Setting Up Asynchronous Communication between ASP.NET Core Microservices

Gill Cleeren
CTO XPIRIT BELGIUM

@gillcleeren   www.snowball.be

Overview

Adding asynchronous communication

Using a bus for communication

Working in the background

Polling a service

Solving the eventual consistency problem

Adding Asynchronous Communication

Limiting communication is the first step
- Autonomous microservices

Shopping basket service
- Send to order service

Asynchronous communication is preferred
- Less impact if things go wrong

Use async all the time?

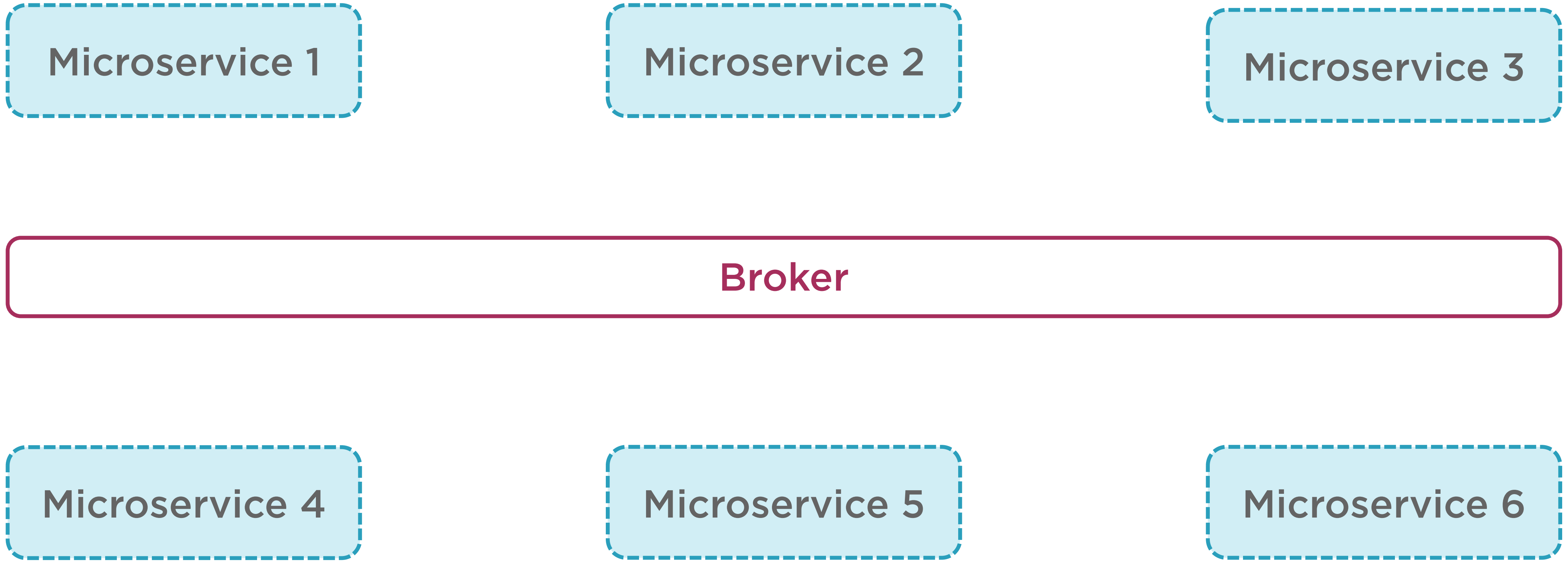
- Some data will need to be replicated
- Integration events

