如何解决Soar无法支持的策略?

一、前言

书接上回，我们提到处理流式安全数据的时候使用了elkeid hub平台进行处理，它能够满足我们的日常需求，但是一些比较复杂的策略就比较难以实现，比如在1分钟内时间我们要告警哪个云密钥调获取了云存储桶对象（GetObject）大于等于2次，同时也列举桶（GetBucket）对象1次及以上。

如何解决这个问题呢？这个得提到Apache Flink。

二、遇到的困难

2.1、处理日志的时间并非事件发生的事件

我们遇到的第一个问题是，日志产生和到日志被处理是有时间差的，假如这个时间差稍微大一点，我们想正确的统计一分钟内的行为就不是很靠谱

2.2、实时处理的日志时间是乱序的

在处理实时处理的日志的时候，会因为各种各样的原因会导致日志时间是乱序的。

举个例子：2024.10.01 00:00:00的日志会比2024.10.01 00:00:10先到。

在处理阿里云的SLS的云日志就是这样，由于SLS的分区机制，会把日志随机存储在不同区域，当你拉取的时候也是依次从这些分区拉，所以处理日志时候发现日志并非严格排序。

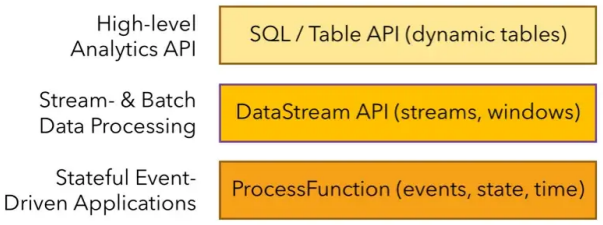
三、Flink

Apache Flink 是一款开源框架，擅长流处理（实时DataStream）与批处理（离线DataSet）。

2.1、基本概念

Flink 的 API 大体上可以划分为三个层次：
1、SQL/Table API
2、DataStream API
3、ProcessFunction

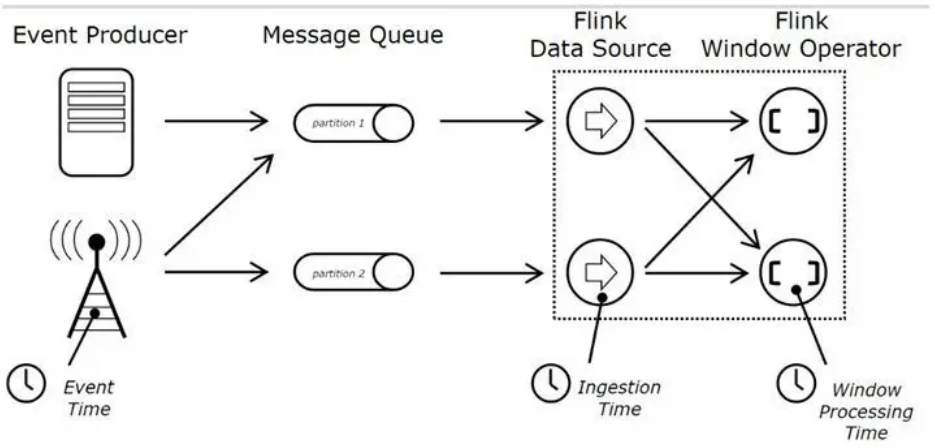
时间在Flink中的地位



这里我们会用最后面两层，SQL/Table API可以在Flink平台上做一些简单的数据处理。

2.2、解决事件差问题

Flink支持三种事件处理方式



Event Time：事件时间是每条事件在它产生的时候记录的时间
Ingestion Time：摄入时间是事件进入flink的时间
Process Time：处理时间是当前机器处理该条事件的时间流处理程序使用该时间进行处理的时候

我们可以使用事件时间进行处理。

2.3、解决时间乱序问题

Flink提供了三种方式去解决乱序的问题：

1、AscendingTimestampExtractor：当数据生成的时间戳是按升序排列时，可以使用AscendingTimestampExtractor来提取时间戳和生成水印。然而，在数据中如果存在时间戳升序排列但值较小的事件（例如序列1, 2, 3, 2, 4），那么较小的事件（如第二个2）可能会丢失。为了解决这个问题，可以通过sideOutputLateData来捕获和处理这些丢失的数据。

