Aviso de Propriedade e Uso Educacional de Material em PDF

Prezados Alunos(as),

Gostaríamos de reforçar que todo o material em PDF disponibilizado nesta plataforma é de propriedade exclusiva da DIO, destinando-se unicamente a fins educacionais. Este conteúdo é compartilhado com vocês com o intuito de enriquecer os seus conhecimentos e apoiar seus estudos.

Ressaltamos a importância de utilizar este material de acordo com o objetivo central, que é o aprendizado e a formação educacional. Qualquer compartilhamento indevido por parte de terceiros, que não estejam alinhados com este propósito, ou em outras plataformas que não sejam as oficialmente designadas pela DIO, é estritamente proibido e considerado uma violação dos direitos autorais da empresa.

A proteção dos direitos autorais é essencial para manter a integridade do conteúdo educacional e garantir que ele continue acessível a todos os alunos. Portanto, pedimos que este aviso seja levado a sério e que todos os usuários desta plataforma ajam com responsabilidade e ética ao utilizar o material compartilhado.

Agradecemos pela vossa compreensão e colaboração em manter a integridade do nosso ambiente educacional.

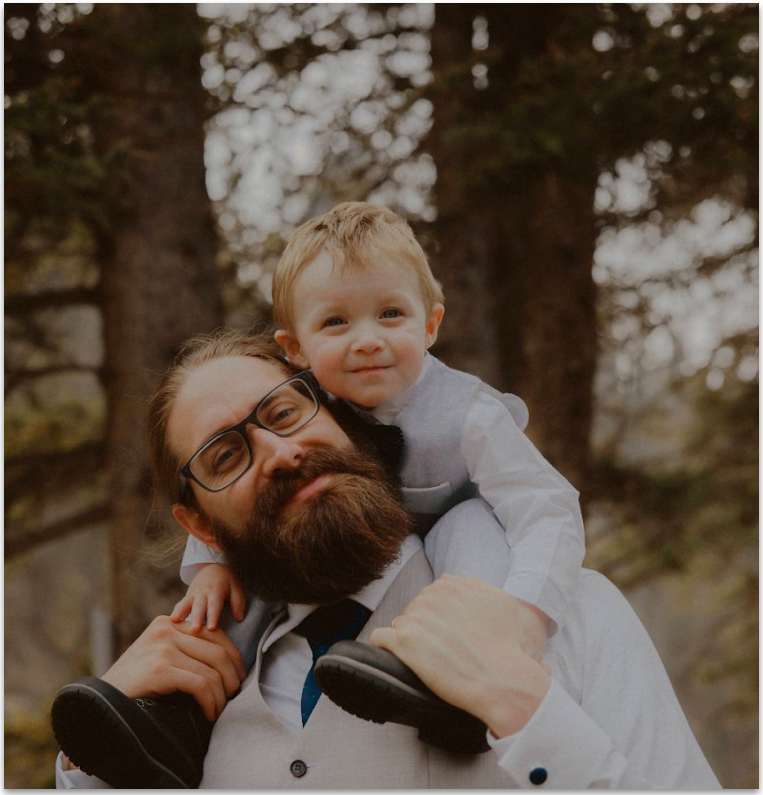
Atenciosamente,

Time de Educação da DIO.

Event-Driven Architecture Fundamentals

Kafka Integration in Spring Boot

Thiago Poiani
Tech Lead at SkipTheDishes 🇨🇦
@thpoiani



VanguardaBrasil

monitora
Innovation. Software. Simplicity.

LadyDriver

SKIP

INSTITUTO FEDERAL
São Paulo

ufscar

USP
Universidade de São Paulo

ORACLE
Certified
Professional
Java SE 11 Developer
2019

aws
certified
Developer
ASSOCIATE

slido

https://app.sli.do

#DIO-EDA

Agenda

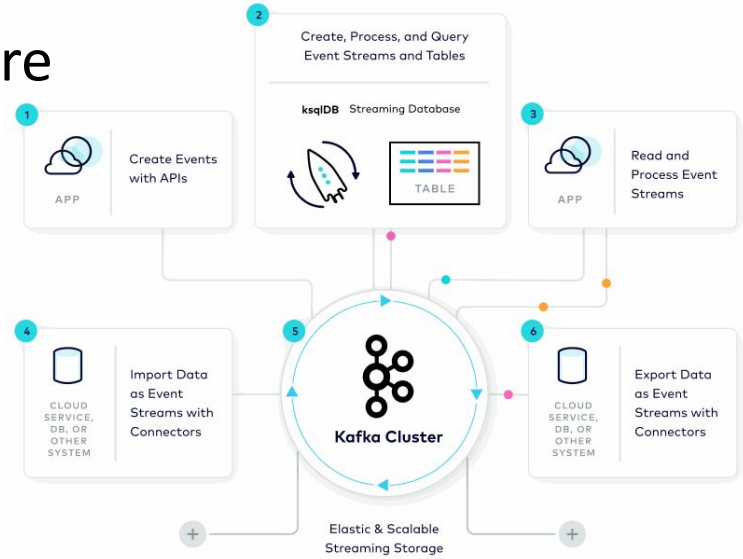
Foundations of Event-Driven Architecture