Async communication uses dumb pipes and smart endpoints

- Mostly between the services
- Outside to services can be sync

Broker-based

Asynchronous Communication

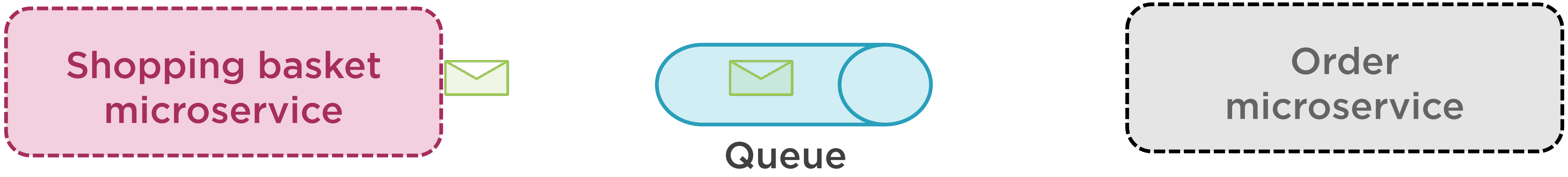


Communication Options

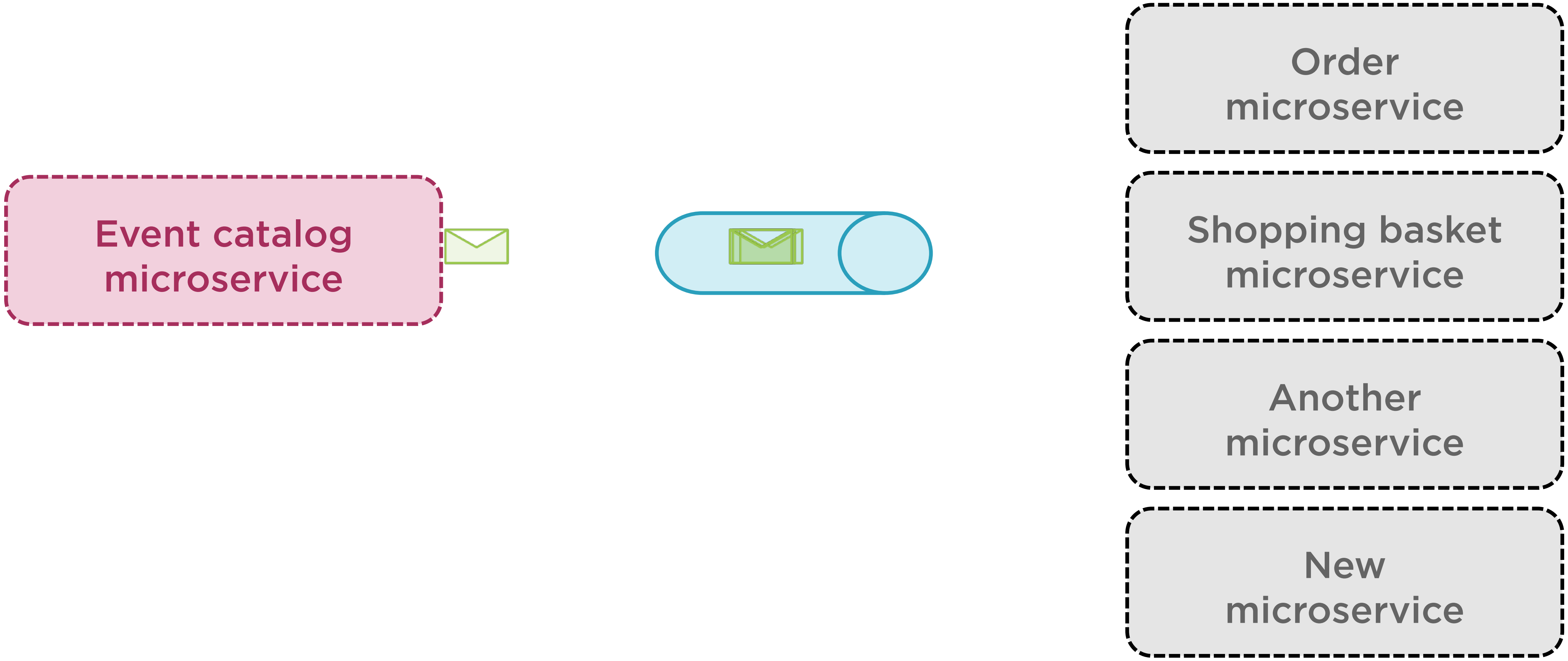
Point-to-point

Publish-subscribe

Point-to-point Communication



Publish-subscribe Communication



Asynchronous Communication

Flexible

Scalable

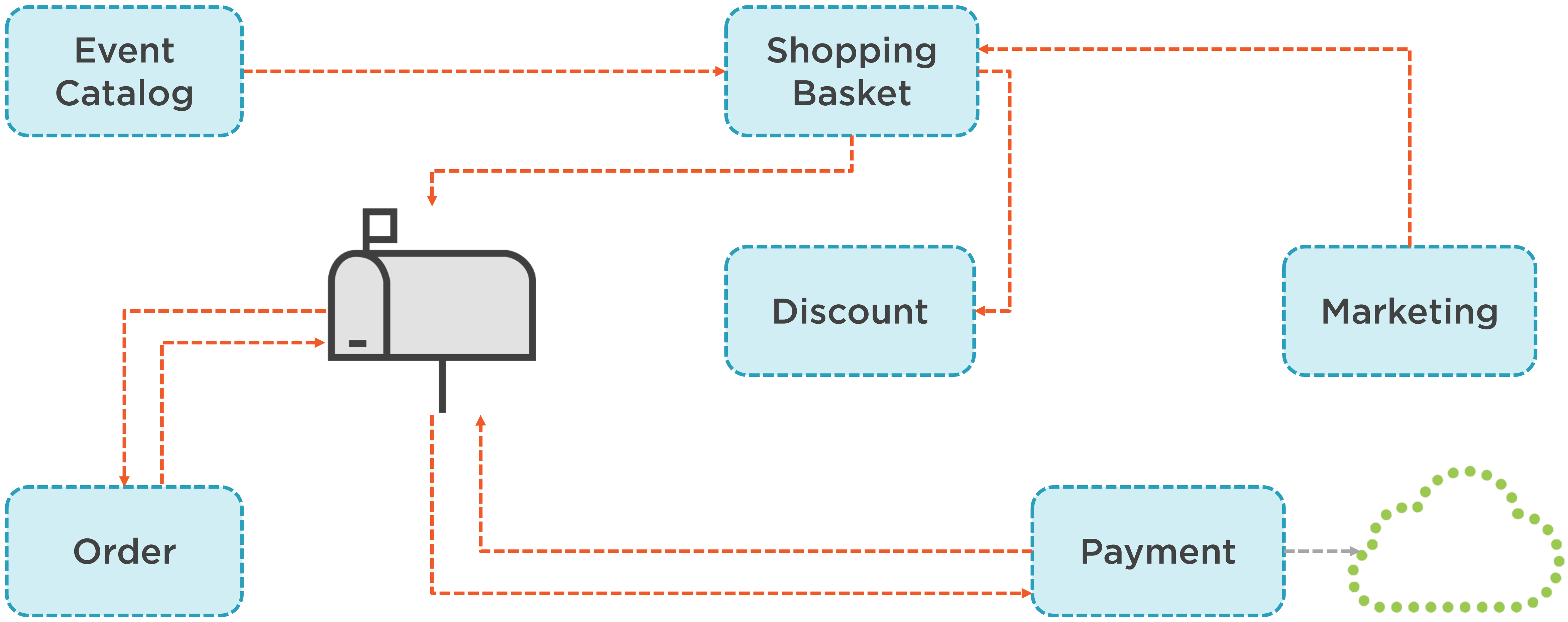
Harder “to follow”