2、BoundedOutOfOrdernessTimestampExtractor：如果数据中存在乱序，可以使用BoundedOutOfOrdernessTimestampExtractor，并指定一个允许的延迟时间。这个实现允许系统处理一定程度的乱序数据，而不会丢失事件。
3、IngestionTimeExtractor是在时间特性被指定为IngestionTime时使用的，它直接基于数据的摄入时间来生成时间戳和水印，而不依赖于数据内的时间戳信息。

BoundedOutOfOrdernessTimestampExtractor进行延迟等待是会解决一部分的问题，但无法解决周期内时间事件排序以及流式期间的的时间排序。

2.4、Flink简单代码解决

使用DataStream类（流式），如果是离线数据的话使用DataSet类

```
DataStream<OssLogEvent> ossDataStream = dataStream
        .flatMap(new Tokenizer())
        .assignTimestampsAndWatermarks(new BoundedOutOfOrdernessTimestampExtractor<OssLogEvent>(Time.seconds(10)) {
            @Override
            public long extractTimestamp(OssLogEvent ossLogEvent) {
                return ossLogEvent.getTimestamp();
            }}).setParallelism(1);

```

注意：这里并不会对时间周期的事件进行排序，如下图

```
timestamp:1733562089000
timestamp:1733562089000
timestamp:1733562089000
timestamp:1733562089000
timestamp:1733562089000
timestamp:1733562088000
timestamp:1733562089000
timestamp:1733562083000
timestamp:1733562083000
timestamp:1733562084000
timestamp:1733562084000
timestamp:1733562084000
timestamp:1733562080000
timestamp:1733562081000
timestamp:1733562081000
timestamp:1733562081000
timestamp:1733562081000
```

ossDataStream处理进行处理的时候不要进行并发处理，会导致一个时间段会重复触发多次。也不要设置全局并发，并且设置时间处理机制为事件事件。

```
env.setStreamTimeCharacteristic(TimeCharacteristic.EventTime);
env.setParallelism(1);
```

最后就是窗口处理的选择上了：首先必须要在定义窗口前确定的是你的 stream 是 keyed 还是 non-keyed。 keyBy(...) 会将你的无界 stream 分割为逻辑上的 keyed stream。 如果 keyBy(...) 没有被调用，你的 stream 就不是 keyed。

[窗口 | Apache Flink](https://nightlies.apache.org/flink/flink-docs-master/zh/docs/dev/datastream/operators/windows/)

我们这里使用non-keyed窗口也就是timeWindowAll函数进行处理。

```
SingleOutputStreamOperator<OssLogEvent> sum = ossDataStream
        .timeWindowAll(Time.seconds(10))
        .process(new ProcessAllWindowFunction<OssLogEvent, OssLogEvent,
            @Override
            public void process(ProcessAllWindowFunction<OssLogEvent, Os

                Map<String,Integer> akGetBucketCountMap = new HashMap<>(
                Map<String,Integer> akGetObjectCountMap = new HashMap<>(
                Set<String> akSet = new HashSet<String>();

                System.out.println("10秒窗口处理"+ context.window().getSta
                for (OssLogEvent event : iterable) {

                    String ip = event.getClientIp();
                    String bucket = event.getBucket();
                    String ak = event.getAccessId();

                    System.out.println("timestamp:"+event.getTimestamp()
                    if (ak.equals("xxxxxxxxxxxxxx")){
                        System.out.println("############################
                        collector.collect(event);

                    }
                    collector.collect(event);

                    akSet.add(ak);
                    if ("GetBucket".equals(event.getOperation())) {
                        if(akGetBucketCountMap.containsKey(ak)){
                            akGetBucketCountMap.put(ak,akGetBucketCountM
                        }else {
                            akGetBucketCountMap.put(ak,1);
                        }
                    } else if ("GetObject".equals(event.getOperation()))
                        if(akGetObjectCountMap.containsKey(ak)){
                            akGetObjectCountMap.put(ak,akGetObjectCountM
                        }else {
                            akGetObjectCountMap.put(ak,1);
                        }
```

```
                    }
                }

                System.out.println(akSet);
                System.out.println(akGetObjectCountMap);
                System.out.println(akGetBucketCountMap);

                for (String ak : akSet) {
                    if (akGetBucketCountMap.containsKey(ak) && akGetBuck
                        System.out.println("告警: 在 " + context.window()
                    }
                }


            }
        });

```

四、总结

看起来挺简单的一个需求，想要处理好，涉及的知识点和东西还挺多的，好在最终也是能够解决文章开头说的需求。

Sls2Flink.zip
60.1KB 预览