

# ACP / 3 + 4

Michael Glienecke, PhD

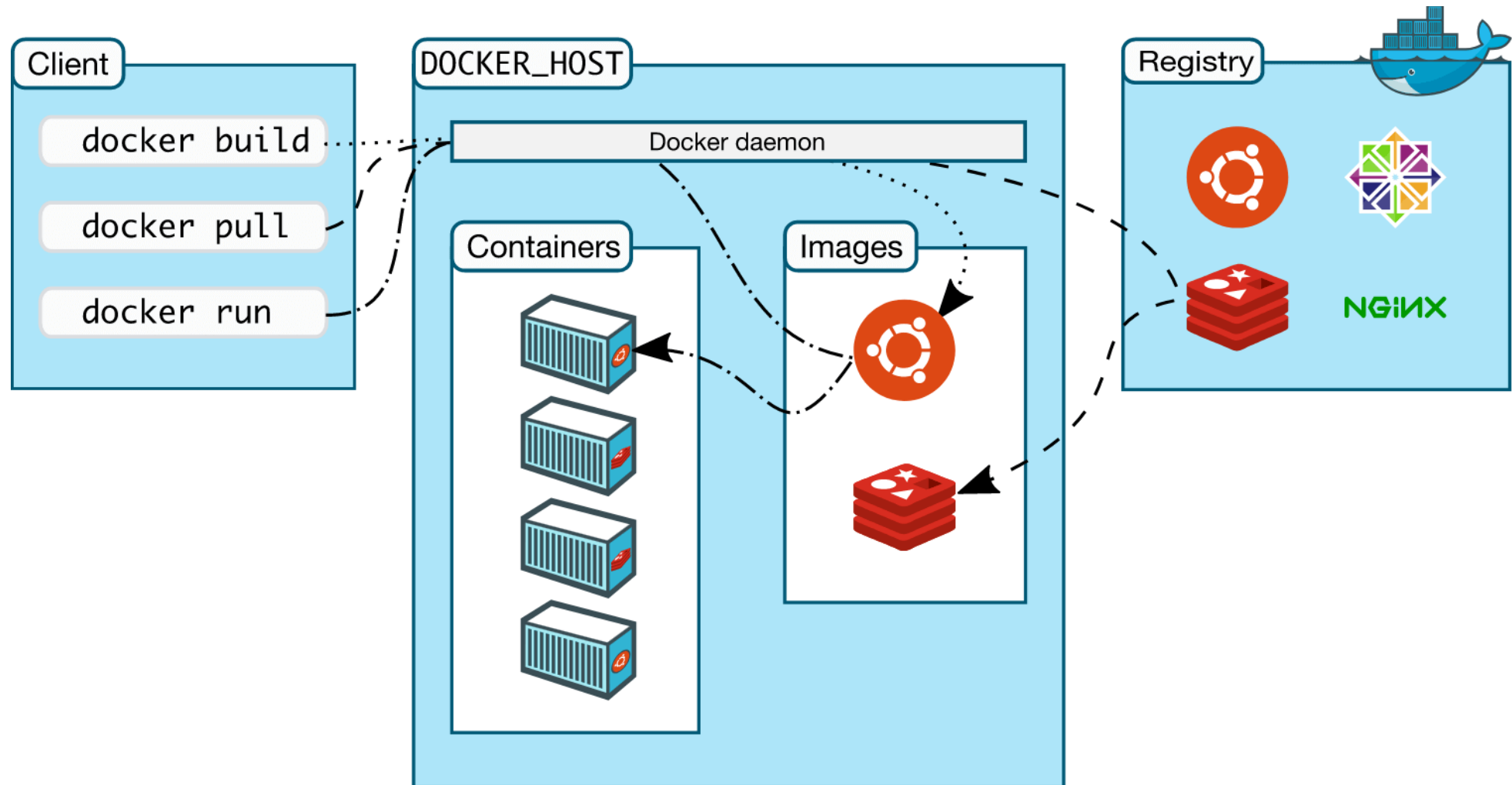
# Welcome again

- A bit about containerization
- Introduction to microservices
- Communication protocols / patterns / methods
- JSON
- REST in more detail
- REST techniques

