Data Engineering

Message Queues, Event Streaming & Apache Kafka

Queuing

Queuing

- Methode um Nachrichten zwischen Applikationen auszutauschen
- Nachrichten werden FIFO (first in first out) verarbeitet
- Es gibt Publisher und Subscriber
- Queues sind exhaustive: Nachrichten die konsumiert wurden werden gelöscht
- Das System das die Queues hält, nennt sich **Message Broker**

Queue Data Structure

Queuing

- Publisher schreiben Nachrichten in eine oder mehrer Queues
- Subscriber lesen Nachrichten aus einer oder mehren Queues
- Applikationen können Publisher und Subscriber zur gleichen Zeit sein

Messages

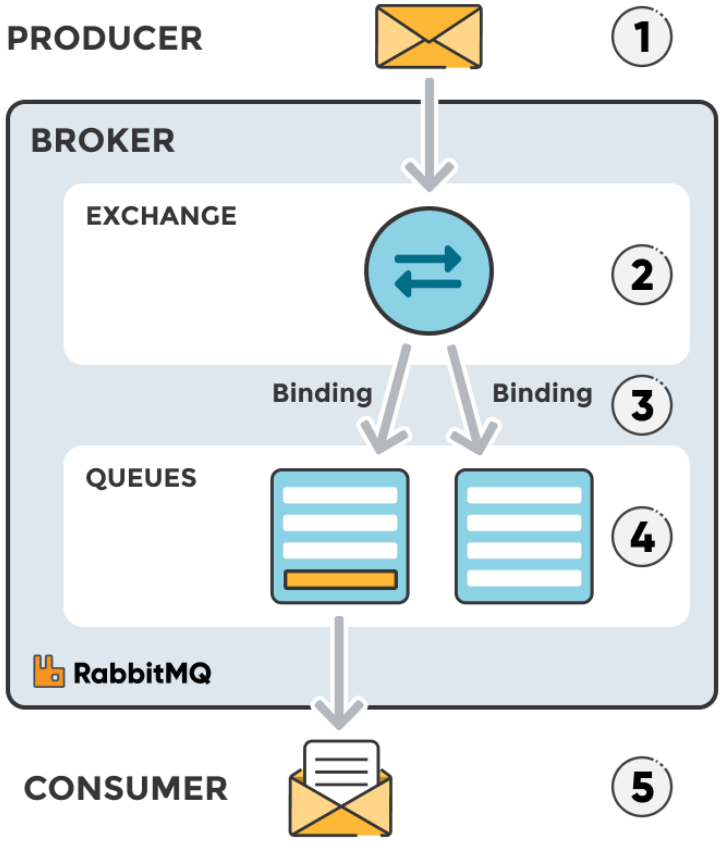
- Routing-Key: Key der für Routing genutzt wird
- Nachrichten verfügen über einen Header mit Metainformationen
- Body enthält Nutzdaten – Eine Message Queue ist agnostisch und kann beliebige Nachrichten “verschicken“ – es gibt keine Konsistenzkriterien

Routing

- Grundsätzlich werden Nachrichten, je nach Inhalt und Empfänger, in unterschiedliche Queues geschrieben – Warum?
  - Eine Queue ist exhaustive – was gelesen wird, wird gelöscht – Wollen wir die gleiche Nachricht an mehrere Empfänger zustellen benötigen wir weitere Queues
  - Nicht jede Nachricht ist für jeden Empfänger relevant
  - Verteilung & Verarbeitung der Nachrichten basiert oft auf den Headern der Nachricht
- Es wird ein Routing-System benötigt, welche produzierte Nachrichten auf Basis von Regeln an die entsprechenden Queues verteilt