Kafka and Confluent Cloud

Unleashing the Power of Event Streaming

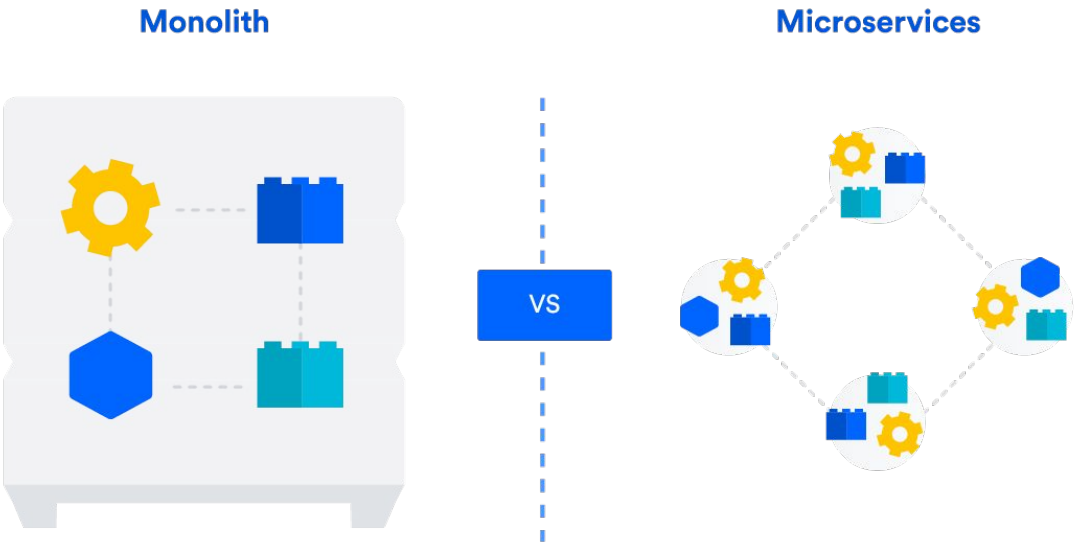
Hands-On

Building Event-Driven Spring Boot Applications



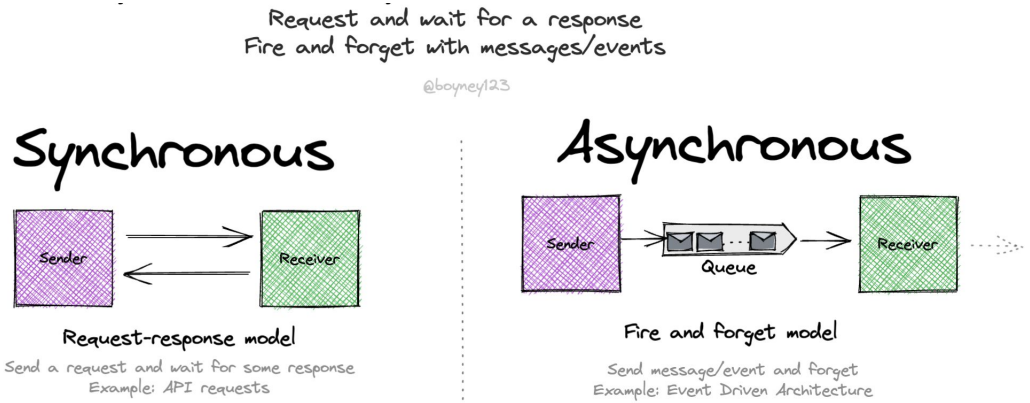
Foundations of Event-Driven Architecture

Communication



https://www.atlassian.com/microservices/microservices-architecture/microservices-vs-monolith

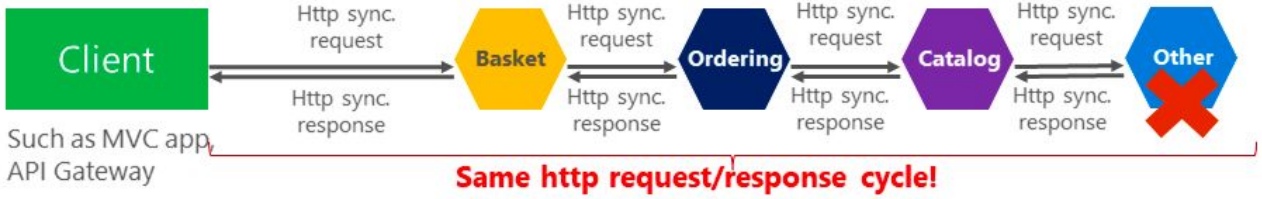
Sync vs Async Protocol



https://serverlessland.com/event-driven-architecture/visuals/sync-vs-async

Sync vs Async Communication

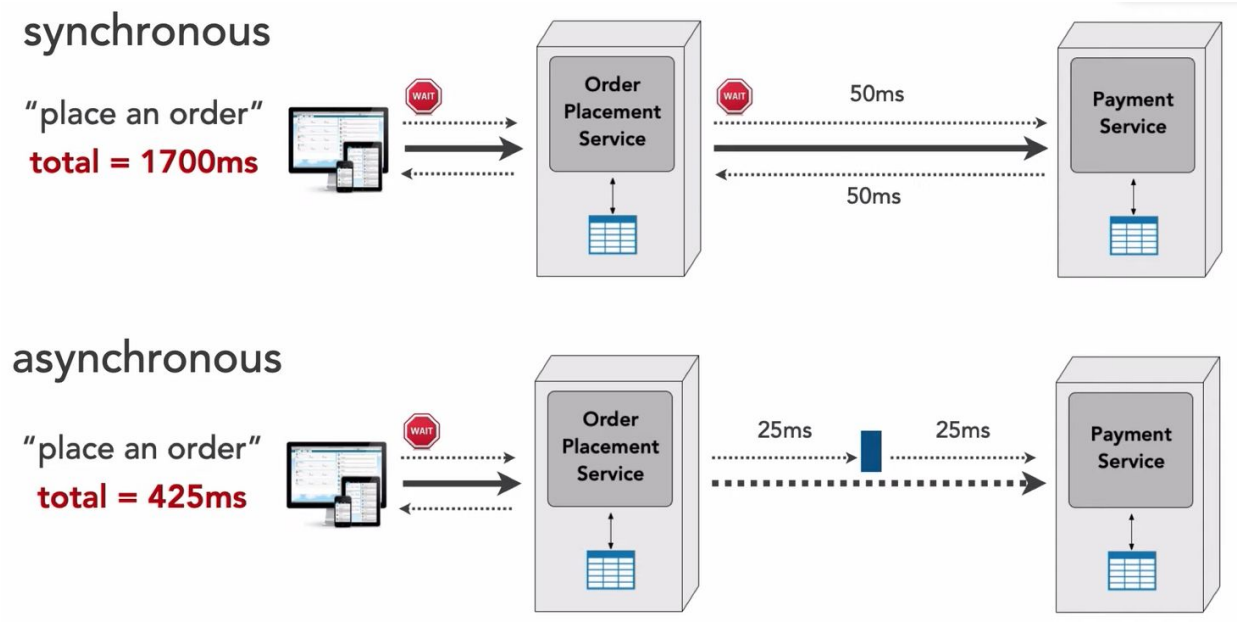
Synchronous
all request/response cycle



Asynchronous
Comm. across internal microservices
(EventBus: like **AMQP**)