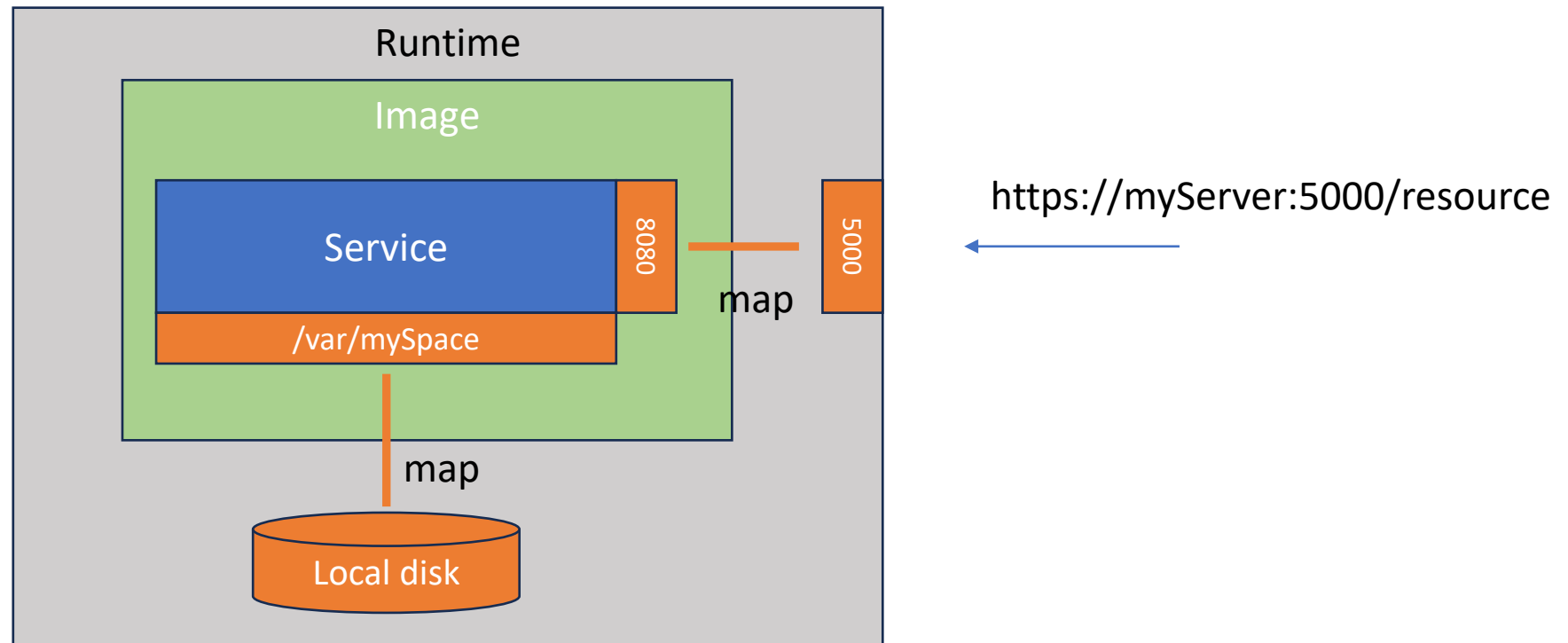
# Containerization - how

- OCI based
  - <https://opencontainers.org/about/overview/>
  - <https://opencontainers.org/community/overview/>
- Images form the runnable applications
- Are loaded on demand
- Executed when needed
- All networking is mapped
- File system access is mapped to volumes

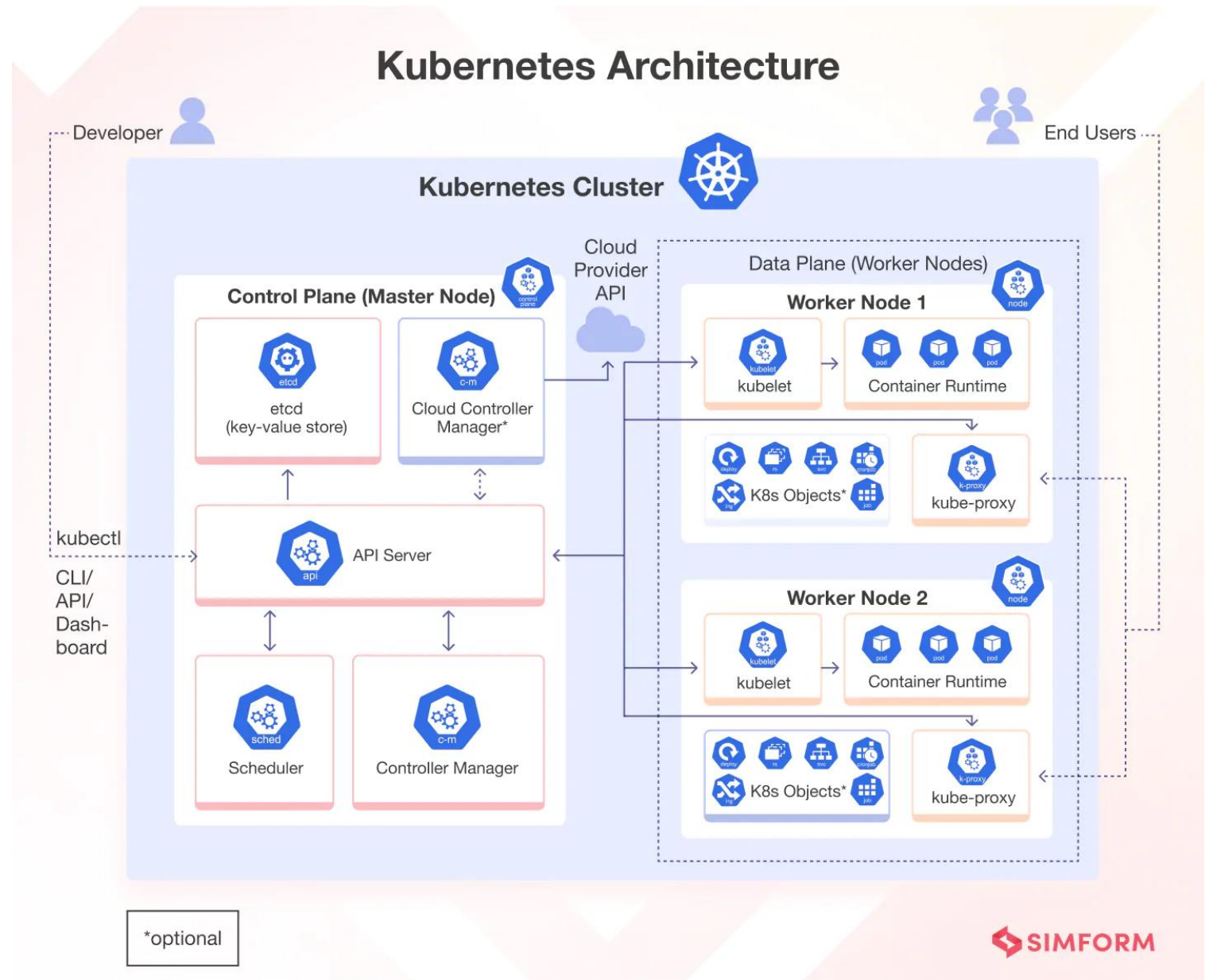
# docker



# A microservice in docker



# Kubernetes



# Communication styles in general

- Synchronous
- Asynchronous

# Asynchronous

- One-to-one
  - Async request / response
  - One way notifications
- One-to-many
  - Publish / subscribe
  - Publish / async responses
- Issue Request - Response will arrive at some time
  - or never -> error handling is crucial
  - Sometimes "fire and forget" is intended (status updates) -> like UDP
  - Usually handled with promises (e.g. JavaScript and AJAX) in client-programming



# Asynchronous

- True async response handling is tricky
  - Where does the server send the response to?
    - Web-Callback
    - Polling (yes, an ugly word, but sometimes useful and cheap – like a Spin-Lock Semaphore)
  - Signal-R (Chat, Async notifications, mostly done using UDP)
- Often the client starts a "sync" background thread to wait for the answer to simulate async behaviour
- Messaging is async by design (Kafka, MQ, AQMP, ...)
  - And reliable