Demo

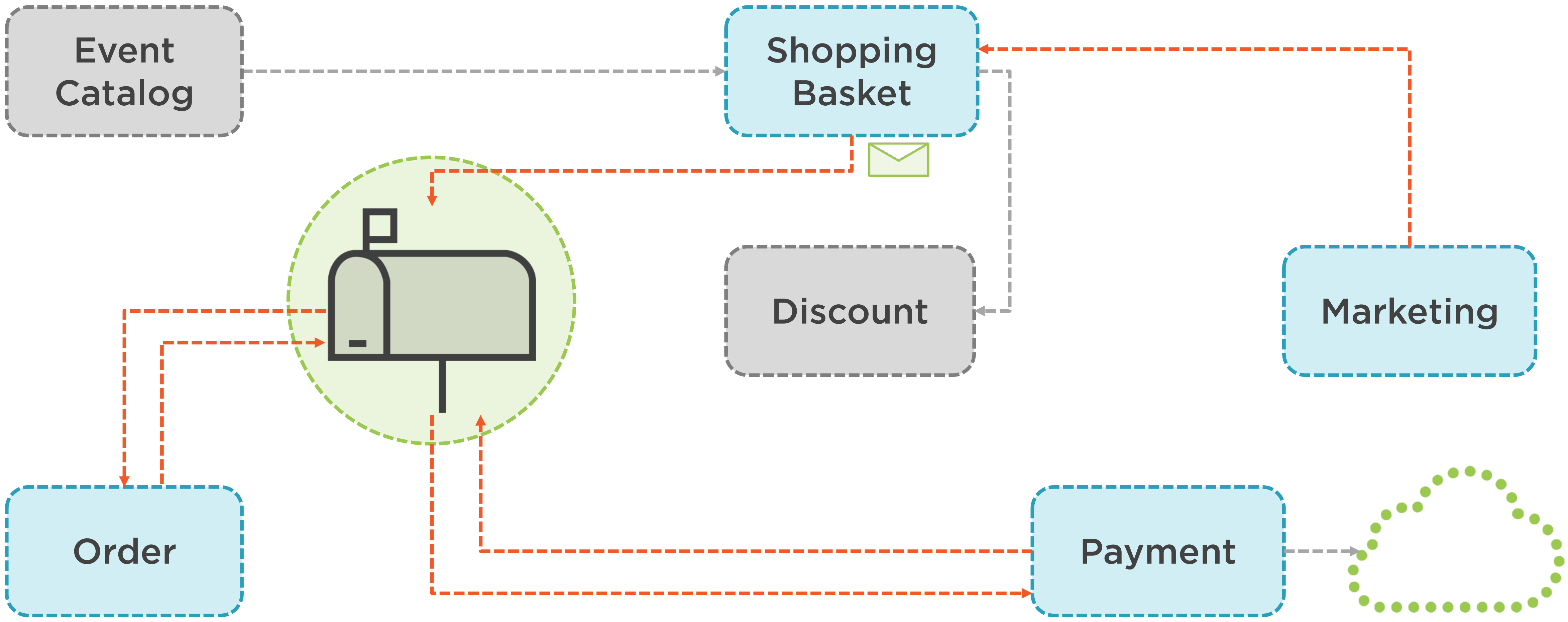
Running the application for this module

Using a Bus for Communication

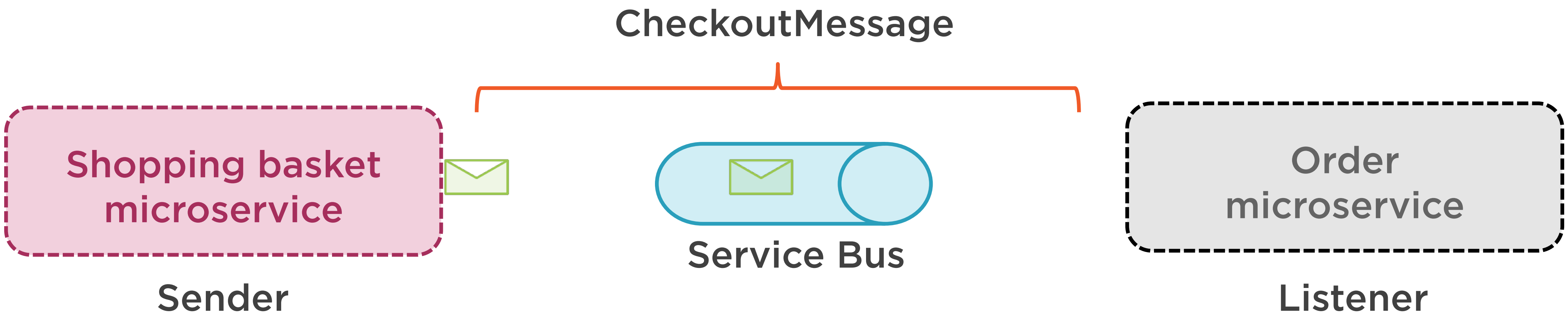
Asynchronous Communication in GloboTicket



