به نام خدا

آشنایی با ابزار کافکا

تهیه‌کنندگان: پارسا عنایتی، مهدی اکبری

برنامه نویسی وب

یک فروشگاه آنلاین را در نظر بگیرید که نیاز دارد تراکنش‌های مشتریان را در زمان واقعی پردازش کند. مشکلاتی که باید حل شود شامل:

1. **ثبت بلادرنگ تراکنش‌ها** :نیاز به ثبت تمام تراکنش‌ها به محض وقوع.
2. **پردازش همزمان** :تحلیل بلادرنگ داده‌ها برای پیشنهادات شخصی‌سازی شده و موجودی انبار.
3. **مقیاس‌پذیری** :مدیریت حجم بالای تراکنش‌ها در زمان تخفیفات یا فروش ویژه.

درصورت استفاده از راه‌حل های قدیمی و سیستم‌های سنتی پیام‌رسانی یا پایگاه‌های داده به یکسری محودیت‌ها برمی‌خوریم:

- **محدودیت مقیاس‌پذیری** :بسیاری از سیستم‌های پیام‌رسانی سنتی مانند ActiveMQ یا RabbitMQ با محدودیت‌های مقیاس‌پذیری مواجه هستند و در صورت افزایش حجم داده‌ها، عملکرد آنها کاهش می‌یابد.

- **پایداری و قابلیت اطمینان** :این سیستم‌ها ممکن است در صورت خرابی سرور، داده‌ها را از دست بدهند یا با مشکلات بازیابی مواجه شوند.

- **تاخیر بالا** :ثبت بلادرنگ تراکنش‌ها در پایگاه داده‌های سنتی ممکن است باعث افزایش تاخیر شود و نتواند نیاز به پردازش بلادرنگ را برآورده کند.

- **محدودیت‌های همزمانی** :پایگاه‌های داده در مواجهه با حجم بالای تراکنش‌ها ممکن است با مشکلات همزمانی مواجه شوند که منجر به کاهش کارایی سیستم می‌شود.

- **محدودیت در حجم داده‌ها** :ابزارهای پردازش بلادرنگ مانند Storm یا Flink ممکن است در مواجهه با حجم بسیار بالای داده‌ها با مشکلات کارایی و مقیاس‌پذیری مواجه شوند.

- **پیچیدگی در مدیریت** :مدیریت و نگهداری این ابزارها ممکن است پیچیده و هزینه‌بر باشد.

- **محدودیت در توزیع بارکاری** :بسیاری از سیستم‌های توزیع‌شده سنتی نمی‌توانند به صورت کارآمد بارکاری را بین سرورها توزیع کنند و با افزایش حجم داده‌ها، کارایی آن‌ها کاهش می‌یابد.

- **مشکلات هماهنگی** :هماهنگی بین سرورها در سیستم‌های توزیع‌شده ممکن است پیچیده و پرهزینه باشد.

راه حل چیست؟

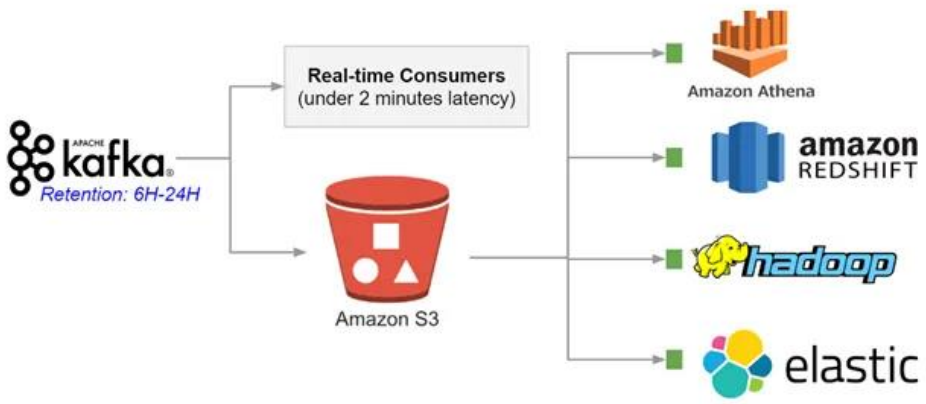
استفاده از ابزاری که قابلیت‌های زیر را داشته باشد:

- **مقیاس‌پذیری بالا** :توانایی مدیریت حجم عظیم داده‌ها و افزایش بار کاری.
- **پایداری و قابلیت اطمینان** :حفظ داده‌ها با استفاده از replication.
- **عملکرد بالا** :پردازش میلیون‌ها پیام در ثانیه با تاخیر کم.
- **انعطاف‌پذیری** :معماری publish-subscribe برای انتشار و اشتراک داده‌ها بین سیستم‌ها.
- **پردازش بلادرنگ** :تحلیل و پردازش داده‌ها به صورت لحظه‌ای.
- **جامعه کاربری فعال** :منابع آموزشی و پشتیبانی گسترده.
- **سازگاری با تکنولوژی‌های موجود** :ادغام آسان باHadoop ، Spark، Flinkو سایر ابزارها.