# Synchronous

- Request -> Response with wait
  - Causes blocking of the requester until response arrives or timeout
- Can be achieved with "promises" in asynchronous manner as well
  - [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\\_Objects/Promise](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise)
- Sometimes the easiest and most useful to do
  - e.g. logon -> you really want to wait until you get the response
- In general, avoid if you can
  - Load issues, scaling, resource blocking

# Architectural styles (not all...)

- Monolith
  - Single process
  - Modular
  - Distributed
- Service oriented architectures (SOA)
  - Services collaborate to achieve some capabilities
  - Often shared databases
  - Larger logical blocks ("services")
  - Scaling often only on the service-level
    - not the service as such on several nodes / services often cannot be spread out

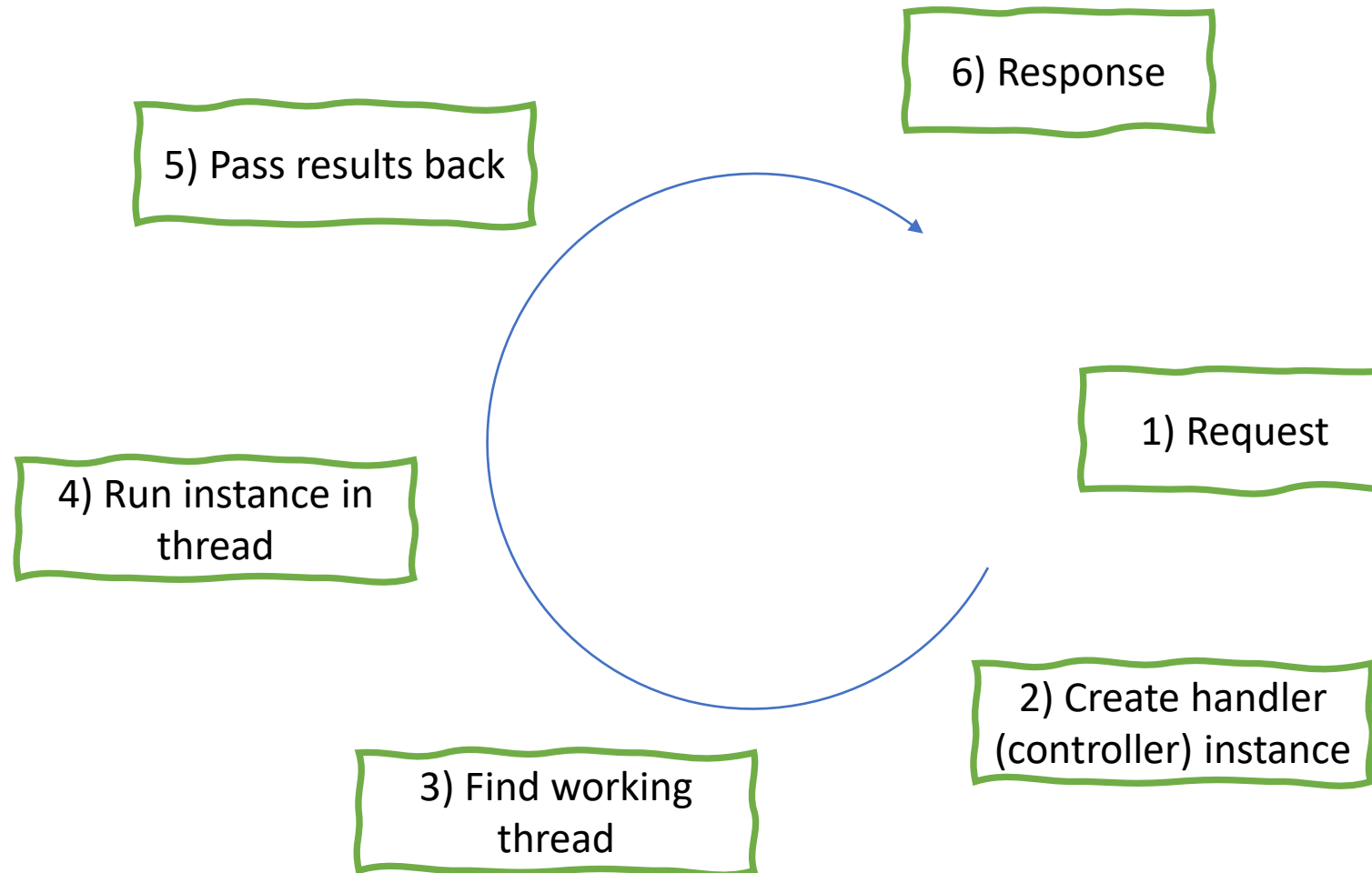
# Architectural styles /2

- Microservices
  - Simple
  - Meshes
- Flow
  - Event based

# Introduction to microservices

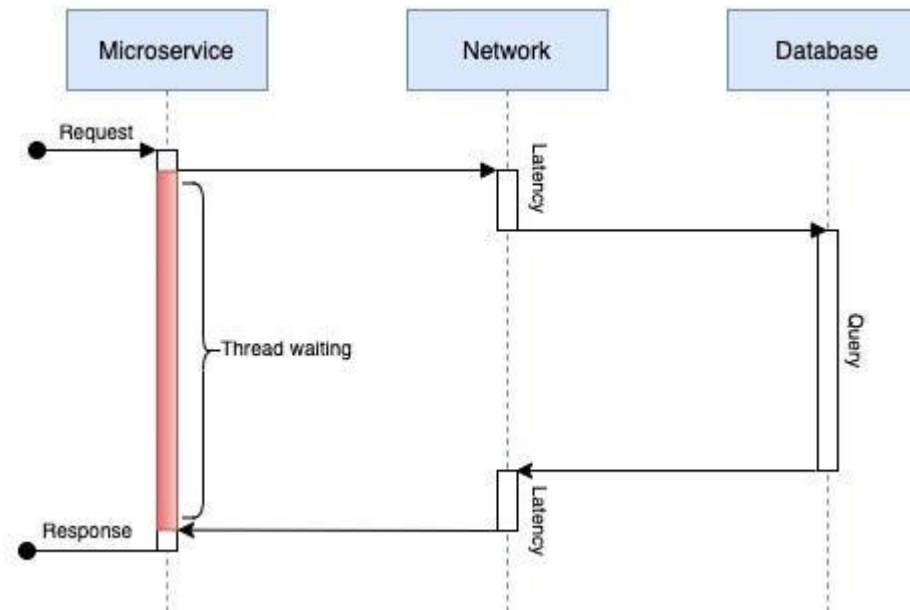
- <https://microservices.io/>
- Independent releasable / deployable services
- Should model a business domain - "DDD"
- All functionality and data is encapsulated
  - Each service has its own data / function
  - "Information hiding"
- All functionality and data is made available to others via networks (as API)
  - The word API is very confusing sometimes...
- Loose coupling / High Cohesion
- Be forgiving...

