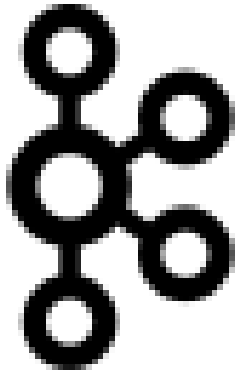
Sistem Terdistribusi (Distributed System)



(Special topics)

Apache Kafka

distributed streaming platform

Henry Saptono, M.Kom

Pengantar

- Setiap perusahaan didukung oleh data. Kita menerima informasi, menganalisis, memanipulasi, dan banyak menghasilkan sebagai output.
- Setiap aplikasi membuat data, apakah itu pesan log, metrik, aktivitas pengguna, pesan keluar, atau yang lainnya. Setiap byte data memiliki cerita sendiri, sesuatu yang penting yang akan menginformasikan tentang hal berikutnya yang harus dilakukan.

Pengantar

- Untuk mengetahui apa itu, kita perlu mendapatkan data dari tempat data itu dibuat ke tempat data itu dapat dianalisis.
- Perhatikanlah apa yang kita lihat setiap hari di situs web seperti Amazon, di mana ketika kita meng‘klik’ suatu item yang menarik bagi kita kemudian oleh Amazon diubah menjadi rekomendasi yang ditampilkan kepada kita beberapa saat kemudian.

Pengantar

- Semakin cepat kita melakukan ini, semakin gesit dan responsif organisasi kita. Semakin sedikit upaya yang kita habiskan untuk memindahkan data, semakin banyak kita dapat fokus pada bisnis inti yang ada. Inilah sebabnya mengapa pipa (*pipeline)* merupakan komponen penting dalam perusahaan berbasis data. Bagaimana cara kita memindahkan data menjadi hampir sama pentingnya dengan data itu sendiri.

Publish/Subscribe Messaging

- Sebelum membahas spesifik Apache Kafka, penting bagi kita untuk memahami konsep **publish/subscribe messaging** dan mengapa itu penting.
- Publish/subscribe messaging adalah pola yang ditandai oleh pengirim (penerbit) yang mengirimkan sepotong data (pesan) yang tidak secara khusus mengarahkannya ke penerima.

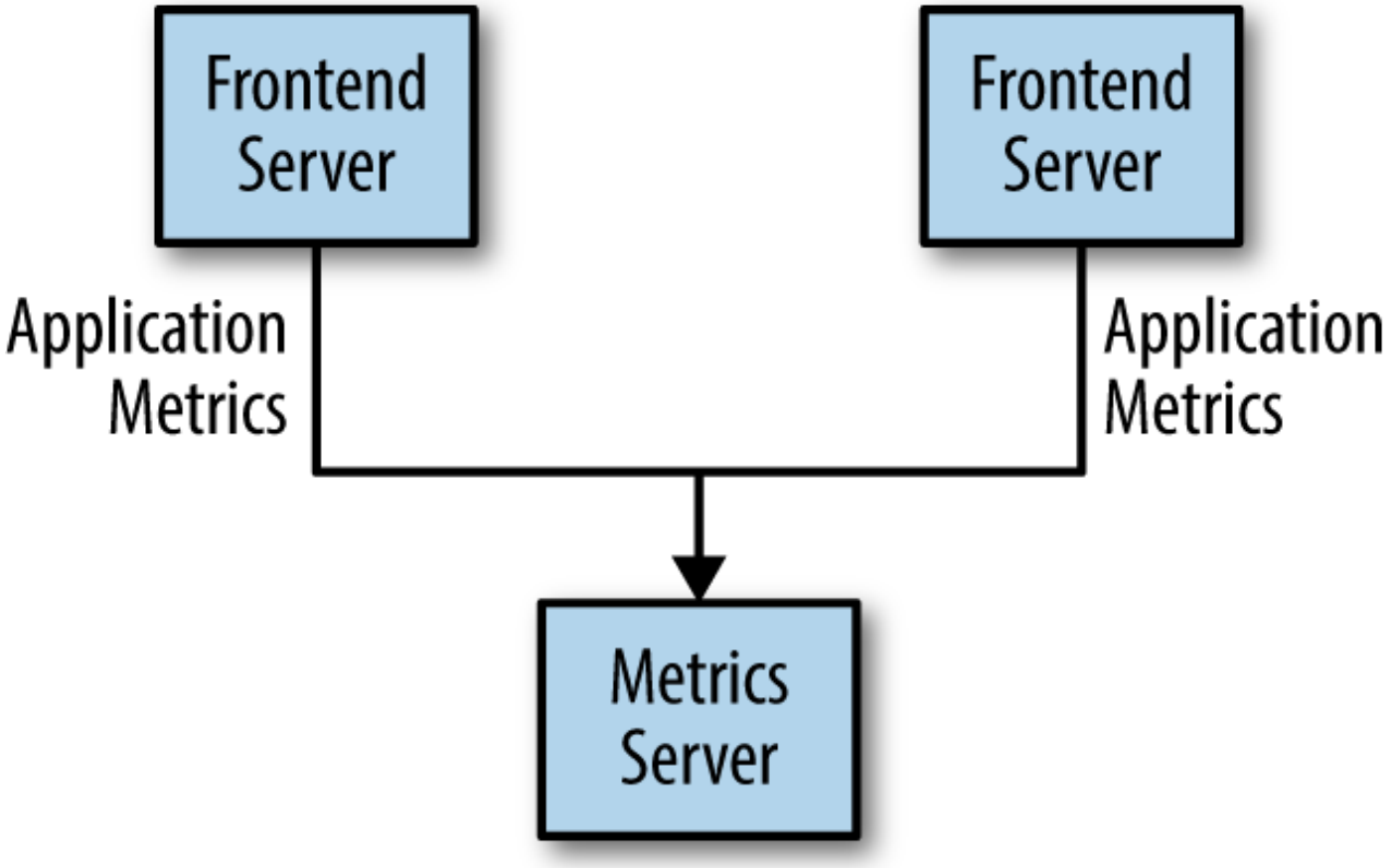
Publish/Subscribe Messaging

- Sebagai gantinya, penerbit (*publisher*) mengklasifikasikan pesan tersebut dengan cara tertentu, dan penerima (*subscriber*) itu berlangganan untuk menerima kelas-kelas pesan tertentu.
- Sistem pub / sub sering memiliki broker, titik pusat di mana pesan diterbitkan

Bagaimana ini bermula?