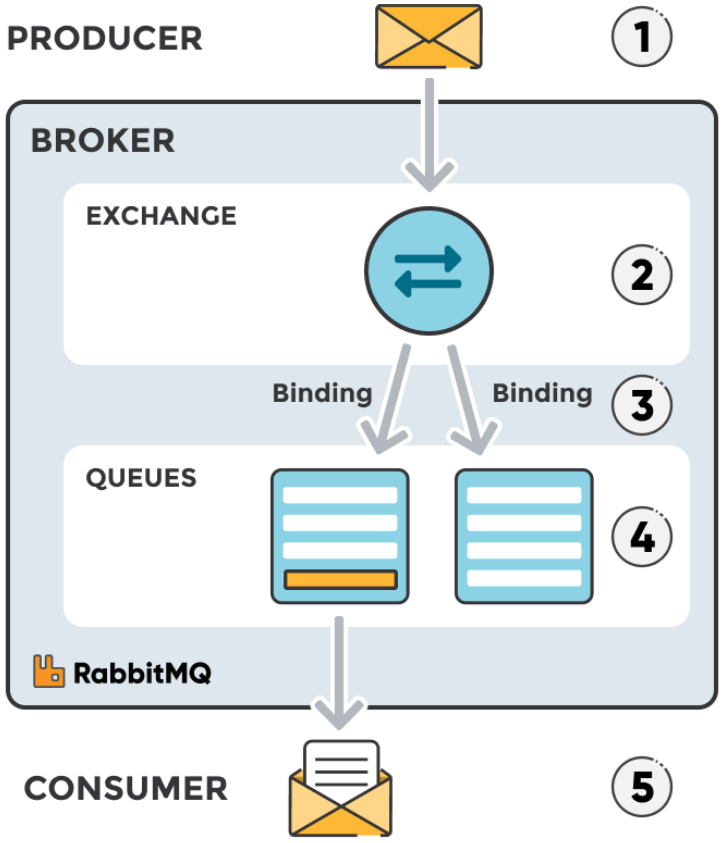
Routing – Exchanges

- In RabbitMQ werden diese „Router“ Exchanges genannt
- Der Producer schickt Nachrichten an den Exchange
- Der Exchange verteilt die Nachrichten auf Queues basierend auf dem Exchange Type



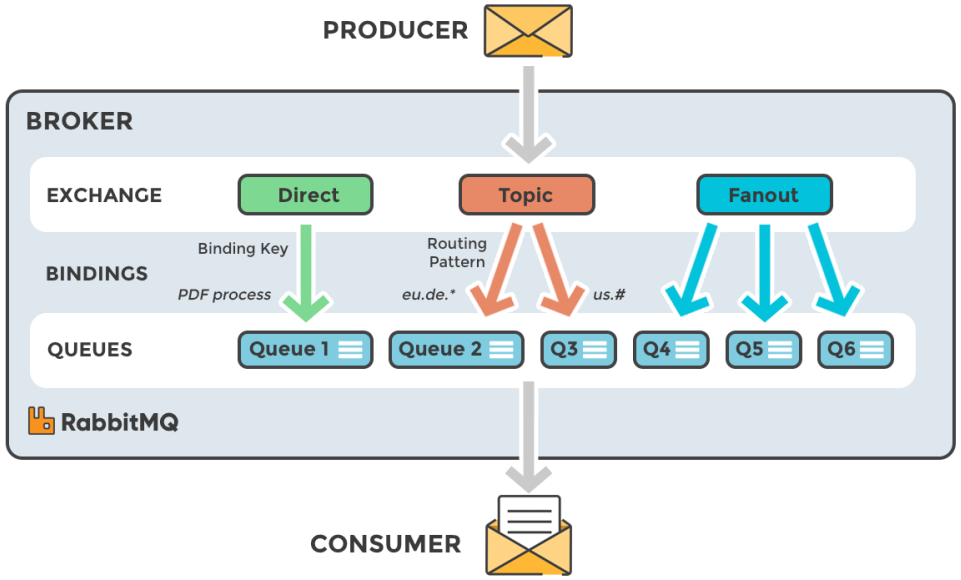
Routing – Bindings

- Bindings verbinden Exchanges mit Queues
- Bindings besitzen Keys, die für das Routing genutzt werden können



Routing – Exchange-Types

- **Direct –** Nachricht werden an die Queues weitergeleitet wenn der Routing Key mit dem Binding Key übereinstimmt
- **Fanout –** Routet Nachrichten an alle Queues die mit dem Exchange verbunden ist
- **Topic –** Wildcard Match auf Basis des Routing Keys
- **Headers:** Routing passiert auf Basis von Werten aus dem Header



