Trilha Microservices

Event Sourcing em uma arquitetura de Micro-serviços

Thiago da Rosa de Bustamante

Dev na ThoughtWorks / Professor PUC

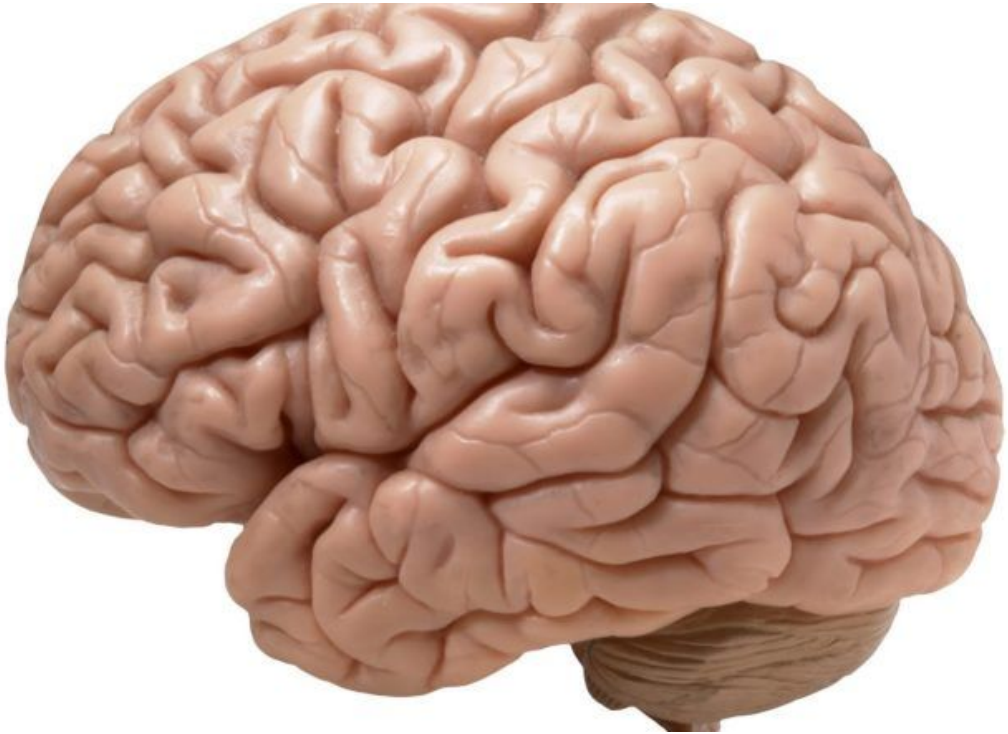
Thalita Nick Pinheiro Gomes

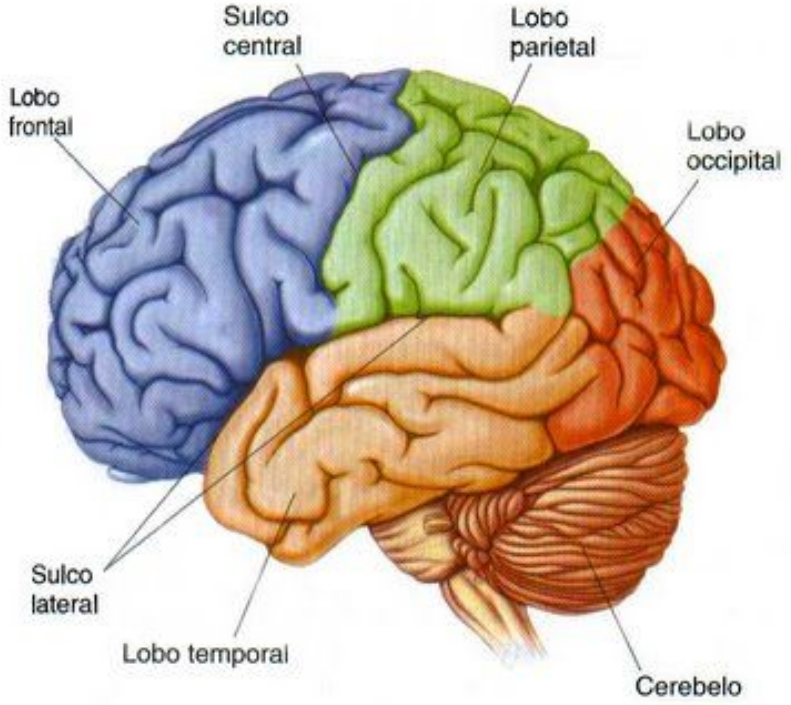
@ThalitaPinheiro -TechLead / Deva na ThoughtWorks