- Banyak kasus penggunaan publish / subscribe berawal dengan cara yang sama: dengan antrian pesan sederhana atau saluran komunikasi antarproses.
- Misalnya, Anda membuat aplikasi yang perlu mengirim informasi pemantauan ke suatu tempat, jadi Anda menulis koneksi langsung dari aplikasi Anda ke aplikasi yang menampilkan metrik Anda di dashboard, dan mendorong metrik melalui koneksi itu, seperti yang terlihat pada Gambar 1

Bagaimana ini bermula?



Gambar 1. A single, direct metrics publisher

Bagaimana ini bermula?

- Ini adalah solusi sederhana untuk sebuah masalah sederhana yang bekerja saat Anda memulai pemantauan. Tak lama, Anda memutuskan ingin menganalisis metrik Anda untuk jangka waktu yang lebih lama, dan itu tidak berfungsi dengan baik di dasbor. Anda memulai layanan baru yang dapat menerima metrik, menyimpannya, dan menganalisisnya.

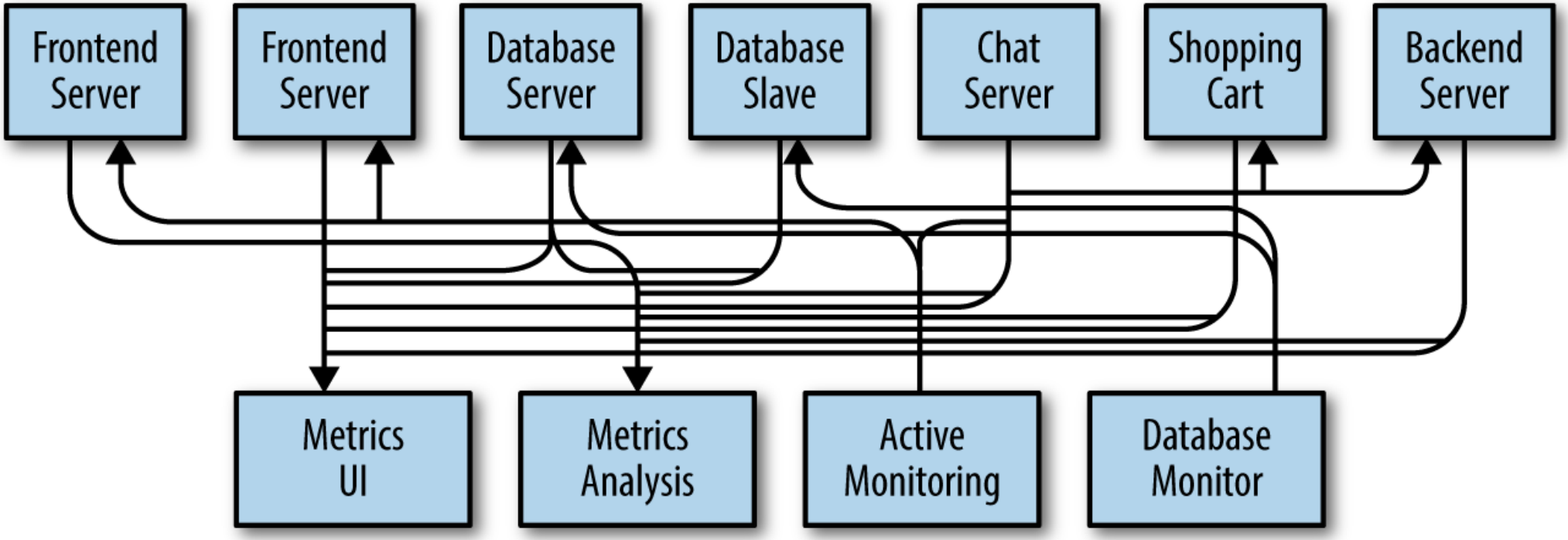
Bagaimana ini bermula?

- Untuk mendukung ini, Anda memodifikasi aplikasi Anda untuk menulis metrik ke kedua sistem. Sekarang Anda memiliki tiga aplikasi lagi yang menghasilkan metrik, dan semuanya membuat koneksi yang sama ke dua layanan ini. Rekan kerja Anda berpikir itu akan menjadi ide yang baik untuk melakukan “jajak pendapat aktif” dari layanan untuk memperingatkan juga, jadi Anda menambahkan server pada masing-masing aplikasi untuk memberikan metrik berdasarkan permintaan.

Bagaimana ini bermula?

- Setelah beberapa saat, Anda memiliki lebih banyak aplikasi yang menggunakan server-server itu untuk mendapatkan metrik individual dan menggunakannya untuk berbagai keperluan. Arsitektur ini dapat terlihat seperti Gambar 2, dengan koneksi yang bahkan lebih sulit untuk dilacak.

Bagaimana ini bermula?