Vorteile von Queues

Vorteile

- Kommunikation zwischen Applikationen ist „decoupled“
- Applikationen kommunizieren nicht mehr Punkt zu Punkt sondern schreiben “Events“ in ein externes System
- Einfache Erweiterbarkeit wenn weitere Applikationen ebenfalls über das Event benachrichtigt werden müssen
- Gesteigerte Resilienz
- Asynchrone Verarbeitung
- Daten werden kontinuierlich verarbeitet
- Software Architektur „Eventdriven Microservices“ basiert auf diesem Prinzip

Event Streaming

Event Streaming

- Event Streaming und Queuing sind sehr ähnlich
- Ebenfalls eine FIFO Datenstruktur für die Verarbeitung von Echtzeitdaten
- Event Streams (auch Topics) sind nicht exhaustive
- Statt auf Queues basieren Event Streams auf Logs
- Es existieren weder Exchanges noch Bindings
- Prominentestes Beispiel: **Apache Kafka**

Kafka Terminologien

- **Kafka Topics** – Äquivalent zu einer Queue ohne exhaustion
- **Producer / Consumer** – Systeme die in Topics produzieren oder daraus lesen
- **Logs –** Datenstruktur in der Nachrichten gespeichert werden – ein Topic besteht aus mehreren Logs
- **Broker –** Kafka besteht aus einer Menge an Brokern, die Requests verarbeiten und die Daten in Logs auf der Festplatte speichern
- **Partitons –** Kafka ist ein verteiltes System und verteilt Daten eines Topics mit Hilfe von Partitions über die Broker