# The structure of a microservice



# Microservices structure

- Spring Boot uses Tomcat
  - Some limitations (<https://oskar-uit-de-bos.medium.com/the-performance-challenge-in-java-microservices-e51cce3977e9>)
- Blocking threads and consuming resources

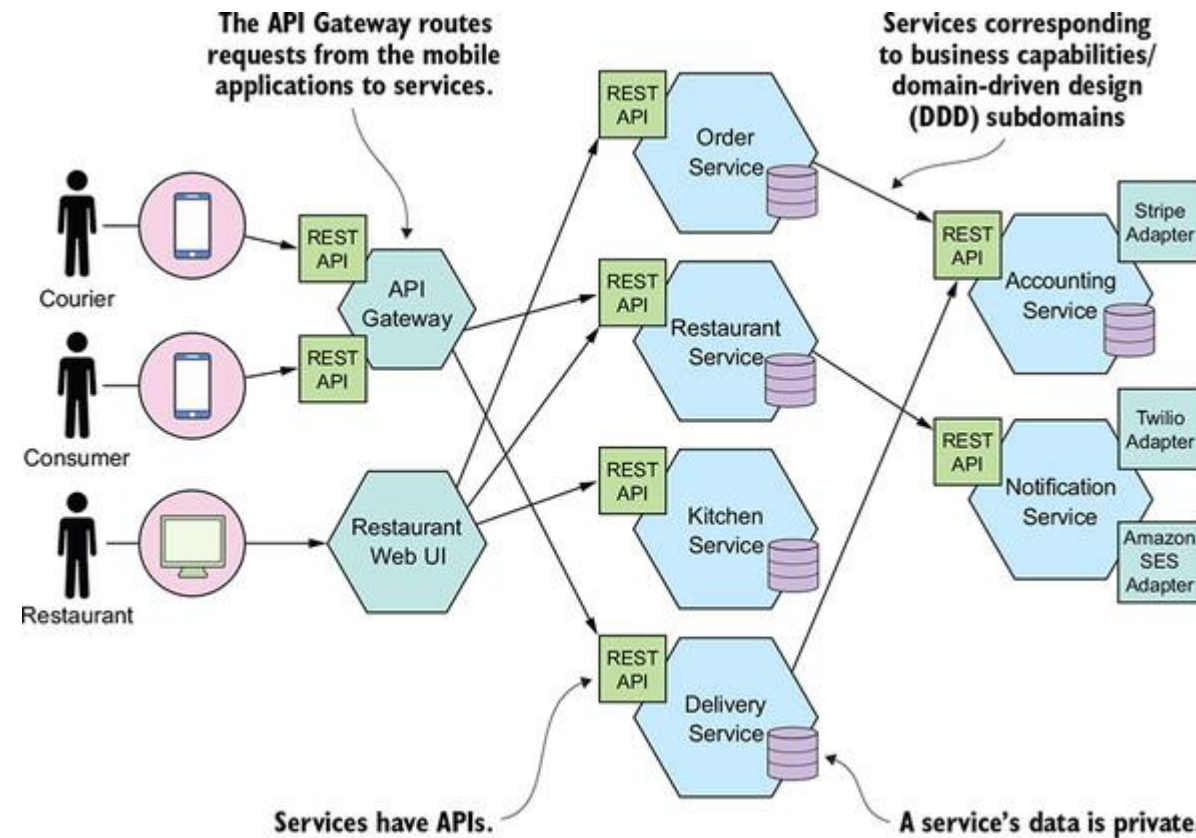


# Microservice styles (some of them)

- Point to point (mesh)
- Event based

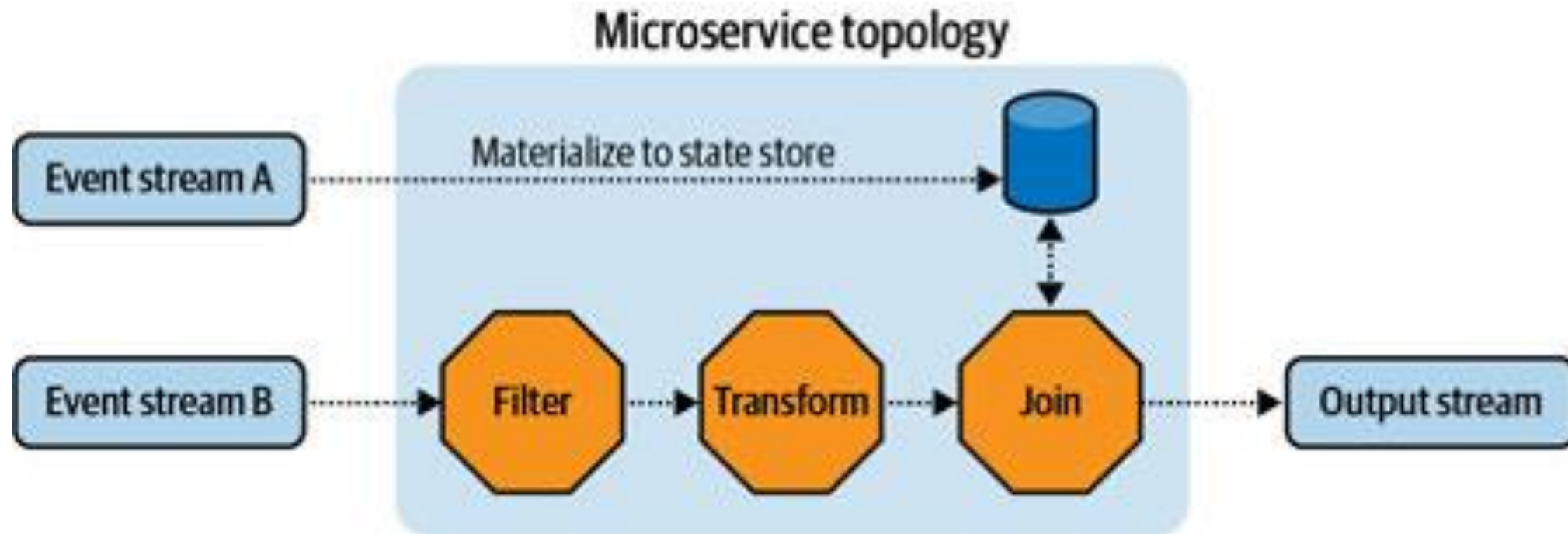


# Point to point - mesh



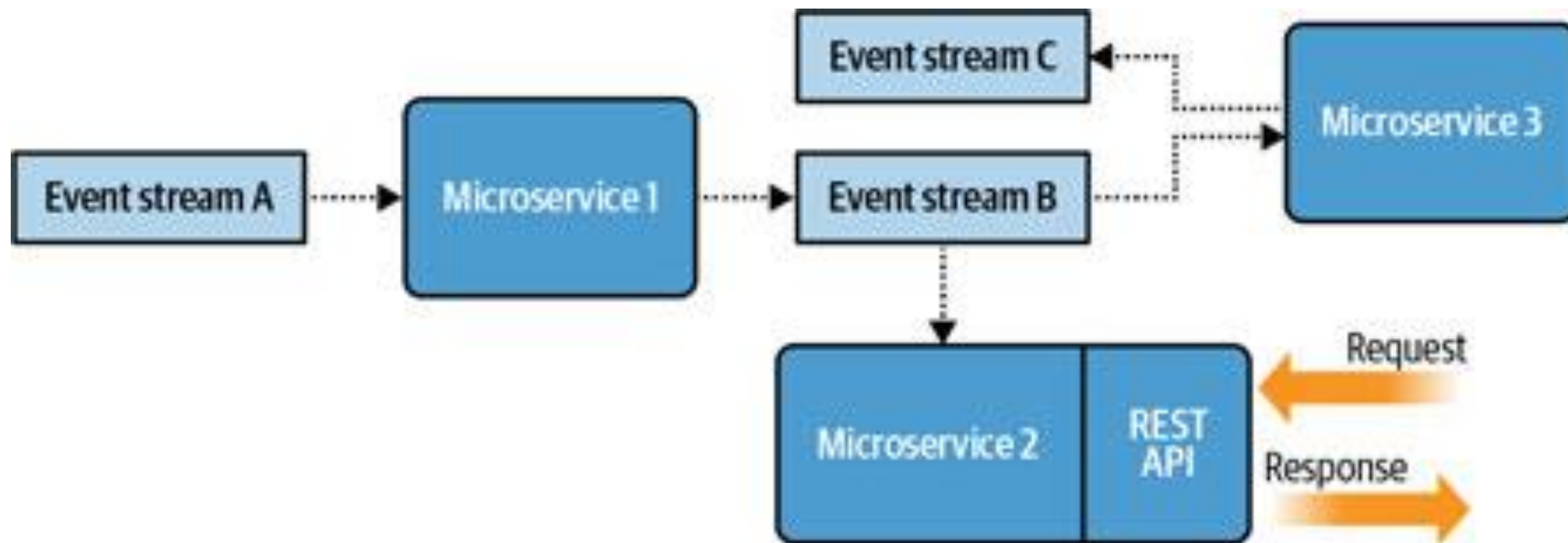
Event is created and it may or may not be executed

# Event based architecture



[https://eu01.alma.exlibrisgroup.com/leganto/public/44UOE\\_INST/citation/45644351890002466?auth=SAML](https://eu01.alma.exlibrisgroup.com/leganto/public/44UOE_INST/citation/45644351890002466?auth=SAML)

# Event based architectures /2



[https://eu01.alma.exlibrisgroup.com/leganto/public/44UOE\\_INST/citation/45644351890002466?auth=SAML](https://eu01.alma.exlibrisgroup.com/leganto/public/44UOE_INST/citation/45644351890002466?auth=SAML)

# Alternatives to microservices

- Monolith
- Service Oriented Architecture (SOA)
  - Well, microservices are a kind of SOA
  - Sometimes, when microservices become macroservices (yes...) the border blurs
- Pure flow architectures (an extension to event based)

# Monolith vs microservices

- Choose the right architecture for the right usage
- Monolith doesn't mean bad – it is a choice
- Can be super powerful, if done right
- Very little effort and loads of benefits
- Don't underestimate microservices – they are harder and more complex as it seems
- Not every new idea is "the killer" for older stuff

# Communication protocols

- Request - Response
  - **gRPC**
  - http native
  - SOAP
  - **REST**
  - **GraphQL**
- Messaging
  - **Kafka**
  - MQTT
  - Pulsar
  - AMQP (RabbitMQ, ...)