THE DEVELOPER'S CONFERENCE

THE
DEVELOPER'S
CONFERENCE

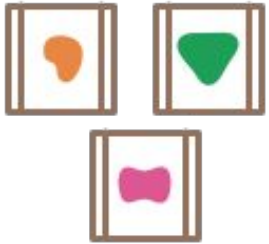
Monolitos e Microserviços

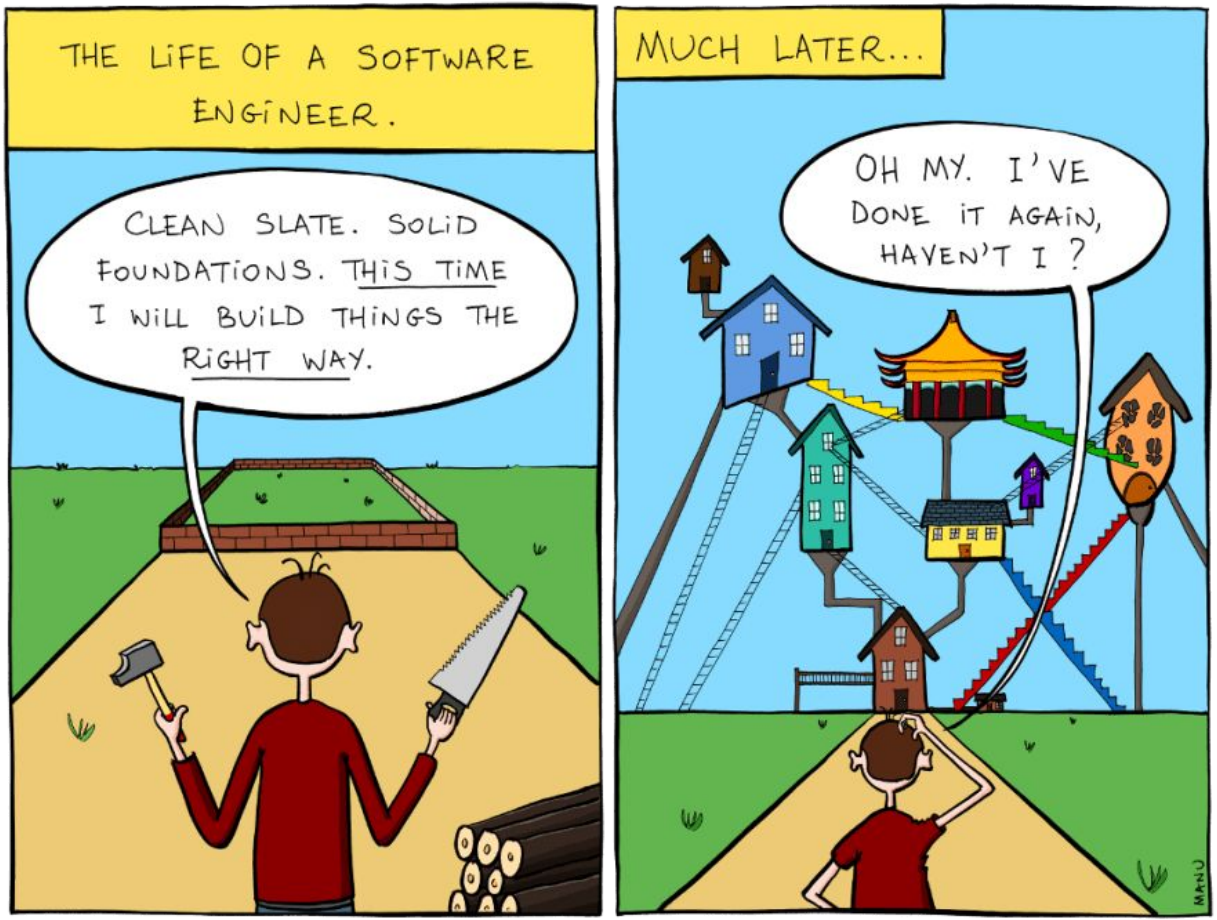



Sulco central
Lobo parietal
Lobo frontal
Lobo occipital
Sulco lateral
Lobo temporal
Cerebelo

A monolithic application puts all its functionality into a single process...

A microservices architecture puts each element of functionality into a separate service...





