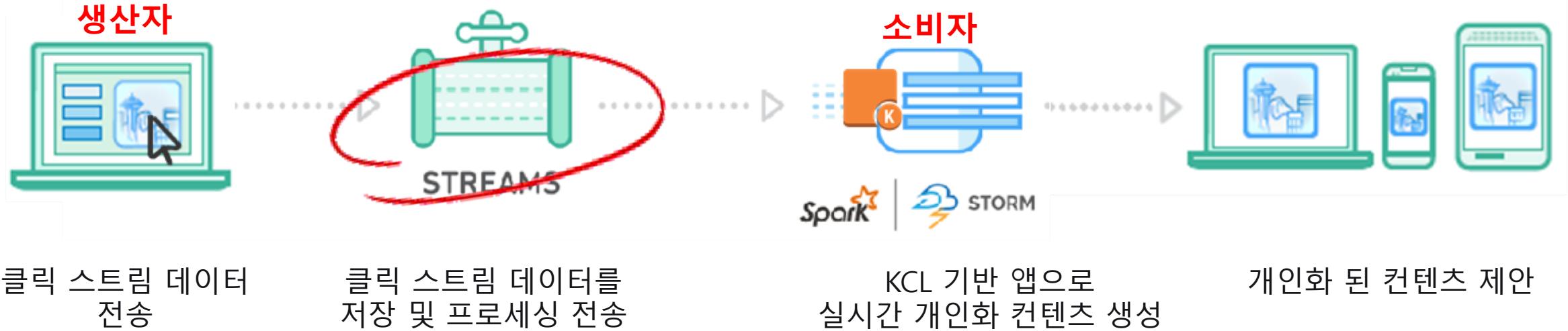
비디오 스트림을 수집, 처리, 저장

Amazon Kinesis Streams

Kinesis Streams에 데이터 넣고 사용하기

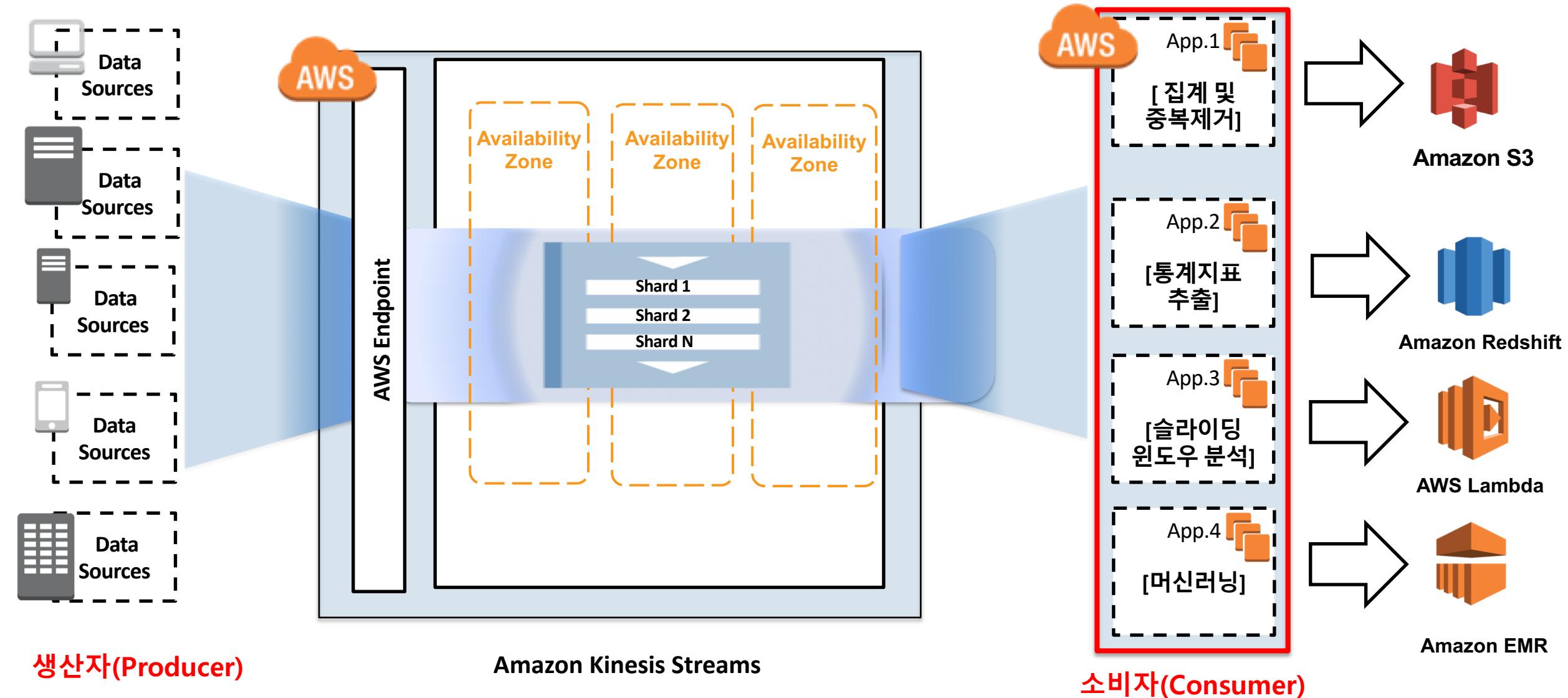


Kinesis Streams 개념