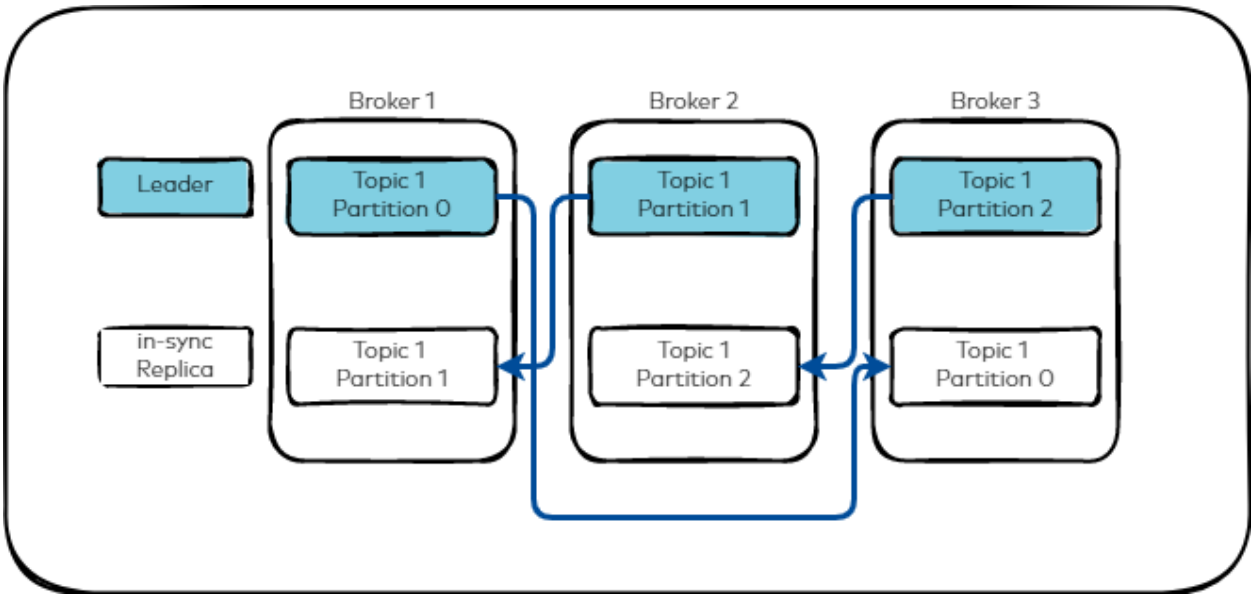
اینجاست که مفهومی با نام Apache Kafka به کمک مهندسان نرم‌افزار می‌آید.

Apache Kafka یک مخزن داده توزیع‌شده و به عبارتی یک Message Broker است که برای دریافت و پردازش داده‌های استریمینگ در زمان واقعی بهینه‌سازی شده است. داده‌های استریمینگ داده‌هایی هستند که به طور پیوسته توسط هزاران منبع داده تولید می‌شوند و معمولاً به صورت همزمان ارسال می‌گردند. یک پلتفرم استریمینگ باید بتواند این جریان پیوسته از داده‌ها را مدیریت کرده و داده‌ها را به صورت ترتیبی و تدریجی پردازش کند.

