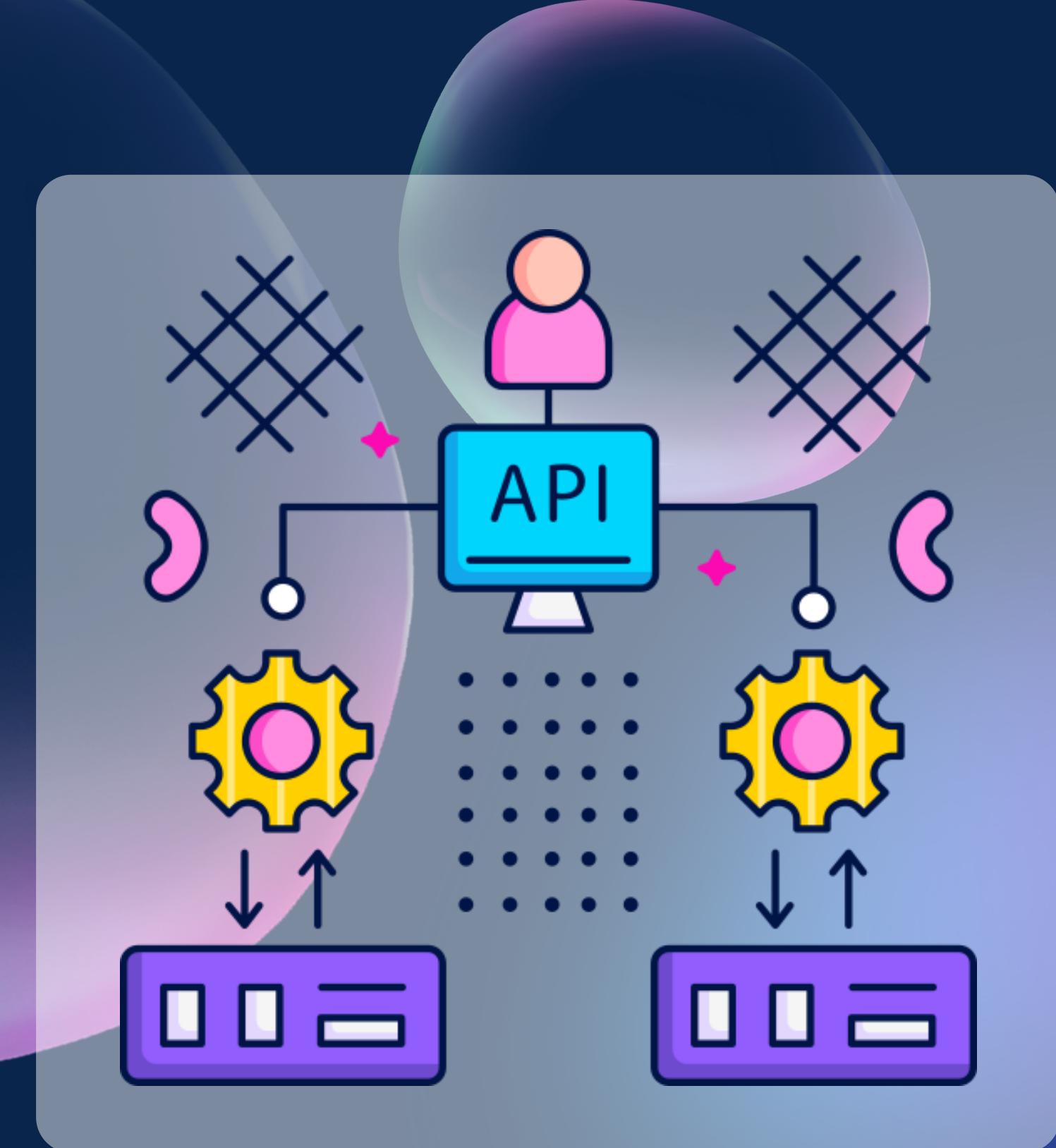
