DYALOG

Elsinore 2023

Something about Kafka

Stefan Krüger

DYALOG

Elsinore 2023

Something about Kafka

Stefan Krüger (rookie)

DYALOG

Elsinore 2023

Something about Kafka

Stefan Krüger (the Mac guy)

DYALOG

Elsinore 2023

Something about Kafka

Stefan Krüger (computer person)

DYALOG

Elsinore 2023

Something about Kafka

Stefan Krüger (docks windows)

kafka

Disclaimer: a distinct lack of APL

What's Kafka?

What's Kafka got to do with APL?

“

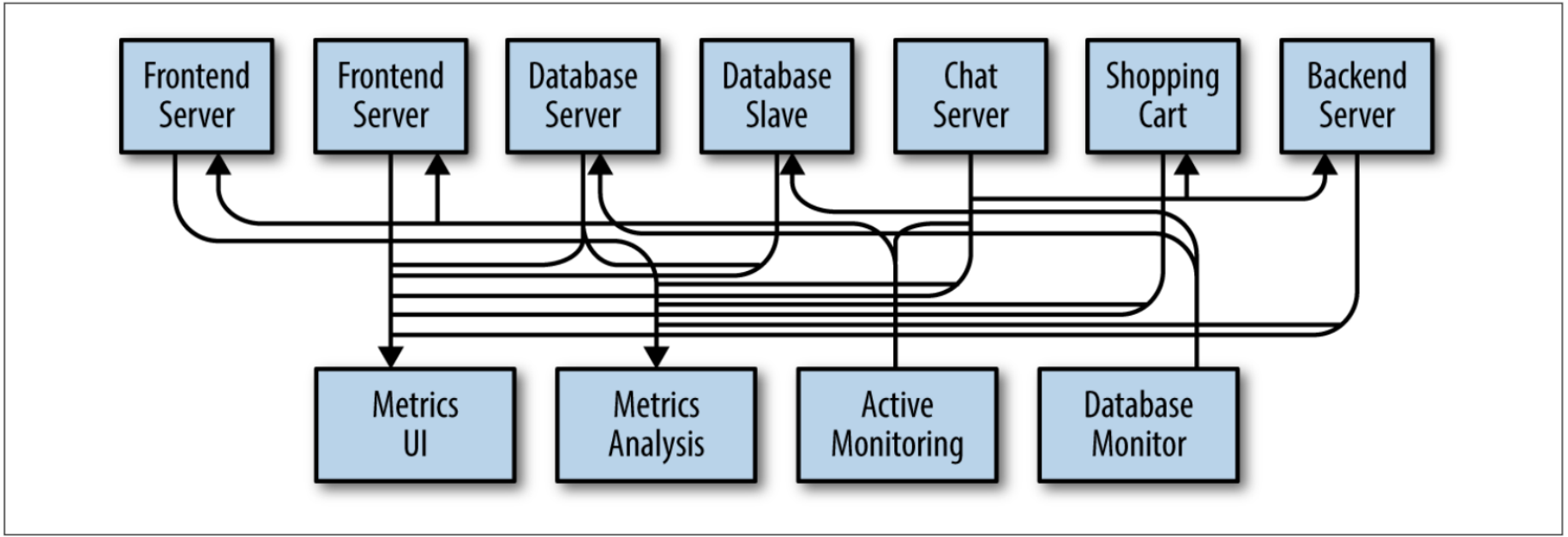
Apache Kafka is a distributed event store and stream-processing platform. It provides a unified, high-throughput, low-latency platform for handling real-time data feeds.