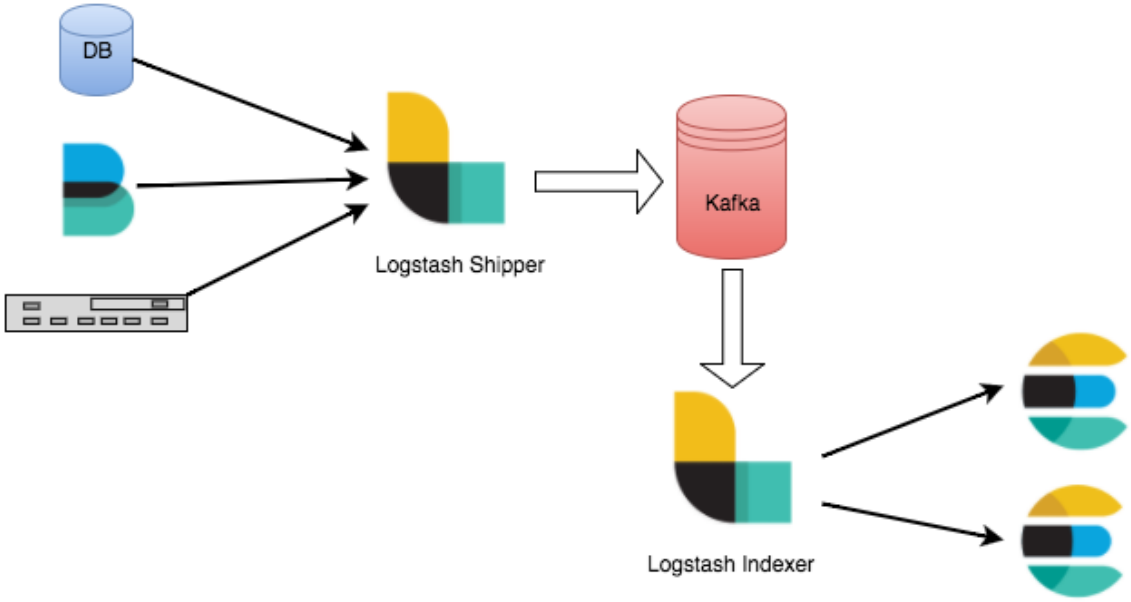
Kafka به گونه‌ای طراحی شده است که بتواند حجم بسیار زیادی از داده‌ها را به صورت کارآمد و با تاخیر کم پردازش کند. ساختار توزیع‌شده آن به کاربران امکان می‌دهد تا سیستم را بر اساس نیازهای خود به سادگی گسترش دهند و منابع را به طور پویا مدیریت کنند.



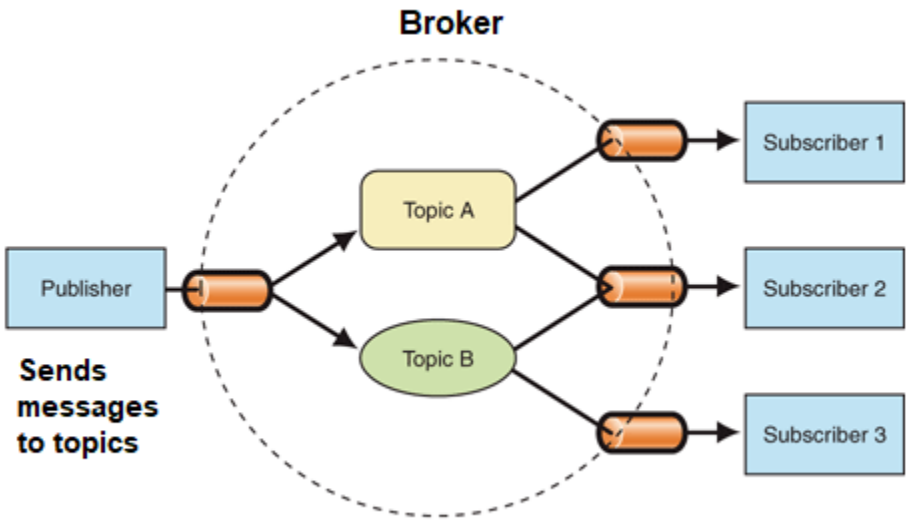
یکی از ویژگی‌های کلیدی Kafka ، استفاده از مفهوم replication است که داده‌ها را بین چندین سرور کپی می‌کند. این ویژگی باعث می‌شود که حتی در صورت بروز خرابی در یک یا چند سرور، داده‌ها از دست نروند و سیستم همچنان به کار خود ادامه دهد.



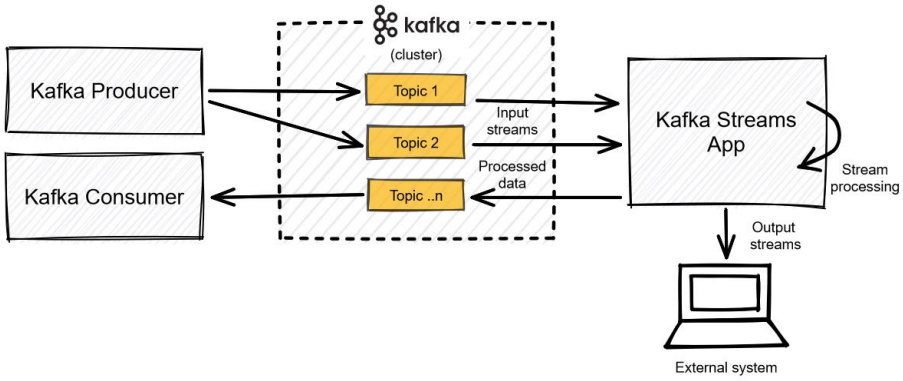
Kafka برای دستیابی به عملکرد بالا بهینه‌سازی شده است. این ابزار می‌تواند میلیون‌ها پیام در ثانیه را با تاخیر کم پردازش کند. به همین دلیل، برای کاربردهایی که نیاز به پردازش سریع داده‌ها دارند، مانند تحلیل داده‌های بلادرنگ و مانیتورینگ، بسیار مناسب است.