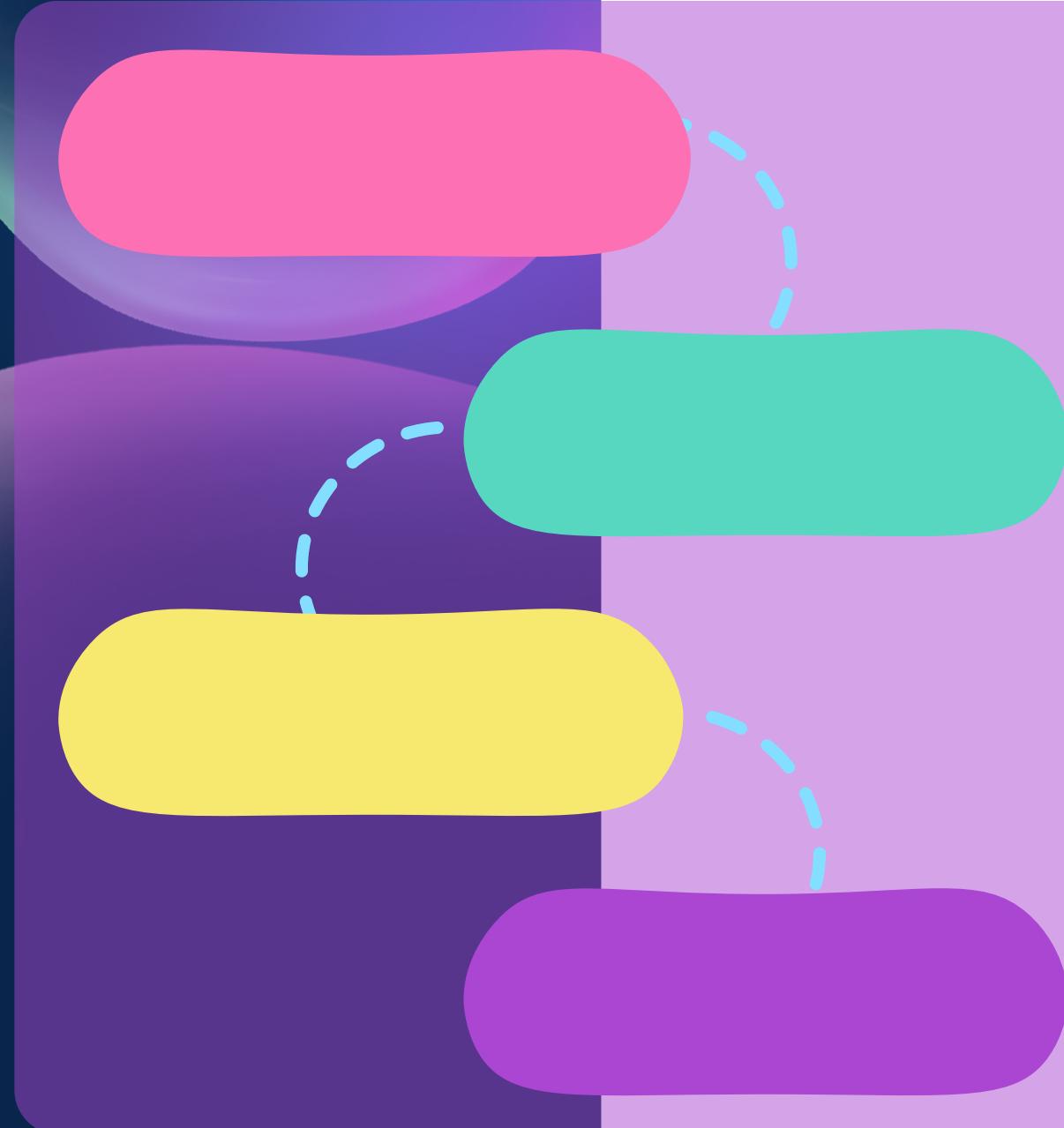