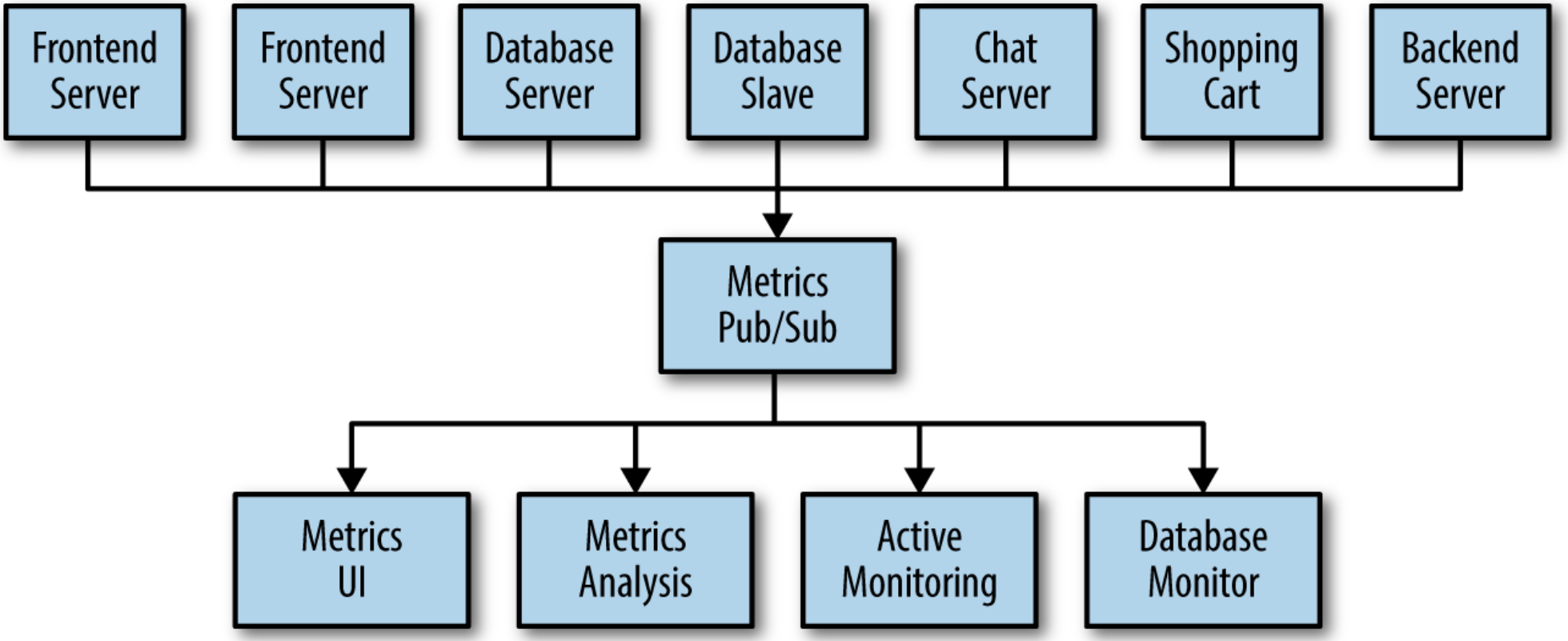


Gambar 2. Many metrics publishers, using direct connections

Bagaimana ini bermula?

- Anda menyiapkan satu aplikasi yang menerima metrik dari semua aplikasi di luar sana, dan menyediakan server untuk menanyakan metrik tersebut untuk sistem apa pun yang membutuhkannya. Ini akan mengurangi kompleksitas , arsitektur menjadi sesuatu yang mirip dengan Gambar 3.
- Disinilah Anda telah membangun sistem perpesanan publish-subscribe!

Bagaimana ini bermula?



Gambar 3. A metrics publish/subscribe system

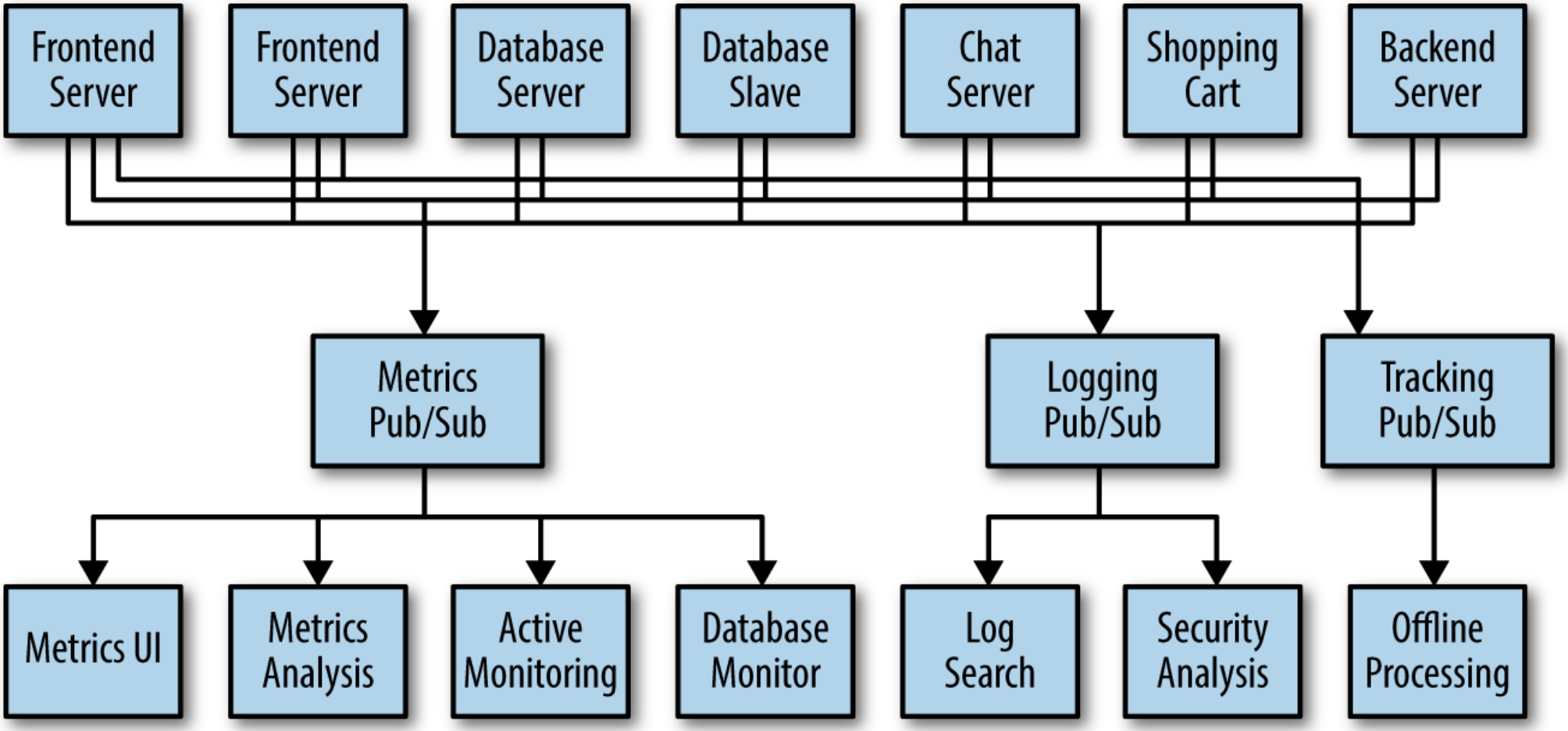
Sistem Antrian Individual (Individual Queue Systems)

- Pada saat yang sama ketika Anda melakukan ‘pertempuran’ dengan metrik-metrik ini, salah satu rekan Anda telah melakukan pekerjaan serupa dengan pesan log. Dan rekan Anda yang lainnya lagi telah bekerja pada pelacakan perilaku pengguna di situs web frontend dan memberikan informasi itu kepada pengembang yang bekerja pada mesin pembelajaran, serta membuat beberapa laporan untuk manajemen.

Sistem Antrian Individual (Individual Queue Systems)

- Dan Anda semua telah mengikuti jalur yang sama untuk membangun sistem yang memisahkan penerbit informasi dari pelanggan ke informasi tersebut.
- Gambar 4 menunjukkan infrastruktur seperti itu, dengan tiga sistem pub / sub yang terpisah.