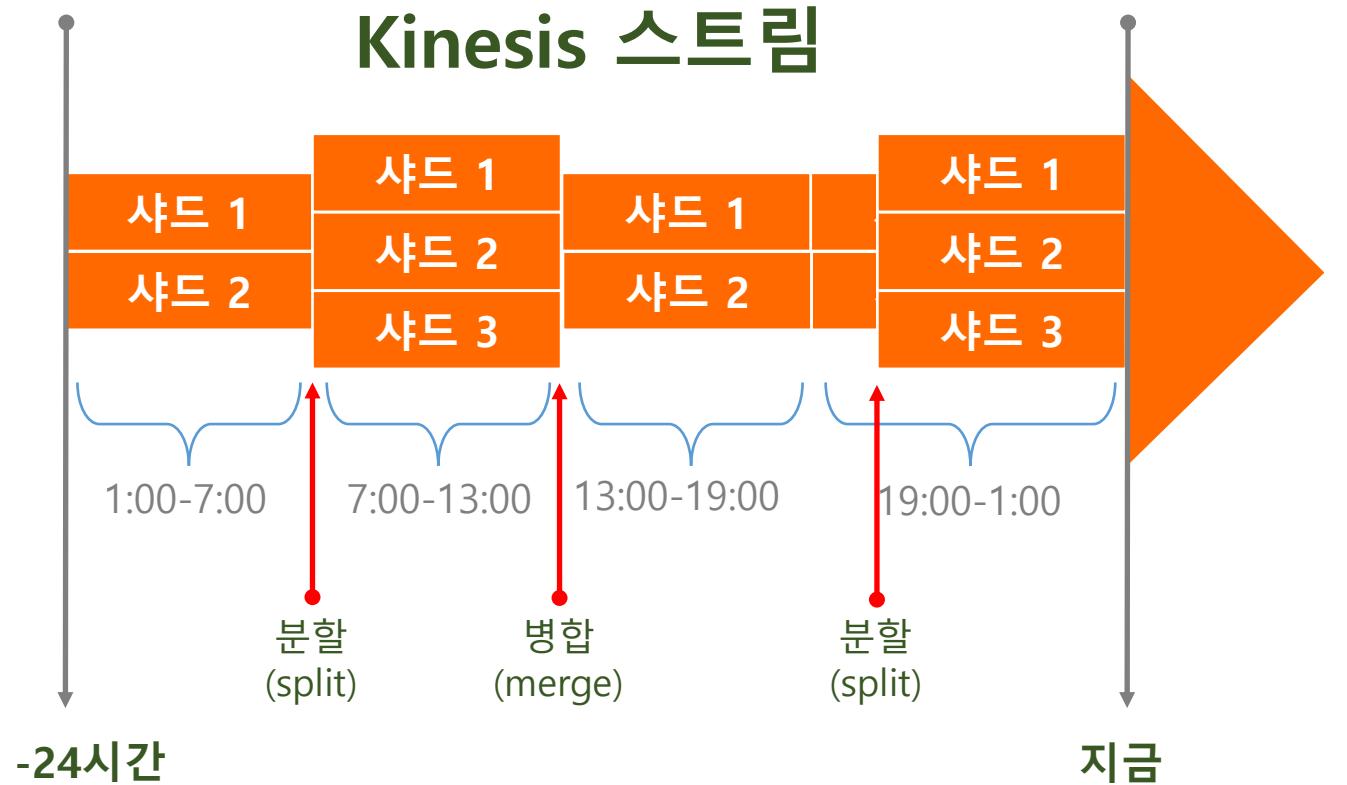


- 전송스트림 : 샤프 / 프로비저닝 / 파티션키 있음
- 생산자, 소비자를 구성하여 읽기, 쓰기 작업
- 추가 처리 또는 분석을 위해 다른 AWS 서비스와 연계 (EMR, Redshift, DynamoDB 등)

