



Microservices Architecture

Aleksandar Dimic

What is a microservice?

What is a microservice architecture?

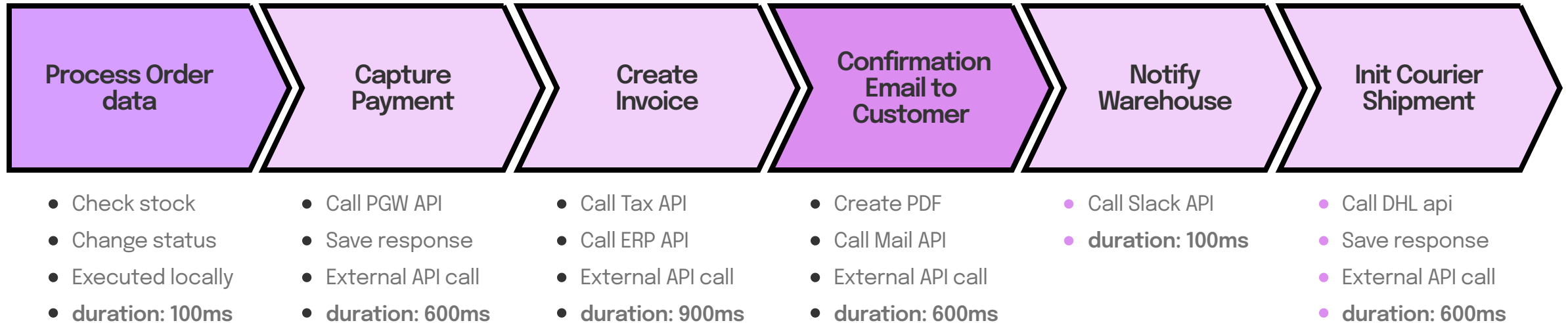
- A large application is broken down into smaller, independent (micro) services
- **Each microservice:**
 - Does one thing well - single responsibility
 - Can be developed, deployed, and scaled independently
 - Has its own database or data storage (**in theory**)
 - Communicates with other services
- **Main goal - improve performance & scalability**

how to transfer

Monolith to Microservice

Order Confirmation on Monolith Arhc.

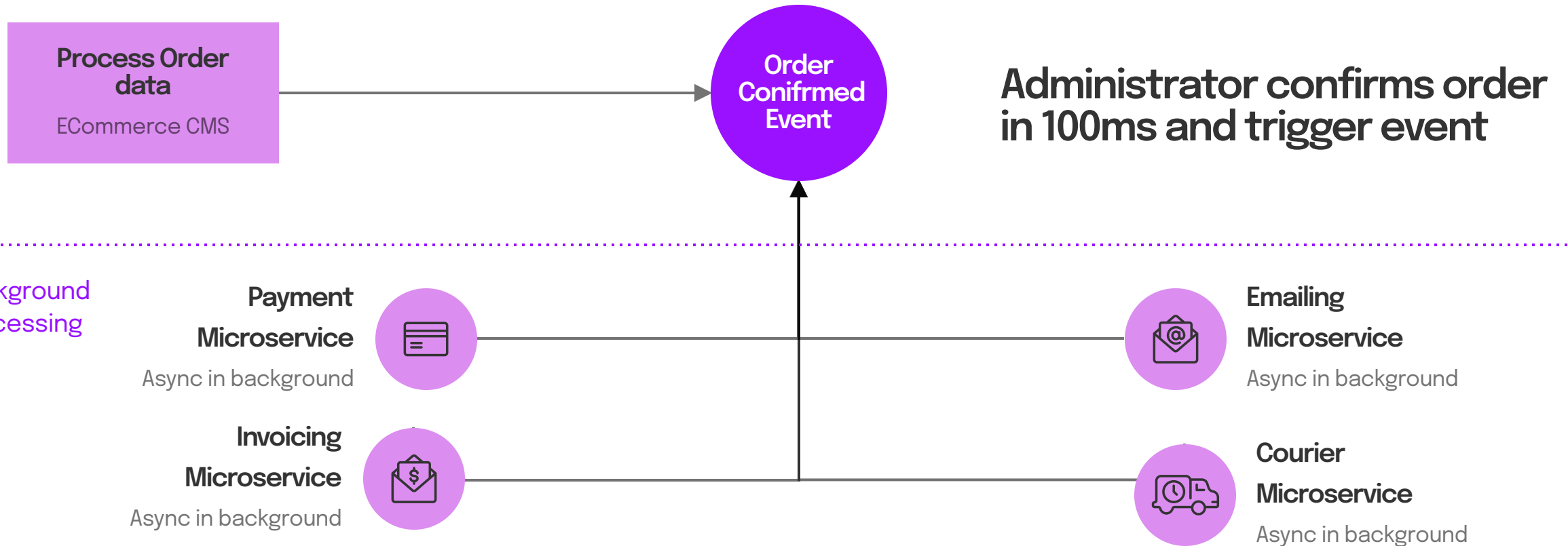
Administrator confirms one order.



Total process duration for one order. ~3 seconds

What about bulk confirmation for 50 orders???