Asynchronous Communication in GloboTicket



Sender and Listener



Messages and Events

Messages

Expect action to be taken
Full data included

Event

Something has happened
Typically small

Different Options

Azure Service Bus

RabbitMQ

NServiceBus

Azure Service Bus

- Messaging system
- Cloud-based integration
- Dead-lettering
- Scalable

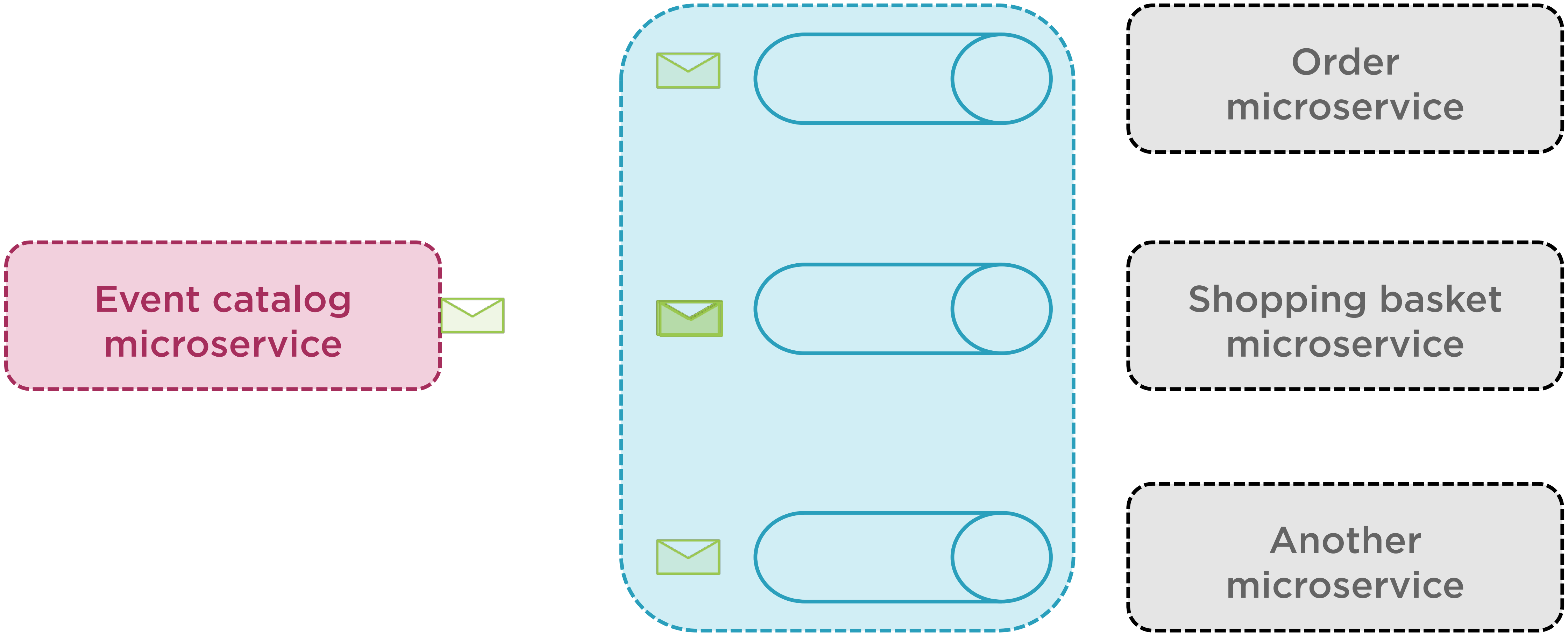
Concepts of Azure Service Bus

Queue

Topic

Subscriber

Publish-subscribe Communication



Demo

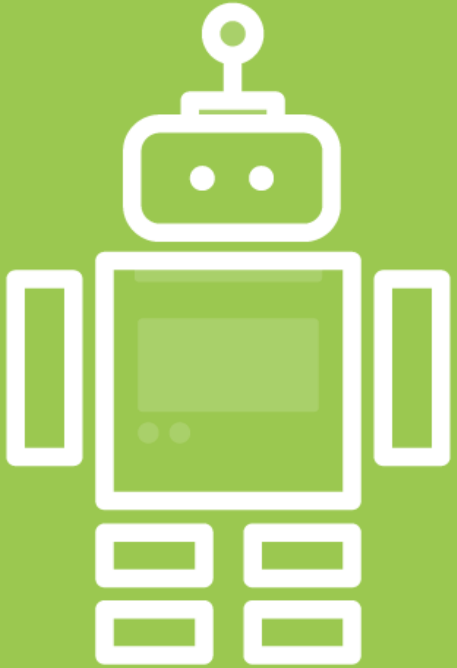
Setting up Azure Service Bus

Demo

Posting messages from the Shopping Basket Service

Reading message asynchronously in the Order Service

Working in the Background to Handle Payments



Adding a new service

Most often based on business capability.