Sistem Antrian Individual (Individual Queue Systems)



Gambar 4. Multiple publish/subscribe systems

Sistem Antrian Individual (Individual Queue Systems)

- Ini tentunya jauh lebih baik daripada menggunakan koneksi point-to-point (seperti pada Gambar 2), tetapi ada banyak duplikasi. Perusahaan Anda memelihara beberapa sistem untuk antrian data, yang semuanya memiliki bug dan batasannya sendiri. Anda juga tahu bahwa akan ada lebih banyak kasus penggunaan untuk pengiriman pesan segera. Apa yang ingin Anda miliki adalah sistem terpusat tunggal yang memungkinkan untuk menerbitkan jenis data generik, yang akan tumbuh seiring pertumbuhan bisnis Anda.

Apache “Kafka”

- Apache Kafka® adalah sistem perpesanan publish / subscribe yang dirancang untuk mengatasi masalah yang sebelumnya kita bahas.
- Kafka sering digambarkan sebagai “log komit terdistribusi” atau yang lebih baru sebagai “platform streaming terdistribusi.”

Kafka

- Sebuah filesystem atau database log komit dirancang untuk memberikan catatan yang tahan lama dari semua transaksi sehingga mereka dapat diputar ulang untuk secara konsisten membangun keadaan suatu sistem.
- Demikian pula, data dalam Kafka disimpan tahan lama, dalam urutan, dan dapat dibaca secara deterministik. Selain itu, data dapat didistribusikan dalam sistem untuk memberikan perlindungan tambahan terhadap kegagalan, serta peluang signifikan untuk meningkatkan kinerja.

Kafka - Pesan dan Batch

- Unit data dalam Kafka disebut **pesan**. Jika Anda mendekati Kafka dari latar belakang basis data, Anda dapat menganggap ini mirip dengan **baris atau catatan**.
- Pesan hanyalah sebuah array byte sejauh yang menyangkut Kafka, sehingga data yang terkandung di dalamnya tidak memiliki format atau makna khusus untuk Kafka.
- Sebuah pesan dapat memiliki sedikit metadata opsional, yang disebut sebagai kunci.

Kafka - Pesan dan Batch

- Kuncinya juga merupakan array byte dan, seperti halnya pesan, tidak memiliki arti khusus untuk Kafka.
- Kunci digunakan ketika pesan harus ditulis ke partisi dengan cara yang lebih terkontrol. Skema yang paling sederhana adalah menghasilkan hash yang konsisten dari kunci, dan kemudian memilih nomor partisi untuk pesan itu dengan mengambil hasil dari modul hash, jumlah total partisi dalam topik. Ini memastikan bahwa pesan dengan kunci yang sama selalu ditulis ke partisi yang sama.

Kafka - Pesan dan Batch

- Untuk efisiensi, pesan ditulis ke dalam Kafka dalam batch. Batch hanya kumpulan pesan, yang semuanya diproduksi untuk topik dan partisi yang sama.
- Perjalanan bolak-balik individual di seluruh jaringan untuk setiap pesan akan menghasilkan overhead dan mengumpulkan pesan bersama menjadi satu batch akan mengurangi hal overhead. Tentu saja, ini merupakan tradeoff antara latensi dan throughput: semakin besar batch, semakin banyak pesan yang dapat ditangani per unit waktu, tetapi semakin lama pesan individu perlu disebarkan.
- Batch juga biasanya dikompresi, memberikan transfer data dan penyimpanan yang lebih efisien

Kafka - Skema

- Meskipun pesan adalah array byte pada Kafka itu sendiri, direkomendasikan bahwa struktur tambahan, atau skema, dikenakan pada konten pesan sehingga dapat dengan mudah dipahami. Ada banyak opsi yang tersedia untuk skema pesan, tergantung pada kebutuhan individu aplikasi Anda.
- Sistem sederhana, seperti Javascript Object Notation (JSON) dan Extensible Markup Language (XML), mudah digunakan dan dapat dibaca oleh manusia. Namun, mereka tidak memiliki fitur seperti penanganan tipe yang kuat dan kompatibilitas antara versi skema.

Kafka - Skema

- Banyak pengembang Kafka mendukung penggunaan Apache Avro, yang merupakan kerangka kerja serialisasi yang awalnya dikembangkan untuk Hadoop.
- Avro menyediakan format serialisasi yang kompak; skema yang terpisah dari payload pesan dan yang tidak memerlukan pengkodean untuk dihasilkan ketika mereka berubah; dan jenis data yang kuat dan evolusi skema, dengan kompatibilitas mundur dan maju.

Kafka - Skema

- Format data yang konsisten penting dalam Kafka, karena memungkinkan penulisan dan pembacaan pesan dipisahkan. Ketika tugas-tugas ini digabungkan dengan erat, aplikasi yang berlangganan pesan harus diperbarui untuk menangani format data baru, secara paralel dengan format lama. Hanya dengan begitu aplikasi yang mempublikasikan pesan dapat diperbarui untuk memanfaatkan format baru. Dengan menggunakan skema yang terdefinisi dengan baik dan menyimpannya di repositori umum, pesan di Kafka dapat dipahami tanpa koordinasi.