# e-Commerce Saga Orchestration Implementation

Dall'ordine al pagamento: semplificare e garantire un percorso fluido ottimizzando la gestione dei processi.



# Cosa è la Saga Orchestration?



Pattern di progettazione utilizzato per gestire transazioni complesse e processi aziendali all'interno di un ambiente di e-commerce.

Nel contesto dell'e-commerce, molte operazioni coinvolgono interazioni tra diversi servizi o sistemi, come l'elaborazione degli ordini, la gestione degli inventari, i pagamenti, la spedizione e la gestione dei resi. La saga orchestration si occupa di **coordinare** e **gestire** queste operazioni in modo **coerente** e **affidabile**, assicurando che tutti i passaggi siano eseguiti correttamente e che i dati siano coerenti.

# Principali vantaggi

Divisione di un processo complesso in una serie di passi o fasi, chiamati "compensating actions".

Se si verifica un errore o un fallimento durante uno dei passi, la saga orchestration può coordinare l'esecuzione delle azioni di compensazione per *annullare* o correggere le operazioni precedenti e **ripristinare uno stato coerente**.

**1** Gestire la complessità delle transazioni distribuite

**2** Mantenere l'integrità dei dati

**3** Migliorare l'affidabilità

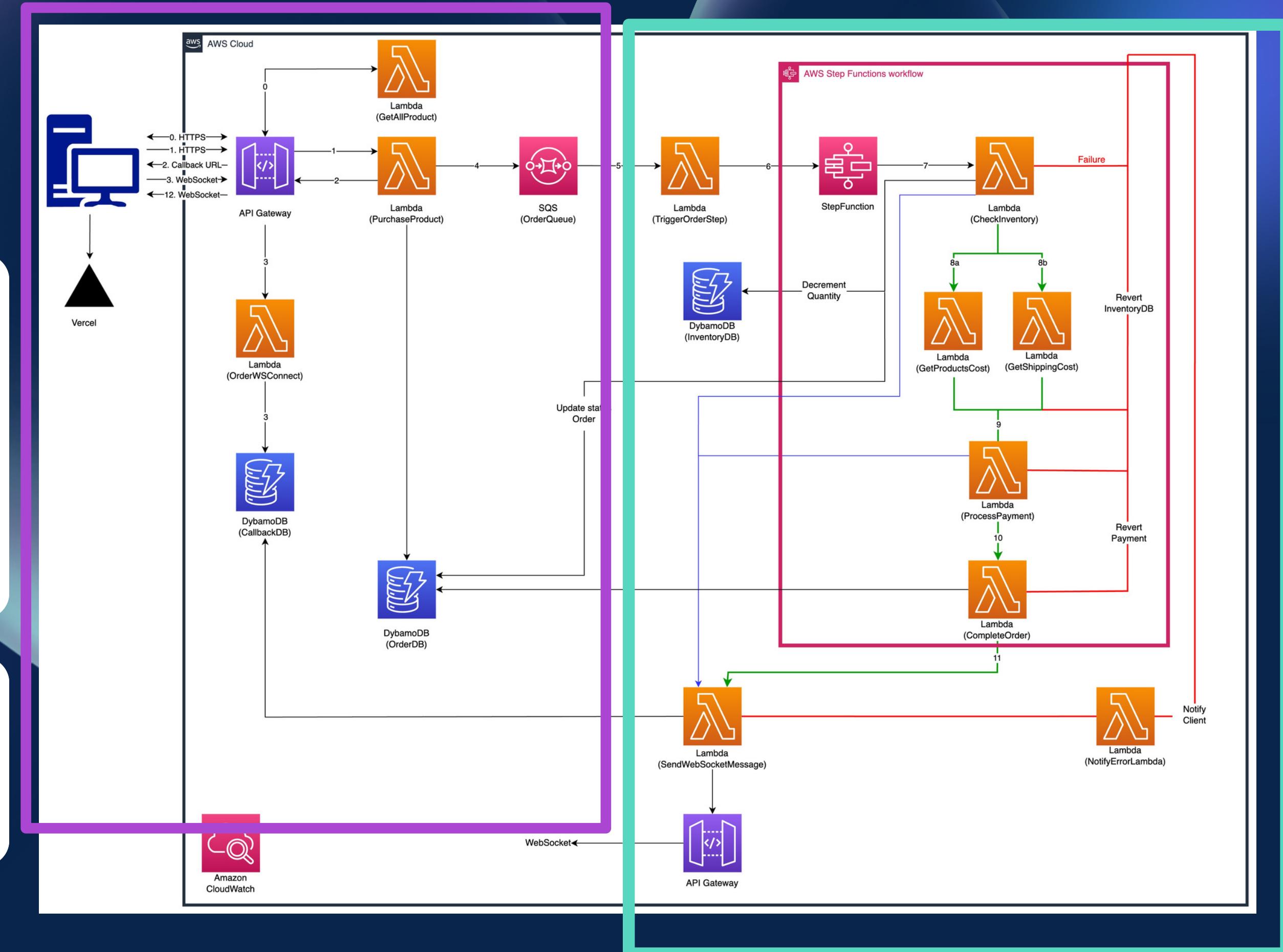
**4** Fornire una migliore esperienza complessiva per i clienti