# Protocols / http based

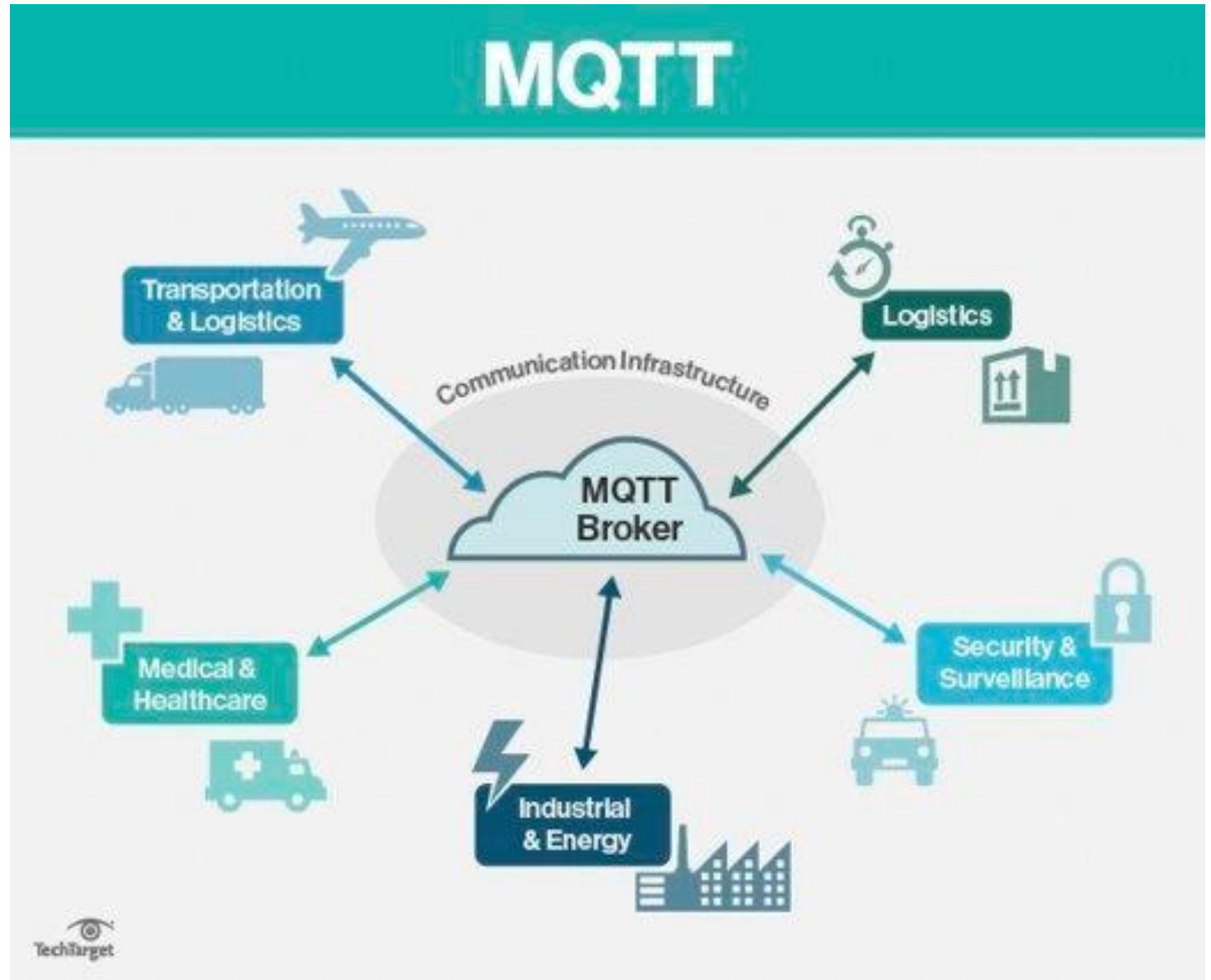
- text-content
  - requests / responses in plain text
  - REST
  - SOAP
  - GraphQL
- gRPC
  - http/2
  - http/3

# MQTT

- A lightweight (initially sending oil-rig telemetry data to a home-station via satellite) **publish / subscribe** messaging protocol for M2M communication
- <http://www.steves-internet-guide.com/mqtt/>



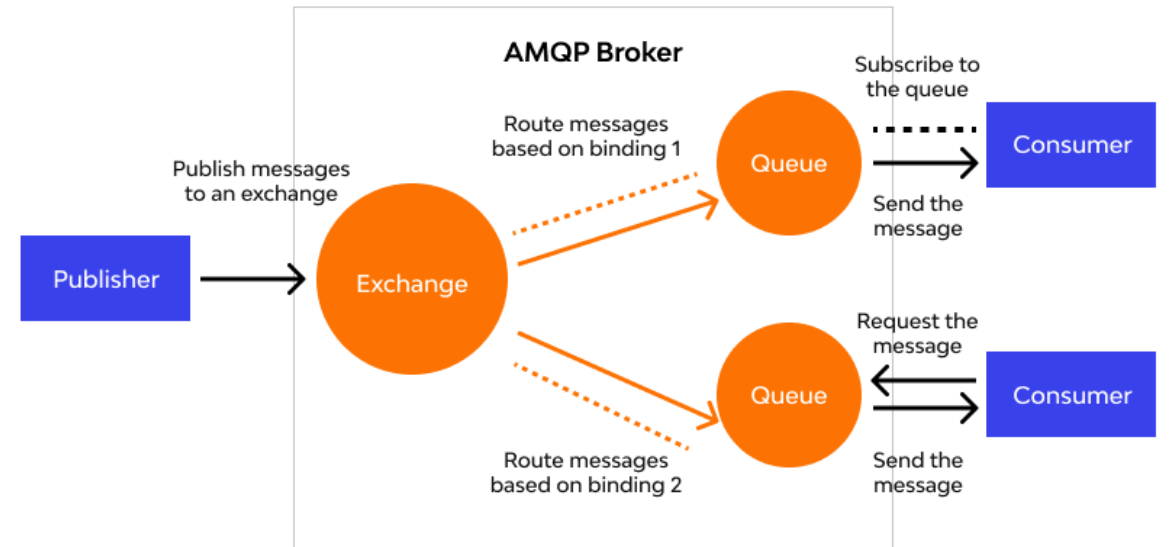
# MQTT



<https://www.techtarget.com/iotagenda/definition/MQTT-MQ-Telemetry-Transport>

# AMQP

- AMQP = Advanced Message Queuing Protocol
- Similar in many things to i.e. Kafka
- RabbitMQ (and easier than Kafka)



# Messaging alternatives

Feature	RabbitMQ	Apache Kafka	ActiveMQ
Messaging Model	Traditional	Publish/Subscribe	Traditional
Scalability	Clustering/Network of Brokers	Partitioning	Clustering/Network of Brokers
Performance	Moderate	High	High
Data Persistence	On Disk (default), In-memory	On Disk	On Disk (default), Database
Integration	Programming Languages, Databases, Web Servers	Data Processing Systems, Databases, Data Sources	JMS Clients, Apache Camel, Apache CXF
Suitable For	Strict Ordering, Reliable Delivery, Moderate-High Message Rates	Streaming Data, High Message Rates	High Data Durability, High Performance

# JSON

- <https://www.json.org/json-en.html>
- Lightweight, human readable data-interchange format
- JSON vs XML: <https://json.org/example.html>
- Downsides:
  - No error handling
  - Less expressive and flexible than XML
  - No comments, namespaces, attributes (hard to add metadata)
  - No versioning
  - No security
  - No XSD
- But... Everybody knows it, everybody uses it...