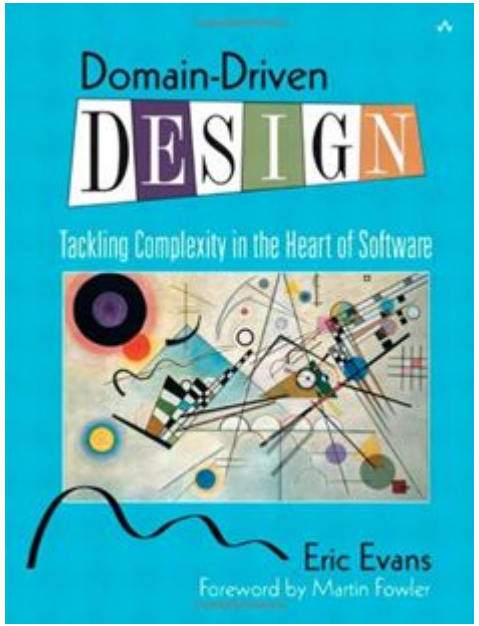
THE DEVELOPER'S CONFERENCE

THE DEVELOPER'S CONFERENCE

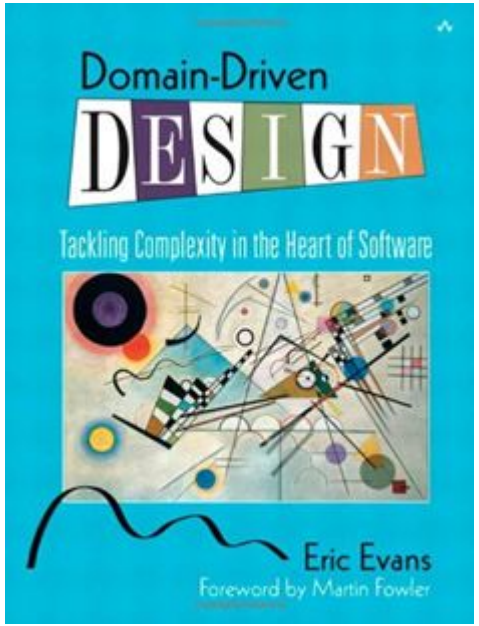
Domain Driven Design

THE DEVELOPER'S CONFERENCE

- Entidades
- Value Objects
- Services
- Repositories
- Agregações



- Entidades
- Value Objects
- Services
- Repositories
- **Agregações**

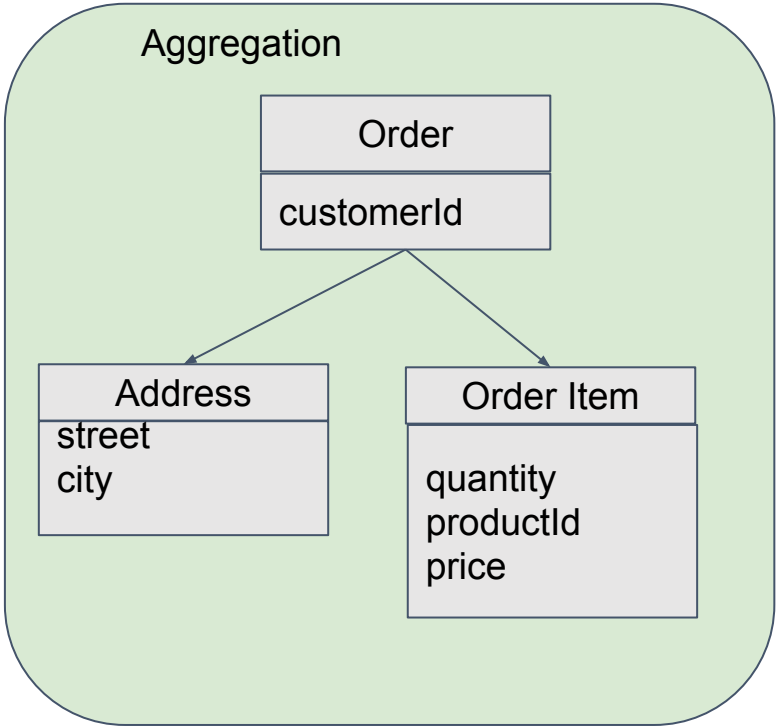


Aggregates

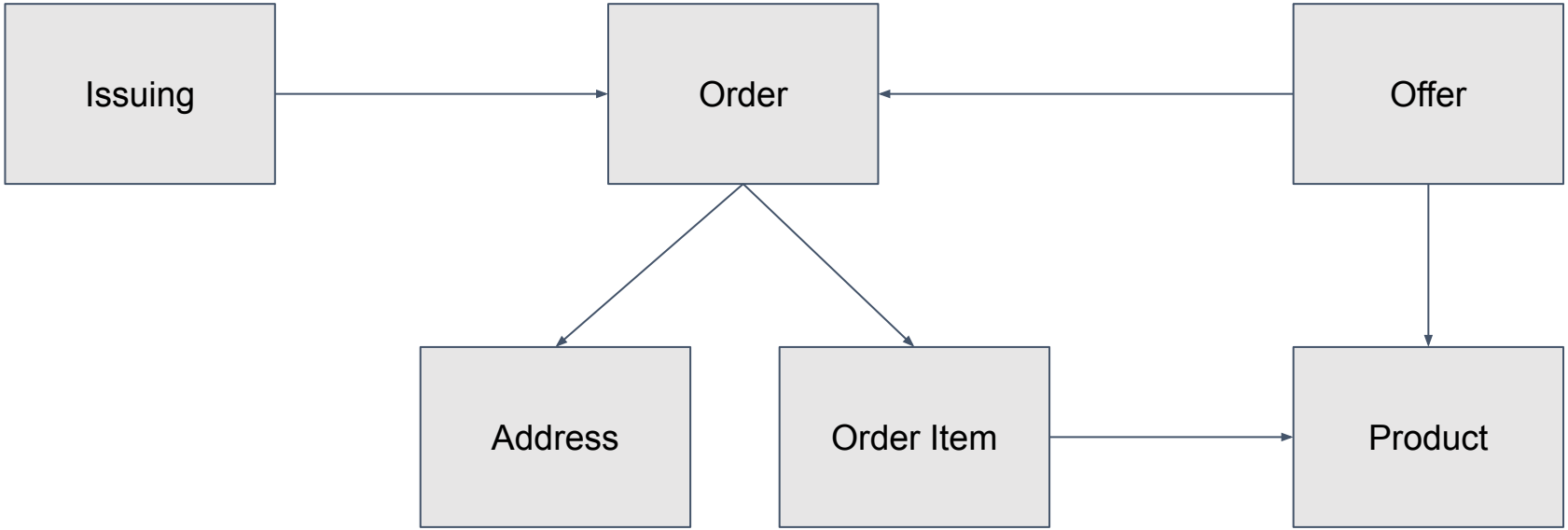
THE DEVELOPER'S CONFERENCE

Definição:

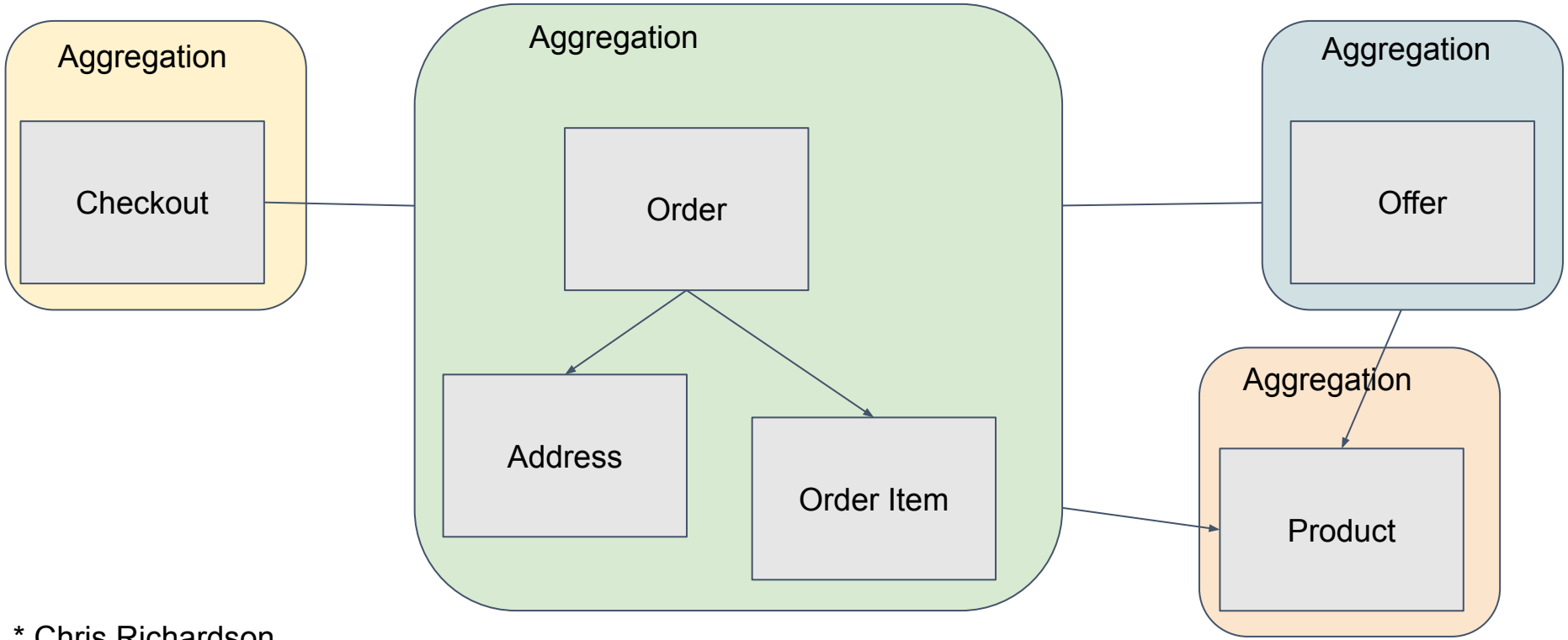
- Conjunto de objetos que podem ser tratados como uma unidade.
- Possui uma entidade "Root" e entidades e value objects associados