Technical capability can be a driver too to decide and create a new service.

Adding the Payment Service

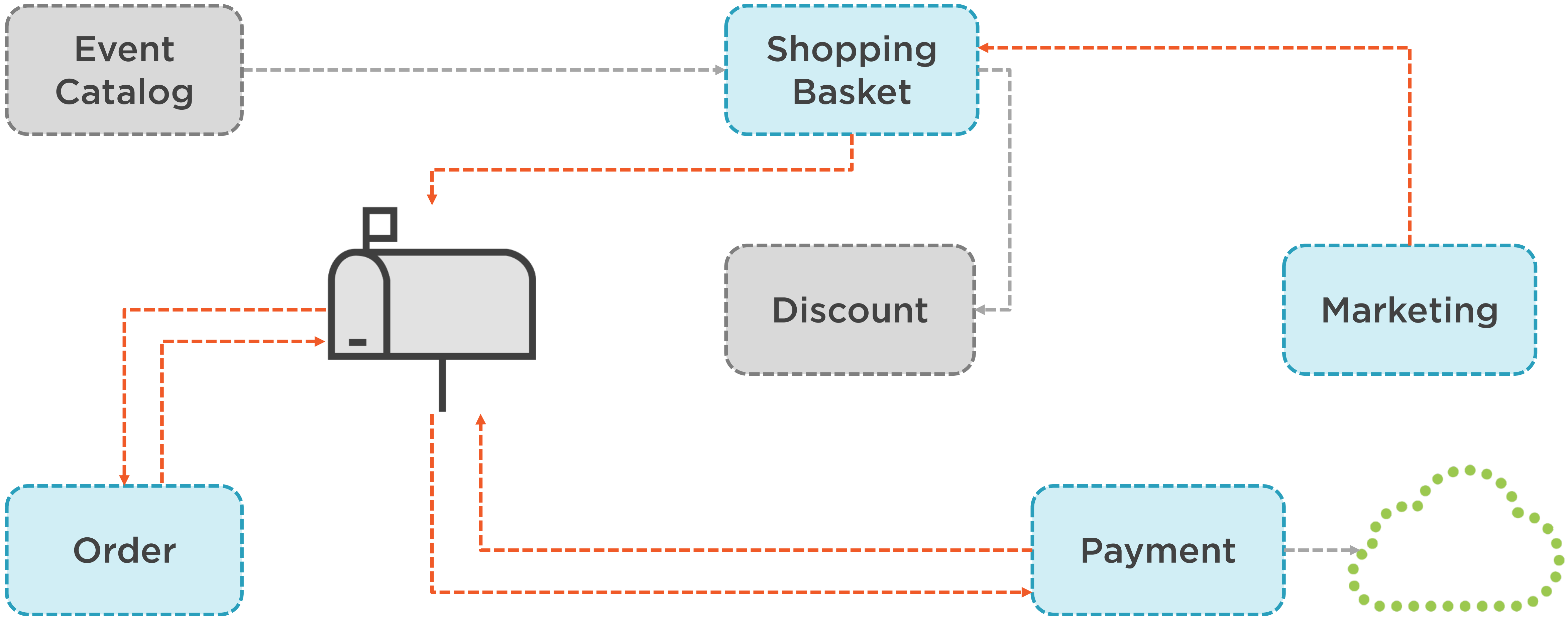
Technical microservice

Support other microservices
Keep other services clean
Reliabilty

Integrate with external payment system

SOAP, other protocol
Knowledge about payment data and format

Asynchronous Communication in GloboTicket



ASP.NET Core Hosted Service

- Background task
- IHostedService
- Keeps running

```
public interface IHostedService
{

    Task StartAsync(CancellationToken cancellationToken);

    Task StopAsync(CancellationToken cancellationToken);

}
```