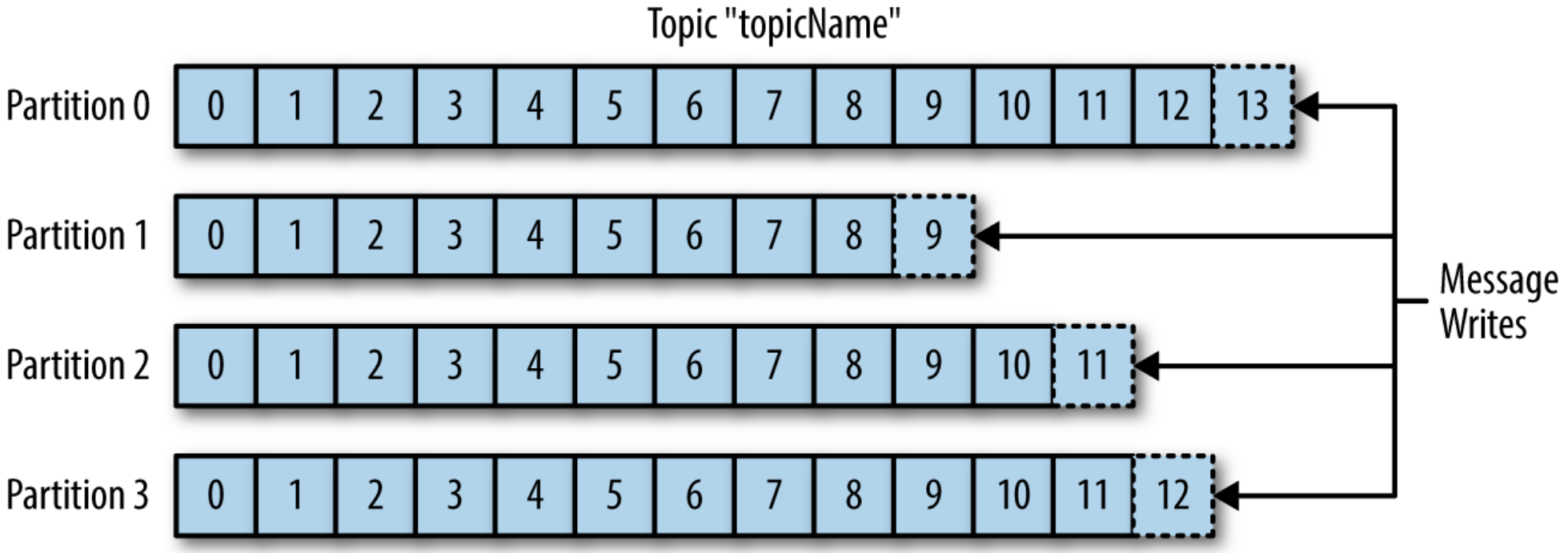
Kafka - Topik dan Partisi

- Pesan-pesan di Kafka dikategorikan ke dalam topik. Analogi terdekat untuk suatu topik adalah tabel basis data atau folder dalam sistem file.
- Topik juga dipecah menjadi beberapa partisi. Kembali ke deskripsi "komit log", partisi adalah log tunggal. Pesan ditulis untuknya dengan cara tambahan saja, dan dibaca secara berurutan dari awal hingga akhir.
- Topik biasanya memiliki beberapa partisi

Kafka - Topik dan Partisi

- Gambar – 5, menunjukkan topik dengan empat partisi, dengan penulisan ditambahkan ke akhir masing-masing.
- Partisi juga merupakan cara Kafka menyediakan redundansi dan skalabilitas.
- Setiap partisi dapat di-host di server yang berbeda, yang berarti bahwa satu topik dapat diskalakan secara horizontal di beberapa server untuk memberikan kinerja yang jauh melampaui kemampuan server tunggal.

Kafka - Topik dan Partisi



Gambar 5. Representasi sebuah topik dengan banyak partisi

Kafka – Produsen dan Konsumen

- Klien Kafka adalah pengguna sistem, dan ada dua tipe dasar: produsen dan konsumen.
- Ada juga API klien tingkat lanjut— Kafka Connect API untuk integrasi data dan Kafka Streams untuk pemrosesan aliran (stream).
- Klien tingkat lanjut menggunakan produsen dan konsumen sebagai blok-blok penyusun dan menyediakan fungsionalitas tingkat tinggi

Kafka – Produsen dan Konsumen

- **Produsen membuat pesan baru**. Dalam sistem publish/subscribe lainnya, ini dapat disebut publishers atau writers.
- Secara umum, pesan akan diproduksi untuk topik tertentu. Secara default, produser tidak peduli ke partisi mana pesan tertentu ditulis dan akan menyeimbangkan pesan di semua partisi topik secara merata.

Kafka – Produsen dan Konsumen

- Dalam beberapa kasus, produsen akan mengarahkan pesan ke partisi tertentu. Ini biasanya dilakukan dengan menggunakan kunci pesan dan partisi yang akan menghasilkan hash kunci dan memetakannya ke partisi tertentu.
- Ini memastikan bahwa semua pesan yang dihasilkan dengan kunci yang diberikan akan ditulis ke partisi yang sama. Produser juga bisa menggunakan partisi kustom yang mengikuti aturan bisnis lain untuk memetakan pesan ke partisi.

Kafka – Produsen dan Konsumen

- **Konsumen membaca pesan**. Dalam sistem publisher/subscriber lainnya, klien ini dapat disebut pelanggan atau pembaca.
- Konsumen berlangganan satu topik atau lebih dan membaca pesan sesuai urutan pembuatannya.