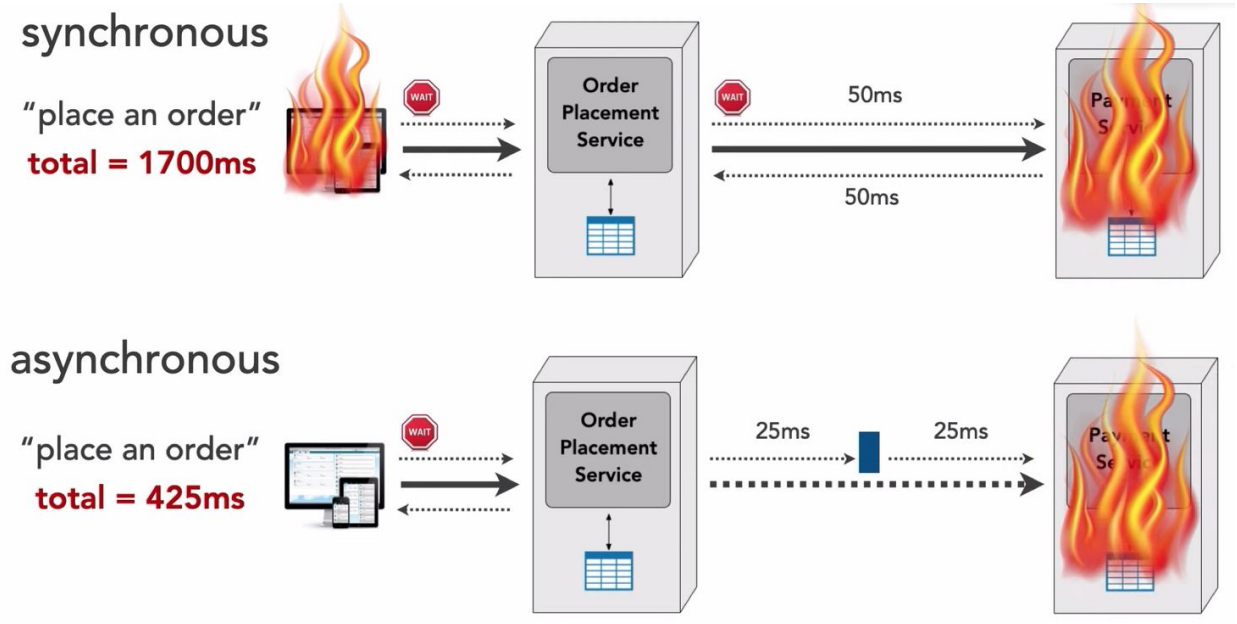
https://learn.microsoft.com/en-us/dotnet/architecture/microservices/architect-microservice-container-applications/communication-in-microservice-architecture

REST vs Messaging: Trade-offs



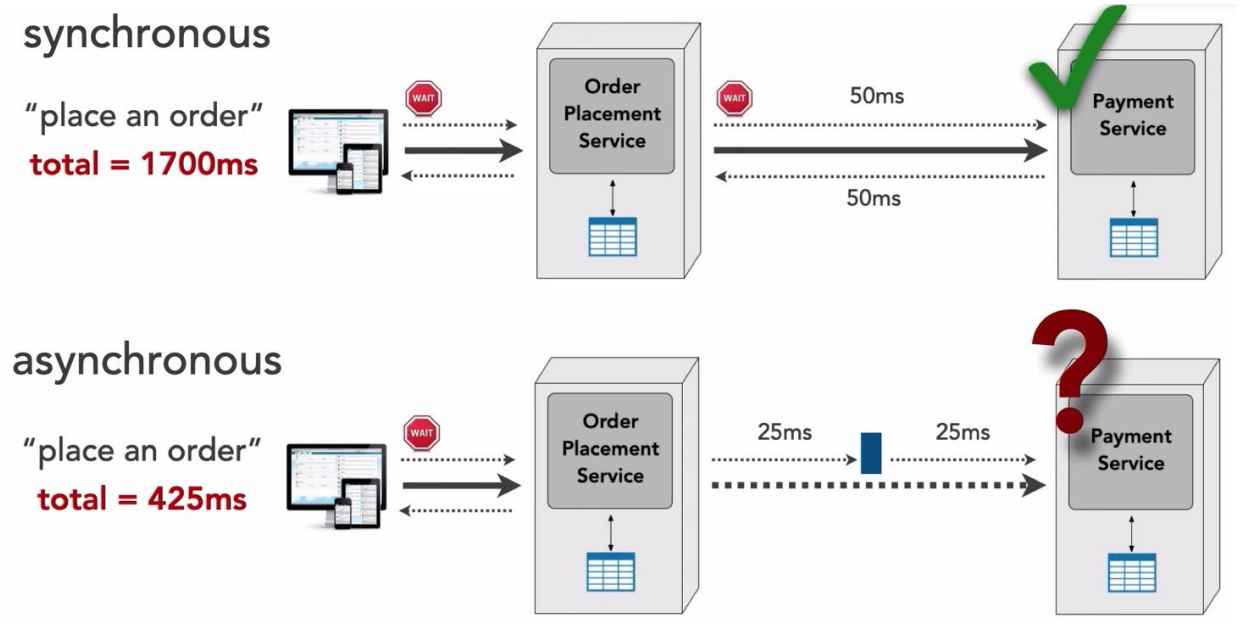
https://developertoarchitect.com/lessons/lesson137.html

REST vs Messaging: Trade-offs



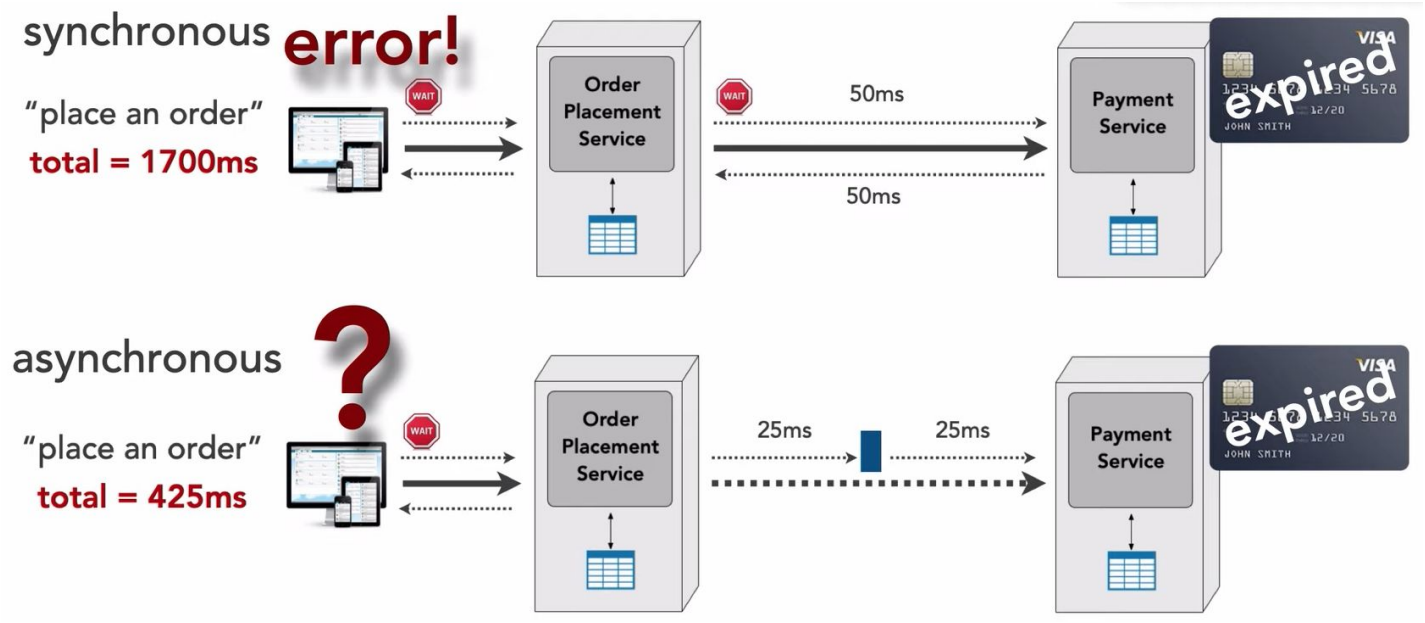
https://developertoarchitect.com/lessons/lesson137.html