Hosted Service

```
services.AddHostedService<ServiceBusListener>();
```

Registering the Hosted Service

Will run a singleton instance

Triggered when a message comes in

Alternative: Azure Function with Service Bus Trigger

Demo

Creating the payment service

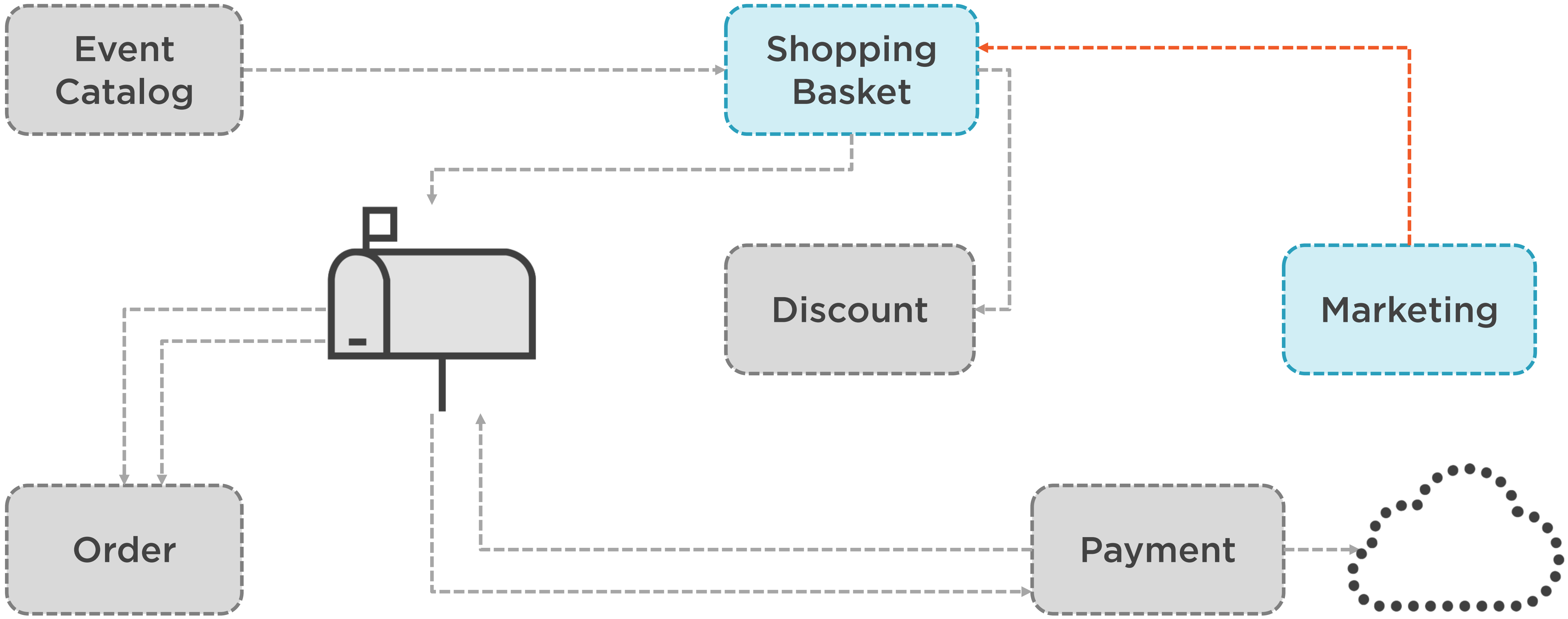
Reading information from the service bus

Polling a Service

Storing events

- User added a ticket to the basket
- Other service can poll periodically
- Multiple services can subscribe