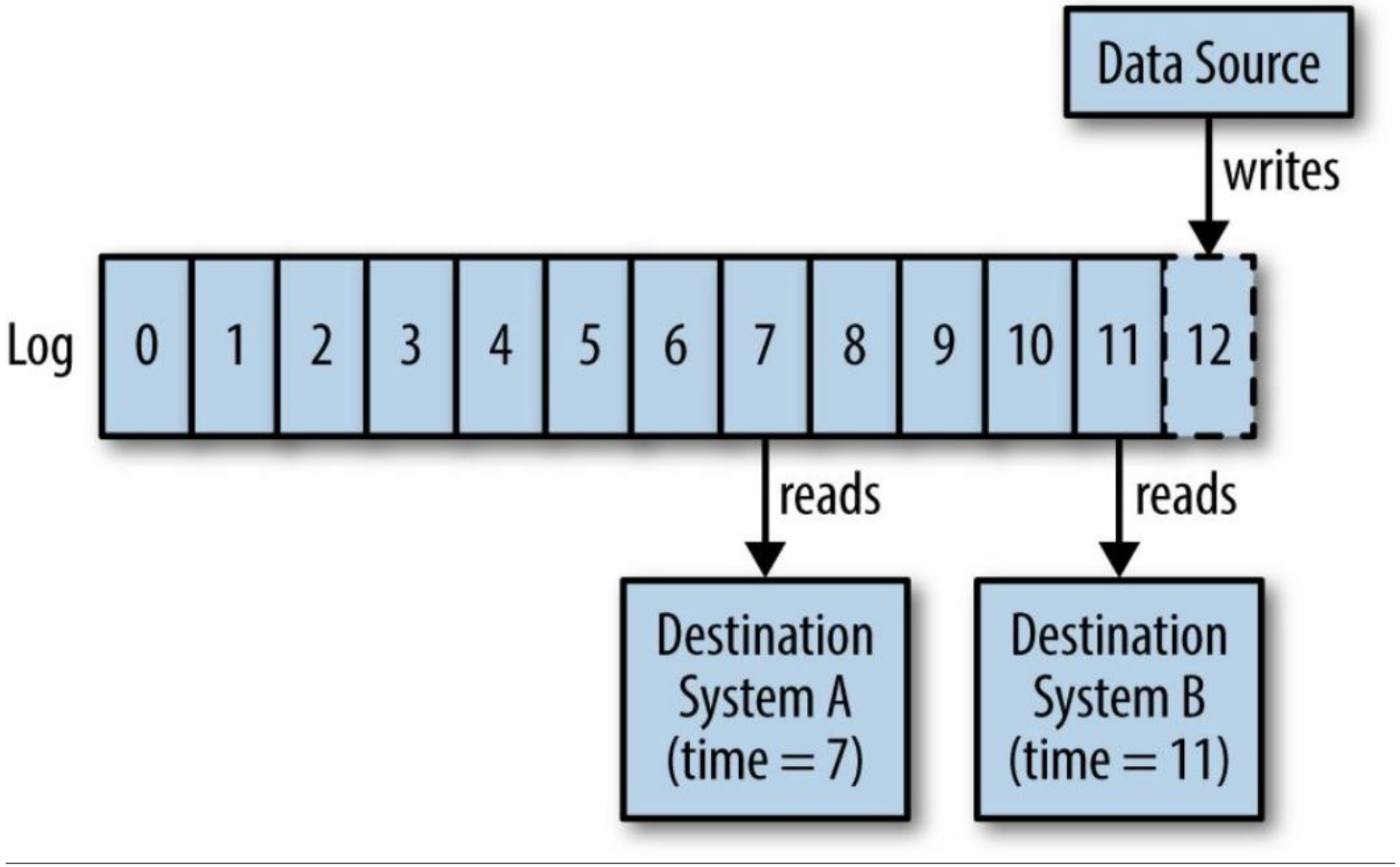
Kafka Terminologie – Event

- Unveränderbarer Fakt
- Auf ein Event kann man reagieren, aber nicht antworten
- Einzelnes Datum eines Event-Streams besitzt 3 Informationen:
  1. Zustandsbeschreibung (Event-Payload)
  2. Timestamp
  3. Kontext (Thema, Topic, Queue, Stream)