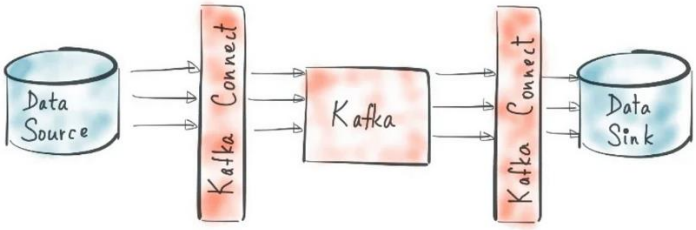


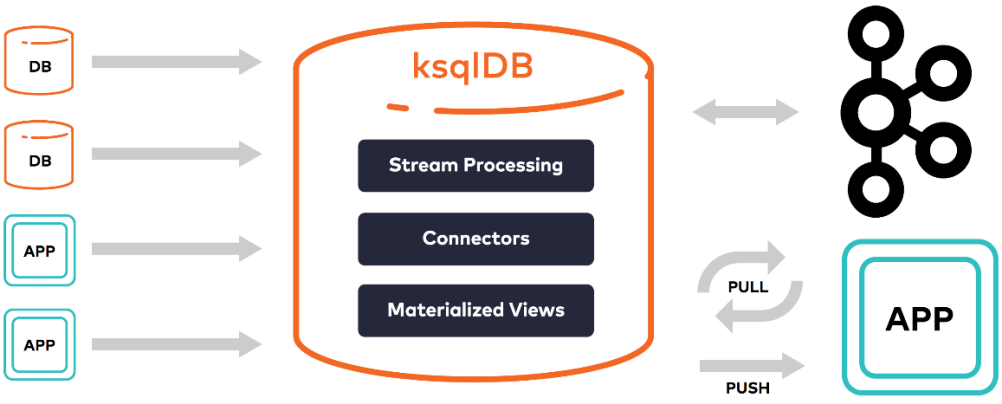
Kafka از معماری publish-subscribe بهره می‌برد که به کاربران امکان می‌دهد تا داده‌ها را به صورت انعطاف‌پذیر بین سیستم‌ها و سرویس‌های مختلف منتشر و مشترک شوند. این معماری امکان ساخت سیستم‌های پیچیده و تطبیق‌پذیر را فراهم می‌کند



Kafka با داشتن اکوسیستم گسترده‌ای از ابزارها و افزونه‌ها، امکانات متنوعی برای پردازش و تحلیل داده‌ها ارائه می‌دهد. برخی از این ابزارها شامل Kafka Streams برای پردازش استریم‌ها، Kafka Connect برای اتصال به سیستم‌های مختلف و KSQL برای تحلیل داده‌ها با استفاده از SQL می‌شوند.