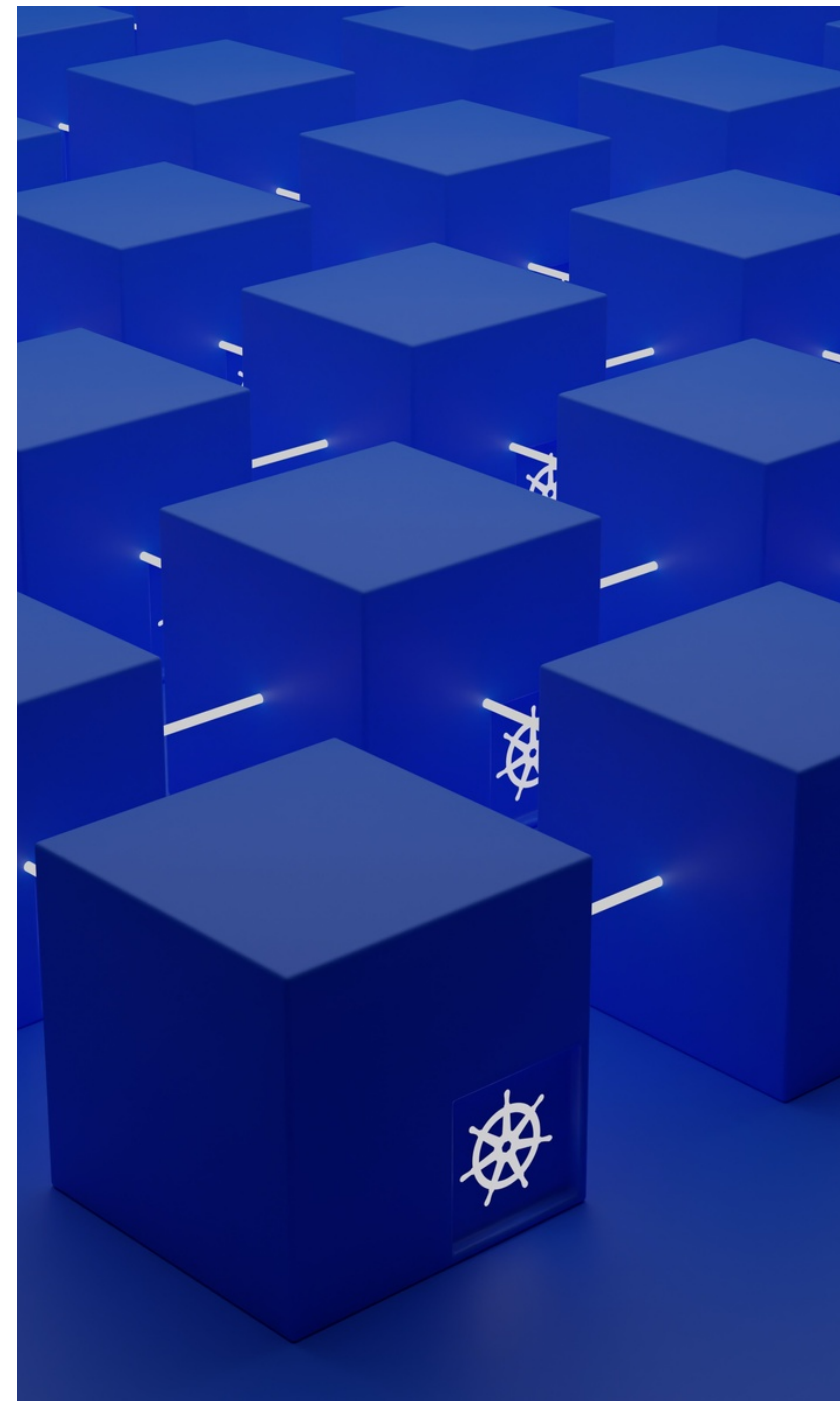
Order Confirmation using Microservices



Microservices run in background listening to events

How microservices communicate?

- Asynchronous (Message Brokers)
Previous example - Increase performance
- Synchronous (Service Discovery)
Improves scalability



Message Brokers

- **Async communication**

Service A sends a message to the **broker**, and then Service B picks it up **later**—whenever it's ready