Granularidade de Aggregates

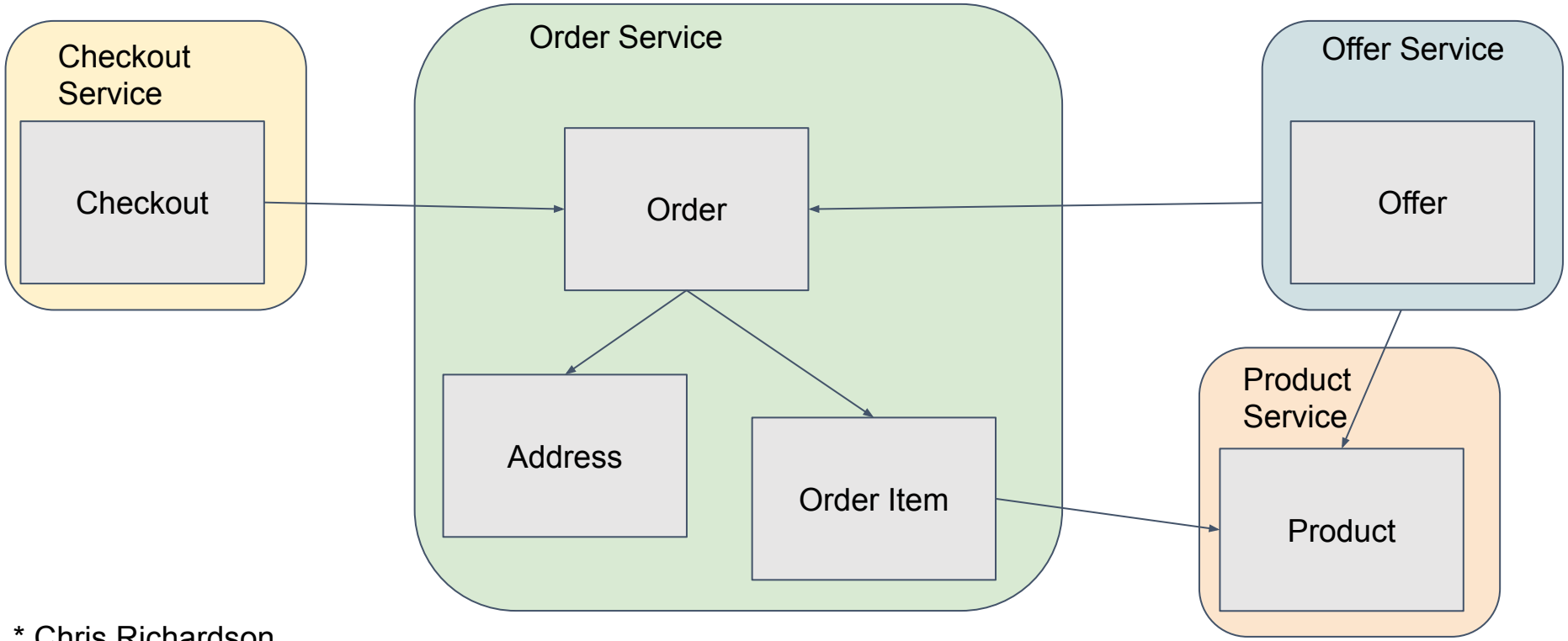


Domain Model = Agregações fracamente acopladas (*)