# Diagram workflow

Utilizzo dei servizi  
di Amazon AWS



Suddivisione del workflow  
con **Lorenzo Marcolli**



# Step Functions Workflow



1 Controllo disponibilità prodotti.



2 Calcolo costi di spedizione e totale carrello.



3 Processazione e validazione pagamento.



4 Completamento ordine.



5 Notifica al client in caso di errori.

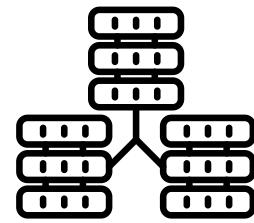




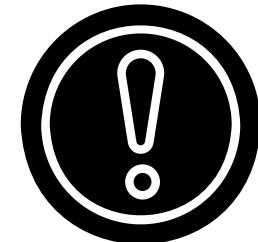
# AWS Step Functions

AWS Step Functions è un **servizio di orchestrazione** di workflow **serverless** che permette agli sviluppatori di utilizzare i servizi AWS per costruire applicazioni distribuite, automatizzare processi, orchestrare microservizi e costruire pipeline di dati e machine learning (ML).

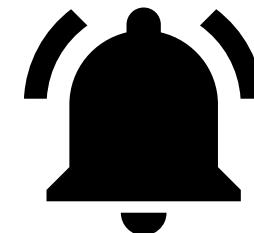
## Fault tolerance



- Ridondanza multi-zona
- Ripetizione e ritentativi

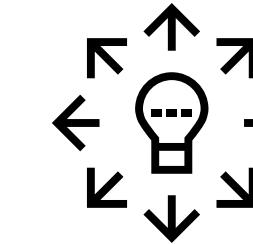


- Gestione delle eccezioni
- Timeout delle attività



- Monitoraggio e allarmi
- Rollback e ripristino
- Backup e ripristino

## Scalabilità



- Scalabilità automatica
- Orchestrazione di servizi scalabili
- Parallelismo
- Gestione delle code

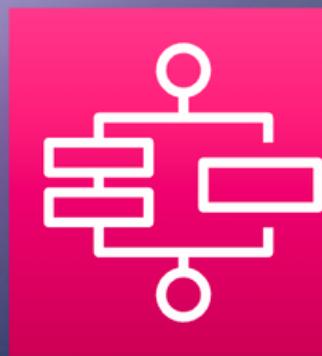


## AWS Lambda

servizio di calcolo basato su eventi serverless

- Scalabilità automatica
- Event-driven

Operazioni sui dati



## API Gateway

servizio di gestione e distribuzione di API e Websocket

- Scalabilità automatica
- Gestione connessioni ed eventi

Notifica status operazioni



## DynamoDB

servizio di database NoSQL completamente gestito

- Modello di dati flessibile
- Scalabilità automatica
- Alta disponibilità e durabilità

Salvataggio ordini e aggiornamento status

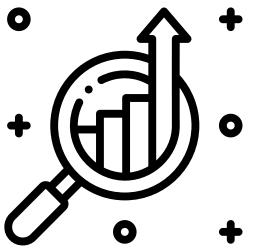


# Operazioni sui dati e invocazione servizi

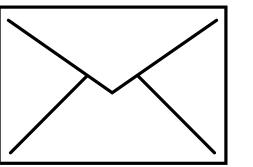


```
const updateItemParams = {
  TableName: 'Inventory',
  Key: {
    productId: { N: product.id.toString() },
  },
  UpdateExpression: 'SET quantity = quantity - :decreaseAmount',
  ExpressionAttributeValues: {
    ':decreaseAmount': { N: product.quantity.toString() }, // Specify the amount to decrease
    ':quantityitem': { N: '0' },
  },
  ConditionExpression: 'quantity > :quantityitem',
}

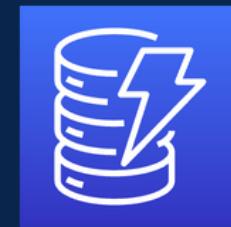
const command = new UpdateItemCommand(updateItemParams)
const data = await dynamoDBClient.send(command)
```



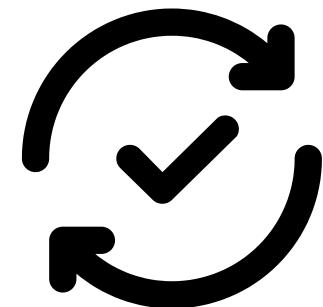
Operazioni sui dati  
invocando i metodi di  
get e update di  
DynamoDB.