REST vs Messaging: Trade-offs



https://developertoarchitect.com/lessons/lesson137.html

REST vs Messaging: Trade-offs



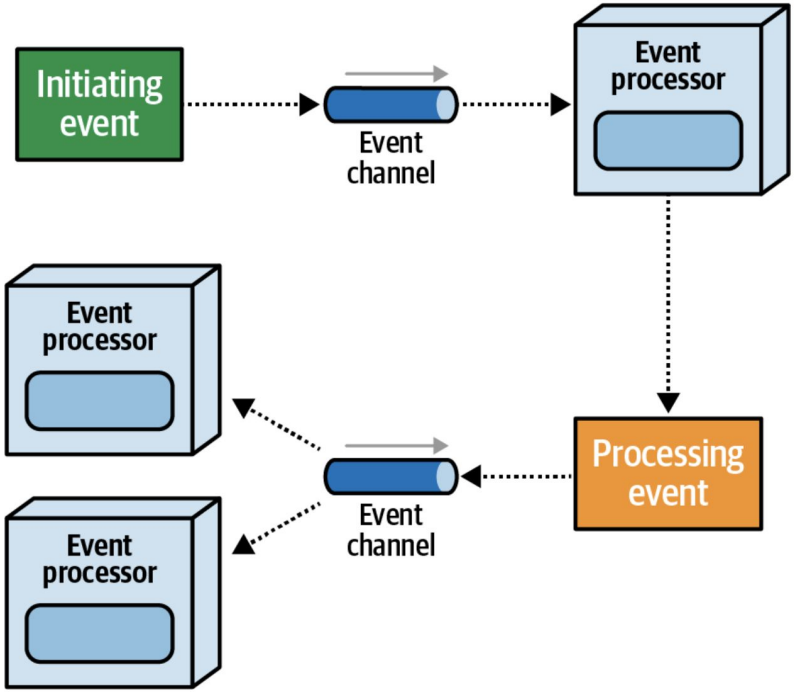
https://developertoarchitect.com/lessons/lesson137.html

Event-Driven Architecture

Broker topology: message flow distributed across services in a chain-like broadcasting

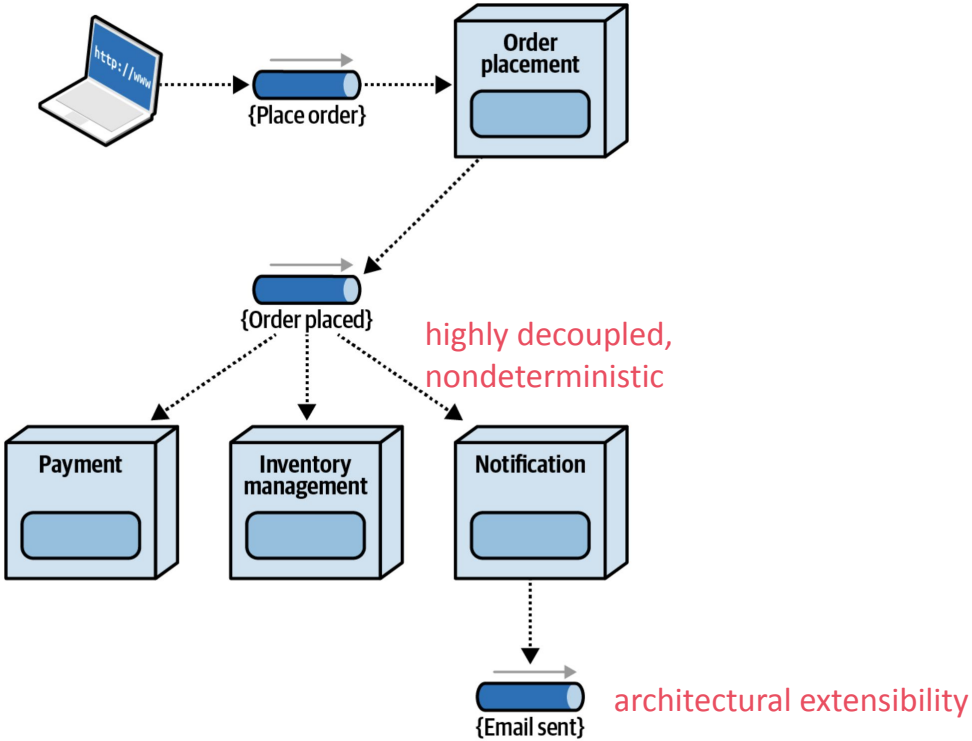
Mediator topology: event mediator manages and controls the workflow for initiating events that require coordination

Event-Driven Architecture: Broker



Mark Richards. Software Architecture Patterns. O'Reilly Media, Inc.

Event-Driven Architecture: Broker



Mark Richards. Software Architecture Patterns. O'Reilly Media, Inc.