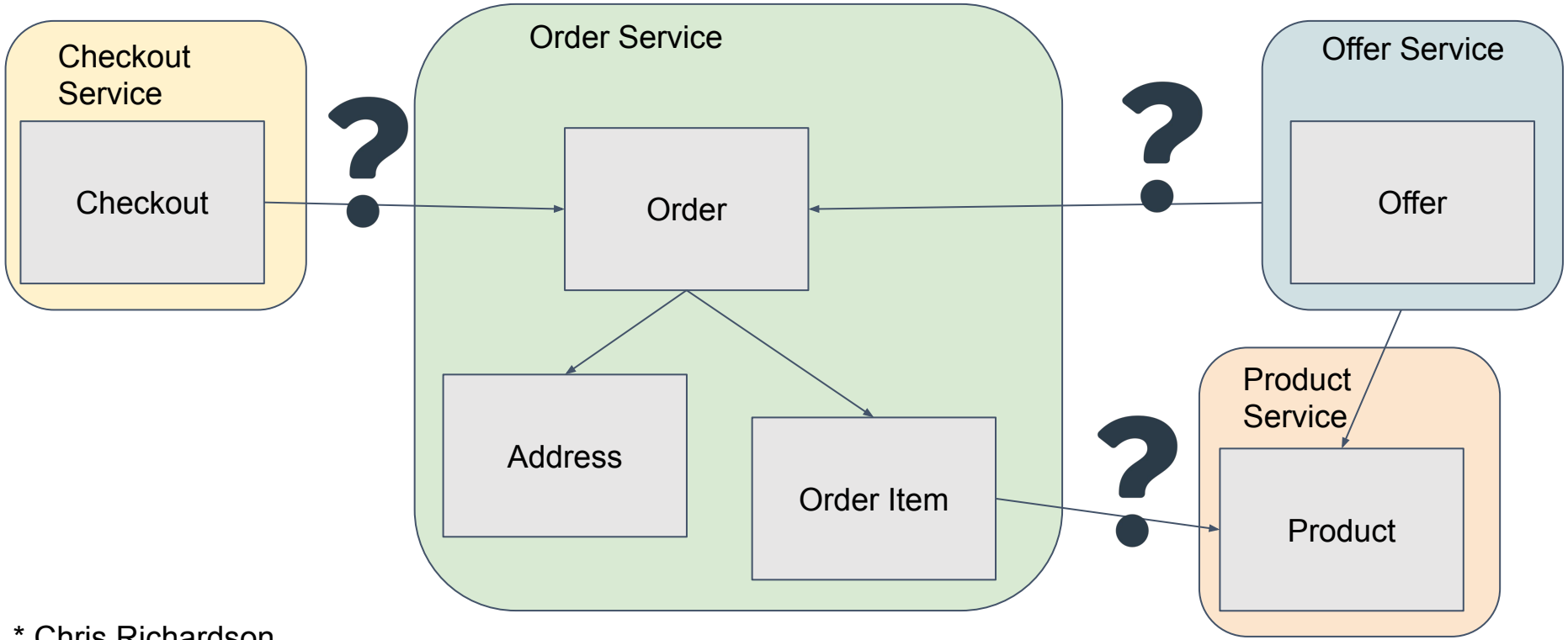


* Chris Richardson

Facilmente particionável em microsserviços



* Chris Richardson



* Chris Richardson

THE
DEVELOPER'S
CONFERENCE

Encapsulamento

BEGIN TRANSACTION

…

SELECT ORDER_TOTAL FROM ORDERS WHERE CUSTOMER_ID = ?

…

SELECT CREDIT_LIMIT FROM CUSTOMERS WHERE CUSTOMER_ID = ?

…

INSERT INTO ORDERS …

…

COMMIT TRANSACTION

ACID

THE DEVELOPER'S CONFERENCE

BEGIN TRANSACTION

…

SELECT ORDER_TOTAL FROM ORDERS WHERE CUSTOMER_ID = ?

…

SELECT CREDIT_LIMIT FROM CUSTOMERS WHERE CUSTOMER_ID = ?

…

INSERT INTO ORDERS …

…

COMMIT TRANSACTION

BEGIN TRANSACTION

…

SELECT ORDER_TOTAL FROM ORDERS WHERE CUSTOMER_ID = ?

…

SELECT CREDIT_LIMIT FROM CUSTOMERS WHERE CUSTOMER_ID = ?

…

INSERT INTO ORDERS …

…

COMMIT TRANSACTION

THE DEVELOPER'S CONFERENCE

Domain Events

definition_processed

THE DEVELOPER'S CONFERENCE

Evento

[ocultar]

Origem: Wikipédia, a enciclopédia livre.

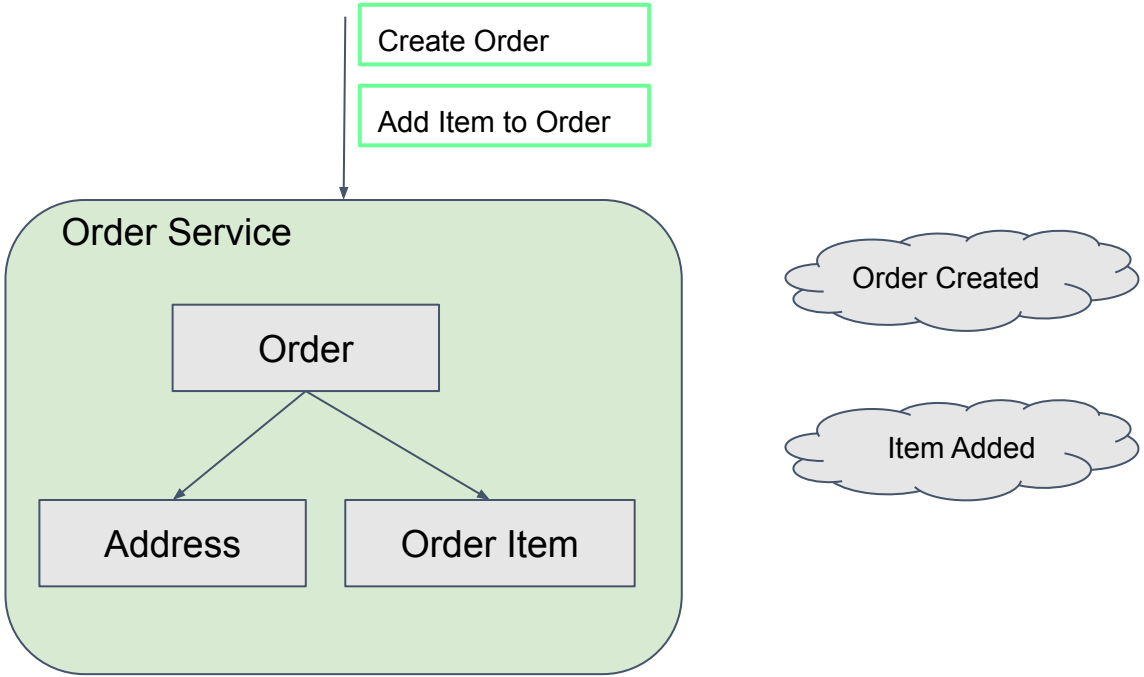
Evento pode referir-se a:

- Evento (computação) — uma mensagem do software indicando que algo aconteceu, como por exemplo o pressionamento de uma tecla ou um clique do mouse

O Wikcionário tem o verbete ***Evento***.