Kinesis Streams 데이터 흐름



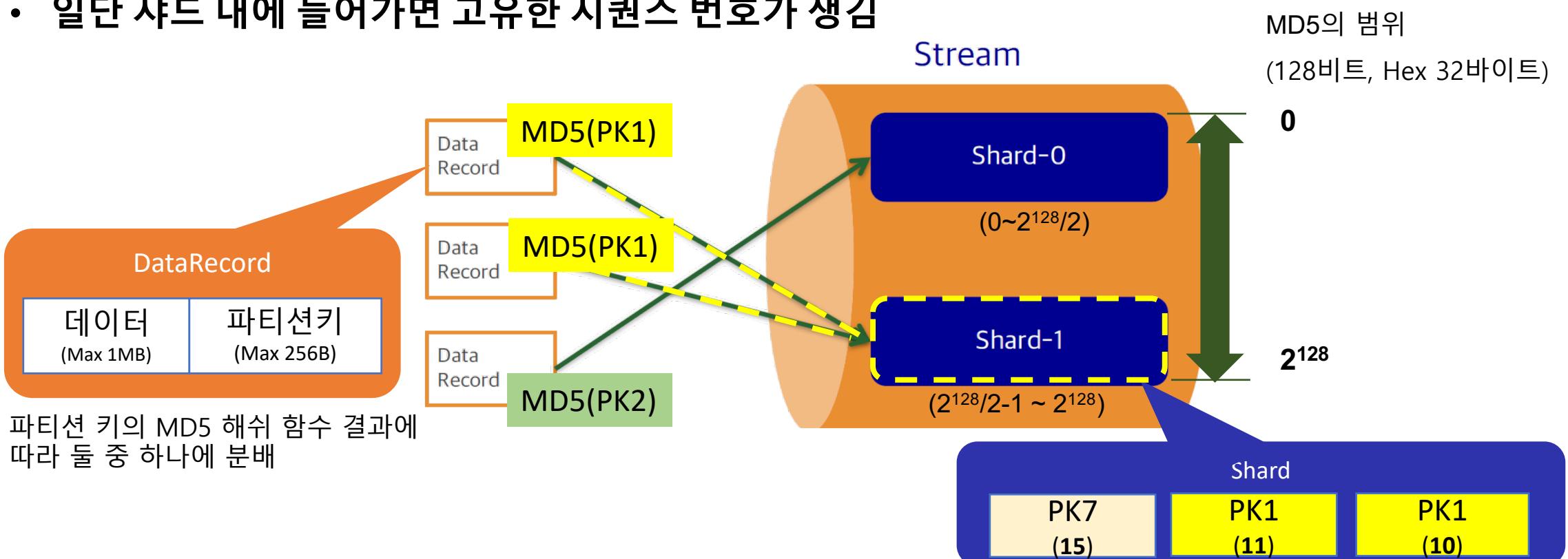
Kinesis Streams 샤드



- 샤드는 시간당 과금이 됨
(\$0.018/샤드/시간, \$0.02/백만개PUT)
- 최대 1MB/초, 최대 1,000 TPS 수신
- 최대 2MB/초, 최대 5TPS 송신
- 모든 데이터는 기본으로 24시간동안 저장됨
- 데이터 보존 기간 연장 가능 (최대 7일)
- 샤드들을 분할하거나 병합을 통해 확장
- 보존 기간 내에 있는 데이터 다시 재생 가능

Kinesis Streams 샤드 분배 로직

- 각 Shard는 128비트 정수의 범위 내에서 지정.
- 파티션 키와 MD5 값이 일치하는 범위의 Shard에 분배
- 파티션 키를 잘 설계하여 좋은 분산을 구현할 필요가 있음
- 일단 샤드 내에 들어가면 고유한 시퀀스 번호가 생김



[참고] AWS콘솔에서 Kinesis Stream 생성하기

시작하기 1단계 : Stream 이름과 Shard의 수량을 입력하여 Stream 생성

Create Kinesis stream

Kinesis stream name* → 스트림 이름

Shards

A shard is a unit of throughput capacity. Each shard ingests up to 1MB/sec and 1000 records/sec, and emits up to 2MB/sec. To accommodate for higher or lower throughput, the number of shards can be modified after the Kinesis stream is created using the API. [Learn more](#)

Producers → Kinesis stream → Consumers

Number of shards* → 샤드 수

You can provision up to 200 more shards before hitting your account limit of 200.
[Learn more](#) or [request a shard limit increase for this account](#)

Total stream capacity Values are calculated based on the number of shards entered above.

Write MB per second
1000 Records per second

Read MB per second

Kinesis Streams에 데이터 넣기 (1/2) – AWS SDK

- 간단한 Put 인터페이스를 사용하여 데이터 저장

- ✓ 파티션 키는 PUT 레코드들을 분산되는데 사용됨
- ✓ 성공적인 Put에 대한 고유 시퀀스 번호가 생산자에 반환됨
- ✓ AWS SDK for Java, JavaScript, Python, Ruby, PHP, .Net

```
스트림 이름          데이터          파티션키
↑                  ↑                  ↑
Python 2.7.5 (default, Mar  9 2014, 22:15:05)
[GCC 4.2.1 Compatible Apple LLVM 5.0 (clang-500.0.68)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> from boto.kinesis.layer1 import KinesisConnection
>>> conn = KinesisConnection()
>>> conn.put_record('TestStream', 'put data.', 'partition_key', b64_encode=True)
{u'ShardId': u'shardId-000000000001', u'SequenceNumber': u'495391436812708109417'
>>> 
```

