archivist’s manifesto.

4 февраля 2021 г. 10:38

Есть много веских причин заархивировать наши события

- архивирование потоков событий, которые проходят через наш унифицированный журнал, делает нашу архитектуру обработки более надежной, позволяет нам выполнять повторную обработку при необходимости и позволяет нам улучшать результаты обработки

Kinesis не предназначены для хранения всего вашего архива событий

- В Kinesis горизонт обрезки установлен на 24 часа и может быть увеличен до 1 недели (или 168 часов). После этого отсечения старые события обрезаются - удаляются из потока навсегда.

теоретически вы можете хранить все данные о событиях в Kafka, но на практике большинство людей ограничивают их хранение одной неделей (что по умолчанию) или месяцем

-

как вариант: Если данные будут обрезаны через несколько часов или дней, давайте просто сделаем всю нашу обработку до истечения этого временного окна!

- Например, если нам нужно рассчитать определенные метрики в этом временном окне, давайте позаботимся о том, чтобы эти метрики были вычислены и безопасно записаны в постоянное хранилище в нужное время.

1 недостаток этого подхода: Resilience

- Мы хотим, чтобы наша унифицированная обработка журналов была максимально устойчивой к сбоям.
- Поскольку мы не можем предсказать и пофиксить все различные вещи, которые могут дать сбой в нашей работе, становится важным, чтобы у нас была надежная резервная копия наших входящих событий. Затем мы можем использовать эту резервную копию для восстановления после любого сбоя обработки потока на нашей собственной скорости.
- пример с рисунком: Если у нас Kinesis или Kafka настроены **на удаление событий навсегда через 24 час**а, это делает наш конвейер событий гораздо более уязвимым: мы должны исправить любые сбои обработки, прежде чем эти события навсегда исчезнут из нашего единого журнала.
- после устранения проблемы нам придется возобновить обработку событий с момента сбоя.

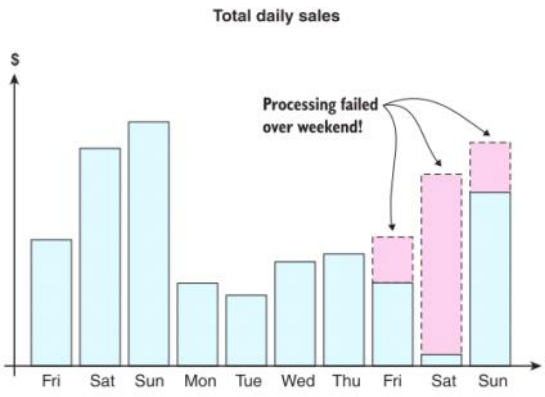

Figure 7.2 A dashboard of daily sales annotated to explain the data missing from the second weekend. From the missing data, we can surmise that the outage started toward the end of Friday, continued through the weekend, and was identified and fixed on Monday (allowing most of Sunday's data to be recovered).

так как сбой каскадный то нужно сохранять исходные события

- Помните также, что наши конвейеры обычно состоят из нескольких этапов обработки, поэтому на всех этапах после сбоя также будут отсутствовать важные входные данные. У нас будет каскадный сбой

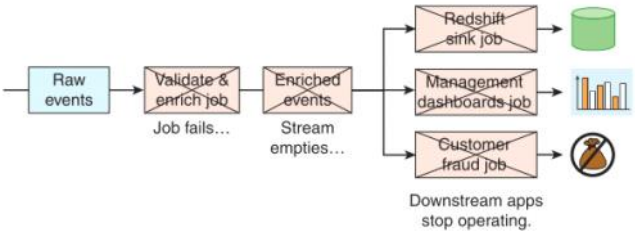

Figure 7.3 A failure in our upstream job that validates and enriches our event causes cascade failures in all of our downstream jobs, because the stream of enriched events that they depend on is no longer being populated.

2 недостаток этого подхода: Reprocessing

- есть несколько причин, по которым мы можем захотеть повторно обработать поток событий из полного архива этого потока
- Все эти варианты использования зависят от наличия доступа ко всей истории потока событий

например, мы хотим исправить ошибку в микросервисе и поэтому нам надо обработать все события заново

- Мы хотим исправить ошибку в существующем вычислении или агрегировании. Например, мы обнаруживаем, что наши ежедневные метрики рассчитываются для неправильного часового пояса.