- Sempre no passado
- É Idempotente
- Afeta uma única agregação

Comandos e Eventos



* Chris Richardson

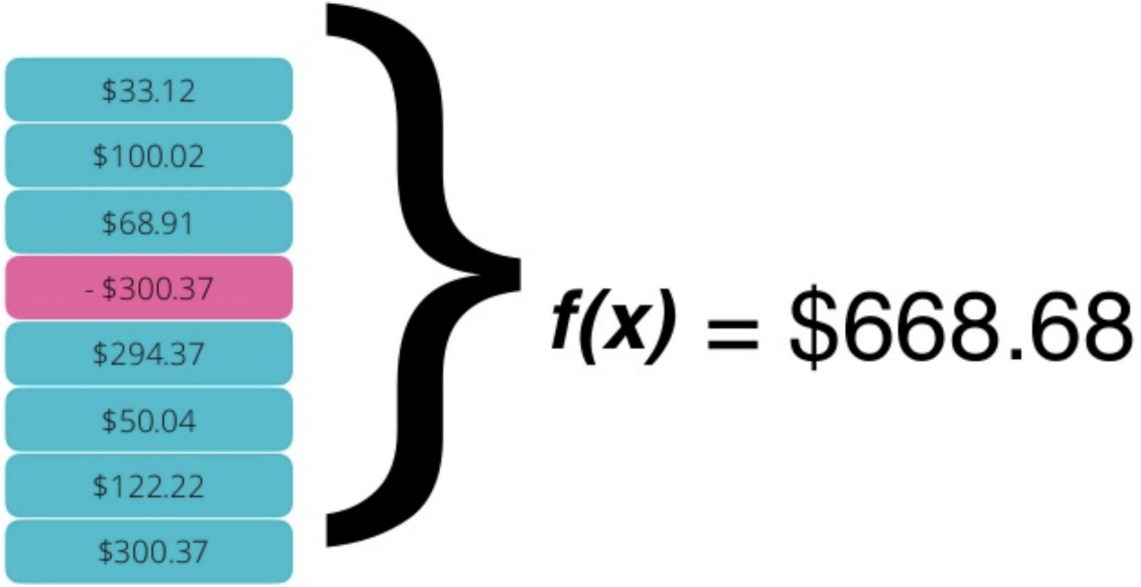
THE DEVELOPER'S CONFERENCE

Event Sourcing

THE
DEVELOPER'S
CONFERENCE

$668.68

Evento



EVENT SOURCING

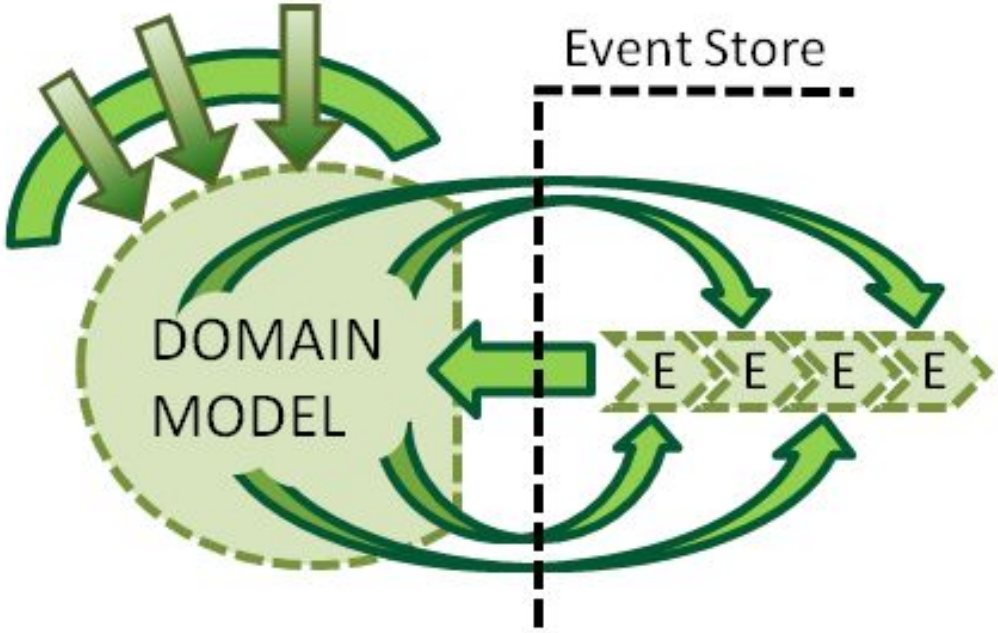
"Event Sourcing ensures that all changes to application state are stored as a sequence of events. Not just can we query these events, we can also use the event log to reconstruct past states, and as a foundation to automatically adjust the state to cope with retroactive changes."