Logs

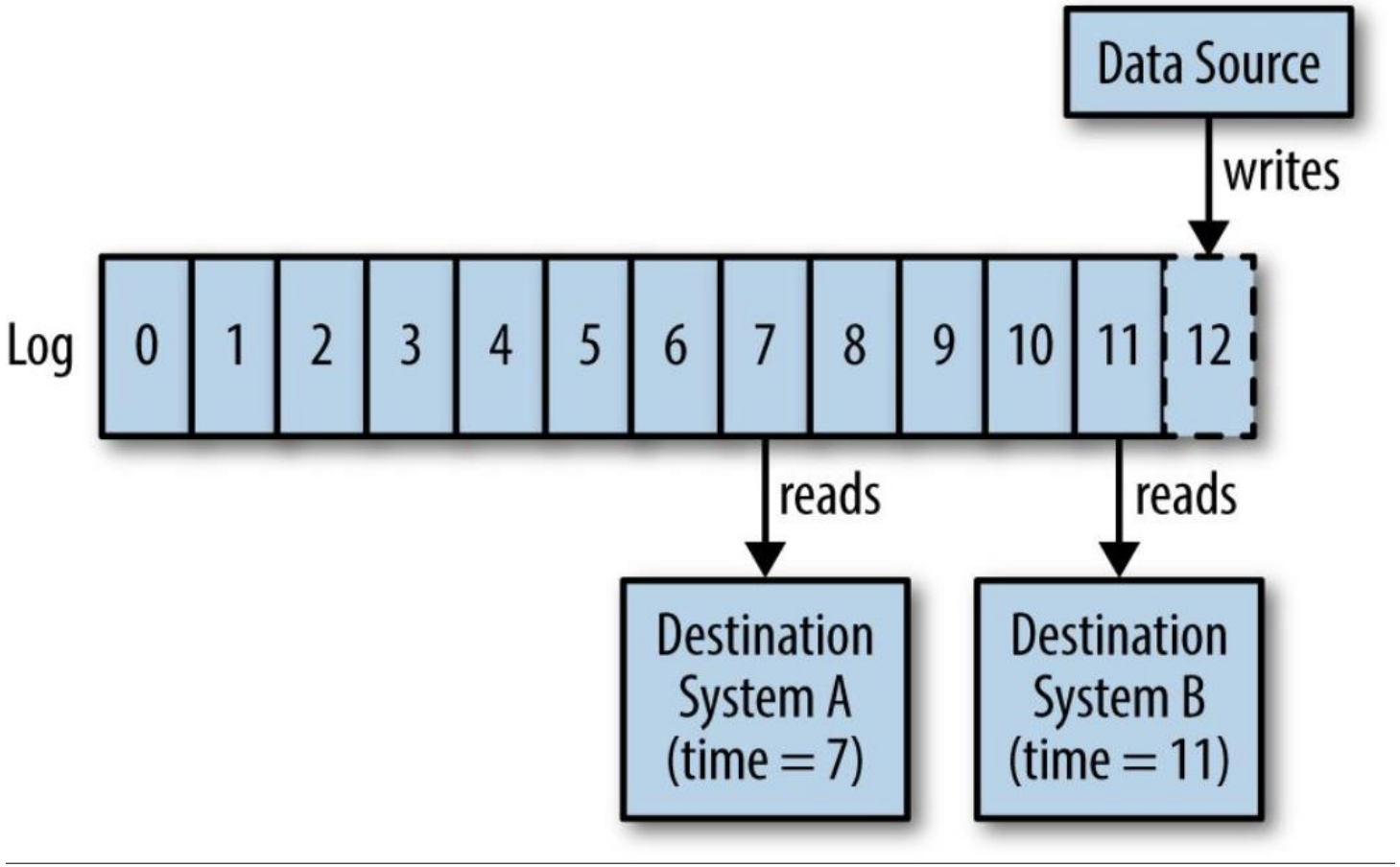
Logs

- Speicherung von Daten in einer zeitlich geordneten Sequenz
- Kein Löschen auf Basis gelesener Nachrichten
- Ein oder mehrere Pointer markieren die Leseposition



Logs

- Mehrere Applikations-Gruppen können gleichzeitig, unabhängig auf dem Log lesen
- Pointer können händisch verschoben werden – Events können erneut in der ursprünglichen Reihenfolge verarbeitet werden



Consumer Groups

Consumer Groups

- Mehrere Applikationen können ohne gegenseitige Beeinflussung gleichzeitig Daten aus einem Topic verarbeiten
- Verhalten bei Applikationen des gleichen Typs (z.B. Microservices) ungewünscht
  - Beispiel: Eine Banktransaktion sollte nur einmal auf dem entsprechenden Konto verbucht werden auch wenn mehrere Instanzen des Konto-Services gleichzeitig Transaktionen verarbeiten

Consumer Groups

- Consumer Groups gruppieren Consumer die sich einen Pointer auf die Daten teilen
- Für die Consumer einer Consumer Group verhält sich das Topic wie eine Queue mit dem Unterschied, dass eine „Replay“ der Nachrichten möglich ist
- Andere Consumer Groups besitzen eigene, unabhängige Pointer
- Vorteile:
  - Weitere Consumer Groups benötigen keine zusätzlichen Topics und kein Routing – Perfekt für Decoupling und Stream Processing
  - Events können jederzeit wiederholt werden

Consumer Groups

- Andere Consumer Groups besitzen eigene, unabhängige Pointer
- Vorteile:
  - Weitere Consumer Groups benötigen keine zusätzlichen Topics und kein Routing – Perfekt für Decoupling und Stream Processing
  - Events können jederzeit wiederholt werden
- Die Anzahl der Consumer pro Consumer Group ist limitiert (gleich mehr)