например, мы хотим переделать алгоритм обработки в микросервисе и поэтому нам надо обработать все события заново

- Мы хотим изменить наше определение метрики. Например, мы решаем, что сеанс просмотра пользователем нашего веб-сайта заканчивается через 15 минут бездействия, а не через 30 минут.

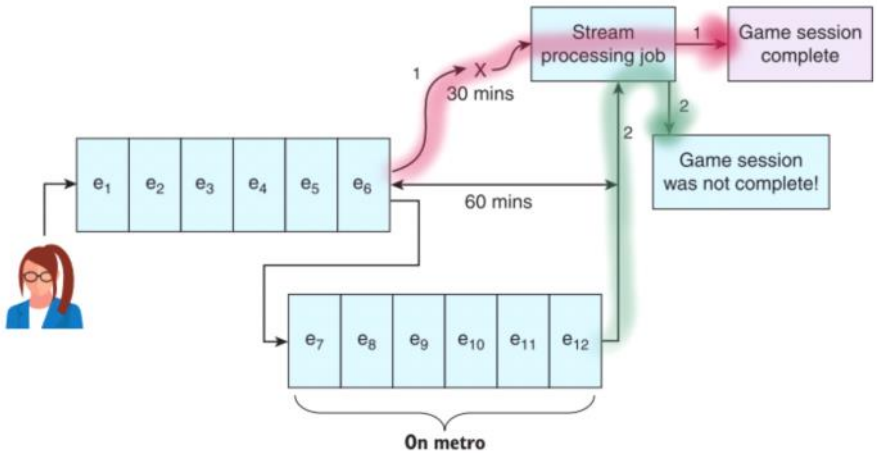

Figure 7.4 Our stream processing job draws the wrong conclusion—that a game session has finished— because relevant events arrive too late to be included in the decision-making process.

3 недостаток этого подхода: Refinement (запоздалая очистка данных)

Late-arriving data

- В реальном мире данные часто поступают с опозданием , и это влияет на точность наших расчетов потоковой обработки.
- Допустим, игрок едет в городском метро и на час теряет сигнал. Игрок продолжает играть в метро, и игра продолжает точно записывать события; затем игра отправляет им кэшированные события одним большим пакетом, когда пользователь снова поднимается над землей. К сожалению, пока пользователь находился в метро, наша задача по обработке потока уже решила, что игрок закончил игру, и соответствующим образом обновила свои показатели по завершенным игровым сессиям. Опоздавший пакет событий делает недействительными выводы, сделанные нашим приложением для обработки потоковой информации, как показано на рисунке

Approximations

- По соображениям производительности вы можете прибегать к приближениям в алгоритмах обработки поток
- Вычисление уникальных посетителей (COUNT DISTINCT в SQL) может быть сложной задачей, потому что метрика не является аддитивной - например, вы не можете подсчитать количество уникальных посетителей веб-сайта за месяц, сложив уникальные номера посетителей для составляющих недели. Поскольку точный подсчет уникальности требует больших вычислительных ресурсов, мы часто выбираем приблизительный алгоритм, такой как HyperLogLog, для обработки потока.
- Хотя эти приближения часто бывают удовлетворительными для потоковой обработки, аналитическая группа обычно хочет иметь возможность уточнить эти вычисления. В случае уникальных посетителей веб-сайта, аналитическая группа может захотеть иметь возможность генерировать истинное значение счетчика по всей истории событий веб-сайта.

Framework limitations

tools for archieveng

4 февраля 2021 г. 10:39

куда складывать данные

Distributed filesystem	Hosted?	API	Description
Amazon Simple Storage Service (S3)	Yes	HTTP	A hosted file storage service, part of Amazon Web Services.
Azure Blob Storage	Yes	HTTP	A hosted unstructured data storage service, part of Microsoft Azure.
Google Cloud Storage	Yes	HTTP	A hosted object storage service, part of Google Cloud Platform.
Hadoop Distributed File System (HDFS)	No	Java, Thrift	A distributed filesystem written in Java for the Hadoop framework.
OpenStack Swift	No	HTTP	A distributed, highly available, eventually consistent object store.
Riak Cloud Storage (CS)	No	HTTP	Built on the Riak database. API compatible with Amazon S3.
Tachyon	No	Java, Thrift	A memory-centric storage system optimized for Spark and Hadoop processing. Implements the HDFS interface.