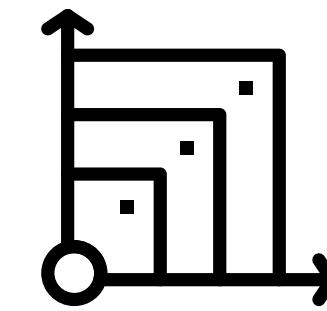
Invocazione del metodo  
di invio messaggi delle  
Websocket.



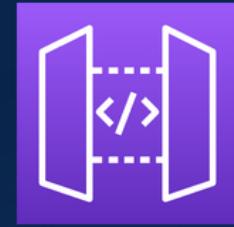
# Salvataggio ordini e aggiornamento status



Aggiornamenti  
continui sullo stato  
degli ordini.



Scalabilità automatica: si  
adatta automaticamente  
per gestire picchi di traffico  
o ridurre la capacità  
durante i periodi di  
inattività.



# Notifica status operazioni

1

Recupero della **connectionId** a partire dall'**orderId**

2

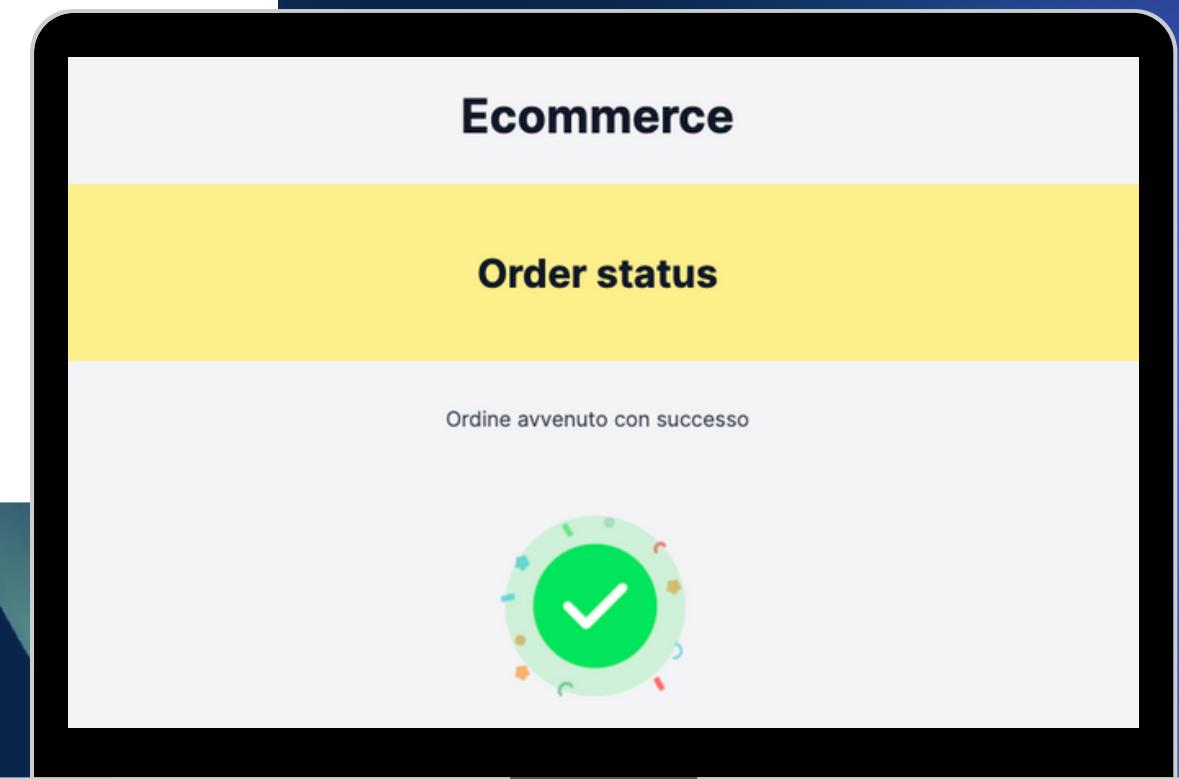
Invio del payload sottoforma di stringa sul canale di comunicazione avente la **connectionId** relativa

