Communicating Between Microservices

Mark Heath
CLOUD ARCHITECT

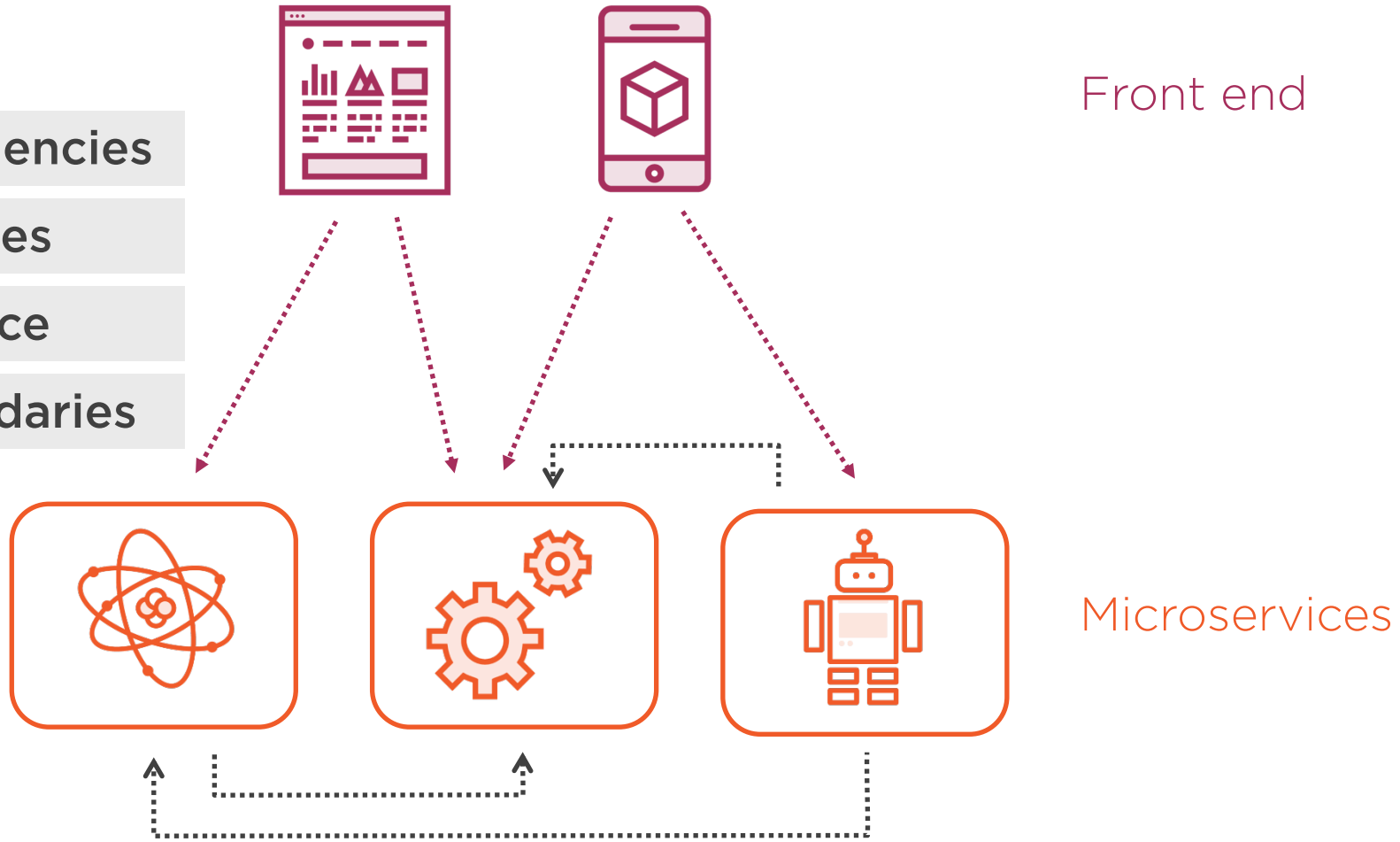
@mark_heath www.markheath.net

Overview

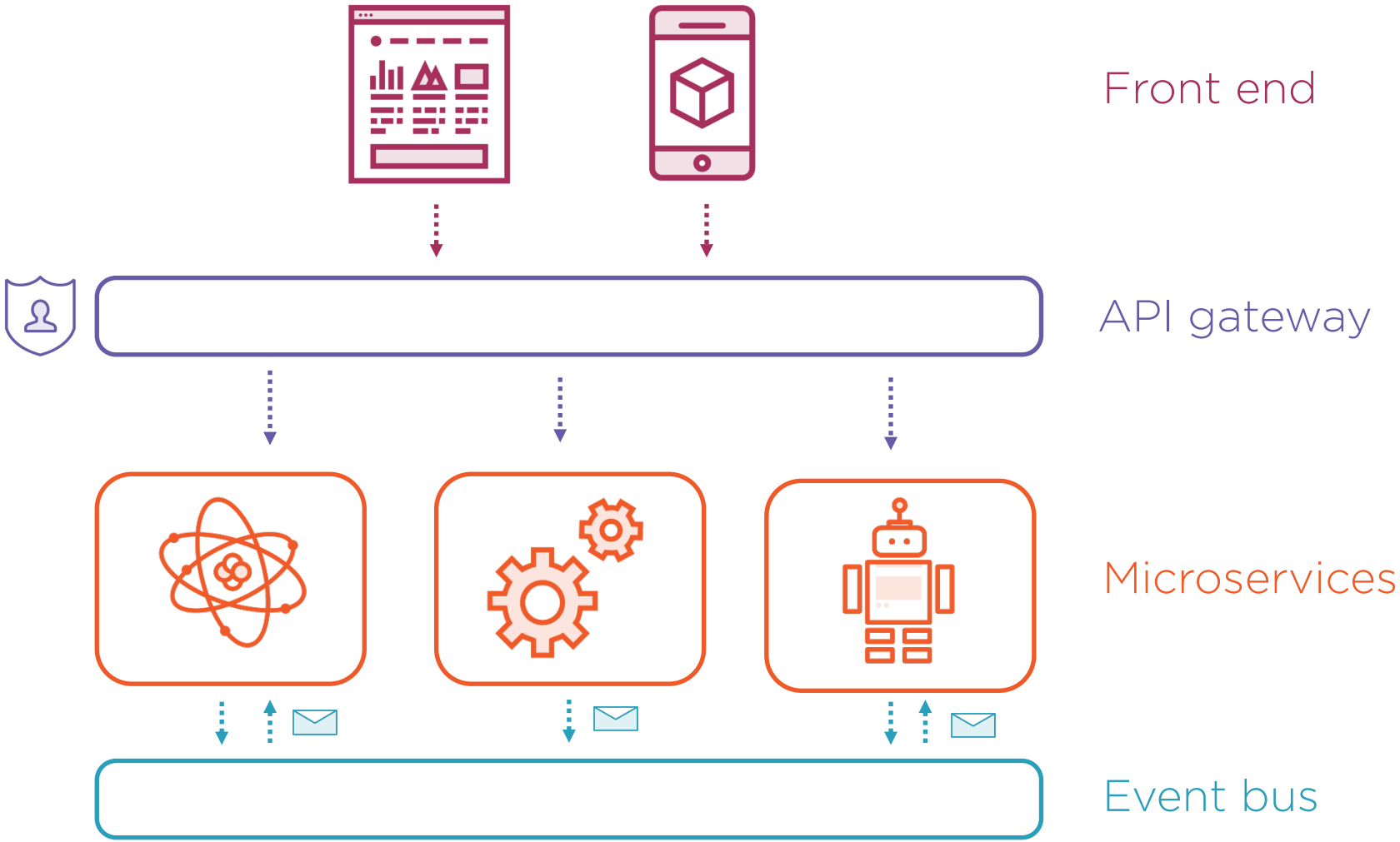
- Should microservices call each other?
- Should client apps directly call microservices?
- API gateways