데이터를 Kinesis Stream으로 보내는 Python 예제

Kinesis Streams에 데이터 넣기 (2/2) – 기타

SDK 외 다른 도구를 사용한 데이터 입력

- **KPL(Kinesis Producer Library)**

- ✓ 생산자 어플리케이션과 Kinesis Streams API 작업간 중간 역할
- ✓ 재시도 매커니즘, 요청 당 여러 Shard에 여러 Record 쓰기
- ✓ 생산자 성능 시각화를 위한 Amazon Cloudwatch 통합

- **Kinesis Agent**

- ✓ 파일 합성을 지속 모니터링 후 새로운 데이터 스트림으로 전송
- ✓ Linux 기반 서버환경에서 동작하는 독립형 Java 어플리케이션

- **Log4J Appender**

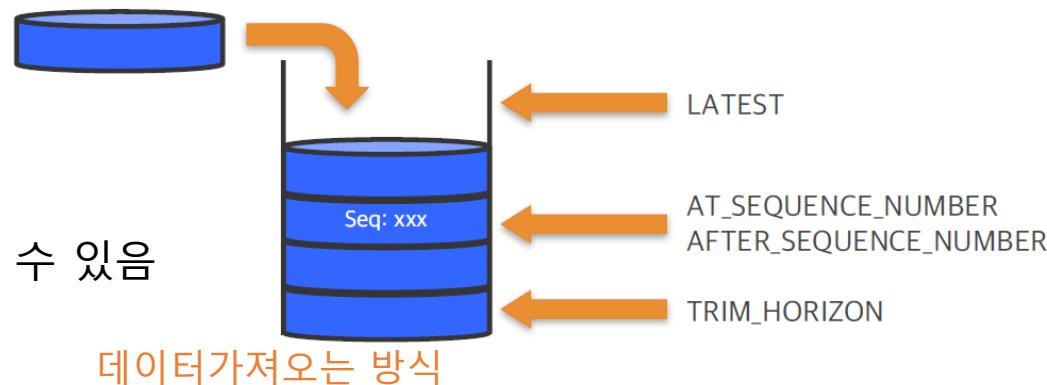
- ✓ Log4J의 출력을 Kinesis로 입력할 수 있는 Appender
- ✓ 버퍼링 크기, 동시 스레드 숫자 설정 가능