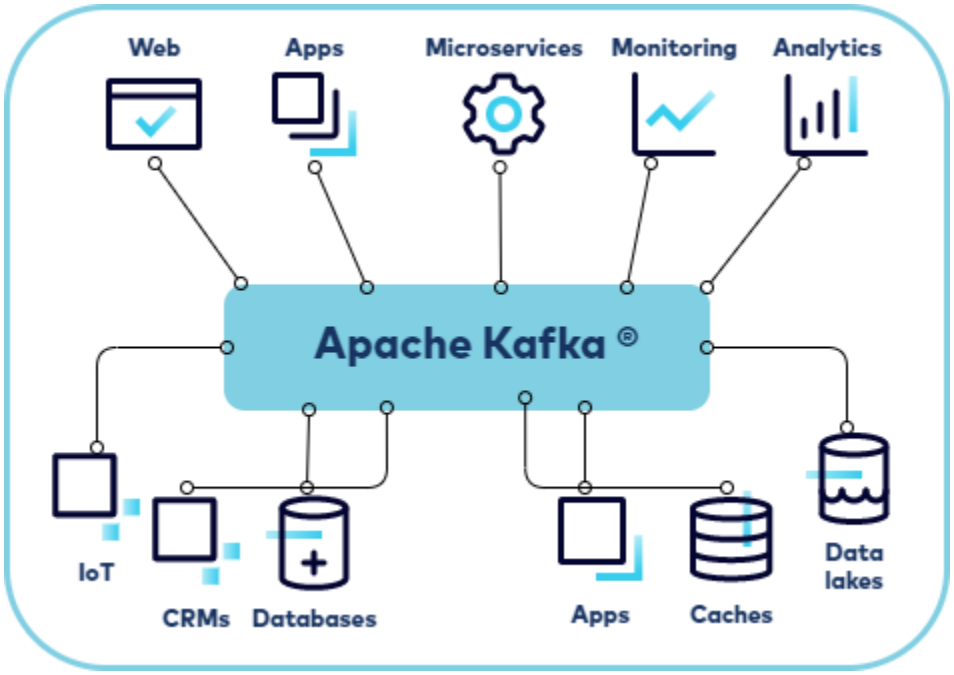






Kafka به کاربران امکان می‌دهد تا داده‌ها را به صورت بلادرنگ پردازش و تحلیل کنند. این ویژگی برای کاربردهایی مانند تحلیل رفتار کاربران، پردازش تراکنش‌های مالی، مانیتورینگ سیستم‌ها و دستگاه‌ها بسیار مفید است.

Apache Kafka دارای یک جامعه کاربری بزرگ و فعال است که به توسعه، بهبود و پشتیبانی از این ابزار کمک می‌کند. وجود منابع آموزشی متعدد، مستندات جامع و انجمن‌های پشتیبانی به کاربران جدید کمک می‌کند تا به سرعت با Kafka آشنا شوند و از آن استفاده کنند.



Kafka با بسیاری از فناوری‌ها و ابزارهای موجود در حوزه‌های مختلف از جمله Hadoop ، Spark، Flink و Elasticsearch سازگاری دارد. این سازگاری امکان ادغام و استفاده از Kafka در معماری‌های موجود را تسهیل می‌کند.

ساختار و مفاهیم کلیدی در Kafka

1. Topic

یک Topic در Kafka نمایانگر یک جریان داده است. داده‌ها در Kafka به صورت دسته‌های جداگانه‌ای که Topic نامیده می‌شوند، سازماندهی می‌شوند.

- **کاربرد** :تولیدکننده‌ها (Producers) داده‌ها را به Topicهای خاصی می‌فرستند و مصرف‌کننده‌ها (Consumers) داده‌ها را از این Topicها دریافت می‌کنند.

2. پارتیشن (Partition)

هر Topicای می‌تواند به چندین پارتیشن تقسیم شود. هر پارتیشن یک بخش از داده‌های یک Topic را نگهداری می‌کند.

- **مزایا:**
  - **مقیاس‌پذیری** :تقسیم داده‌ها به پارتیشن‌ها امکان پردازش موازی و افزایش مقیاس‌پذیری را فراهم می‌کند.
  - **پایداری** :هر پارتیشن می‌تواند بر روی چندین سرور (بروکر) نگهداری شود تا در صورت خرابی یکی از سرورها، داده‌ها از دست نروند.

4. مصرف‌کننده (Consumer)

مصرف‌کننده‌ها داده‌ها را از Topicهای خاص در Kafka دریافت و پردازش می‌کنند.

5. بروکر(Broker)

سرورهایی که پیام‌ها را دریافت و ذخیره می‌کنند. هر Kafka cluster شامل یک یا چند بروکر است.

- **کاربرد** :بروکرها پیام‌ها را در پارتیشن‌های Topicها ذخیره می‌کنند و در اختیار مصرف‌کننده‌ها قرار می‌دهند.

6. Replication

هر پارتیشن می‌تواند چندین نسخه (replica) داشته باشد که بر روی بروکرهای مختلف نگهداری می‌شوند.