- Synchronous and asynchronous communication
- RESTful APIs
- Messaging patterns
- Resilient communications
- Service discovery

Calling Microservices

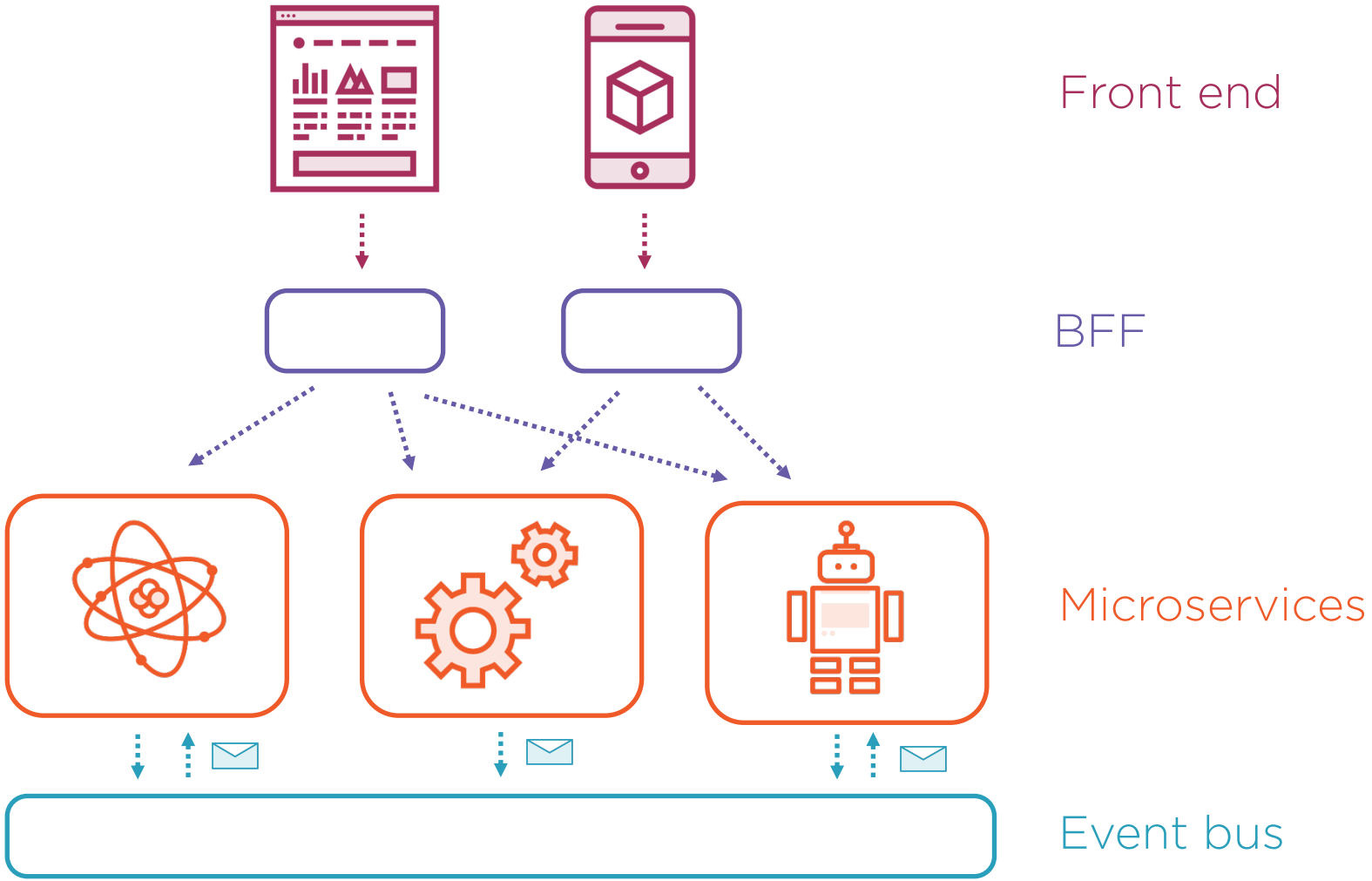
- Tangled dependencies
- Cascading failures
- Poor performance
- Misplaced boundaries