Kinesis Streams에서 데이터 가져오기 (1/2) - AWS SDK

GetShardIterator의 동작

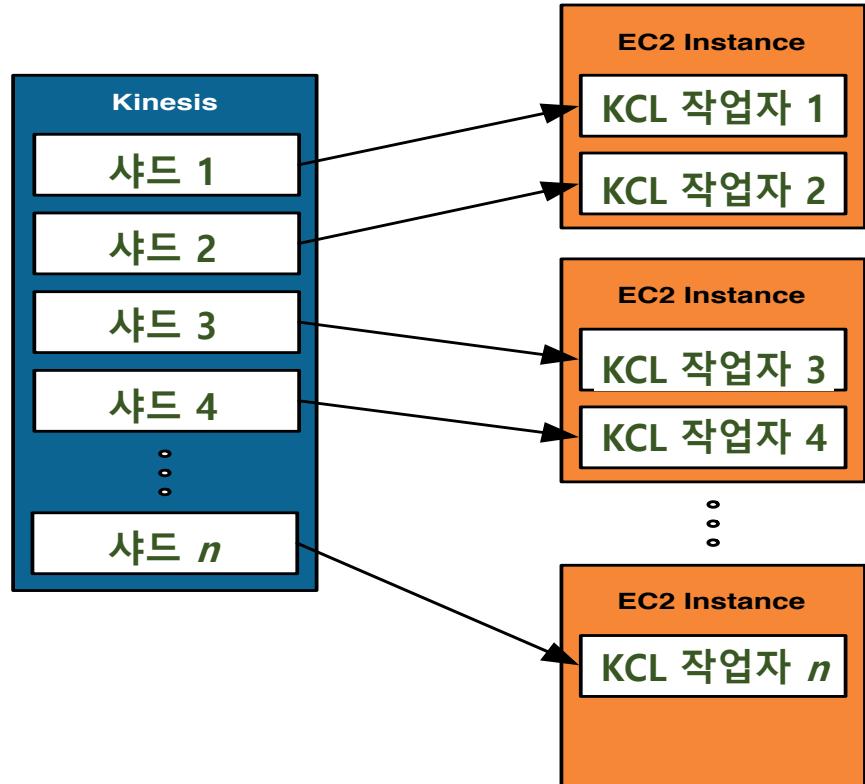
- 간단한 Get 인터페이스를 사용하여 데이터 이용

- ✓ GetShardIterator API에서 Shard의 위치를 검색
 - ✓ GetRecords API를 사용하여 Kinesis에 저장된 데이터를 가져올 수 있음



```
>>> from boto.kinesis.layer1 import KinesisConnection
>>> conn = KinesisConnection()
>>> shardIterator = conn.get_shard_iterator('TestStream', 'shardId-000000000001', 'LATEST')
>>> print shardIterator
{u'ShardIterator': u'AAAAAAAAAGJhY0yi0wdb/hu0DLh/KIppszESCNNNQcxVSVNF0xtnSwTI19WXEq6yFb2d/GcRZUZ0ctcoC9ZgKZ5dWPH99
PzQSRqQjVB4jv5+JcNz2fceEetb+X/Mc9VBAMngggPpeAFEtuP0IEdlLV2iwmM774K0WP2uczk='}
>>> iterator = shardIterator['ShardIterator']
>>> conn.get_records(iterator, 10000, True)
{u'Records': [], u'NextShardIterator': u'AAAAAAAAAFmKnGpLl7h7v93xknMKKuqQWb0aTZrfZuX5wzwY9CmhiCHeRH1CAyrXh90dbVnTX
KW/IzrUDHEWx//mwfUMM16aDNkM2Zu0wDjgkJHbGrCLSL97j7S6mimPvVsg2zEheis4Sxjrb4KB13R4eq5HJphcqIqxMx8='}
>>> []
```

Kinesis Streams에서 데이터 가져오기 (2/2) - 기타



SDK 외 다른 도구를 사용한 데이터 가져오기

- **Kinesis Client Library (KCL)**

- ✓ 샤드의 수가 변경될 때마다 KCL에서 자동으로 처리
- ✓ 각 샤드마다 자동적으로 Kinesis 작업자(worker) 시작
- ✓ KCL 위치를 추적하기 위해 체크포인트 사용 (Dynamo DB 사용)
- ✓ 작업자들이 실패하는 경우 재시작

- **Connector Library**

- ✓ S3, DynamoDB, Redshift, ElasticSearch 등과 손쉽게 연동

* 참고

https://docs.aws.amazon.com/ko_krstreams/latest/dev/developing-consumers-with-kcl.html