Vorteil des Konzepts

Whiteboard Session

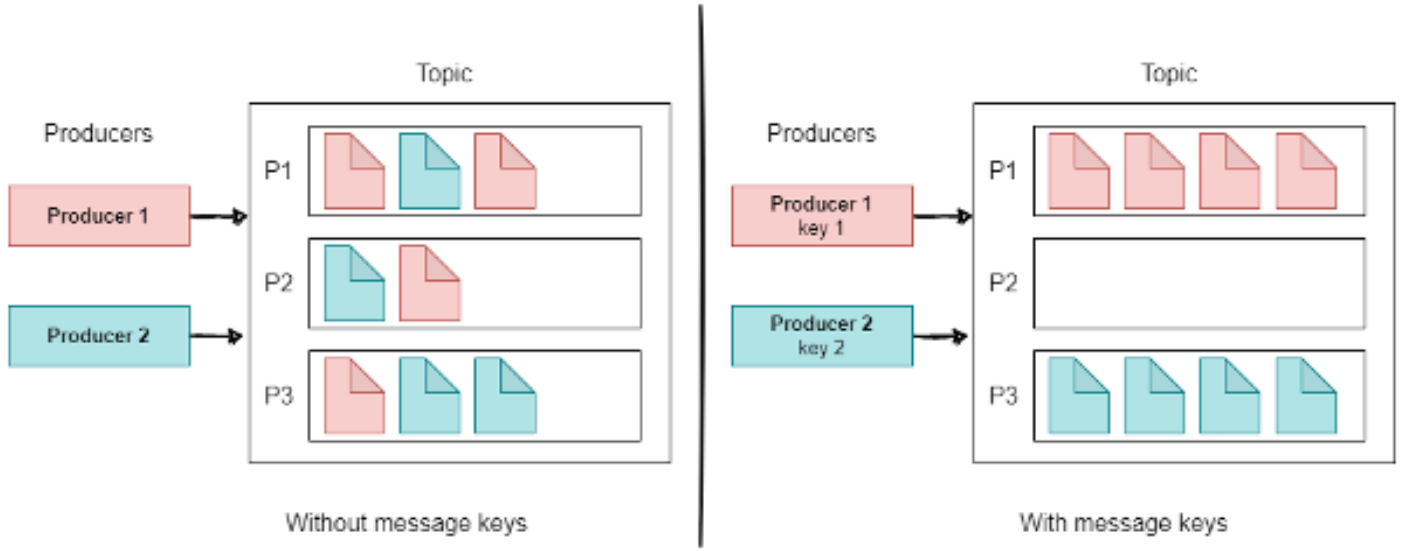
Partitionierung

Partitionierung

- Kafka ist ein hoch effizientes Verteiltes System ohne Limitierung von Skalierungen
- Topics werden Partitioniert und über mehrere Broker verteilt um diese Skalierbarkeit sicher zu stellen

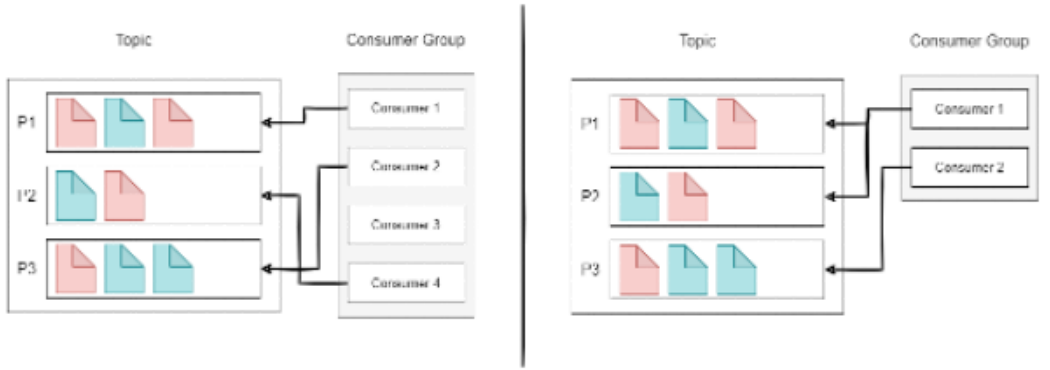
Partitionierung

- Topic wird in mehrere Partitionen aufgeteilt
- Partitionen bestehen aus unabhängigen Logs
- Partitionen werden über Broker verteilt
- Partition geschieht auf Basis des Keys oder Round Robin