# Tools to test microservices

- REST
  - Postman
  - curl
  - Swagger
- gRPC
  - gRPCurl
  - Postman
- Kafka
  - Command line (shell) tools

# Running a microservice in docker

- One service / image
- Sidecar as an option
- Showing a small example

# Example microservice in Java

- Main entry point:

<https://github.com/mglienecke/IlpTutorialRestService/blob/main/src/main/java/uk/ac/ed/inf/ilptutorialrestservice/IlpTutorialRestServiceApplication.java>

# Example microservice in C#

- <https://learn.microsoft.com/en-us/dotnet/architecture/microservices/multi-container-microservice-net-applications/data-driven-crud-microservice>



# Example microservice in Go

- <https://github.com/PacktPublishing/Microservices-with-Go>
- Main entry point:  
<https://github.com/PacktPublishing/Microservices-with-Go/blob/main/Chapter02/src/movie/cmd/main.go>

# How does this map to the Azure Blob API?

- <https://learn.microsoft.com/en-us/rest/api/storageservices/blob-service-rest-api>
- Azure Blob is a collection of endpoints in REST
- There presumably are several "controllers" which instantiate handlers for requests
- These perform the actual request and return the results

# REST details

- HTTP methods: <https://restfulapi.net/http-methods/>
- Style
- URL
- Exchanging URLs (remote / local)
- Mocking
- Unit testing

# General

- REST <> http
- REST is
  - Uniform
  - Stateless
  - Client-Server
  - Cacheable
- URL
  - Syntax: <https://www.w3.org/Addressing/URL/url-spec.html#:~:text=URL%20syntax,in%20an%20a%20later%20section>.
  - BNF: [https://www.w3.org/Addressing/URL/5\\_BNF.html](https://www.w3.org/Addressing/URL/5_BNF.html)

# http-methods

- <https://restfulapi.net/http-methods/>
- Mainly used is:
  - GET
  - POST
- Sometimes
  - PUT
  - DELETE (most businesses either use POST or do a "soft-delete" approach)
- Almost never
  - PATCH

# REST-styles

- Query string
- Body (not for GET!)

# Designing an API in REST

- Do it very careful as it will remain for a long time and cannot be easily changed
- API Management is a big issue in the industry
- RPC based designs
  - POST <https://..../createOrder> JSON data in Body
  - POST <https://..../createOrderPosition> JSON data in Body
- Resource based designs
  - GET <https://.../resource/sub/identifier>
  - PUT <https://.../orders/> JSON data in Body
  - PUT <https://.../orders/1/position> JSON data in Body

# Testing

- Unit-testing
- Integration-testing
- Round-trip testing
  
- Fitness functions



# Mocking

- Development takes time – services even longer
- UI depends on services and often needs something to work with
- Mock to the rescue
- What is mocking?

# gRPC

- <https://piotrminkowski.com/2023/08/29/introduction-to-grpc-with-spring-boot/>
- Protobuf as definition
- grpcUrl
- Sample proto files

# Create a service with REST and gRPC

# Combined services (multi-protocol)

- The ability to handle several protocols in the same service
- Often REST + gRPC
- Alternative is:
  - Conversion layers (REST to gRPC) - BFF
  - Separate layers for e.g. GraphQL to gRPC
  - ...

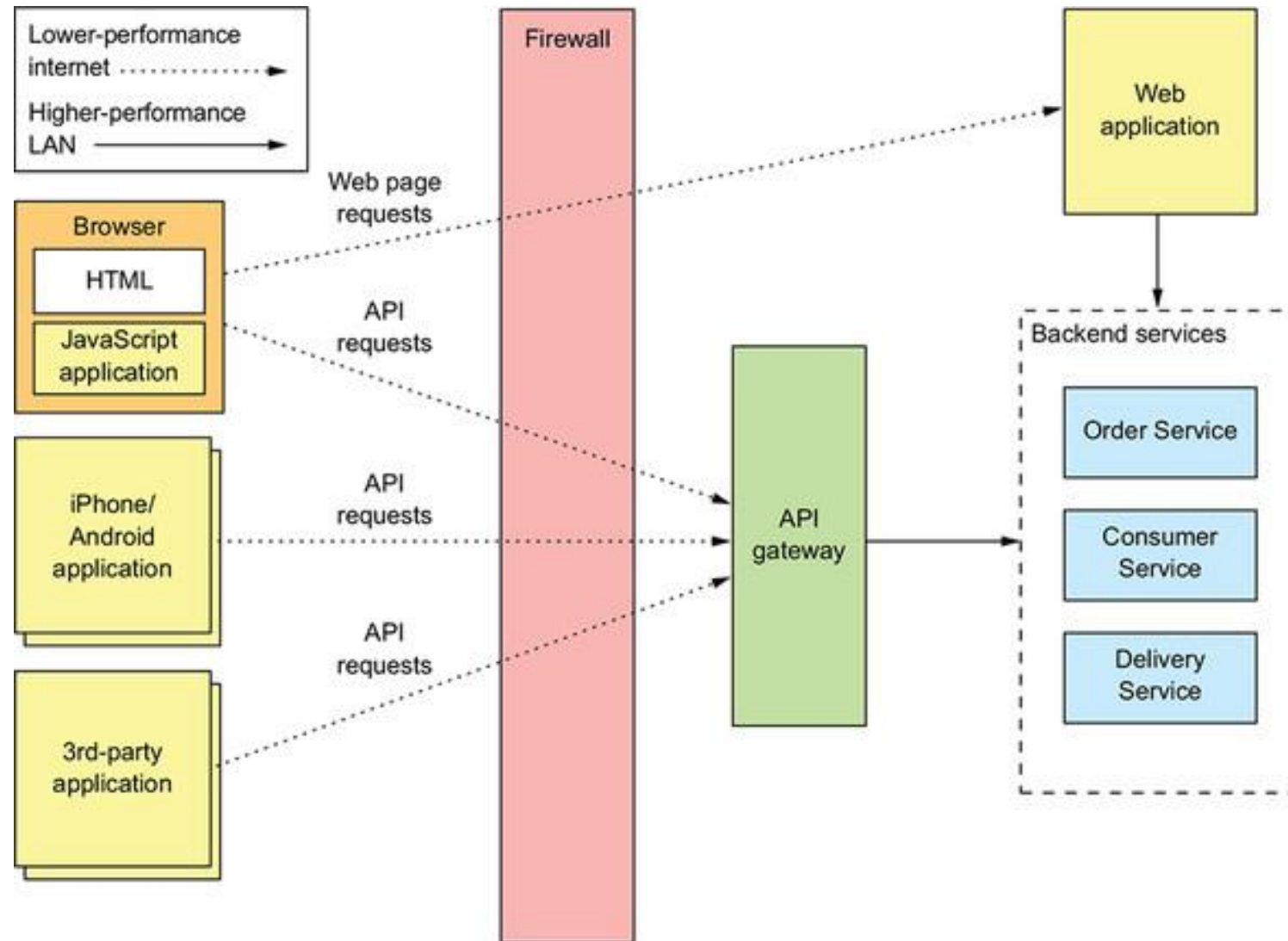
# BFF (patterns)