<https://github.com/awslabs/amazon-kinesis-connectors>

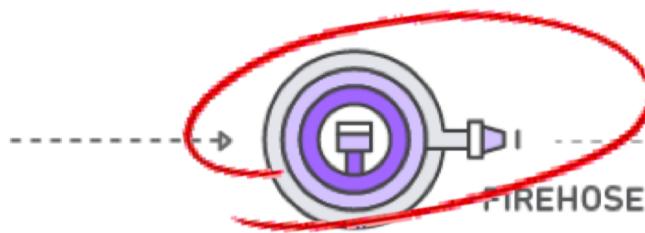
Amazon Kinesis Firehose

Kinesis Firehose 개념

생산자



스트리밍 데이터
캡쳐 및 전송

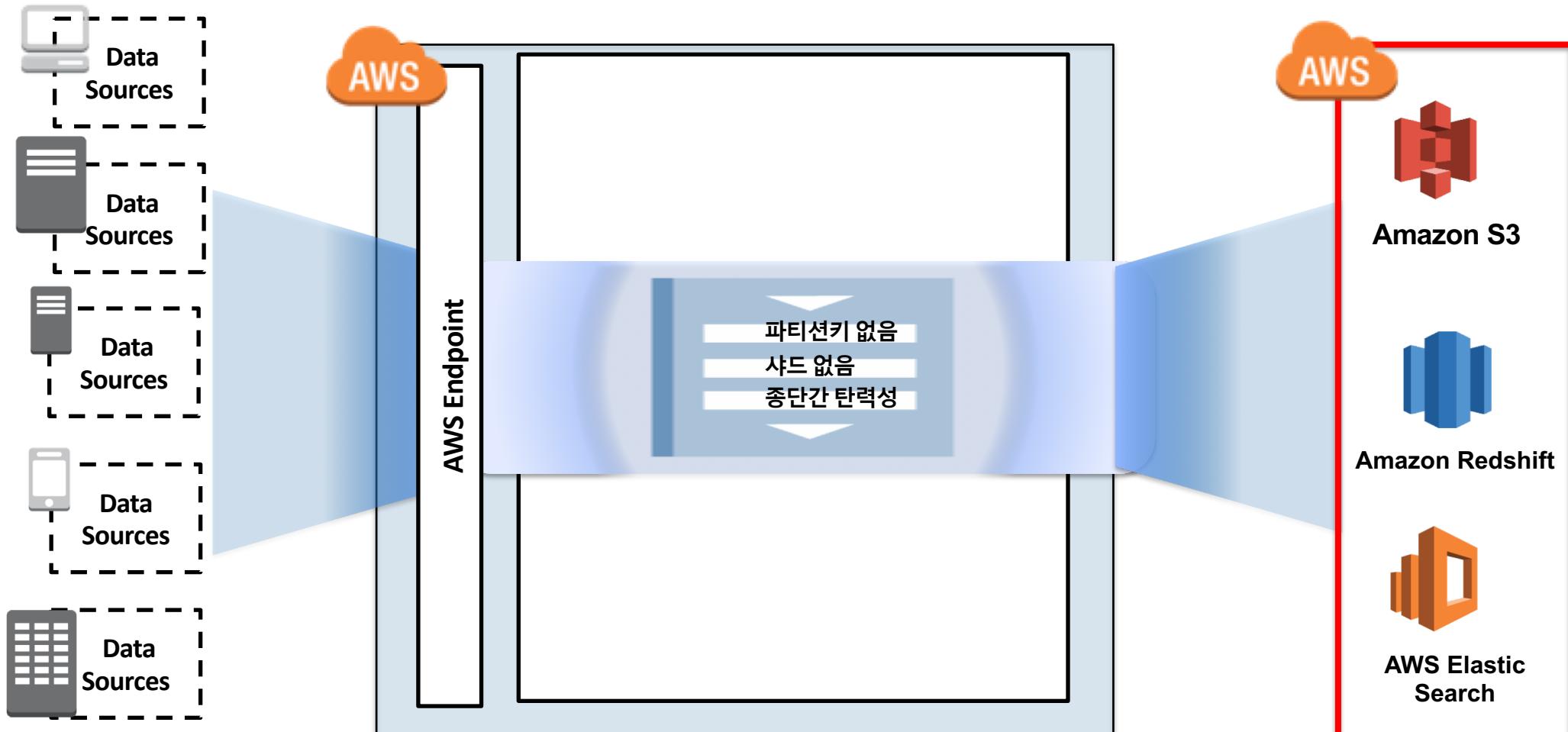


FireHose 는 Amazon S3, Redshift 및 Elasticsearch에
지속적으로 스트리밍 데이터 로드

선호하는 BI 도구를 사용하여
스트리밍 데이터 분석

- **Firehose 전송스트림** : 색드 / 프로비저닝/ 파티션키 없음
- **레코드** : 각 데이터 Blob(레코드)는 절차에 따라 전송스트림으로 전송, (최대크기 1MB)
- **버퍼링** : Firehose가 유입스트림 데이터를 대상으로 전달하기전 버퍼링할 크기 및 기간 (MB, 초)

Kinesis Firehose 데이터 흐름



생산자(Producer)

Amazon Kinesis Streams

[참고] AWS콘솔에서 Kinesis Firehose 생성하기

시작하기 3단계 : Stream 이름과 Destination 지정

Kinesis Firehose - Create delivery stream

Step 1: Name and source
Step 2: Transform records
Step 3: Choose destination
Step 4: Configure settings
Step 5: Review

Select destination

Destination

- Amazon S3 i
- Amazon Redshift i
- Amazon Elasticsearch Service i
- Splunk i

Firehose to S3 data flow overview

S3 destination

S3 bucket* tw-dp-rt View bucket tw-dp-rt in S3 console

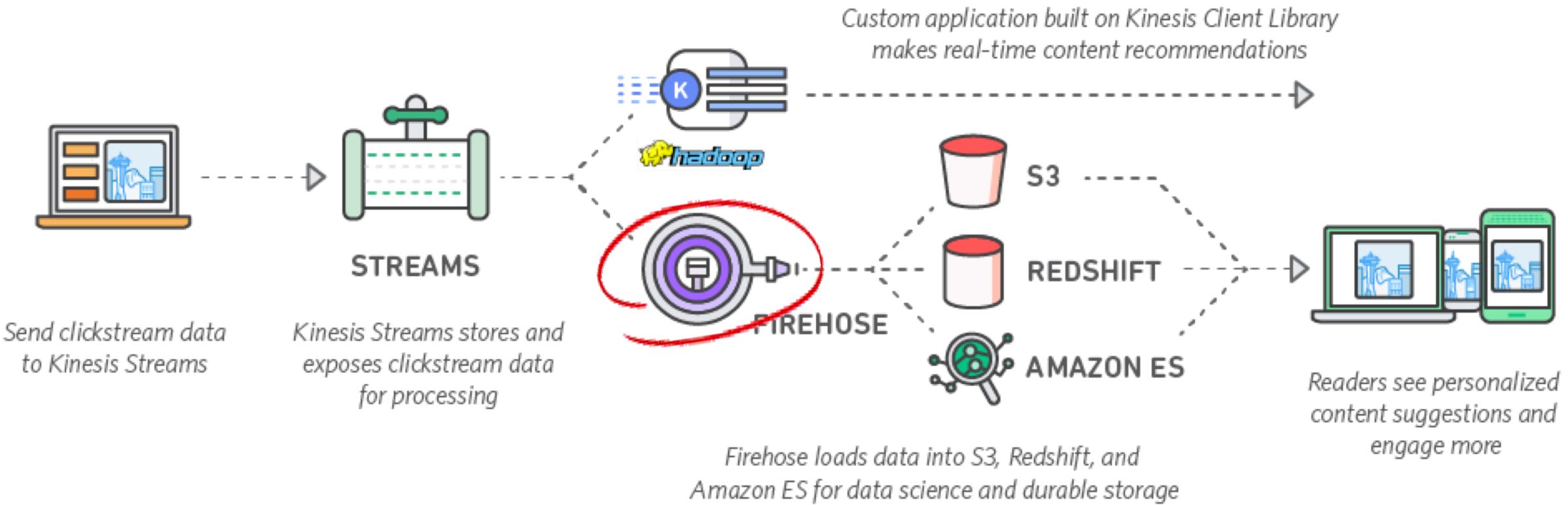
Prefix stream/ i

* Required

Destination 종류

Destination 상세 정보

Kinesis Firehose (사용 예)