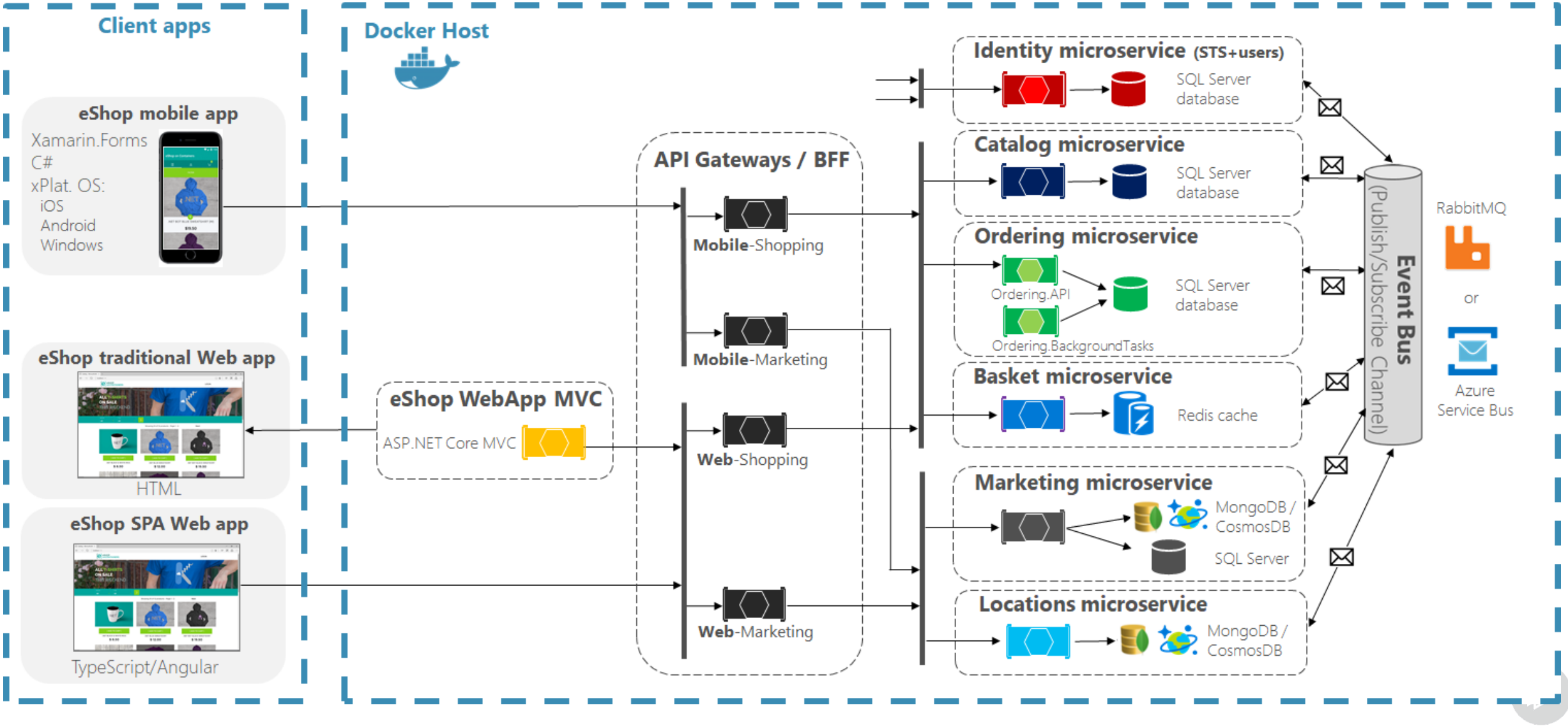
Microservice Communication Patterns



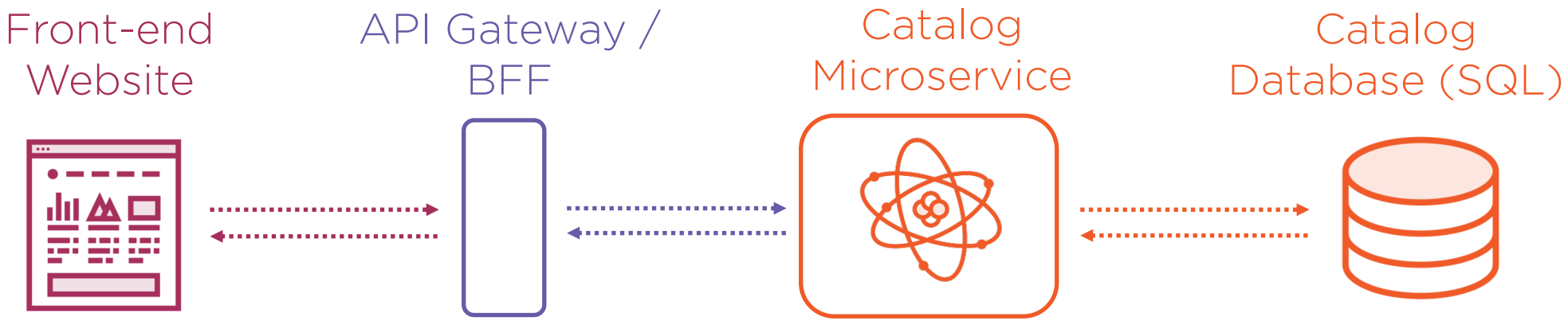
Backend for Frontends (BFF)



eShopOnContainers Architecture



Synchronous Communication

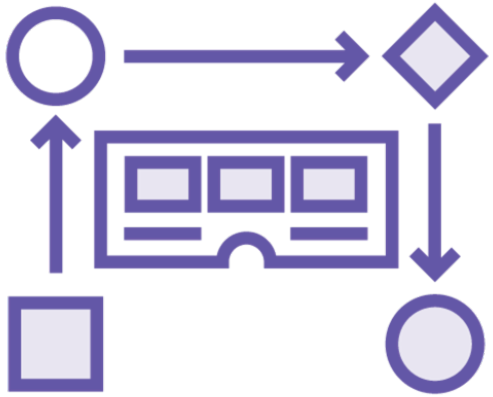


Performance is important

HTTP

Industry standard

Easily used by any programming language



Standard error codes

Caching

Proxies

Payloads

JSON or XML

RESTful APIs

Represent information as “resources”

- e.g. CatalogItem, Order

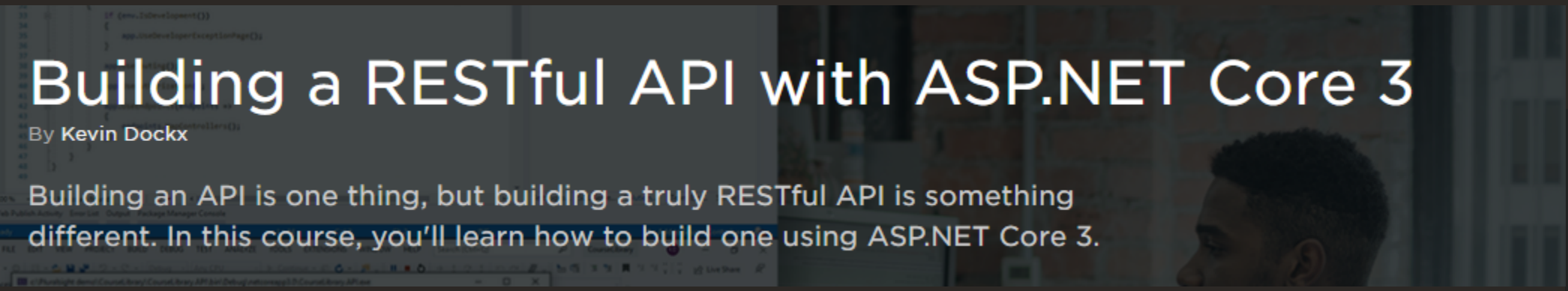
Use standard HTTP methods:

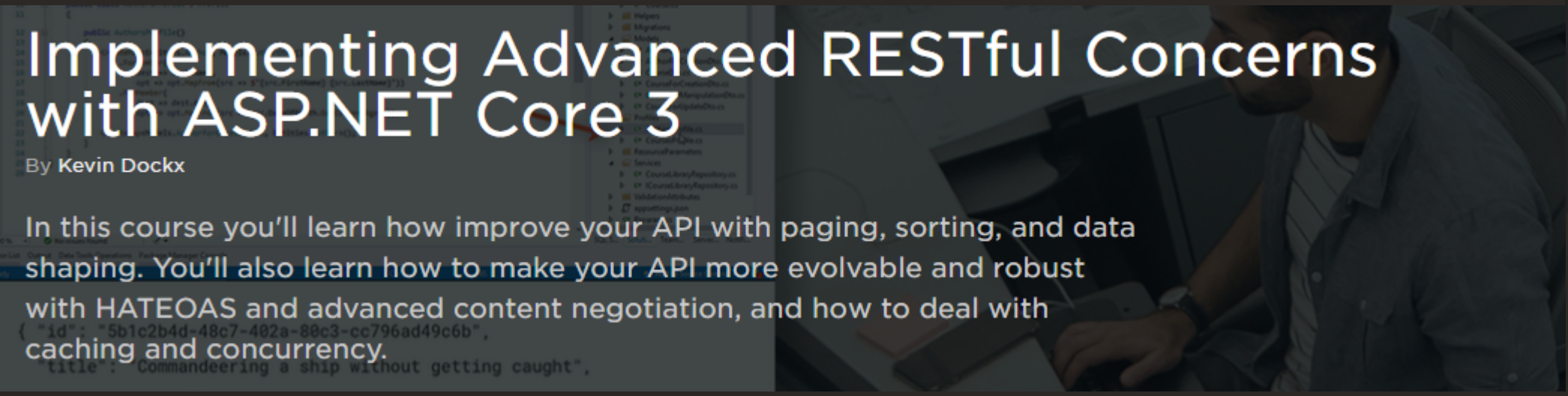
- GET to retrieve
- POST to create
- PUT to update

Use HTTP status codes

Media type headers (Content-Type)