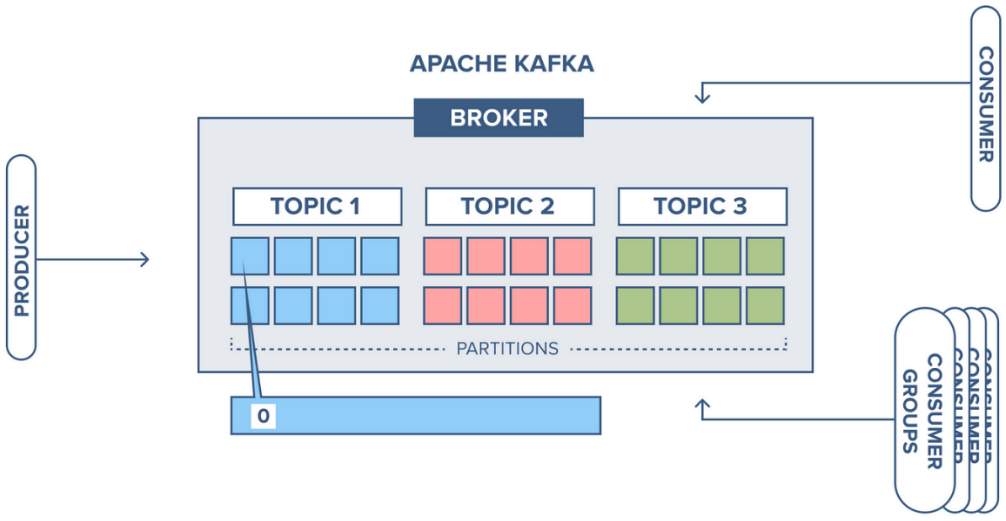
- **مزایا** :این ویژگی باعث افزایش پایداری و قابلیت اطمینان Kafka می‌شود، زیرا در صورت خرابی یکی از بروکرها، داده‌ها از نسخه‌های دیگر بازیابی می‌شوند.

7. گروه مصرف‌کننده (Consumer Group)

مجموعه‌ای از مصرف‌کننده‌ها که با هم کار می‌کنند تا پیام‌ها را از یک Topic دریافت و پردازش کنند.

گروه‌های مصرف‌کننده باعث افزایش مقیاس‌پذیری و توزیع بارکاری می‌شوند. هر پیام در یک Topic تنها توسط یک مصرف‌کننده از هر گروه پردازش می‌شود.

همچنین یکی از اعضای گروه (معمولا عضوی که زودتر عملیات مصرف را شروع کرده است) به عنوان leader انتخاب می‌شود تا در صورت بروز اتفاقاتی همچون dead شدن یکی از اعضا، عملیات assign یک پارتیشن را که حالا عضو آن مرده است، به عضوی دیگر انجام دهد.



ساختار داخلی پیام‌ها در Kafka

اجزای اصلی پیام درKafka

1. **Record**:
   - هر پیام در Kafka به عنوان یک رکورد (Record) شناخته می‌شود.
   - هر رکورد شامل یک کلید(key) ، یک مقدار (value) و متادیتاهای دیگری است.
2. **Key**:
   - کلید می‌تواند به توزیع پیام‌ها در پارتیشن‌ها کمک کند. پیام‌هایی با کلید یکسان به یک پارتیشن خاص ارسال می‌شوند.
   - کلید می‌تواند اختیاری باشد. اگر کلید استفاده نشود، توزیع پیام‌ها به صورت تصادفی انجام می‌شود.
3. **Value**:
   - مقدار اصلی پیام است که حاوی داده‌های مورد نظر برای انتقال است.
   - مقدار می‌تواند هر نوع داده‌ای باشد که به صورت بایت‌ها قابل ذخیره و انتقال باشد. (UTF-8, Base-64...)
4. **Offset**:
   - هر پیام در یک پارتیشن دارای یک شناسه منحصر به فرد به نام offset است.
   - Offset نشان‌دهنده موقعیت پیام در پارتیشن است و توسط Kafka برای مدیریت پیام‌ها و پیگیری وضعیت مصرف‌کننده‌ها استفاده می‌شود.
5. **Timestamp**:
   - هر رکورد می‌تواند یک تایم‌استمپ داشته باشد که زمان تولید یا زمان دریافت پیام توسط Kafka را نشان می‌دهد.
   - تایم‌استمپ به تحلیل زمانی و مدیریت پیام‌ها کمک می‌کند.
6. **Headers**:
   - Kafka از نسخه 0.11.0.0 به بعد، از هدرها (Headers) پشتیبانی می‌کند.
   - هدرها شامل جفت‌های کلید-مقدار اختیاری هستند که متادیتای اضافی در مورد پیام ارائه می‌دهند.

یک پیام کامل در Kafka می‌تواند به صورت زیر باشد:

```
Record {
    Key: "user123",
    Value: "OrderID: 456, Amount: 789",
    Offset: 42,
    Timestamp: 1627584000,
    Headers: {
        "Source": "web",
        "Content-Type": "application/json"
    }
}
```