3

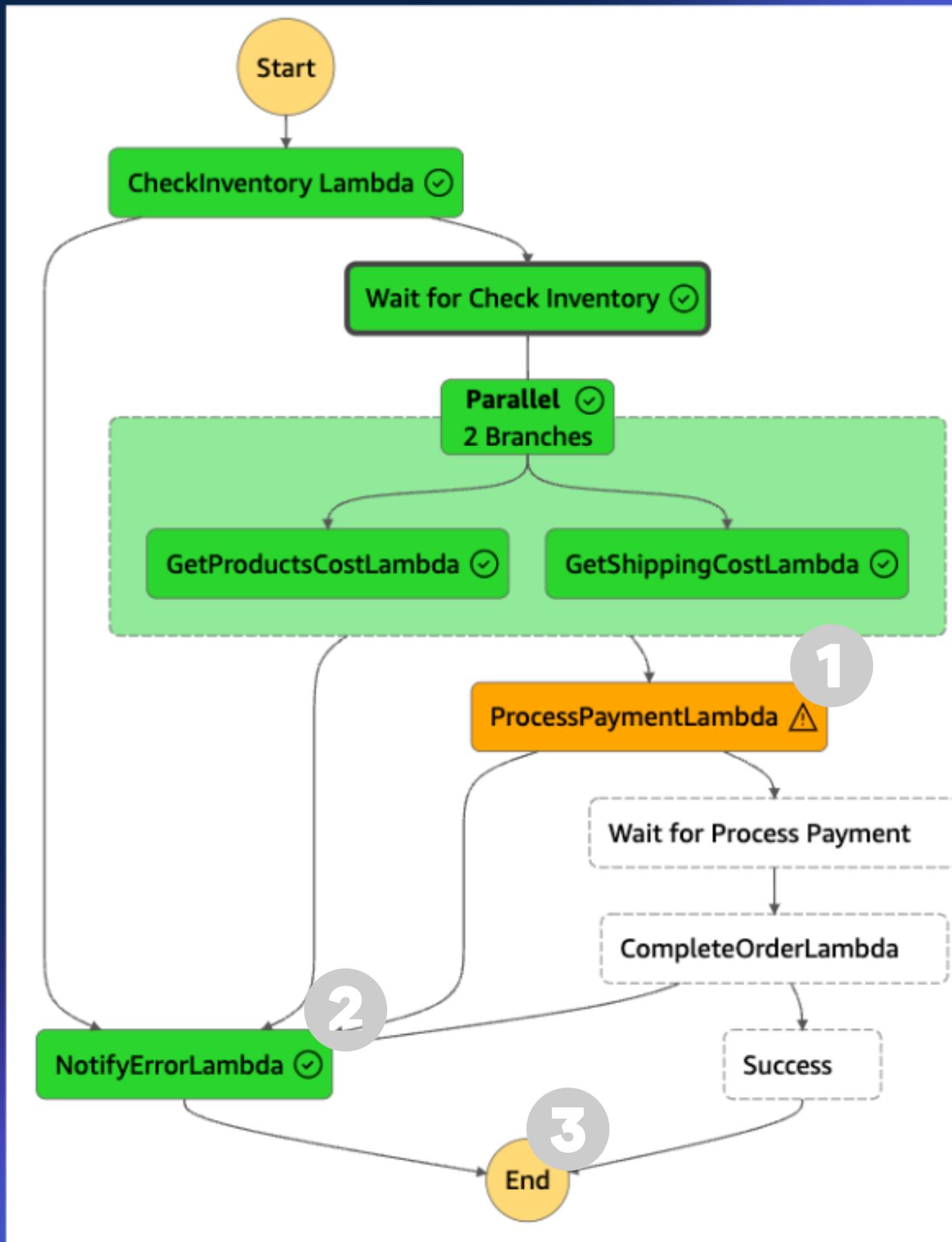
Ricezione a frontend del messaggio e visualizzazione a schermo

```
// API Gateway Management API client initialization
const apigatewaymanagementapi = new ApiGatewayManagementApiClient({
  apiVersion: '2018-11-29',
  endpoint:
    'https://wkr7p95088.execute-api.eu-central-1.amazonaws.com/production/',
})

// Prepare and send the WebSocket message using the connection ID
const paramsApiGateway = {
  ConnectionId: connectionId,
  Data: JSON.stringify(messageObj),
}
const postToConnectionCommand = new PostToConnectionCommand(paramsApiGateway)
await apigatewaymanagementapi.send(postToConnectionCommand)
```