Learn More About RESTful APIs





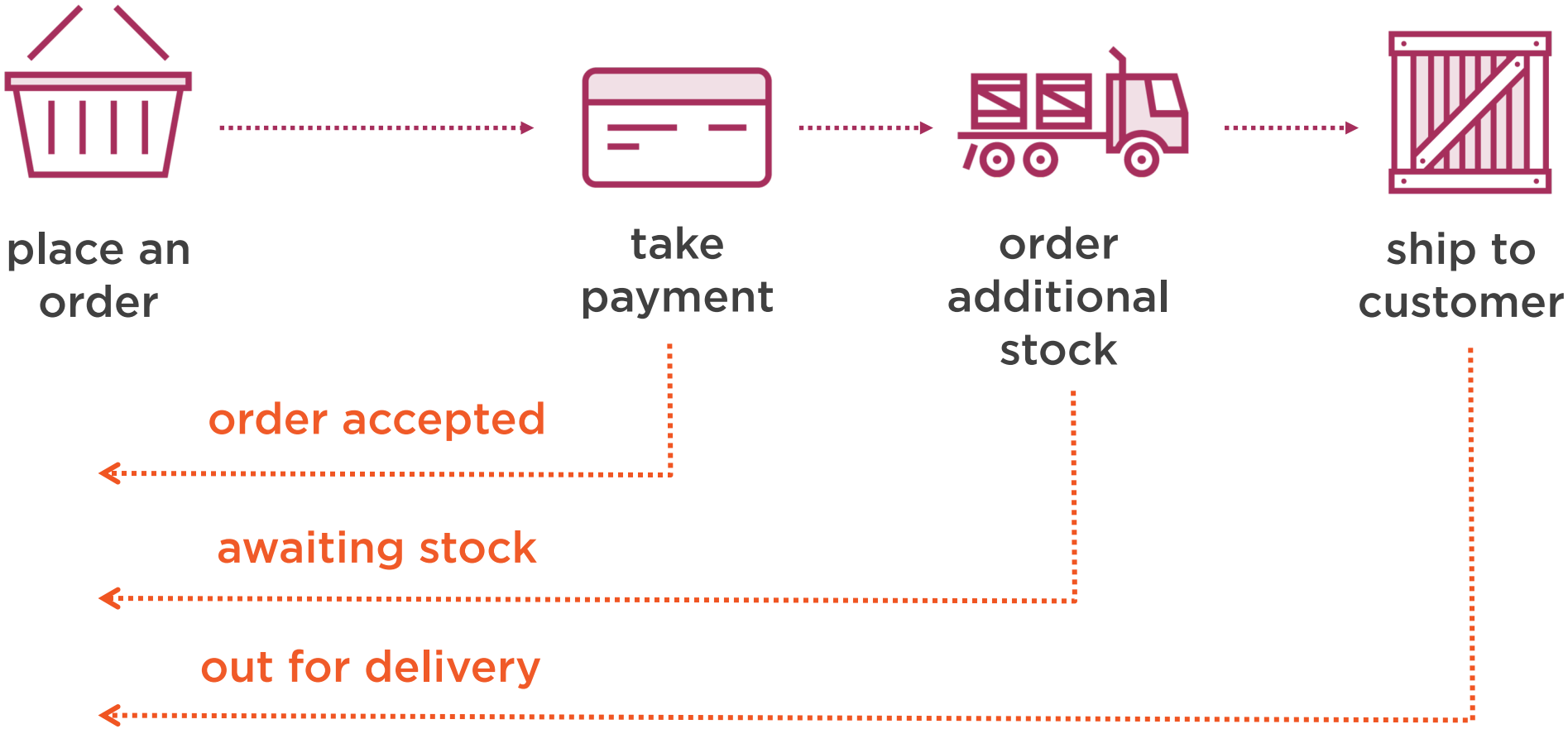
Course author

Kevin Dockx

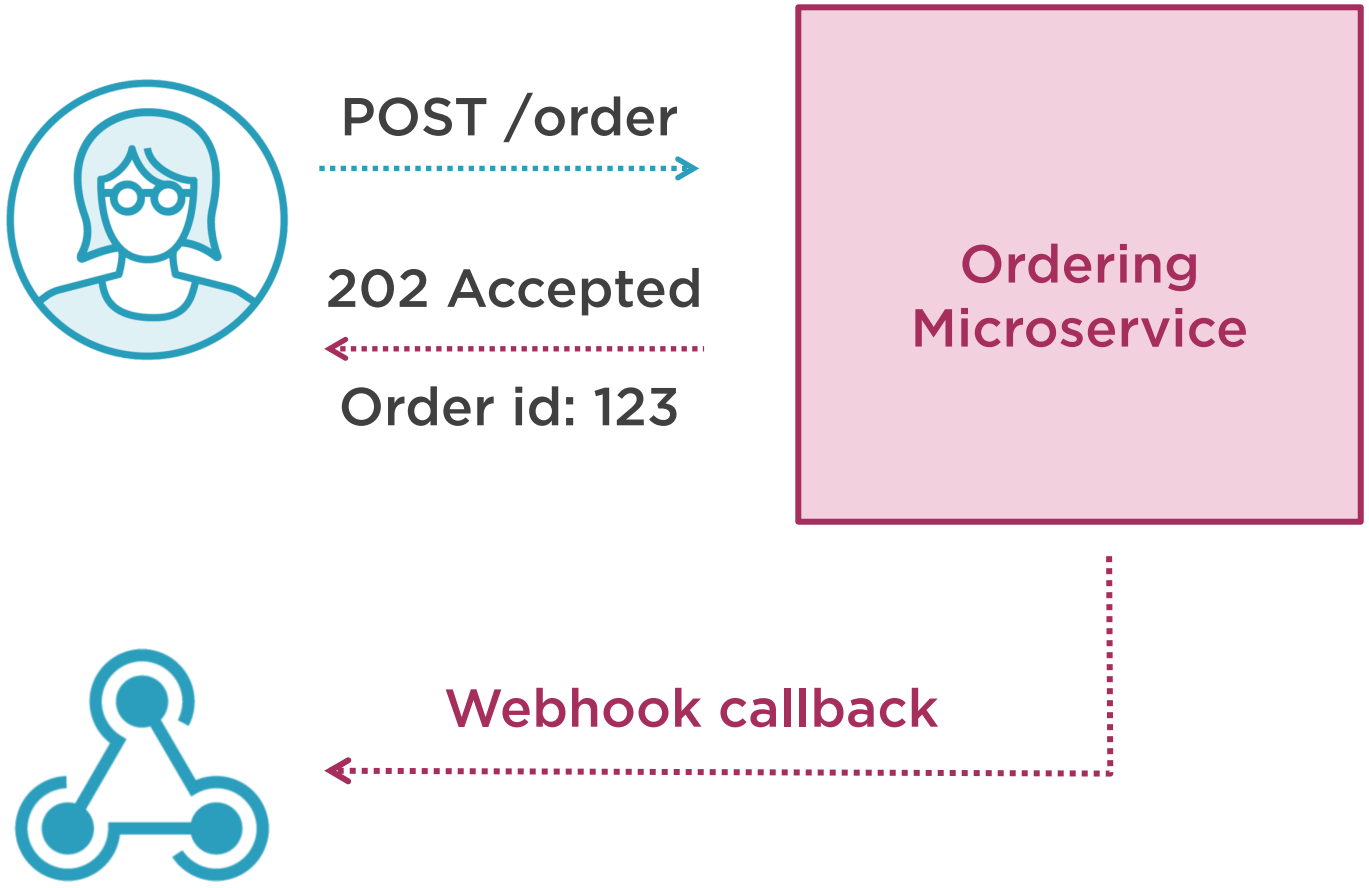
Kevin Dockx is a freelance solution architect, author & consultant, living in Antwerp (Belgium). These days he's mainly focused on RESTful architectures & security for web applications and mobile...