- **Message (Job) Queues**

Messages (Jobs) are executed by queue workers

- **Event driven (publish-subscribe)**

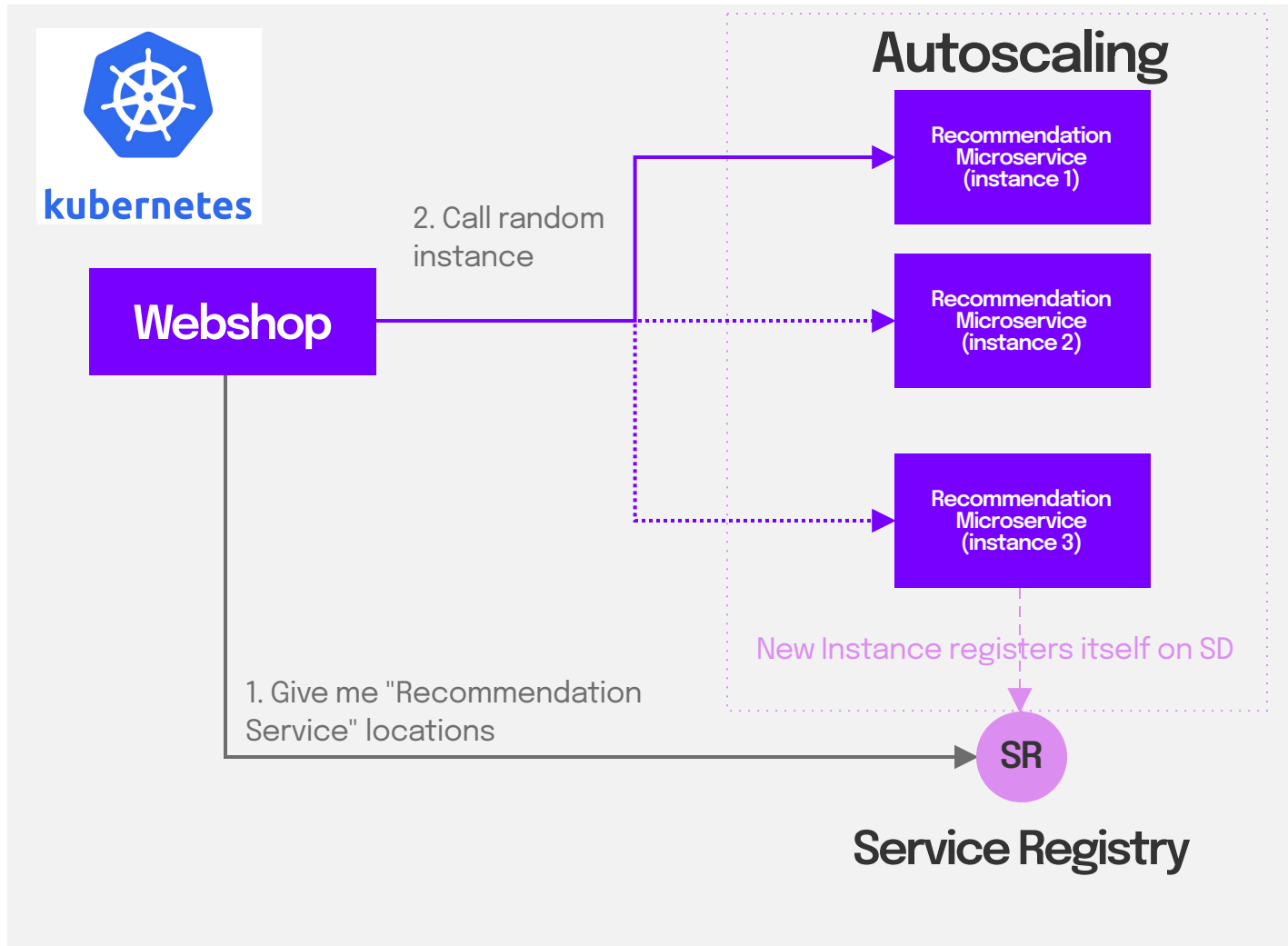
Message (Event) is published and subscribers are notified

- **Goal: Service decoupling**

Add different listeners regardless of what generates event



Service Discovery - Synchronous



- **Service Registry keeps locations of all microservices**
- **Used in distributed systems (kubernetes)**
- **Goal: Scale synchronous communication**
- **Best known service registries:**
 - Netflix Eureka
 - Consul
 - Zookeeper

Why is containerization important?

- **Isolation**

Each microservice runs in its own container, with its own dependencies. No conflicts.

- **Portability**

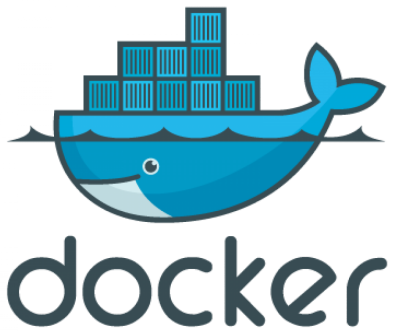
Containers work the same on any environment—local dev, staging, production, or cloud.

- **Scalability**

Containers can be easily scaled with orchestrators like Kubernetes.

- **Fast Deployment**

Containers start quickly, making deployments and rollbacks smoother.



1

Break application into smaller parts

Identify processes and organize microservices.

NOT EVERYTHING IS A MICROSERVICE!

2

Use appropriate frameworks

Java Spring - defacto microservice standard

NextJS - NodeJS framework

3

Containerize applications

Pack your application into Docker Images.

It will help you versioning your deployments.

4

Implement CI/CD

Define procedures for deployment and implement them on some CI/CD platform

All major GIT platforms offer CI/CD

5

Message Brokers & Service discovery

Use **Kafka, RabbitMQ, or SQS** for async communication between services.

Implement **service discovery** to load balance

6

Orchestrate with Kubernetes

Use **Kubernetes** to deploy, manage, scale, and monitor containers.

How reach microservice architecture?

Explore microservices further

Thank you for listening!