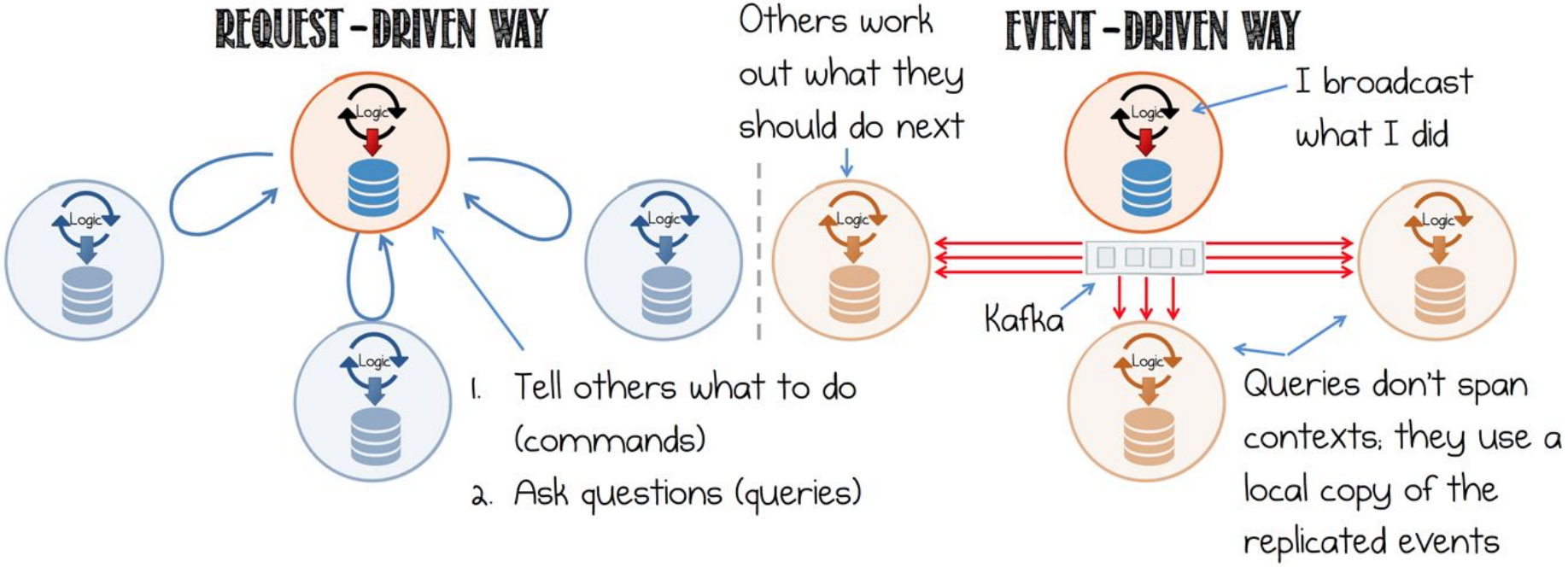
Martin Fowler

THE DEVELOPER'S CONFERENCE

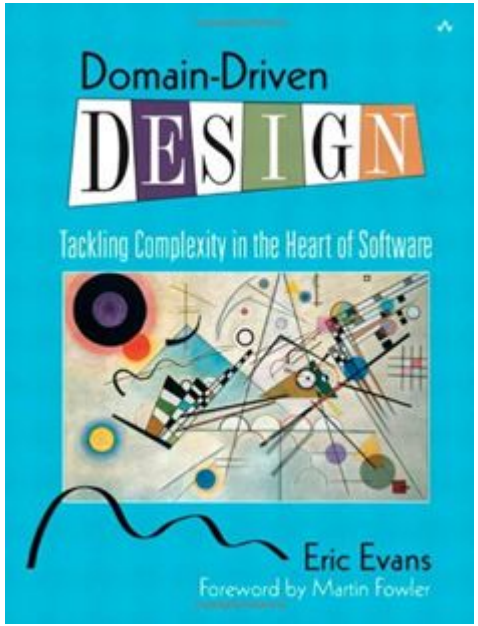
Stream id	Aggregate	Sequence	Event type	Event data	Timestamp
X	Order	1	OrderCreated	{ … }	1560429907
X	Order	2	OrderApproved	{ … }	1560431997
X	Order	3	OrderShipped	{ … }	1560433507

Event Store




REQUEST-DRIVEN WAY
Logic
Logic
Logic
Logic
1. Tell others what to do (commands)
2. Ask questions (queries)
Others work out what they should do next
EVENT-DRIVEN WAY
Logic
I broadcast what I did
Logic
Logic
Kafka
Logic
Queries don't span contexts; they use a local copy of the replicated events

Referencias



https://www.slideshare.net/SpringCentral/developing-microservices-with-aggregates?from_action=save

https://martinfowler.com/bliki/

https://herbertograca.com/2017/11/16/explicit-architecture-01-ddd-hexagonal-onion-clean-cqrs-how-i-put-it-all-together/

THE DEVELOPER'S CONFERENCE

Estudo de caso 1

Orders Microservice

Entidades


'use strict';
export interface Order {
customerId: string;
id?: string;
items?: Array<OrderItem>;
}
export interface OrderItem {
id?: string;
productId: string;
amount: number;
}

Service

THE DEVELOPER'S CONFERENCE

```
@Singleton
@AutoWired
export class OrdersService {
    @Inject private orders: OrdersCommands;

    public async createOrder(order: Order): Promise<Order> {
        if (order) {
            order.id = await this.orders.createOrder(order);
        }

        return order;
    }

    public async createOrderItem(orderId: string, item: OrderItem): Promise<OrderItem> {
        if (item) {
            item.id = await this.orders.createOrderItem(orderId, item);
        }

        return item;
    }
}
```

Commands

THE DEVELOPER'S CONFERENCE

```
10  export class OrdersCommands {
11      @Inject private ordersRepository: OrdersRepository;
12
13      public async createOrder(order: Order): Promise<string> {
14          order.id = uuid();
15          await this.ordersRepository.addEvent(order.id, {
16              data: order,
17              type: Events.OrderCreated
18          });
19          return order.id;
20      }
21
22      public async createOrderItem(orderId: string, item: OrderItem): Promise<string> {
23          item.id = uuid();
24          await this.ordersRepository.addEvent(orderId, {
25              data: item,
26              type: Events.OrderItemAdded
27          });
28          return item.id;
29      }
30  }
```