نحوه ذخیره‌سازی پیام‌ها درKafka

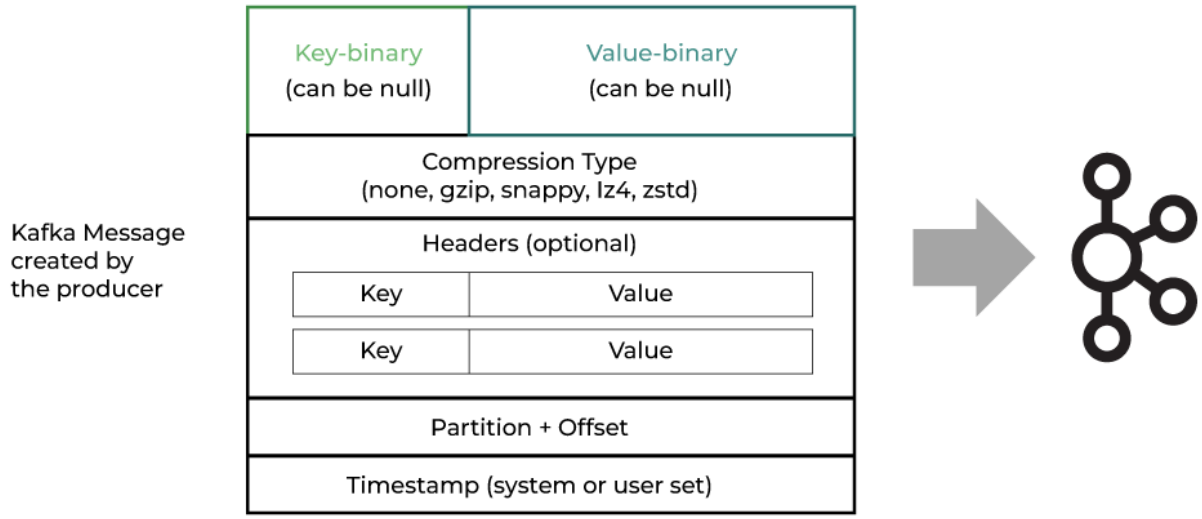
پیام‌ها در Kafka در داخل لاگ‌ها ذخیره می‌شوند که در پارتیشن‌ها تقسیم شده‌اند. هر پارتیشن یک لاگ ترتیب‌دار از پیام‌هاست که به ترتیب ورود ذخیره می‌شوند. این لاگ‌ها به صورت توالی‌دار نوشته و خوانده می‌شوند.

Log Segments:

- هر پارتیشن به چندین بخش (Segment) تقسیم می‌شود که هر کدام یک فایل فیزیکی در دیسک هستند.
- وقتی اندازه یک بخش از حد معین عبور کند یا زمان معینی بگذرد، Kafka یک بخش جدید ایجاد می‌کند.

Indexes:

- برای دسترسی سریع به پیام‌ها، Kafka از ایندکس‌ها استفاده می‌کند.
- هر پارتیشن دارای ایندکس‌های جداگانه‌ای برای Offset و Time است که دسترسی به پیام‌ها را سریع‌تر می‌کند.



مفاهیم پیشرفته درKafka

Consumer Offset

- Offset :یک شناسه منحصر به فرد است که نشان‌دهنده موقعیت هر پیام در داخل یک پارتیشن است. مصرف‌کننده‌ها (Consumers)از offset برای پیگیری پیام‌هایی که پردازش کرده‌اند استفاده می‌کنند.
- **ذخیره‌سازی** : Offsetها می‌توانند به صورت دستی یا خودکار commit شوند و در یک Topic خاص به نام __consumer_offsets__ذخیره می‌شوند.
- **مدیریت** :مصرف‌کننده‌ها می‌توانند با استفاده ازAPI های Kafka ، offsetهای خود را مدیریت کنند و تصمیم بگیرند که از کجا خواندن پیام‌ها را شروع کنند.

Acknowledgment در Kafka

- Acknowledgment (ACK) در Kafka فرایندی است که در آن تولیدکننده (Producer) تأیید می‌کند که پیام به درستی توسط بروکر (Broker) دریافت شده است.
- **انواع:ACK**
  - **acks=0** :تولیدکننده منتظر تأیید دریافت پیام از بروکر نیست. (بدون اطمینان)
  - **acks=1** :تولیدکننده منتظر تأیید دریافت پیام از رهبر پارتیشن (Leader) است.
  - **acks=all :**تولیدکننده منتظر تأیید دریافت پیام از همه رپلیک‌ها (Replicas) است. (بیشترین اطمینان)

Commit

- Commit در Kafka فرایندی است که در آن مصرف‌کننده‌ها offset هایی را که پردازش کرده‌اند، به Kafka اعلام می‌کنند.
- **انواع:**
  - **خودکار(Auto Commit)** : Kafka به صورت دوره‌ای و خودکار offset های پردازش شده را commit می‌کند.
  - **دستی(Manual Commit)** : مصرف‌کننده‌ها به صورت دستی و با استفاده ازAPI های Kafka، offsetها را commit می‌کنند. این روش انعطاف‌پذیری بیشتری برای مدیریت پیام‌ها و خطاها فراهم می‌کند.