When to Consider

- High performance, decoupling, scalability and fault tolerance

- System is reacting to user-initiated events, rather than responding to user requests

- Complex, nondeterministic workflows that are challenging to model

When Not to Consider

- Most of the system is request-based

- Synchronous processing is business crucial, and users must wait for processing to be completed for specific requests

- High levels of data consistency are required

  Because all processing is eventually consistent in EDA

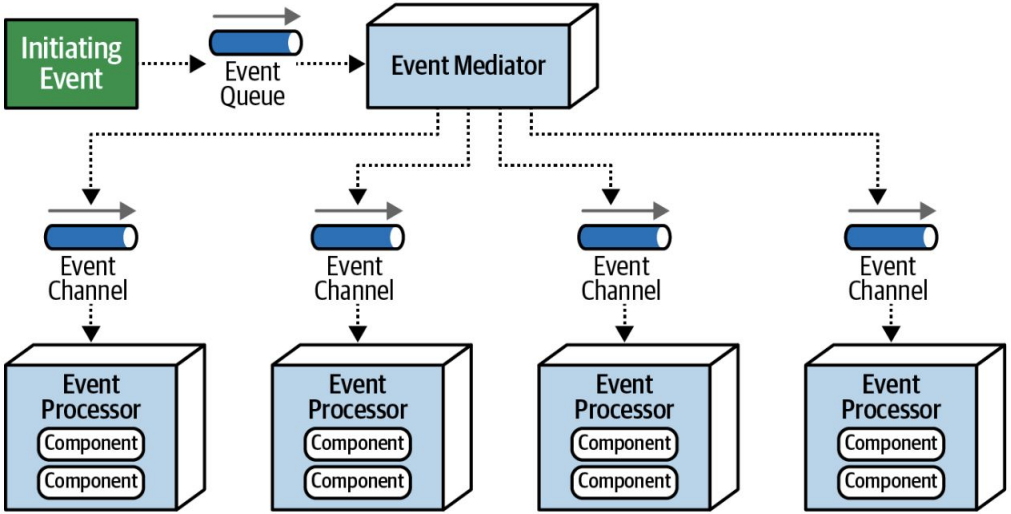
- Precise control over the workflow and timing of events is necessary (coordination)

- Error handling, complex error scenarios

	layered	modular monolith	microkernel	microservices	service-based	service-oriented	event-driven	space-based
agility	★	★★	★★★	★★★★★	★★★★	★	★★★	★★
abstraction	★	★	★★★	★	★	★★★★★	★★★★	★
configurability	★	★	★★★★	★★★	★★	★	★★	★★
cost	★★★★★	★★★★★	★★★★★	★	★★★★	★	★★★	★★
deployability	★	★★	★★★	★★★★★	★★★★	★	★★★	★★★
domain part.	★	★★★★★	★★★★★	★★★★★	★★★★★	★	★	★★★★★
elasticity	★	★	★	★★★★★	★★	★★★	★★★★	★★★★★
evolvability	★	★	★★★	★★★★★	★★★	★	★★★★★	★★★
fault-tolerance	★	★	★	★★★★★	★★★★	★★★	★★★★★	★★★
integration	★	★	★★★	★★★	★★	★★★★★	★★★	★★
interoperability	★	★	★★★	★★★	★★	★★★★★	★★★	★★
performance	★★★	★★★	★★★	★★	★★★	★★	★★★★★	★★★★★
scalability	★	★	★	★★★★★	★★★	★★★	★★★★★	★★★★★
simplicity	★★★★★	★★★★★	★★★★	★	★★★	★	★	★
testability	★★	★★	★★★	★★★★★	★★★★	★	★★	★
workflow	★	★	★★	★	★	★★★★★	★★★★★	★

https://www.developertoarchitect.com/downloads/architecture-styles-worksheet.pdf

Event-Driven Architecture: Mediator



Mark Richards and Neal Ford. Software Architecture Fundamentals. O'Reilly Media, Inc.