Asynchronous Communication in GloboTicket



Demo

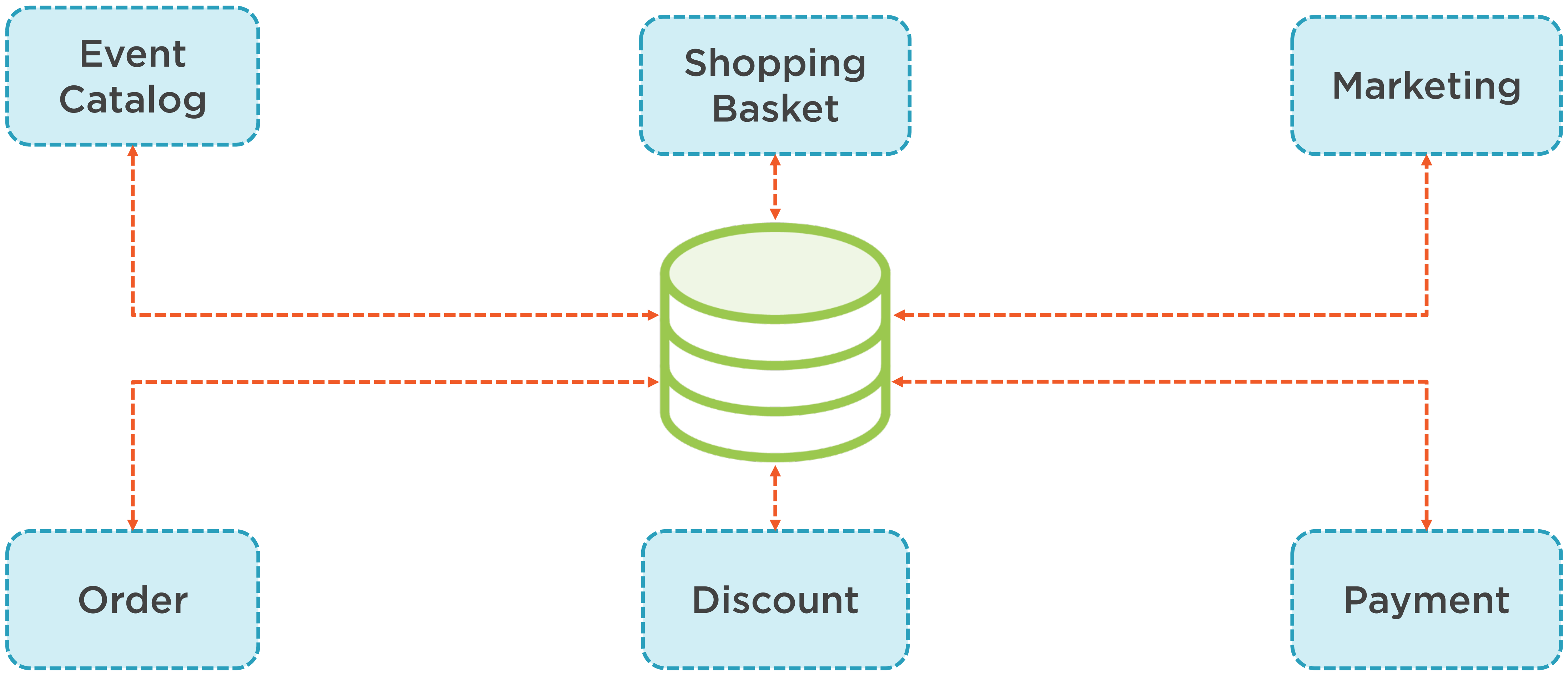
Storing events in the shopping basket

Adding the marketing service

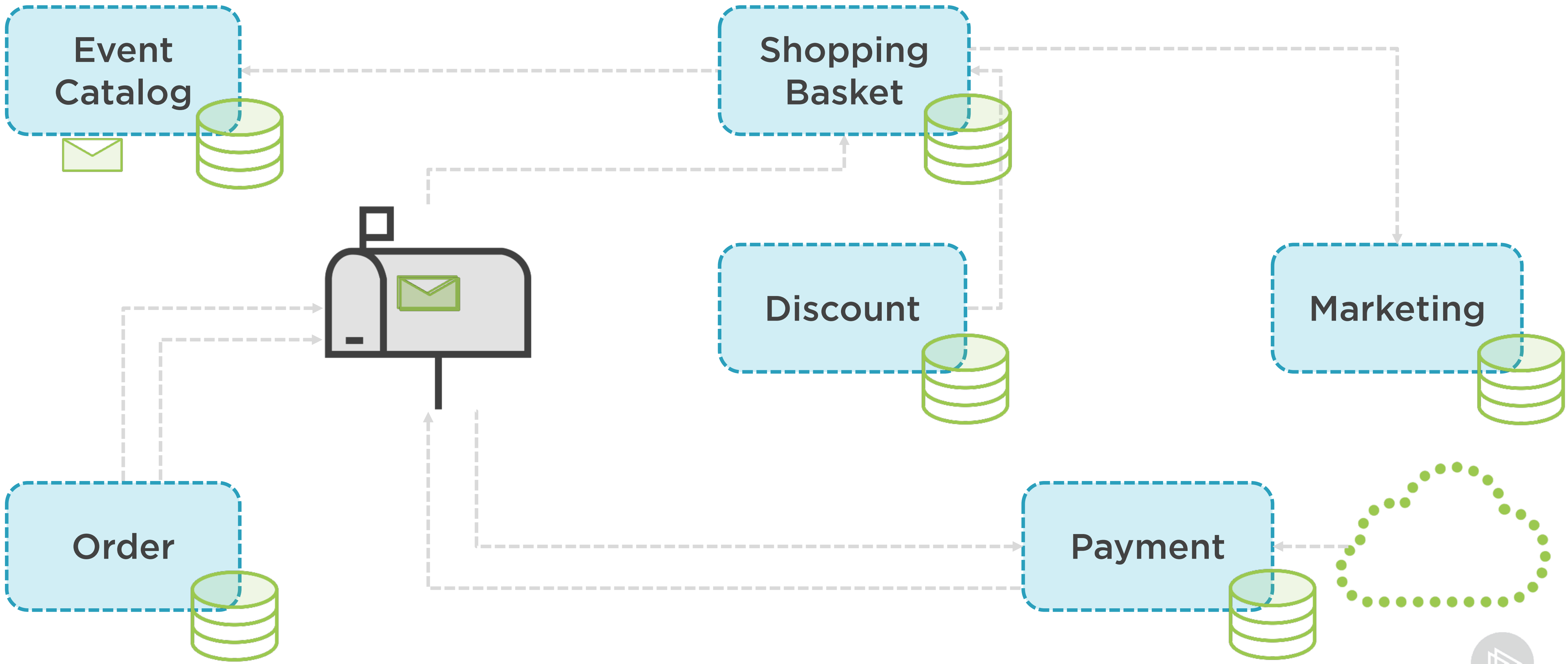
Reading basket information

Solving the Eventual Consistency Problem

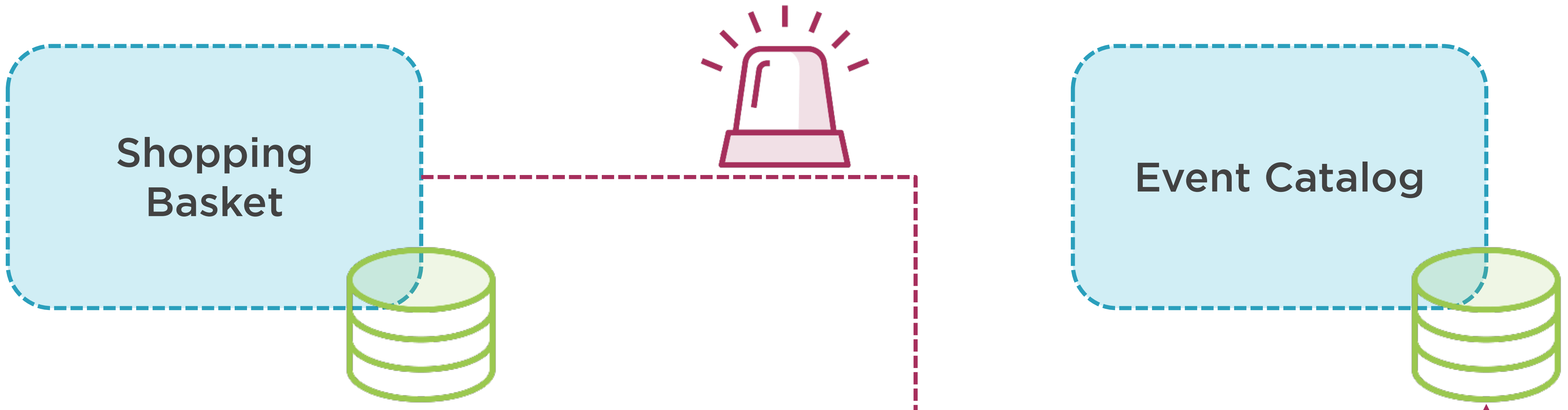
Consistency in a Monolithic Application



Data Changes in GloboTicket



Updating a Database in a Different Microservice



Demo

Exploring the data consistency problem in GloboTicket

Eventual Consistency

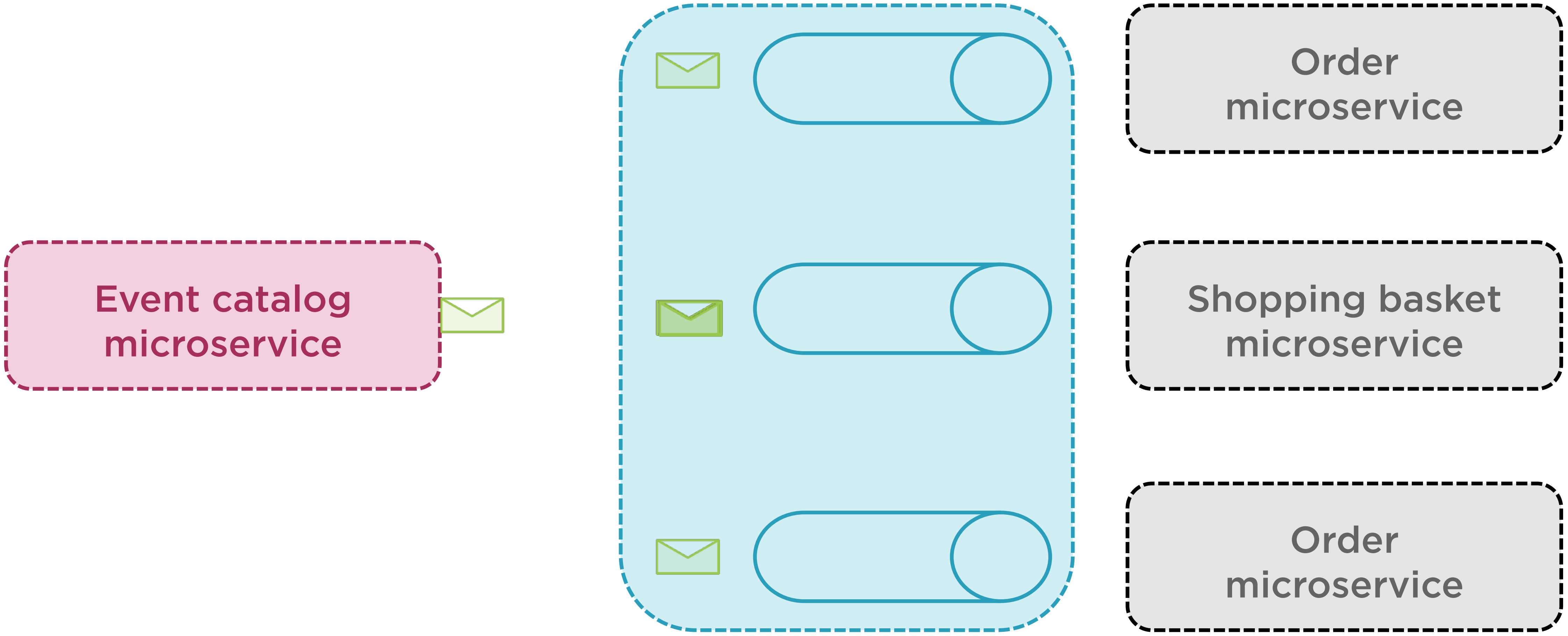
Common in distributed systems

Data may change in one service

Other services will be consistent at some point

Awareness

Adding Integration Events



Events

Let others know that something has changed.

Fire and forget.

Can cause other (integration) events to trigger.

Demo

Solving eventual consistency

Adding an integration event

Summary

Asynchronous communication is a natural fit for microservices

Based around service bus

Integration events for data sync between different microservices

Up next:

Adding resiliency to the services