https://www.pluralsight.com/courses/asp-dot-net-core-3-restful-api-building

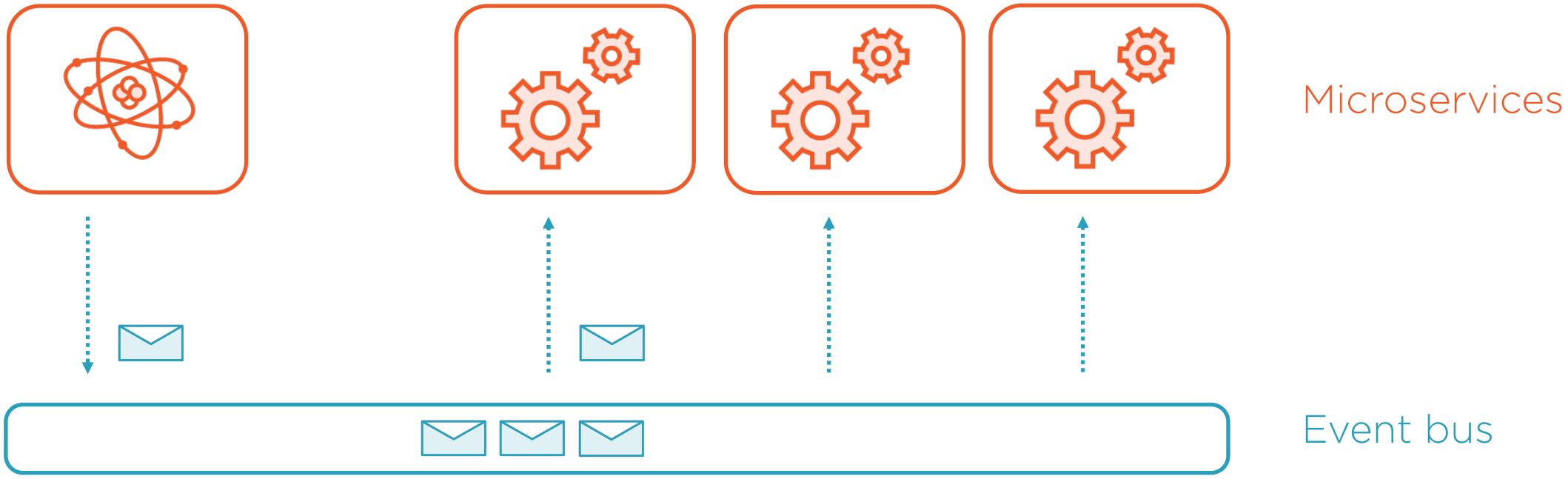
Asynchronous Communication



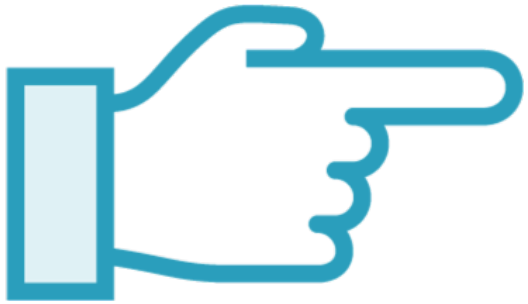
Asynchronous Communication over HTTP



Asynchronous Communication via Messaging



Message Types



Commands

"Do this please"

Post a command

e.g. SendEmail

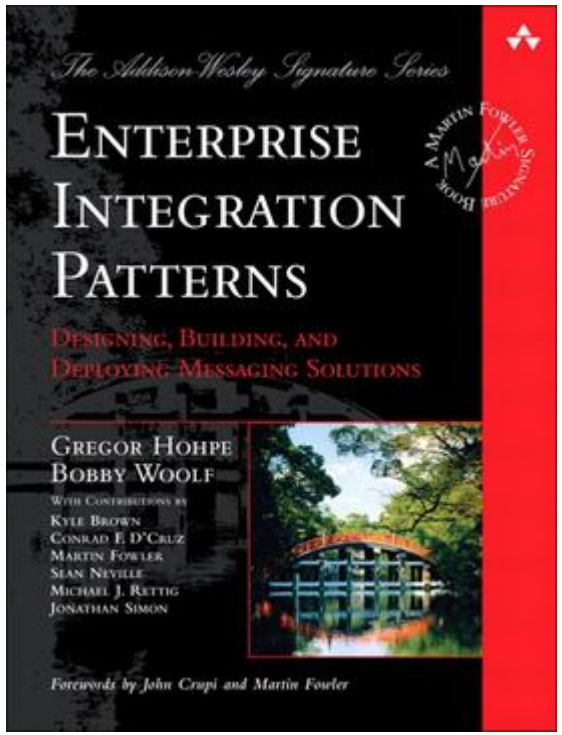
Events

"This happened"

Subscribe to an event

e.g. OrderPlaced

Book Recommendation

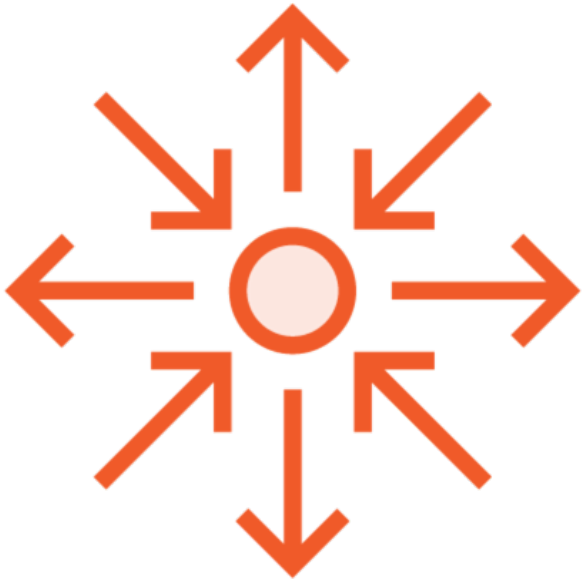


Enterprise Integration Patterns

Gregor Hohpe & Bobby Woolf

https://www.enterpriseintegrationpatterns.com/

Resilient Communication Patterns



Expect transient errors!