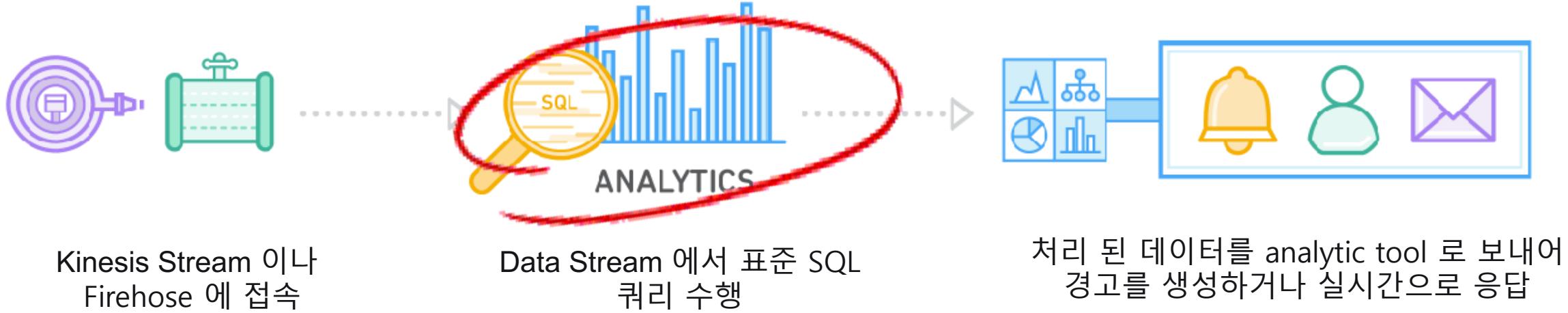


어떤 스트리밍 솔루션을 써야 하는가?

	Amazon SQS 	Kinesis Stream 	Kinesis Firehose 	Apache Kafka 
AWS 관리형	✓	✓	✓	✗
순서 보장	✗	✓	✗	✓
데이터 유지 기간	14일	7일	해당사항 없음	구성 가능
가용성	3AZ	3AZ	3AZ	구성 가능
규모/처리량	무제한/자동	무제한/~샤드	무제한/자동	무제한/~노드
병렬 소비	✗	✓	✗	✓
스트림 MapReduce	해당사항 없음	✓	해당사항 없음	✓
레코드 크기	256KB	1M	1M	구성 가능

Amazon Kinesis Analytics

스트리밍 데이터에 표준 SQL 쿼리 작성



- **Kinesis Stream 또는 Kinesis Firehose**에 쉽게 연결하고 표준 SQL로 검색
- 빅데이터 스트리밍에 대한 지속적인 처리를 초단위로 수행 가능
- 데이터 처리량에 따라 쉽게 확장

스트리밍 데이터에 표준 SQL 쿼리 작성



1단계. 입력 인애플리케이션 스트림 구성



2단계. 스트리밍 데이터를 프로세스 하기 위한 SQL 코드 작성



3단계. 출력 스트림으로 SQL 결과를 계속적으로 확인

[참고] AWS콘솔에서 Kinesis Analytics 사용하기

The screenshot shows the AWS Kinesis Analytics console. At the top, there is a code editor window containing the following SQL code:

```
1 CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" ("sensor_id" VARCHAR(32), "sensor" VARCHAR(15),
2   "station_id" VARCHAR(32), "sensor_avg_value" double, "sensor_smooth_avg_value" double,
3   "60sec_sum_of_sensor_value" double, "60sec_number_of_msg" int, "record_timestamp" TIMESTAMP);
4
5 CREATE OR REPLACE PUMP "STREAM_PUMP" AS INSERT INTO "DESTINATION_SQL_STREAM"
6 SELECT STREAM "sensor_id", "sensor", "station_id",
7   AVG("sensor_value"), AVG("sensor_value_smoothed"),
8   SUM("sensor_value") AS "60sec_sum_of_sensor_value",
9   count(*) AS "60sec_number_of_msg", ROWTIME AS "record_timestamp"
10 FROM "SOURCE_SQL_STREAM_001"
11 GROUP BY "sensor_id", "sensor", "station_id",
12   FLOOR(("SOURCE_SQL_STREAM_001".ROWTIME - TIMESTAMP '1970-01-01 00:00:00') SECOND / 60 TO SECOND);
```

An orange box highlights the code area, and an arrow points from it to the text "SQL 코드 작성" (SQL code creation).

Below the code editor is a main interface with tabs: Source data, Real-time analytics (which is selected), Destination, and Application. The Real-time analytics tab has sections for In-application streams, Pause results (with a refresh icon), and a scroll checkbox.

An arrow points from the scroll checkbox to the text "SQL 결과 확인" (SQL result verification).