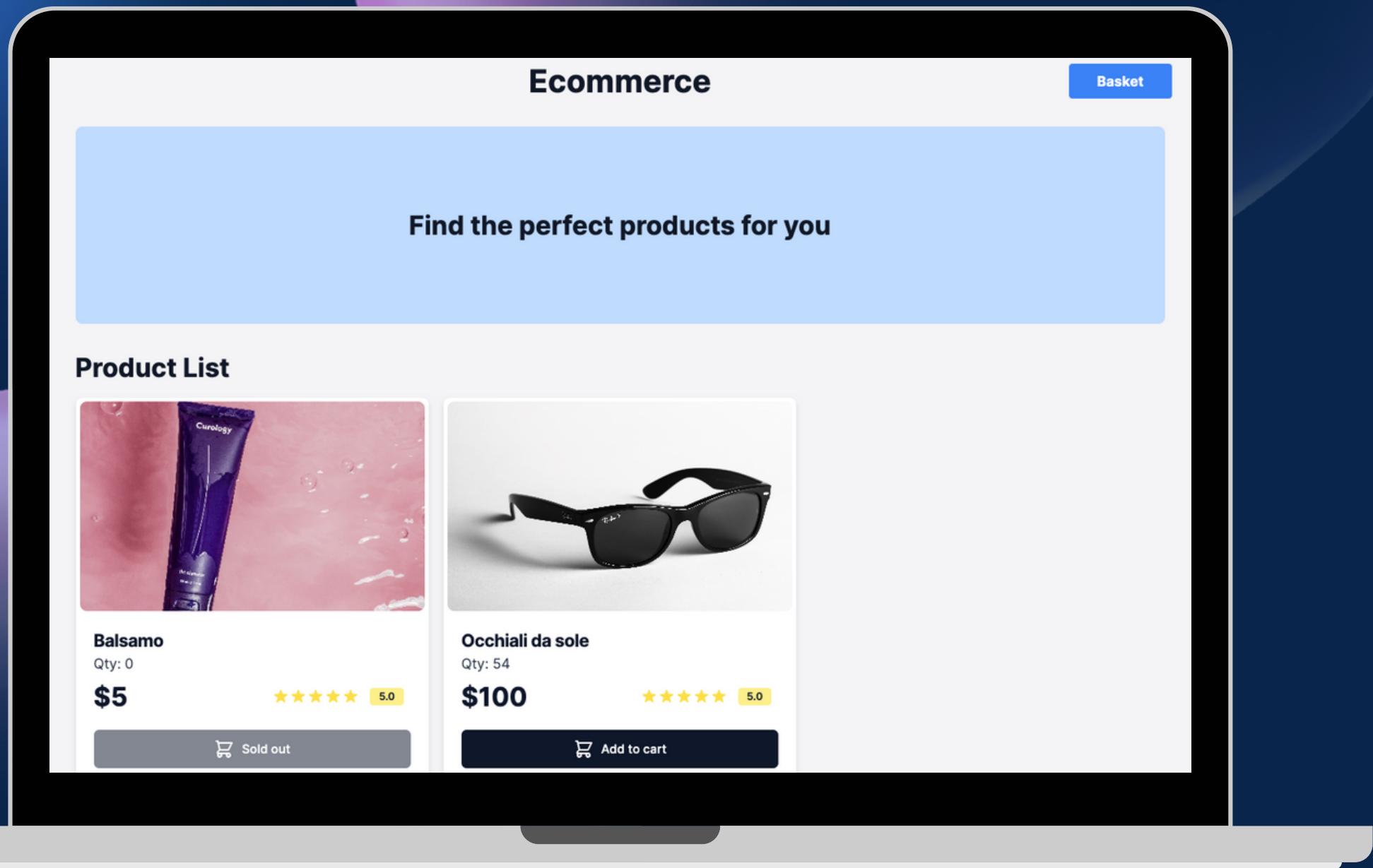
# Error handler



1 La lambda ProcessPayment scatena l'errore.

2 AWS Step Functions gestisce l'errore ed interrompe il flusso regolare deviandolo alla funzione di NotifyError.

3 Terminazione del workflow.



Alberto Rizzi



Relazione e codice