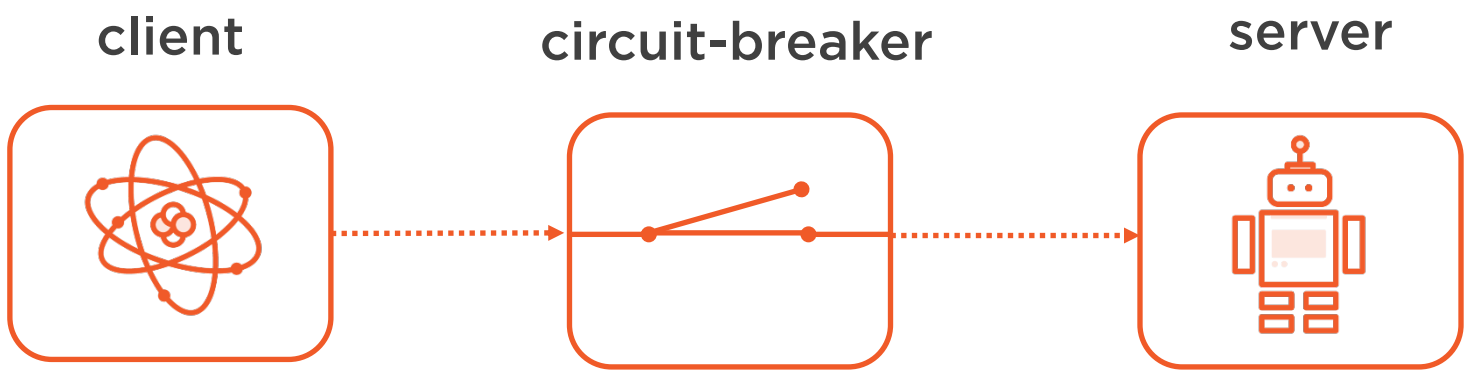
Beware of cascading failures

Implement retries with back-off

- e.g. using Polly in .NET

Circuit breaker

Circuit Breaker



Passes call through (circuit breaker is “closed”)

If enough errors are detected, blocks further calls (circuit breaker is “open”)

After a timeout, allows some calls through to see if the problem has been resolved

Caching can improve resilience

Fallback to cached data if the downstream service is temporarily unavailable.

Messaging Resilience

Messages can be posted if the recipient is offline

- It can catch up later

Message brokers support re-deliveries

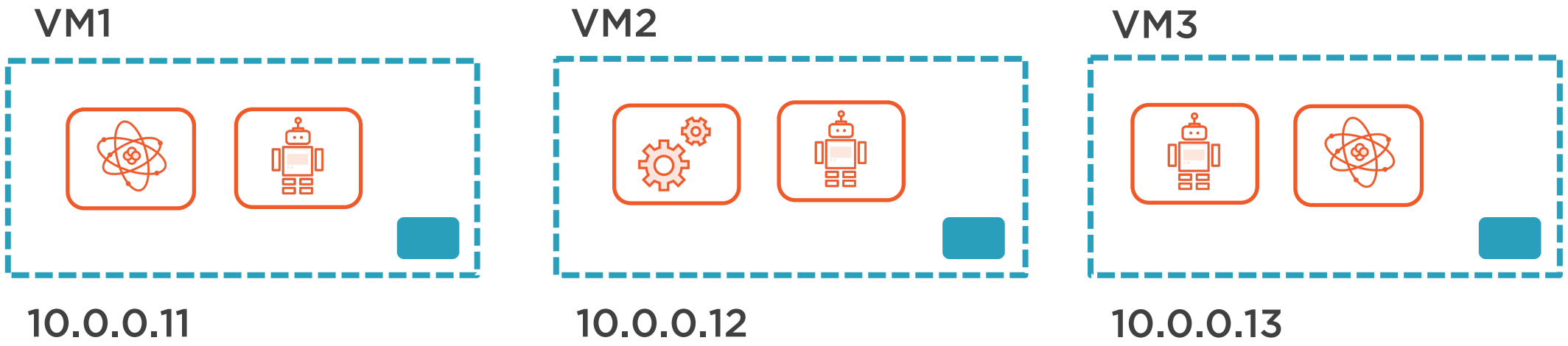
- Dead-letter after multiple failures

Messages may be received out of order

Messages may be received multiple times

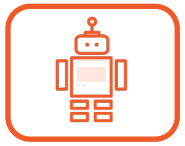
- Idempotent handlers

Service Discovery



Service Registry:

10.0.0.11, 10.0.0.13



10.0.0.11, 10.0.0.12, 10.0.0.13

10.0.0.12

Service Discovery Alternatives

Microservice hosting platforms (PaaS)

- DNS entry per microservice
- Auto-configured load-balancer

Container orchestrators (e.g. Kubernetes)

- Built-in DNS
- Routing handled for you

Summary

- Microservice communications
- RESTful HTTP APIs
- Publishing messages to an event bus
- Both approaches can be combined
- API Gateway / Backend for Frontend

Summary

Resilient communications

- Retries with back-off
- Circuit breakers
- Caching
- Message retries
- Idempotent message handlers

Summary

Service discovery

- Service registry
- Provided by PaaS platforms and orchestrators

Up next...

Securing microservices