The interface also shows a table of results for the stream "DESTINATION_SQL_STREAM". The table has columns: ROWTIME, sensor_id, sensor, station_id, sensor_avg_value, and sensor_sm. Two rows of data are visible:

ROWTIME	sensor_id	sensor	station_id	sensor_avg_value	sensor_sm
2016-11-11 01:44:00.0	813c174428fc4843	temp	qwbKAMlbZW	63.23000000000001	63.2300000
2016-11-11 01:44:00.0	caaed84a552deebf	rain	qwbKAMlbZW	0.0	0.0

Streaming SQL 작성

Streams (in memory tables)

```
CREATE STREAM calls_per_ip_stream(  
    eventTimeStamp TIMESTAMP,  
    computationType VARCHAR(256),  
    category VARCHAR(1024),  
    subCategory VARCHAR(1024),  
    unit VARCHAR(256),  
    unitvalue BIGINT  
) ;
```

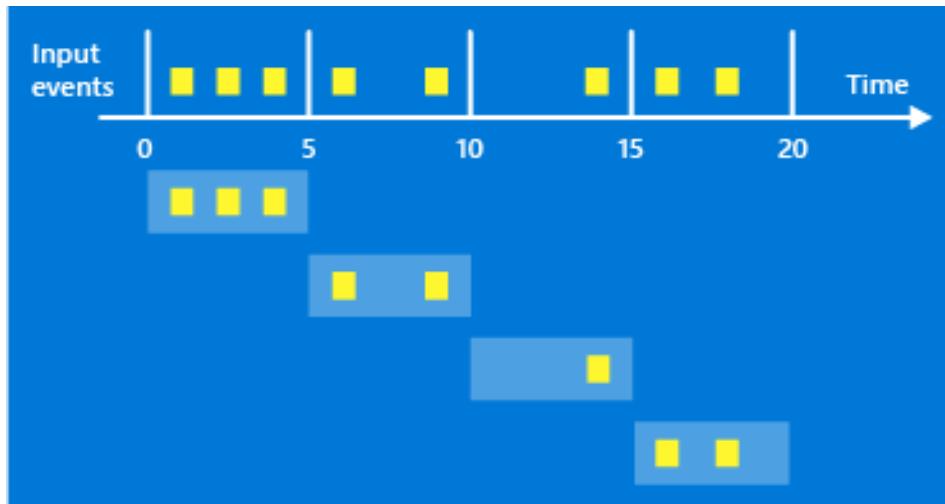
Streaming SQL 작성

Pumps (continuous query)

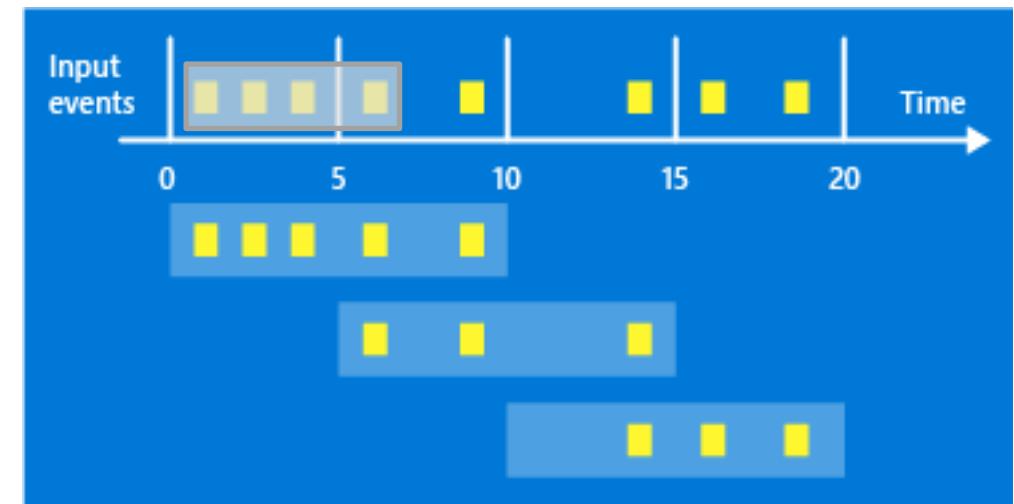
```
CREATE OR REPLACE PUMP calls_per_ip_pump AS
INSERT INTO calls_per_ip_stream
SELECT STREAM "eventTimestamp",
       COUNT(*),
       "sourceIPAddress"
FROM source_sql_stream_001 ctrail
GROUP BY "sourceIPAddress",
         STEP(ctraill.ROWTIME BY INTERVAL '1' MINUTE),
         STEP(ctraill."eventTimestamp" BY INTERVAL '1' MINUTE);
```

Window 모드 Query

- 일정 기간 내 Streaming data Aggregation (count, sum, min...)
- Windowed Query : Tumbling, Sliding, Stagger
- Window는 고정 길이 (예 : 1분, 1시간...)
- ROWTIME이라는 타임스탬프 열 사용



Tumbling window
Aggregate per time interval



Sliding window
Windows constantly re-evaluated

IoT 디바이스 실시간 모니터링 (사용 예)