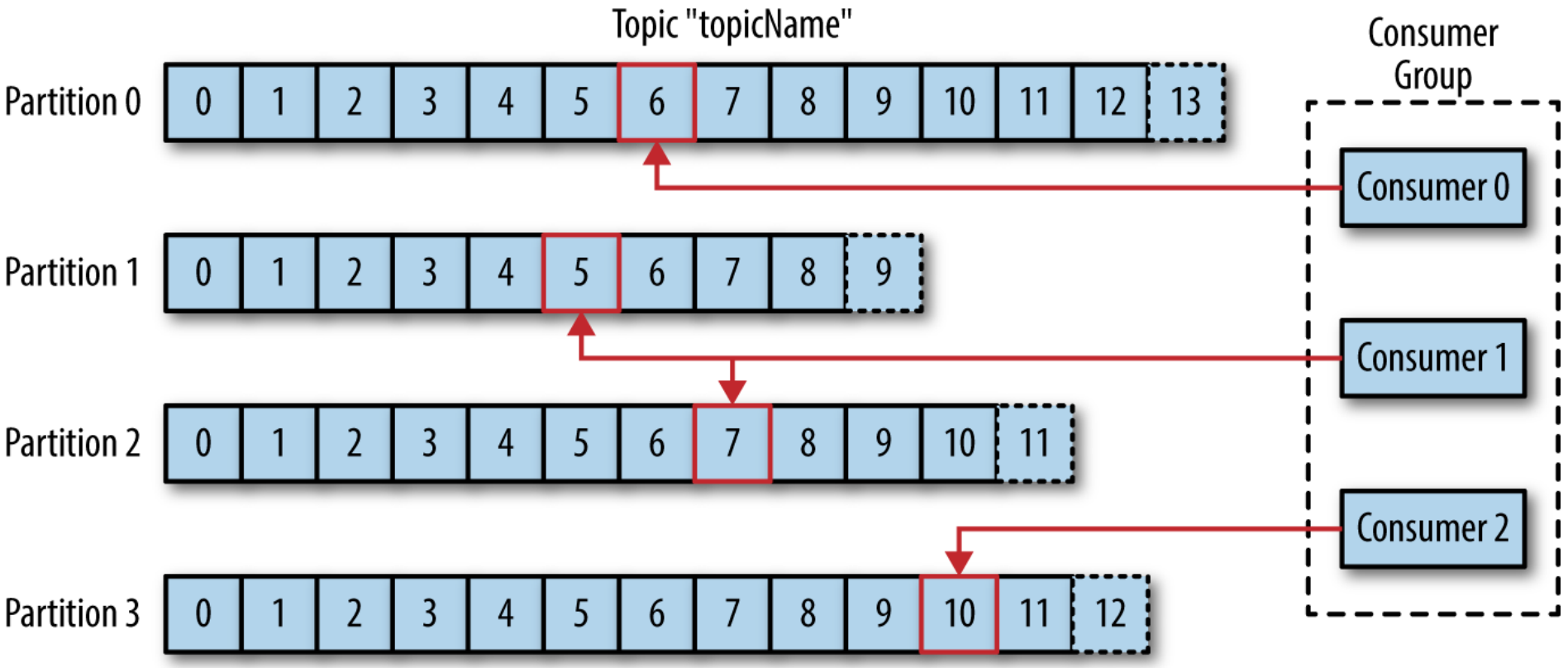
Kafka – Produsen dan Konsumen

- Konsumen melacak pesan mana yang telah dikonsumsi dengan melacak offset pesan.
- Offset adalah sedikit metadata lain — nilai integer yang terus meningkat — yang ditambahkan Kafka ke setiap pesan saat pesan itu diproduksi.
- Setiap pesan dalam partisi yang diberikan memiliki offset unik. Dengan menyimpan offset pesan yang terakhir dikonsumsi untuk setiap partisi, baik di Zookeeper atau di Kafka sendiri, konsumen dapat berhenti dan memulai kembali tanpa kehilangan tempatnya.

Kafka – Produsen dan Konsumen

- Konsumen bekerja sebagai bagian dari kelompok konsumen, yang merupakan satu atau lebih konsumen yang bekerja sama untuk mengkonsumsi suatu topik.
- Grup memastikan bahwa setiap partisi hanya dikonsumsi oleh satu anggota. Pada Gambar 6, ada tiga konsumen dalam satu kelompok yang mengkonsumsi suatu topik. Dua konsumen bekerja dari satu partisi masing-masing, sedangkan konsumen ketiga bekerja dari dua partisi. Pemetaan konsumen ke partisi sering disebut kepemilikan partisi oleh konsumen.

Kafka – Produsen dan Konsumen



Gambar 6. Kelompok konsumen membaca dari suatu topik

Brokers and Clusters

- Server Kafka tunggal disebut broker. Broker menerima pesan dari produsen, memberikan offset kepada mereka, dan melakukan komit pesan untuk penyimpanan pada disk. Ini juga melayani konsumen, merespons untuk mengambil permintaan untuk partisi dan menanggapi dengan pesan yang telah dikomit ke disk.
- Bergantung pada perangkat keras tertentu dan karakteristik kinerjanya, satu broker dapat dengan mudah menangani ribuan partisi dan jutaan pesan per detik.

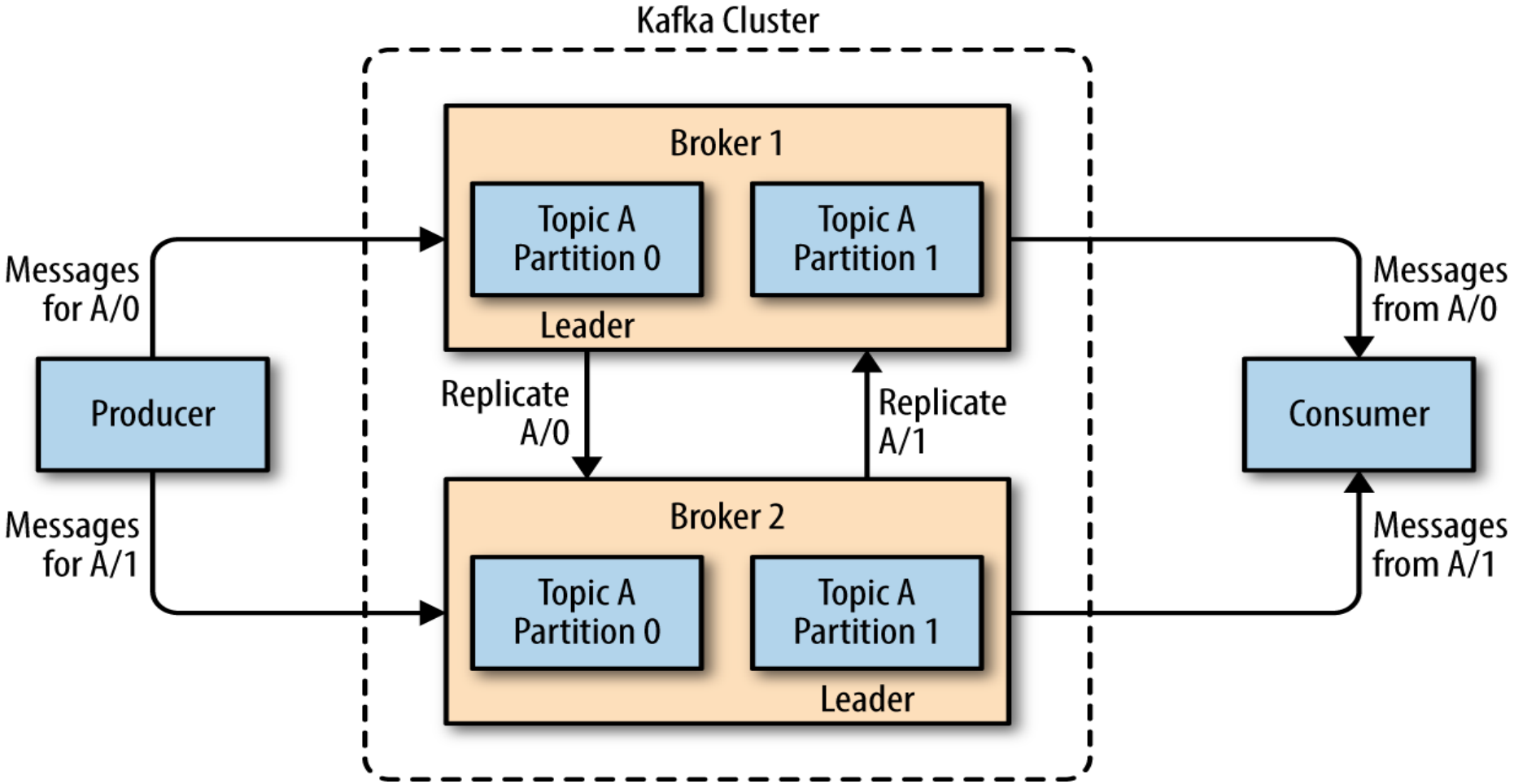
Brokers and Clusters

- Broker Kafka dirancang untuk beroperasi sebagai bagian dari cluster. Dalam sebuah cluster broker, satu broker juga akan berfungsi sebagai pengontrol cluster (dipilih secara otomatis dari anggota cluster yang hidup).
- Pengontrol bertanggung jawab untuk operasi administrasi, termasuk menugaskan partisi untuk broker dan memantau kegagalan broker.