Repository

THE DEVELOPER'S CONFERENCE

```
7   export class OrdersRepository {
8       private eventStore: EventStore;
9
10      constructor() {
11          this.eventStore = EventStoreFactory.get();
12      }
13
14      public async addEvent(orderId: string, orderEvent: OrderEvent<any>): Promise<void> {
15          const eventStream = this.eventStore.getEventStream('orders', orderId);
16          await eventStream.addEvent(orderEvent);
17      }
18  }
```

Event Store


import { EventStore, RedisProvider } from '@eventstore.net/event.store';
import { Configurations } from '../command-line';
export class EventStoreFactory {
public static get() {
if (!EventStoreFactory.eventStore) {
EventStoreFactory.eventStore = new EventStore(
new RedisProvider(Configurations.eventStore));
}
return EventStoreFactory.eventStore;
}
private static eventStore: EventStore;
}

THE DEVELOPER'S CONFERENCE

Estudo de caso 2

Offers Microservice

THE DEVELOPER'S CONFERENCE

Events

```
5   export enum Events {
6       OrderCreated = 'OrderCreated',
7       OrderItemAdded = 'OrderItemAdded'
8   }
9
10  export interface OrderEvent<T> {
11      data?: T;
12      type: string;
13  }
14
15  export interface OrderCreated extends OrderEvent<Order> {
16      type: Events.OrderCreated;
17  }
18
19  export interface OrderItemAdded extends OrderEvent<OrderItem> {
20      type: Events.OrderItemAdded;
21  }
```

THE DEVELOPER'S CONFERENCE

Service

```
10  @Singleton
11  @AutoWired
12  export class OffersService {
13      @Inject private ordersService: OrdersService;
14      @Inject private offersRepository: OffersRepository;
15
16      public async getOffers(orderId?: string, customerId?: string): Promise<Array<Offer>> {
17          let order: Order = null;
18          if (orderId) {
19              order = await this.ordersService.getCurrent(orderId);
20          }
21          return await this.getOffersForOrder(order);
22      }
23
24      private async getOffersForOrder(order: Order) {…
35      }
36
37      private getOrderAmount(order: Order) {…
42      }
43  }
```

THE DEVELOPER'S CONFERENCE

Service

```
 9  export class OrdersService {
10      @Inject private ordersRepository: OrdersRepository;
11      @Inject private ordersEvents: OrderEvents;
12
13      public async loadHistory(orderId: string): Promise<Array<OrderEvent<any>>> {
14          return await this.ordersRepository.getEvents(orderId);
15      }
16
17      public async getCurrent(orderId: string) {
18          const history: Array<OrderEvent<any>> = await this.loadHistory(orderId);
19          if (history && history.length) {
20              return history.reduce((order: Order, event: OrderEvent<any>) => {
21                  return this.ordersEvents.process(order, event);
22              }, { id: orderId, items: [] } as Order);
23          }
24          return null;
25      }
26  }
```

THE DEVELOPER'S CONFERENCE

Repository

```
7   export class OrdersRepository {
8       private eventStore: EventStore;
9
10      constructor() {
11          this.eventStore = EventStoreFactory.get();
12      }
13
14      public async getEvents(orderId: string): Promise<Array<OrderEvent<any>>> {
15          const eventStream = this.eventStore.getEventStream('orders', orderId);
16          const events = await eventStream.getEvents();
17
18          if (events) {
19              return events.map((event: Event) => event.payload);
20          }
21          return null;
22      }
23  }
```

THE DEVELOPER'S CONFERENCE

Event Processor

```
7  export class OrderEvents {
8
9      public process(order: Order, event: OrderEvent<any>) {
10         if (event.type === Events.OrderCreated) {
11             return this.createOrder(order, event as OrderCreated);
12         }
13         else if (event.type === Events.OrderItemAdded) {
14             return this.createOrderItem(order, event as OrderItemAdded);
15         }
16         return order;
17     }
18
19     private createOrder(order: Order, event: OrderCreated) {
20         return _.merge(event.data || {}, order);
21     }
22
23     private createOrderItem(order: Order, event: OrderItemAdded) {
24         if (!order.items) {
25             order.items = [];
26         }
27         order.items.push(event.data);
28         return order;
29     }
30 }
```

THE DEVELOPER'S CONFERENCE

Estudo de caso 3

Issuing Microservice

THE DEVELOPER'S CONFERENCE

Events

```
5   export enum Events {
6       OrderCreated = 'OrderCreated',
7       OrderItemAdded = 'OrderItemAdded',
8       OrderPaid = 'OrderPaid'
9   }
10
11  export interface OrderEvent<T> {
12      data?: T;
13      type: string;
14  }
15
16  export interface OrderCreated extends OrderEvent<Order> {
17      type: Events.OrderCreated;
18  }
19
20  export interface OrderItemAdded extends OrderEvent<OrderItem> {
21      type: Events.OrderItemAdded;
22  }
23
24  export interface OrderPaid extends OrderEvent<OrderPayment> {
25      type: Events.OrderPaid;
26  }
```

THE DEVELOPER'S CONFERENCE

Interface

```
7   export class IssuingListener {
8       @Inject private ordersRepository: OrdersRepository;
9       @Inject private issuingService: IssuingService;
10
11      public async issue() {
12          this.ordersRepository.onOrderPaid((orderId: string) => {
13              this.issuingService.issue(orderId);
14          });
15      }
16  }
```

Repository

```
7   export class OrdersRepository {
8       private eventStore: EventStore;
9
10      constructor() {
11          this.eventStore = EventStoreFactory.get();
12      }
13
14      public async onOrderPaid(callback: (orderId: string) => void) {
15          this.eventStore.subscribe('orders', message => {
16              if (message.event.payload.type === Events.OrderPaid) {
17                  callback(message.stream.id);
18              }
19          });
20      }
21  }
```

THE
DEVELOPER'S
CONFERENCE

DEMO


LOONEY TUNES
"That's all Folks!"