Heartbeat

- Heartbeat :سیگنال‌هایی هستند که توسط مصرف‌کننده‌ها به گروه مصرف‌کننده (Consumer Group) ارسال می‌شوند تا وضعیت فعال بودن خود را اعلام کنند.
- **کاربرد:**
  - اطمینان از اینکه مصرف‌کننده‌ها هنوز فعال و در حال کار هستند.
  - شناسایی مصرف‌کننده‌های مرده یا غیر فعال و توزیع مجدد پارتیشن‌های آن‌ها بین مصرف‌کننده‌های دیگر.

همچنین بهتر است بدانیم که کافکا باید برروی یک سیستمی که وظیفه ایجاد هماهنگی بین clusterهای آن را دارد، پیاده‌سازی شود در اینجاست که از Zookeeper استفاده میکنیم:

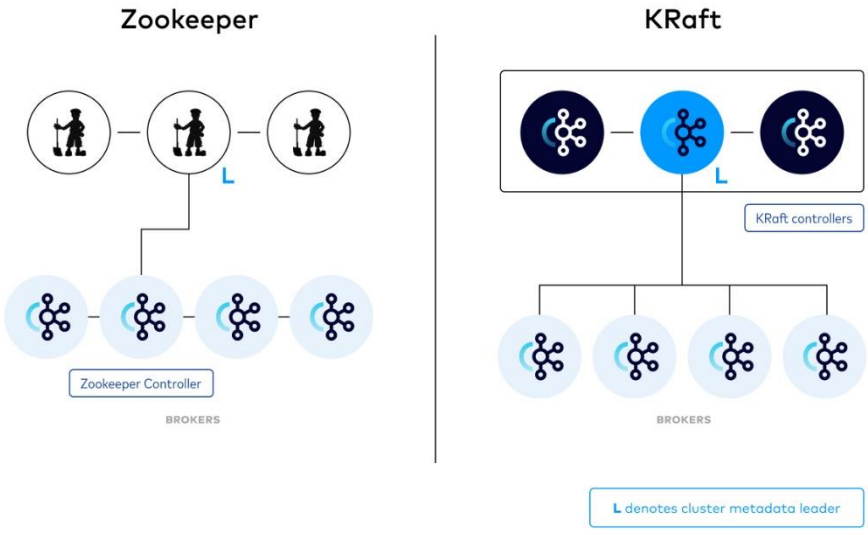
Zookeeper:

- Zookeeper :یک سرویس هماهنگی توزیع‌شده است که Kafka از آن برای مدیریت اطلاعات متا، هماهنگی بین بروکرها و پیگیری وضعیت خوشه‌ها استفاده می‌کند.
- **وظایف**:
  - مدیریت رهبران پارتیشن‌ها(Leader election)
  - ذخیره و مدیریت تنظیمات بروکرها
  - پیگیری وضعیت مصرف‌کننده‌ها وoffset ها

البته در سال‌های اخیر و با پیشرفت دانش و تکنولوژی از ابزارهای دیگری هم به عنوان جایگزین برایZookeeper استفاده می‌شود که نمونه آن را در ذیل میتوانیم بررسی کنیم:

KRaft (Kafka Raft Metadata Mode):

- Kafka در حال انتقال به یک مدل جدید مدیریت متا به نام KRaft است که وابستگی به Zookeeper را حذف می‌کند.
- **مزایا**:
  - کاهش پیچیدگی معماری
  - بهبود عملکرد و مقیاس‌پذیری
  - افزایش پایداری و قابلیت اطمینان



Apache Kafka یک پلتفرم پیام‌رسانی توزیع‌شده است که برای دریافت، ذخیره‌سازی و پردازش داده‌های استریمینگ در زمان واقعی طراحی شده است. این سیستم با استفاده از مفاهیمی چون Topics، پارتیشن‌ها(Partitions) ، وoffset ها، پیام‌ها را به صورت مؤثر و قابل مقیاس‌پذیری مدیریت می‌کند . Kafka ازACK ها برای تأیید دریافت پیام‌ها، و از commitها برای مدیریت offsetهای پردازش شده توسط مصرف‌کننده‌ها استفاده می‌کند. این پلتفرم از Zookeeper یا Kraft برای هماهنگی و مدیریت متادیتا بهره می‌برد و با استفاده از سیگنال‌های heartbeat وضعیت مصرف‌کننده‌ها را مانیتور می‌کند . Kafka با ویژگی‌های مقیاس‌پذیری بالا، پایداری و قابلیت اطمینان، به یکی از ابزارهای محبوب در معماری‌های میکروسرویس و پردازش بلادرنگ تبدیل شده است.