Brokers and Clusters

- Partisi dimiliki oleh satu broker di cluster, dan broker itu disebut pemimpin partisi. Partisi dapat ditugaskan ke beberapa broker, yang akan menghasilkan partisi yang direplikasi (seperti yang terlihat pada Gambar 7). Ini memberikan redundansi pesan di partisi, sehingga broker lain dapat mengambil alih kepemimpinan jika ada kegagalan broker. Namun, semua konsumen dan produsen yang beroperasi pada partisi itu harus terhubung ke pemimpin.

Brokers and Clusters



Gambar 7. Replikasi partisi dalam sebuah cluster

Brokers and Clusters

- Fitur utama dari Apache Kafka adalah retensi, yang merupakan penyimpanan pesan yang tahan lama untuk beberapa periode waktu.
- Broker Kafka dikonfigurasikan dengan pengaturan penyimpanan default untuk topik, baik menyimpan pesan untuk jangka waktu tertentu (mis., 7 hari) atau hingga topik mencapai ukuran tertentu dalam byte (mis., 1 GB). Setelah batas ini tercapai, pesan kedaluwarsa dan dihapus sehingga konfigurasi penyimpanan adalah jumlah minimum data yang tersedia setiap saat.

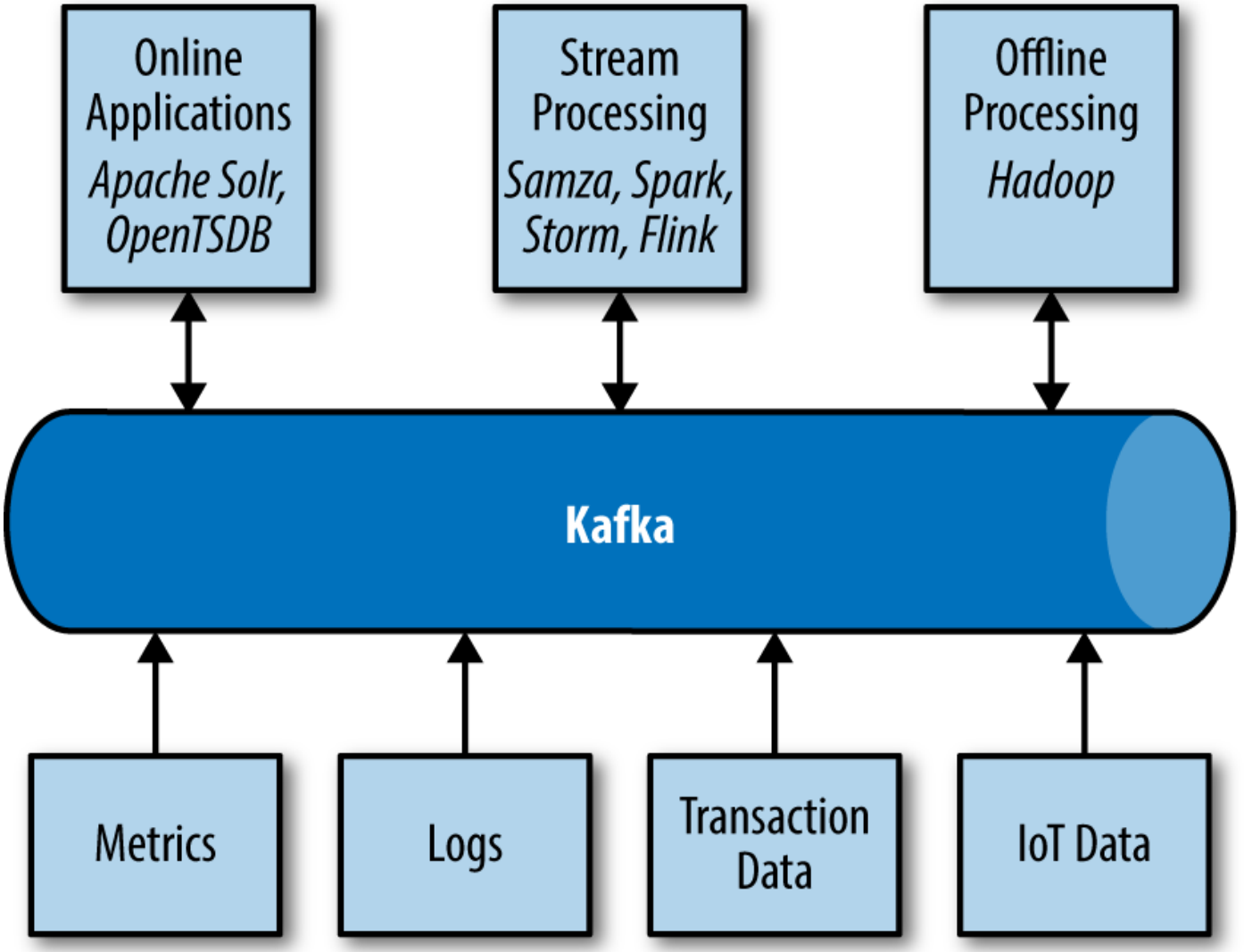
Brokers and Clusters

- Topik individual juga dapat dikonfigurasikan dengan pengaturan penyimpanannya sendiri sehingga pesan disimpan hanya selama berguna. Misalnya, topik pelacakan mungkin dipertahankan selama beberapa hari, sedangkan metrik aplikasi mungkin dipertahankan hanya beberapa jam.
- Topik juga dapat dikonfigurasikan sebagai log yang dipadatkan, yang berarti bahwa Kafka hanya akan mempertahankan pesan terakhir yang dihasilkan dengan kunci tertentu. Ini bisa berguna untuk data tipe changelog, di mana hanya pembaruan terakhir yang menarik.

Kenapa Kafka?

- Multiple Producers
- Multiple Consumers
- Disk-Based Retention
- Scalable
- High Performance

Kenapa Kafka?



Gambar 8. A big data ecosystem

Use Cases

- Activity tracking
- Messaging
- Metrics and Logging
- Commit Log
- Stream processing

Instalasi Kafka

- Sebelum memasang Kafka, Anda membutuhkan lingkungan Java yang diatur dan berfungsi. Ini harus versi Java 8 atau lebih besar, dan bisa versi yang disediakan oleh OS Anda atau yang diunduh langsung dari java.com.
- Kafka akan bekerja dengan edisi runtime Java, mungkin lebih nyaman saat mengembangkan alat dan aplikasi untuk memiliki Java Development Kit (JDK) lengkap.