далее мы можем анализировать этот архив с помощью pache Hadoop and Apache Spark

-

Ваш выбор хранилища будет зависеть от того, использует ли ваша компания предложение «инфраструктура как услуга» (IaaS), такое как AWS, Azure или Google Compute Engine, или создала собственную инфраструктуру обработки данных.

-

какими тулами архивировать кафку

Table 7.2 Tools for archiving our unified log from a distributed filesystem

Tool	From	To	Creator	Description
Camus	Kafka	HDFS	LinkedIn	A MapReduce job that loads Kafka into HDFS. Can autodiscover Kafka topics.
Flafka	Kafka	HDFS	Cloudera / Flume	Part of the Flume project. Involves configuring a Kafka source plus HDFS sink.
Bifrost	Kafka	S3	uSwitch.com	Writes events to S3 in uSwitch.com's own baldr binary file format.
Secor	Kafka	S3	Pinterest	A service for persisting Kakfa topics to S3 as Hadoop SequenceFiles.
kinesis-s3	Kinesis	S3	Snowplow	A Kinesis Client Library application to write a Kinesis stream to S3.
Connect S3	Kafka	S3	Confluent	Allows exporting data from Kafka topics to S3 objects in either Avro or JSON formats.

Secor, a tool from Pinterest: может зеркально отображать Темы Kafka в bucket в Amazon S3.

- we choose Secor over Bifrost because Secor's storage format—Hadoop SequenceFiles—is much more widely adopted than Bifrost's baldr format.
- Мы также могли бы выбрать Kafka Connect S3 от Confluent, но для этого нам потребуется установить дистрибутив Kafka от Confluent вместо Apache.

7.3.3 Setting up Secor

There are no prebuilt binaries for Secor, so we will have to build it from source ourselves. The Vagrant development environment has all the tools we need to get started:

```
$ cd /vagrant
$ wget https://github.com/pinterest/secor/archive/v0.26.tar.gz
$ cd secor-0.26
```

Next, we need to edit the configuration files that Secor will run against. First, load this file in your editor of choice:

```
/vagrant/secor/src/main/config/secor.common.properties
```

Now update the settings within the `MUST SET` section, as in the following listing.

Listing 7.1 secor.common.properties

```
...
# Regular expression matching names of consumed topics.
secor.kafka.topic_filter=raw-events-ch07

# AWS authentication credentials.
aws.access.key={{access-key}}
aws.secret.key={{secret-key}}
...
```

Restricts our archiving to only Nile's raw events from this chapter

Same as aws_access_key_id in ~/.aws/credentials

Same as aws_secret_access_key in ~/.aws/credentials

Next you need to edit this file:

```
/vagrant/secor/src/main/config/secor.dev.properties
```

We have only one setting to change here: the `secor.s3.bucket` property. This needs to match the bucket that we set up in section 7.3.2. When that's done, your secor.dev.properties file should look similar to that set out in the following listing.

Listing 7.2 secor.dev.properties

```
include=secor.common.properties

############
# MUST SET #
############

# Name of the s3 bucket where log files are stored.
secor.s3.bucket=ulp-ch07-archive-{{your-first-pets-name}}

#################
# END MUST SET #
#################

kafka.seed.broker.host=localhost
kafka.seed.broker.port=9092
```

Imports the file we edited previously

Set to your bucket's name

```
zookeeper.quorum=localhost:2181

# Upload policies.                    <-- Rules for uploading files to S3
# 10K
secor.max.file.size.bytes=10000
# 1 minute
secor.max.file.age.seconds=60
```

From this listing, we can see that the default rules for uploading our event files to S3 are to wait for either 1 minute or until our file contains 10,000 bytes, whichever comes sooner. These defaults are fine, so we will leave them as is. And that's it; we can leave the other configuration files untouched and move on to building Secor:

```
$ mvn package
...
[INFO] BUILD SUCCESS...
...
$ sudo mkdir /opt/secor
$ sudo tar -zxvf target/secor-0.26-SNAPSHOT-bin.tar.gz -C /opt/secor
...
lib/jackson-core-2.6.0.jar
lib/java-statsd-client-3.0.2.jar
```