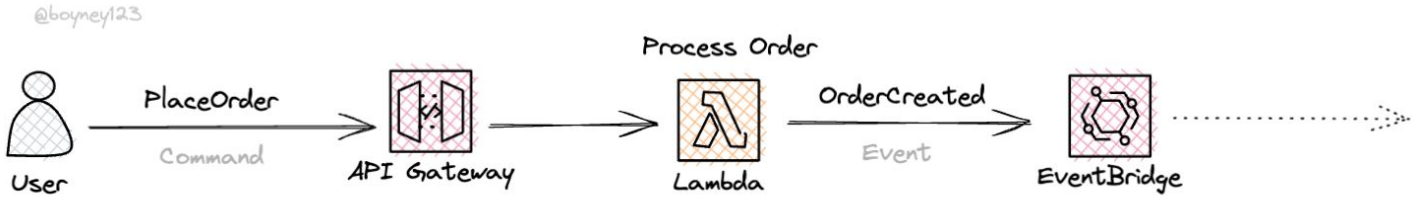
Recommendations

Reactive Manifesto: https://www.reactivemanifesto.org/

Reactive Principles: https://www.reactiveprinciples.org/

Patterns

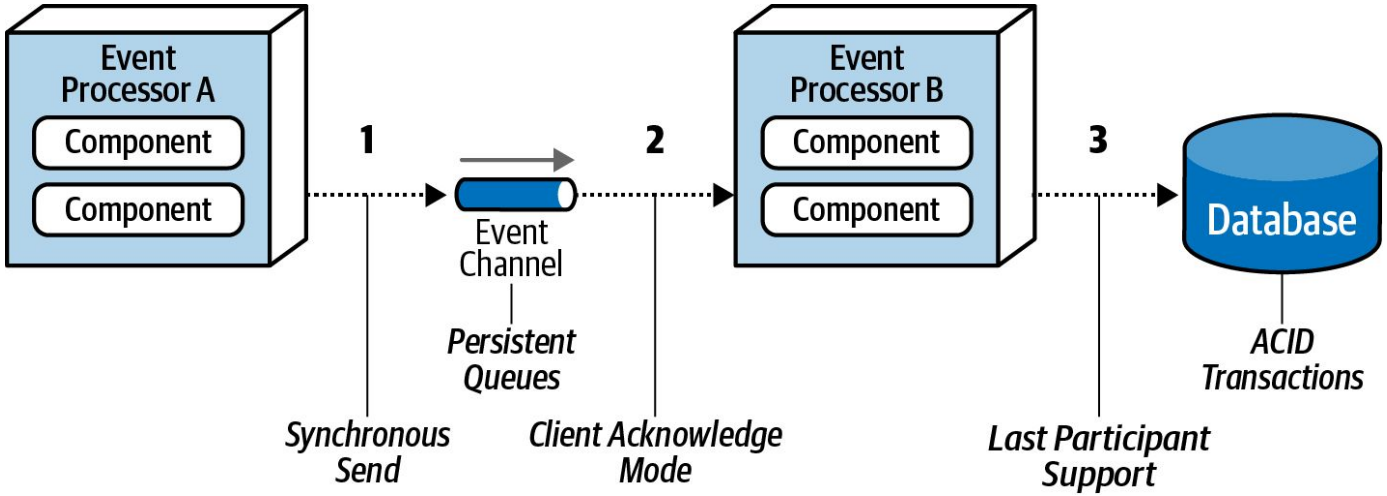
Commands vs Events



https://serverlessland.com/event-driven-architecture/visuals/commands-vs-events

Patterns

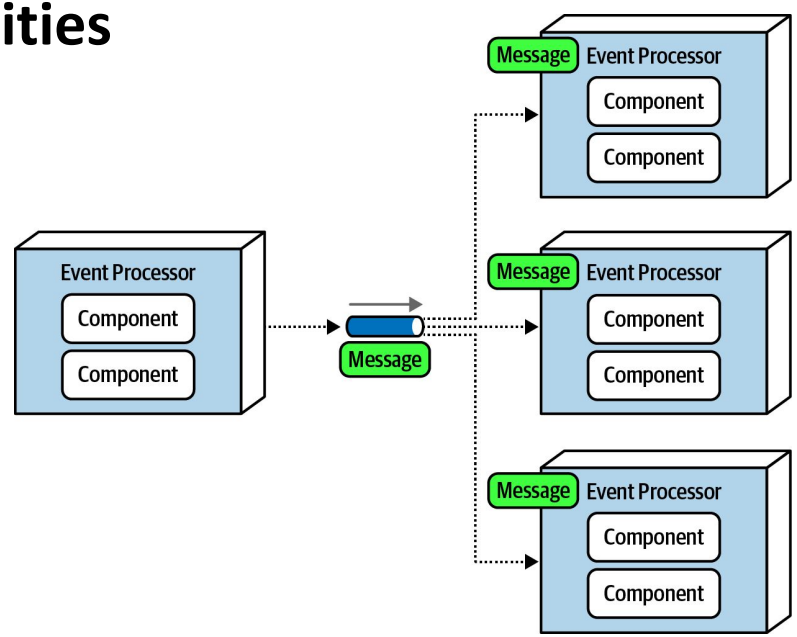
Preventing Data Loss



Mark Richards and Neal Ford. Software Architecture Fundamentals. O'Reilly Media, Inc.

Patterns

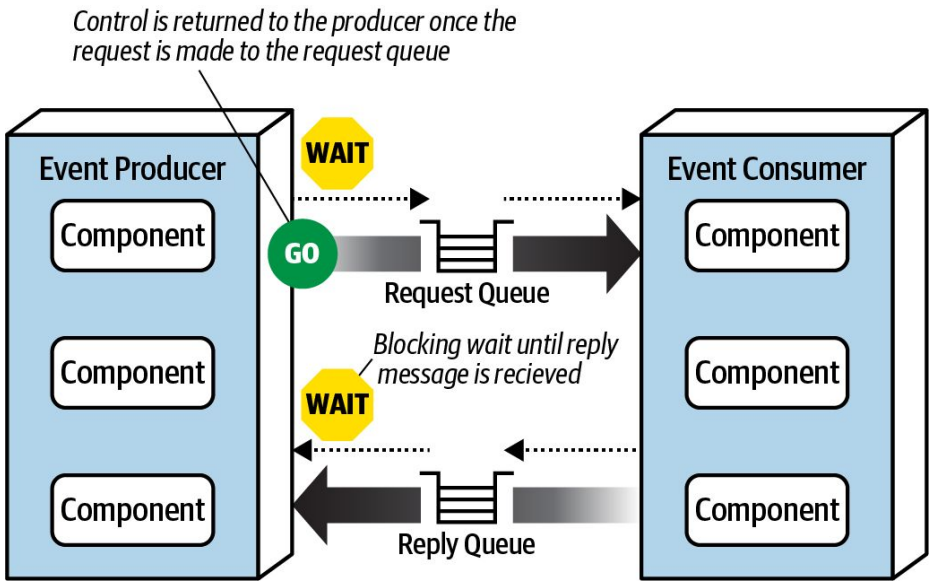
Broadcast Capabilities



Mark Richards and Neal Ford. Software Architecture Fundamentals. O'Reilly Media, Inc.

Patterns

Request-Reply



Mark Richards and Neal Ford. Software Architecture Fundamentals. O'Reilly Media, Inc.

Patterns

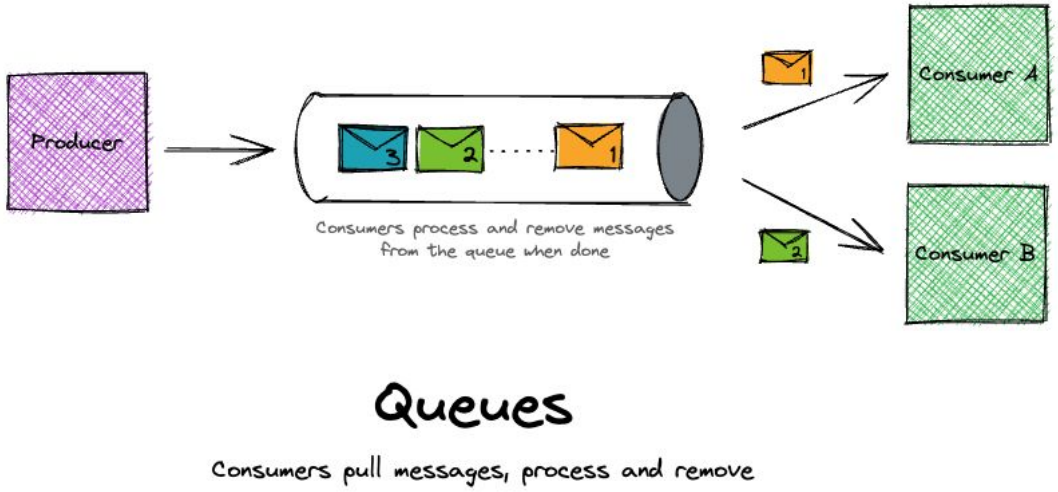
Idempotent Consumer

- Process the same message multiple times will result in the same outcome as processing it once