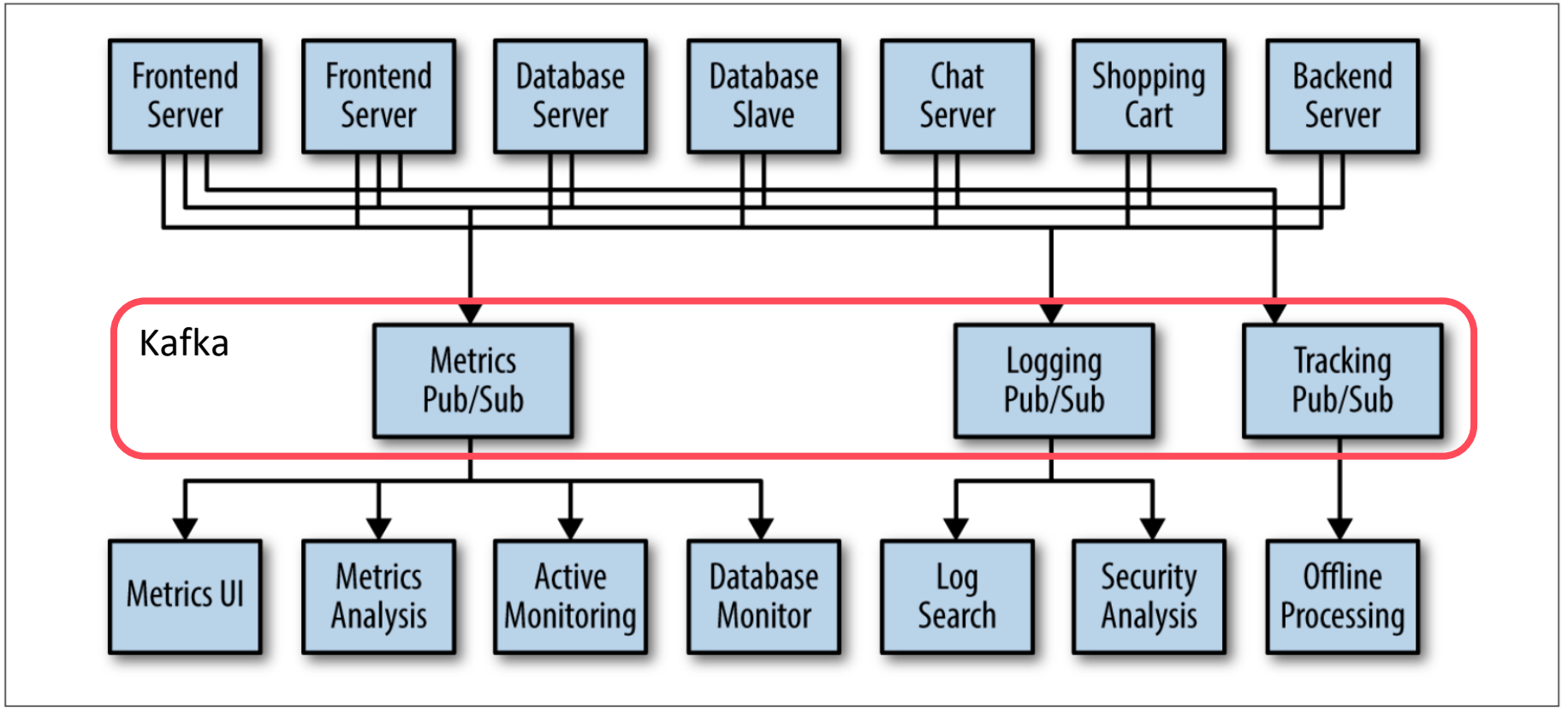
not just

It's a message queue

pub-sub

It's a distributed streaming platform


Frontend Server
Frontend Server
Database Server
Database Slave
Chat Server
Shopping Cart
Backend Server
Metrics UI
Metrics Analysis
Active Monitoring
Database Monitor


Frontend Server
Frontend Server
Database Server
Database Slave
Chat Server
Shopping Cart
Backend Server
Kafka
Metrics Pub/Sub
Logging Pub/Sub
Tracking Pub/Sub
Metrics UI
Metrics Analysis
Active Monitoring
Database Monitor
Log Search
Security Analysis
Offline Processing


amazon
web services
IBM Cloud
Microsoft
Azure
Google Cloud

Photo by Aedrian on Unsplash

Kafka Terminology

- `message` - fundamental unit of data ("row")
- `topic` - stream name ("table"). Routes `messages` from `producers` to `consumers`
- `producer/consumer` - topic writer/reader
- `partition` - fundamental unit of parallelism ("shard")
- `retention` - configurable per topic

Kafka Terminology

- `message` - fundamental unit of data ("row")
- `topic` - stream name ("table"). Routes `messages` from `producers` to `consumers`
- `producer/consumer` - topic writer/reader
- `partition` - fundamental unit of parallelism ("shard")
- `retention` - configurable per topic

Kafka Terminology

- `message` - fundamental unit of data ("row")
- `topic` - stream name ("table"). Routes `messages` from `producers` to `consumers`
- `producer/consumer` - topic writer/reader
- `partition` - fundamental unit of parallelism ("shard")
- `retention` - configurable per topic

Kafka Terminology

- `message` - fundamental unit of data ("row")
- `topic` - stream name ("table"). Routes `messages` from `producers` to `consumers`
- `producer/consumer` - topic writer/reader
- `partition` - fundamental unit of parallelism ("shard")
- `retention` - configurable per topic

Kafka Terminology

- `message` - fundamental unit of data ("row")
- `topic` - stream name ("table"). Routes `messages` from `producers` to `consumers`
- `producer/consumer` - topic writer/reader
- `partition` - fundamental unit of parallelism ("shard")
- `retention` - configurable per topic

Kafka guarantees message ordering

Kafka topic

Consumer group

Producer

Partitioned topic

Running Kafka

```
% docker compose up -d
[+] Running 3/3
 ✓ Network kafka_default  Created                    0.0s
 ✓ Container zookeeper    Started                    0.2s
 ✓ Container broker       Started                    0.3s

% kafka-topics --create \
    --topic dyalog \
    --partitions 1 \
    --replication-factor 1 \
    --bootstrap-server localhost:9092

Created topic dyalog.
```

CONFLUENT

Stream Catalog | LEARN

- Home
- Environments
- Cluster links
- Stream shares

Welcome back to Confluent Cloud!