Finally, we are ready to run Secor:

```
$ sudo mkdir -p /mnt/secor_data/logs
$ cd /opt/secor
$ sudo java -ea -Dsecor_group=secor_backup \
    -Dlog4j.configuration=log4j.prod.properties \
    -Dconfig=secor.dev.backup.properties -cp \
    secor-0.26.jar:lib/* com.pinterest.secor.main.ConsumerMain
Nov 05, 2018 11:26:32 PM com.twitter.logging.Logger log
INFO: Starting LatchedStatsListener
...
INFO: Cleaning up!
```

Note that some few seconds will elapse before the final `INFO: Cleaning up!` message appears; in this time, Secor is finalizing the batch of events, storing it in a Hadoop SequenceFile, and uploading it to Amazon S3.

Let's quickly check that the file has successfully uploaded to S3. From the AWS dashboard:

1. Click the S3 icon.
2. Click the bucket `ulp-ch07-archive-{{your-first-pets-name}}`.
3. Click each subfolder until you arrive at a single file.

This file contains our seven events, read from Kafka and successfully uploaded to S3 by Secor, as you can see in figure 7.8.

как архивировать

4 февраля 2021 г. 11:19

что именно мы должны архивировать: rawest events you can, as far upstream in your event pipeline as possible.

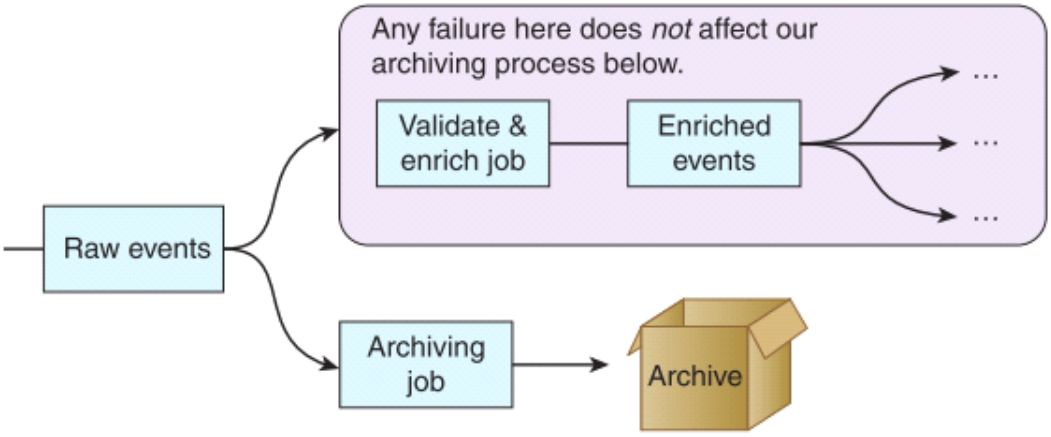

Figure 7.5 By archiving as far upstream as possible, we insulate our archiving process from any failures that occur downstream of the raw event stream.

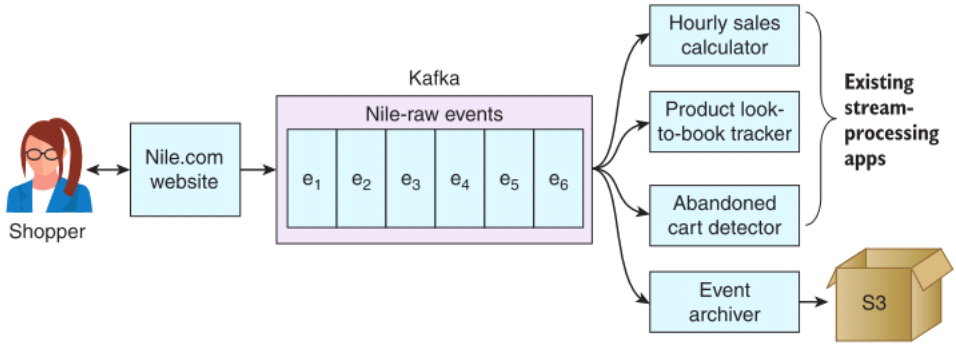

Figure 7.6 Alongside Nile's three existing stream-processing applications, we will be adding a fourth application, which archives the raw event stream to Amazon S3.

как разархивировать

4 февраля 2021 г. 13:15

как мы можем повторно обработать сбойную единицу работы (например, событие), если и когда основная проблема, вызвавшая сбой, устранена