- Backend for frontend (BFF) pattern
- One frontend access point for each unique frontend technology / style
- Often
  - a limiter to a larger service API
  - A conversion bridge (REST – gRPC)
- Authorization / data segregation is sometimes handled here as well
- Quite a lot of repetitive boilerplate code (esp. gRPC – REST is mostly converting data structures)

# Patterns in general

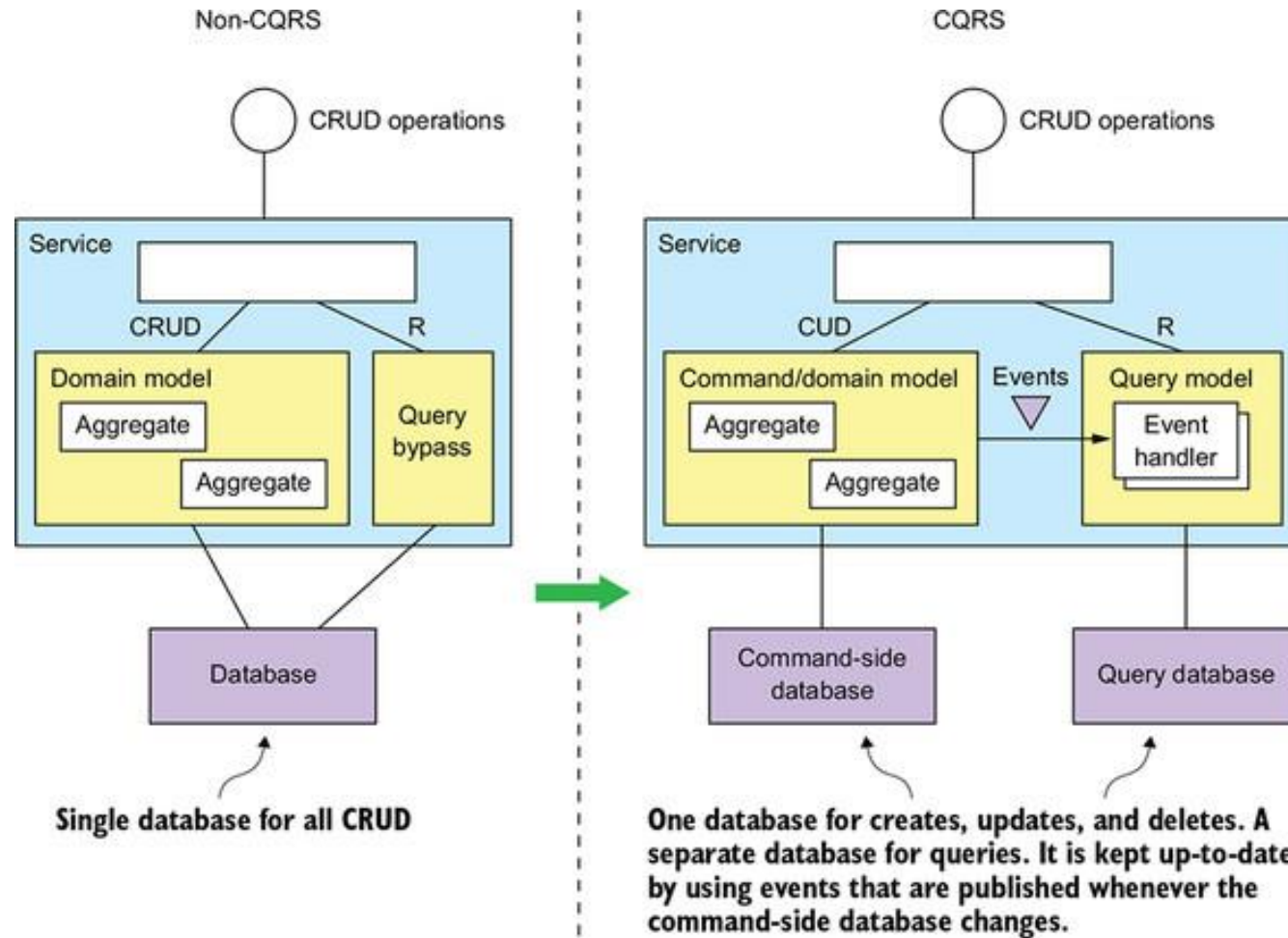
- API Gateway
- CQRS
- Circuit Braker

# API Gateway



# CQRS

## Command Query Responsibility Segregation





# CQRS Pros / Cons

- Enables the efficient implementation of queries in a microservice architecture
- Enables the efficient implementation of diverse queries
- Makes querying possible in an event sourcing-based application
- Improves separation of concerns
- More complex architecture
- Dealing with the replication lag

# Circuit Breaker