- Essential when using a message broker with **at-least once delivery**

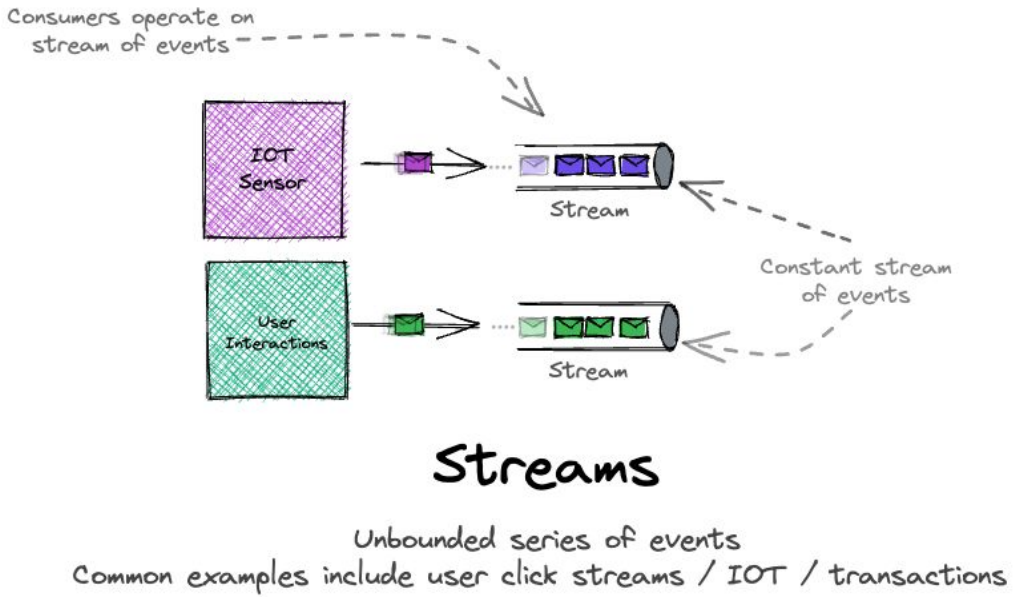
https://microservices.io/patterns/communication-style/idempotent-consumer.html

Queues, Streams, Pub/Sub



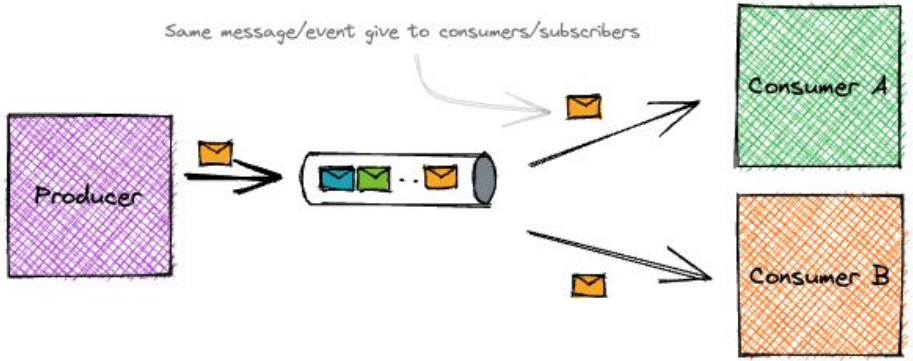
https://serverlessland.com/event-driven-architecture/visuals/queues-vs-streams-vs-pubsub

Queues, Streams, Pub/Sub



https://serverlessland.com/event-driven-architecture/visuals/queues-vs-streams-vs-pubsub

Queues, Streams, Pub/Sub



Pub/Sub

Publish messages/events to many subscribers
Each subscriber gets copy of event to process

https://serverlessland.com/event-driven-architecture/visuals/queues-vs-streams-vs-pubsub

Patterns

Dead Letter Queue (DLQ)

- Mechanism designed to handle messages that have failed to be delivered or processed

https://www.enterpriseintegrationpatterns.com/patterns/messaging/DeadLetterChannel.html

Spec/Doc: AsyncAPI

asyncapi.yaml

```yaml
asyncapi: 2.6.0
info:
  title: Account Service
  version: 1.0.0
  description: This service is in charge of processing user
signups :rocket:
channels:
  user/signedup:
    subscribe:
      message:
        $ref: '#/components/messages/UserSignedUp'
components:
  messages:
    UserSignedUp:
      payload:
        type: object
        properties:
          displayName:
            type: string
            description: Name of the user
          email:
            type: string
            format: email
            description: Email of the user
```

Account Service Documentation

Account Service 1.0.0

This service is in charge of processing user signups 🚀

SUB user/signedup

Accepts the following message:

Payload **Object**

displayName	**String** Name of the user
email	**String** **email** Email of the user

Additional properties are allowed.

```
// Example

{
  "displayName": "Eve & Chan",
  "email": "info@asyncapi.io"
}
```

https://www.asyncapi.com/

Spec/Doc: AsyncAPI

Playground: https://studio.asyncapi.com/

New File > Apache Kafka > Use Template

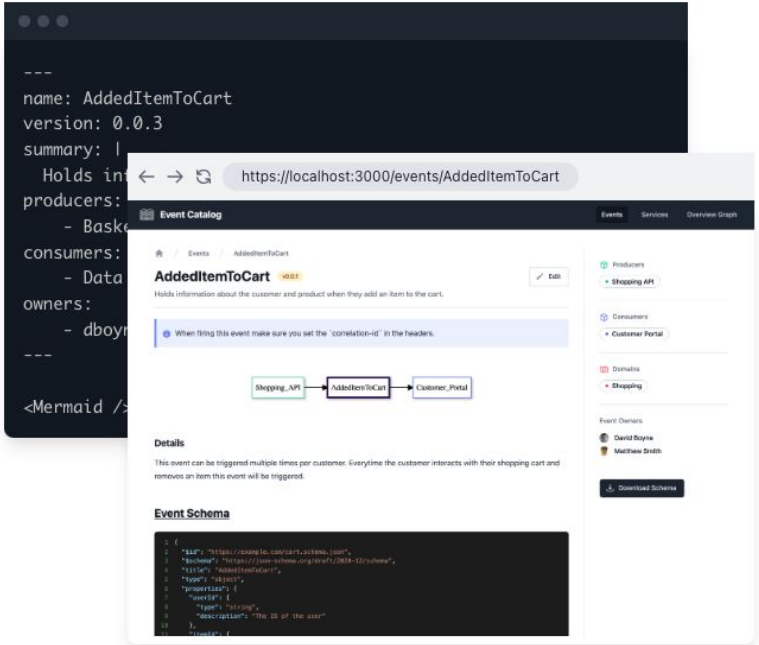
Tutorial: https://killercoda.com/asyncapi/scenario/streetlight-tut

~~Start the application: npm start~~

```
// output/main.js
const { client } = require('./src/api')
client.init()
```