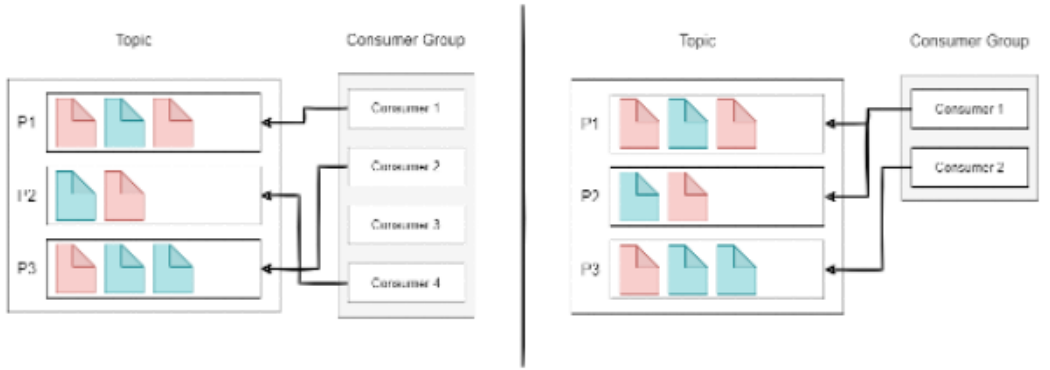
Partitioning

- Partitionierung führt zu mehreren, unabhängigen Logs
- Verarbeitung in der ursprünglichen Reihenfolge kann nur auf Basis der Partition sichergestellt werden
- Es kann maximal ein Consumer aus einer Group aus einer Partition lesen
  - Damit wird die einhaltung der Reihenfolge auf Partitionsebene sichergestellt



Partitioning

- Ist die Reihenfolge in der Events konsumiert werden wichtig, kann dies über clever gewählte Keys erreicht werden
- Beispiel Bank: Reihenfolge der Kontobewegung essentiell aber nur auf Ebene des individuellen Kontos. Konto-ID als Partition Key sorgt dafür dass alle Events eines Kontos garantiert in der richtigen Reihenfolge verarbeitet werden
- Vorsicht: Wird der falsche Key verwendet kommt es zu einem unbalancierten Cluster – Skalierbarkeit eingeschränkt



Replication

Replication

- Kafka bietet die Replication von Logs um Datenverlust vorzubeugen
- Replication ist ein wichtiges Konzept da Broker (Queue und Streaming) – im Gegensatz zu Datenbanken – konzeptionell garnicht oder nur sehr schlecht zu backupen sind

Replication

- Replication Setting pro Topic gibt an wie oft das Topic repliziert wird
- Replication von 1 bedeutet, dass keine Replication existiert
- Jede Partition hat eine Leader und 0-n Follower
- Alle reads und writes werden von der Leader Partition gehandelt

Replication

- Follower konsumieren Nachrichten von der Leader Partition und fügen sie ihrem eigenen Log hinzu
- Problem bei verteilten Systemen wann ist ein Fehlerzustand eines Nodes erreicht:
  1. Node hält die Verbindung mit dem Controller
  2. Im Falle eines Followers: Muss die Writes eines leaders replizieren und nicht „zu weit“ hintendran sein
- Nodes die diese Bedingungen erfüllen sind „in-sync“ vs. „alive“ oder „failed“
- Je nach Anwendungsfall kann konfiguriert werden wann der Zustand erreicht wird

Replication - Acks

- Kafka ist ein System zur Verarbeitung von Daten in naher Echtzeit
- Streaming Broker verarbeiten nicht selten mehrere 100.000de Nachrichten in der Sekunde
- Daten sicher zu replizieren führt deswegen zu einem Delay, welcher sich auf die Latenz der Verarbeitung auswirkt
- In manchen Usecases ist der Verlust einzelner Nachrichten irrelevant: Temperatur-Datum eines Ofens
- In anderen ist der Verlust verheerend: Banktransaktion
- **Sichere Replication geht immer zu Lasten der Latenz und umgekehrt**

Replication– Acks

- Wie kann eine sichere Replication erreicht werden?
- Der Kafka Broker schickt acknowledgement an Producer wenn Nachricht commited ist
  - Commited ist eine, dann wenn alle in-sync replicas einer partition die Nachricht gespeichert haben
- Producer entscheidet via Parameter ob er auf acknowledgements warten möchte oder nicht
- Parameter erlaubt verschiedene Einstellung
  - acks=all => alle in-sync replicas haben die Nachricht verarbeitet (Anzahl der notwendigen In-Sync Replicas wird auf Topic Ebene eingestellt)
  - acks=1 => nur der Leader hat die Nachricht verarbeitet
  - Und so weiter…