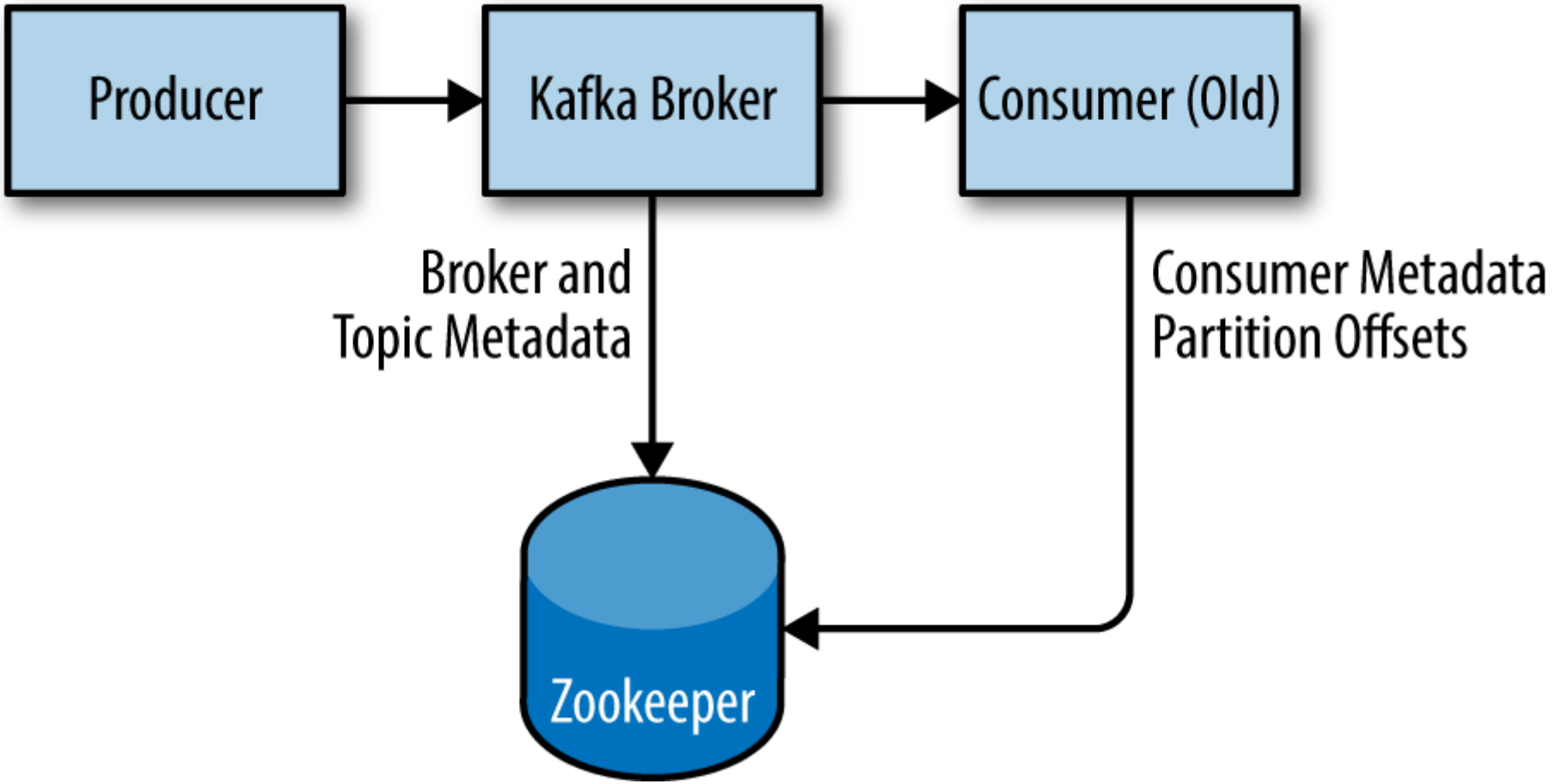
Instalasi Kafka

- Instalasi java pada ubuntu:

```
#sudo apt-get install openjdk-*
```

- Download & install zookeeper.
  - Apache Kafka menggunakan Zookeeper untuk menyimpan metadata tentang cluster Kafka, serta detail klien konsumen, seperti yang ditunjukkan pada Gambar 9.

Instalasi Kafka



Gambar 9. Kafka dan Zookeeper

Instalasi Kafka

- Download binary zookeeper terkini dari url https://downloads.apache.org/zookeeper/zookeeper-3.6.1/apache-zookeeper-3.6.1-bin.tar.gz
- Ekstrak zoookeper:

  # tar -xzvf apache-zookeeper-3.6.1-bin.tar.gz
- Copy sample config zookeeper menjadi konfigurasi zookeeper yang akan digunakan.

  # cd apache-zookeeper-3.6.1-bin

  # cp conf/zoo_sample.cfg conf/zoo.cfg

Instalasi kafka

- Aktifkan zookeper:

  # cd ./bin

  # ./zkServer.sh start
- Periksa status:

  # ./zkServer.sh status

Instalasi kafka

- Download kafka dari url https://downloads.apache.org/kafka/2.5.0/kafka_2.12-2.5.0.tgz
- Ekstrak kafka.

```
# tar xzvf kafka_2.12-2.5.0.tgz
```

- Menjalankan broker kafka.

```
#cd kafka_2.12-2.5.0

# bin/kafka-server-start.sh
config/server.properties
```

Membuat topik

- Pesan diterbitkan dalam topik. Gunakan perintah ini untuk membuat topik baru.

```
# bin/kafka-topics.sh --create --zookeeper
localhost:2181 --replication-factor 1 --
partitions 1 --topic TEST
```

- Anda juga dapat menampilkan daftar semua topik yang tersedia dengan menjalankan perintah berikut.

```
# bin/kafka-topics.sh --list --zookeeper
localhost:2181
```

Mengirim pesan

- Selanjutnya, kita akan mengirim pesan, produser digunakan untuk tujuan ini. Mari kita mulai produser.

```
# bin/kafka-console-producer.sh --
broker-list localhost:9092 --topic
TEST

>Hello World

>Halo Dunia
```

Mengkonsumsi Pesan

- Pesan yang disimpan juga harus dikonsumsi. Untuk itu diperlukan konsumen. Untuk memulai konsumen dan membaca pesan dapat Anda lakukan denga menjalankan perintah berikut:

```
# bin/kafka-console-consumer.sh --
bootstrap-server localhost:9092 --topic
TEST –from-beginning
```

--from-beginning akan menyebabkan pesan diambil semua dari awal. Jika Anda hanya tertarik pada pesan saat menjalankan konsumer maka hilangkan opsi --from-beginning