Start the application: **node main.js**

Spec/Doc: Event Catalog



https://www.eventcatalog.dev/

Spec/Doc: Event Catalog

Demo: https://app.eventcatalog.dev/

Spec/Doc: CloudEvents

cloudevents

https://cloudevents.io/

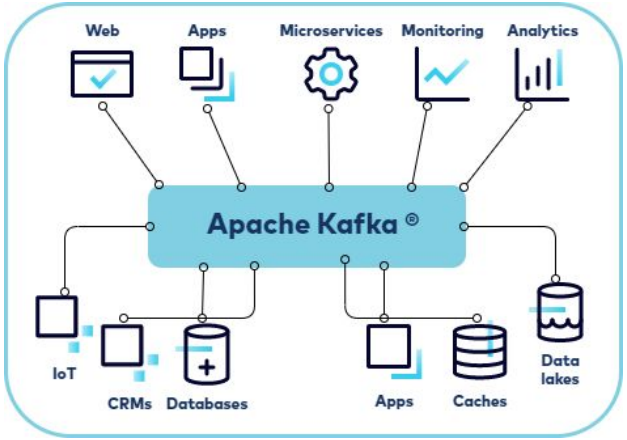
Spec: https://github.com/cloudevents/spec/blob/v1.0.2/cloudevents/spec.md

Kafka and Confluent Cloud

Introduction to Kafka

Apache Kafka® is a distributed event streaming platform that:

- Publishes and subscribes to **streams of events**
- Stores streams of events in a fault-tolerant durable way
- Processes streams of events as they occur



Terminology - Brokers

A broker refers to a server in the Kafka storage layer that **stores event streams** from one or more producers, and later allows consumers to fetch them

A Kafka cluster is typically comprised of several brokers

Terminology - Topics

The Kafka cluster organizes and stores streams of events in topics

- a topic is append only
- events in the topic are immutable
- a consumer reads a event by looking for an **offset** and then reading the events that follow sequentially
- topics are always multi-producer and multi-subscriber
- topics cannot be queried, however, events in a topic can be read as often as needed
- unlike other messaging systems, events are not deleted after they are consumed
- topics can be configured to expire data after it has reached a certain age or when the topic has reached a certain size
- a topic logically and physically splits into partitions

Terminology - Producers

Producers are clients that write events to Kafka.

The producer specifies the topics they will write to and the producer controls how events are assigned to partitions within a topic.

Terminology - Consumers

Consumers are clients that read events from Kafka, subscribing to one or multiple topics.

Normally a consumer will advance its offset linearly as it reads records, however, because the position is controlled by the consumer it can consume records in any order.

Terminology - Partitions

Kafka's topics are divided into several partitions.

While the topic is a logical concept in Kafka, a partition is the smallest storage unit that holds a subset of records owned by a topic.

When a new event is published to a topic, it is actually appended to one of the topic's partitions.

Kafka Visualization

Apache Kafka is a distributed event streaming platform. Using the tool below you can simulate how data flows through a replicated Kafka topic, to gain a better understanding of the message processing model.

Configuration

Choose the number of partitions - between which data will be evenly distributed. Experiment with various counts of brokers, turning them on and off, and seeing how the system adapts. Make sure to store data in replicas, so that they are not lost! Simulate load by increasing the consume interval. Finally, verify how offsets are commited, and see how this impacts redelivery when consumers or brokers are added/removed.

Partitions : 1 2 3 4 5

Brokers : 1 2 3 4 5

Replication factor : 1 2 3 4 5

Producer

Producing interval : 1 ticks

Consumers

Hide Configuration | Hide Descriptions

> What is Apache Kafka?

Producer:

P 1 2 3 4 5 6 7 8 9 10

> What is a Kafka producer?

Brokers:

1

> What is a Kafka broker?

Consumers:

A 1

> What is a Kafka consumer?

Kafka Connect

Kafka Stream

ksqlDB

…

Confluent Cloud

Confluent Cloud is a resilient, scalable, streaming data service based on Apache Kafka®, **delivered as a fully managed service**.

CONFLUENT

https://confluent.io/

Get started with Confluent, for free

Confluent makes it easy to connect your apps, systems, and entire organization with real-time data flows and processing. We provide a solution for data in motion that is cloud native, complete, and available everywhere you need it.

Cloud Native

Stream data at massive scale without the operational burden or expense of open source software

Complete

Go way beyond Kafka to build real-time apps quickly, reliably, and securely

Everywhere

Maintain flexibility with deployments across clouds and on-premises that sync in real time

Choose your deployment:

CLOUD | SELF-MANAGED

CONFLUENT CLOUD

A fully managed, cloud-native service for Apache Kafka®

New signups receive $400 to spend during their first 30 days
No credit card required.

SIGN IN WITH GOOGLE

SIGN IN WITH GITHUB

OR

Full Name

Company

Email

Canada

Create cluster

1. Select cluster type

Create a Kafka cluster

Select a cluster type to determine the features, usage limits, and price of your cluster.

Recommended

Basic

For learning and exploring Kafka and Confluent Cloud.

Ingress	up to 250 MB/s
Egress	up to 750 MB/s
Storage	up to 5,000 GB
Client connections	up to 1,000
Partitions	up to 4,096 (includes 10 free partitions)
Uptime SLA	up to 99.5%

Begin configuration

Starting at

FREE

Upgrade to Standard at any time

Ready to get started?

You have earned $400 to spend in the first 30 days! Claim the free credit to start your streaming journey.

Enter a payment method to keep your services up and running beyond the free trial. Please note that you will **not be charged** before the free trial ends. Learn more

Claim free credit

Dedicated

For use cases with high traffic or that require private networking.

Price as sized: 2 CKUs

Ingress	up to 100 MB/s
Egress	up to 300 MB/s
Storage	unlimited
Client connections	up to 18,000
Partitions	up to 9,000
Uptime SLA	up to 99.99%

Begin configuration

Starting at

$5.33 /hr

Starting at

$1.50 /hr

Starting at

$4.50 /hr

Pricing scales based on usage.

Learn more

Environment

Create Environment

Stream Governance Packages: essentials

Cloud: AWS (Ohio us-east-2)

Environment ID: env-kgkxmg

Environment Name: dio-eda

Cluster

Add Cluster to the Environment dio-eda

Basic

Region: AWS (Ohio us-east-2, Single Availability)

Skip Payment

Cluster Name: staging-us-east-2

Cluster ID: lkc-6wd1zq

Cluster Settings

Bootstrap Server Endpoint:

pkc-921jm.us-east-2.aws.confluent.cloud:9092

API Keys

Create key

Global access

Key: NFYBTLXI32YO6BZL

Secret:
GTLqzWmeWZvDFo/anQos4K3vtSAVyMKzLlFrZIugrGFFBtFM3IytZwhLW9E8/Eds

Description: spring-boot-application

Create Topic

Topic name: students.twitter.posts

Partitions: 3

Hands-On

Spring Boot

https://github.com/thpoiani/dio-international-acceleration-kafka

Challenges and Discussions

+ Serialization: Avro + CloudEvents
+ Schema Registry
+ Change Data Capture (CDC): Database Connector

Q&A

Thank You!
@thpoiani