View environments

Your Confluent overview

Summary of your existing resources

Environments	Clusters	Topics	Partitions	ksqlDB	Connectors	Links
1	1	1	6	0	0	0

There's no payment information for your account

Free trial — 27 days left

Remaining free credit — $400/$400

Enter payment to avoid disruptions Recommended

- ✓ You won't be charged until your free trial is over
- ✓ Avoid service interruptions when your free trial ends
- ✓ Cancel or change services anytime

Update payment info

Dyalog + Kafka = True?

Roll yer own

librdkafka (C++)

Confluent.Kafka (C#)

DKaf.NET

- C# ideally placed for prototyping this
- Performant - still wraps the native library
- Disadvantage: no AIX
- Disadvantage: 'modern' C# (generics) means we can't quite use Confluent.Kafka directly

DKaf.NET

- C# ideally placed for prototyping this
- Performant - still wraps the native library
- Disadvantage: no AIX
- Disadvantage: 'modern' C# (generics) means we can't quite use Confluent.Kafka directly

DKaf.NET

- C# ideally placed for prototyping this
- Performant - still wraps the native library
- Disadvantage: no AIX
- Disadvantage: 'modern' C# (generics) means we can't quite use Confluent.Kafka directly

DKaf.NET

- C# ideally placed for prototyping this
- Performant - still wraps the native library
- Disadvantage: no AIX
- Disadvantage: 'modern' C# (generics) means we can't quite use Confluent.Kafka directly

The problem with generics

```
using (var prod = new ProducerBuilder<string, string>(_config).Build()) {
    for (int i=0; i<10; ++i) {
        prod.Produce(topic, new Message<string, string> { Key = ..., Value = ... },
            (deliveryReport) => {
                if (deliveryReport.Error.Code != ErrorCode.NoError) {
                    Console.WriteLine($"{deliveryReport.Error.Reason}");
                } else {
                    // Message delivered
```

```
∇ Ctor (type arg)
  :Access public
  :Implements constructor
  ⍝ arg is either a charvec or a Confluent.Kafka.ProducerConfig.
  :If type=0
    producer ← ⎕NEW DyKa.StringProducer(⊂arg)
  :Else
    producer ← ⎕NEW DyKa.Producer(⊂arg)
  :End
∇
```

Type selector

Concrete types

```
public class Producer : IDisposable {
    public IProducer<string, byte[]>? Prod { get; private set; }
    private ProducerConfig _config;

    public Producer(string bootstrapServers) {
        _config = new ProducerConfig {
            BootstrapServers = bootstrapServers
        };

        Prod = new ProducerBuilder<string, byte[]>(_config).Build();
    }
```

DKaf.NET

- Write thin C# classes that instantiate a set message signature
- Message key is UTF8 string, value is UTF8 string or byte[]

DKaf.NET

- Write thin C# classes (Producer, Consumer) that instantiate a set message signature
- Message key is UTF8 string, value is UTF8 string or byte[]

```
      ]link.create # /Users/stefan/work/DKaf
Linked: # ↔ /Users/stefan/work/DKaf

      p ← ⎕NEW Producer (⍬'localhost:9092') ⍝ String producer
      p.Produce('dyalog' 'key1' 'This is a string')
      p.Produce('dyalog' 'key2' 'Hello Kafka from Dyalog APL')
```

Type selector

consumer mode

```
% kcat -Ct dyalog -b localhost:9092
This is a string
Hello Kafka from Dyalog APL
% Reached end of topic dyalog [0] at offset 2
```

topic

```
% kcat -Ct dyalog -b localhost:9092
This is a string
Hello Kafka from Dyalog APL
% Reached end of topic dyalog [0] at offset 2
```

Kafka broker

```
% kcat -Ct dyalog -b localhost:9092
This is a string
Hello Kafka from Dyalog APL
% Reached end of topic dyalog [0] at offset 2
```

Demo time

Summary

- Kafka -- your data backbone: fast, scalable, robust, durable
- Dyalog in the cloud -- talk to Kafka, talk to anything
- DKaf: Kafka via .NET

Summary

- Kafka -- your data backbone: fast, scalable, robust, durable
- Dyalog in the cloud -- talk to Kafka, talk to anything
- DKaf: Kafka via .NET

Summary

- Kafka -- your data backbone: fast, scalable, robust, durable
- Dyalog in the cloud -- talk to Kafka, talk to anything
- DKaf: Kafka via .NET