Stream vs. Batch Processing

Brief introduction

Batch Processing

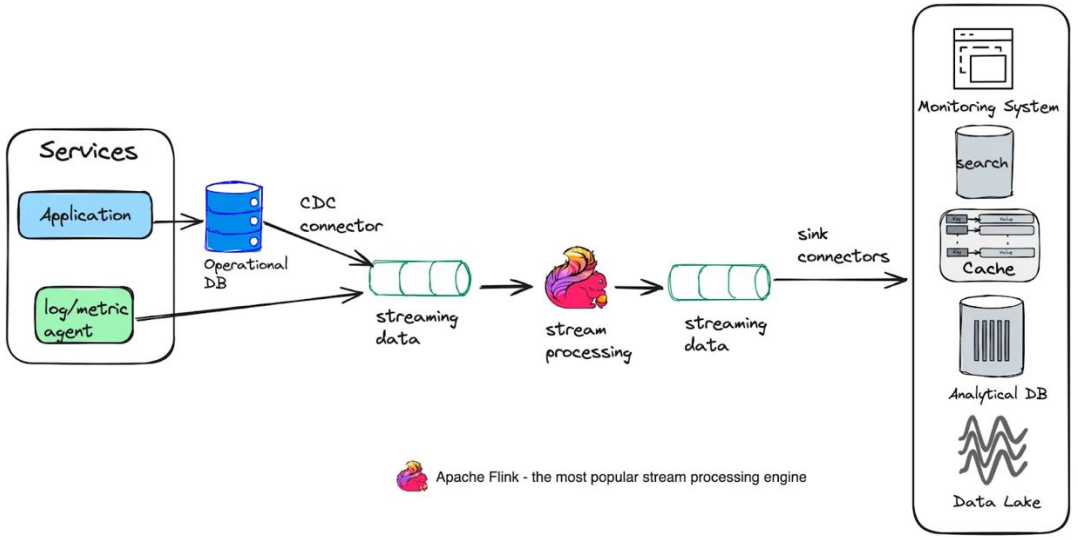
- Auch Stapel-Vearbeitung
- Periodisch wird die gesamte Menge der angefallenen Daten auf einmal verarbeitet

Probleme:

- Change Data Capture: Welche Daten haben sich verändert?
- Hohe Latenz
- Zeitpunkt der Verarbeitung
- Hohe Belastung für Systeme
- Zeitliche Abhängigkeiten sind teilweise schwer zu modellieren
- Fehlertoleranz

Stream Processing

- Daten Verarbeitung auf Basis von Events
- Events werden einzelnen und zum Zeitpunkt ihres Auftretens verarbeitet
- Stream Prozessoren konsumieren aus Topics und erzeugen Events die wiederum in Topics geschrieben werden
- So können Daten vor der weiteren Verarbeitung oder der finalen Speicherung analysiert, angereichert oder gefiltert werden



Nachteile Stream Processing

- Je nach Usecase komplexer
- Nicht jeder Anwendungsfall lässt sich mittels stream processing lösen
- Unter Umständen teurer

Prüfungsleistung

Abschlussprojekt

- **Projektziel:**
  Entwicklung einer End-to-End-Lösung zur Erfassung, Verarbeitung und Speicherung von Daten, inklusive einer Beispielanalyse.
- **Drei Säulen des Projekts:**

1.Scraping von Daten:

1. Daten müssen **periodisch** abgegriffen werden.
2. **Scheduling** idealerweise mit Tools wie **Airflow**.

2.Vorverarbeitung der Daten:

1. Externe und umfangreiche Vorverarbeitung der Daten.

3.Speicherung der Daten:

1. Speicherung in einer für den Usecase sinnvollen **Datenbank**.
2. **Einstellung und Optimierung** der Datenbank.
3. Sinnvolles Deployment mit entsprechenden Einstellungen.

- **Wichtig:** Beispielanalyse ist **kein integraler Bestandteil** des Data Engineerings.

Abschlussprojekt

- **Ablauf:**
- **Projektidee:**
  Einreichung bis zur **letzten Woche im Dezember**.
- **Freigabe:**
  Durch Projektleitung nach Prüfung.
- **Bearbeitungszeit:**
  Bis zum finalen **Präsentationstermin (tbd)**.
- **Abgabe und Bewertung:**

1. **Abschlusspräsentation:**
   1. Darstellung der getroffenen Entscheidungen.
2. **Code-Abgabe:**
   1. Muss in einem zugänglichen **Git Repository** vorliegen.
3. **Einstellungen und Deployment:**
   1. Alle Datenbank- und Deployment-Einstellungen als **Code** verfügbar (z. B. Skripte).
   2. Keine manuellen Änderungen (z. B. via Kibana).