



Event-driven Microservices in the Serverless Age

Serverless Meetup Boston, May 23, 2019

Marius Bogoevici

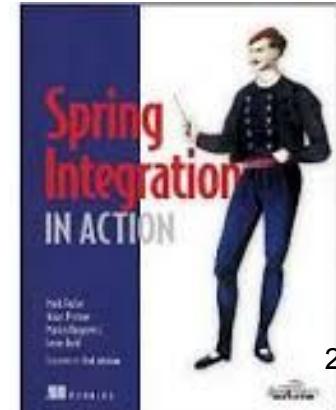
Principal Specialist Solution Architect

mariusb@redhat.com

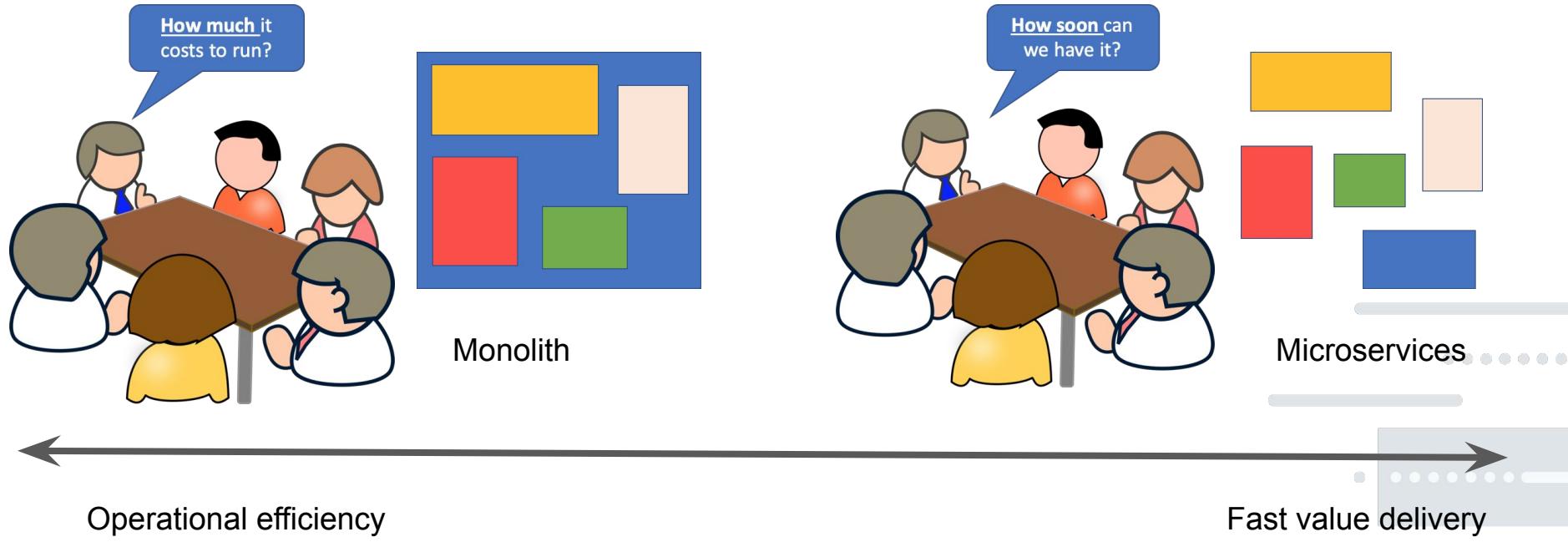
twitter: mariusbogoevici

Marius Bogoevici

- Principal Specialist Solutions Architect at Red Hat
 - Specialize in Integration/Messaging/Data Streaming
- OSS contributor since 2008
 - Spring Integration
 - JBoss ecosystem
 - Spring XD, Spring Integration Kafka
 - Former Spring Cloud Stream project lead
- Co-author “Spring Integration in Action”, Manning, 2012



WHY MICROSERVICES? WHY SERVERLESS?



Still, **why** fast value delivery?

Fast value delivery

New features

Experimentation

**Increased
confidence**

Unfortunately, we cannot predict the future. As an organization, we must be able to observe and experiment in our environments and react accordingly.

We need to be agile.



Purpose
built

agility

On the other hand we must be mindful of our resources;

We want to eliminate waste, reduce time to experiment, and make it cheap so we can increase our returns.

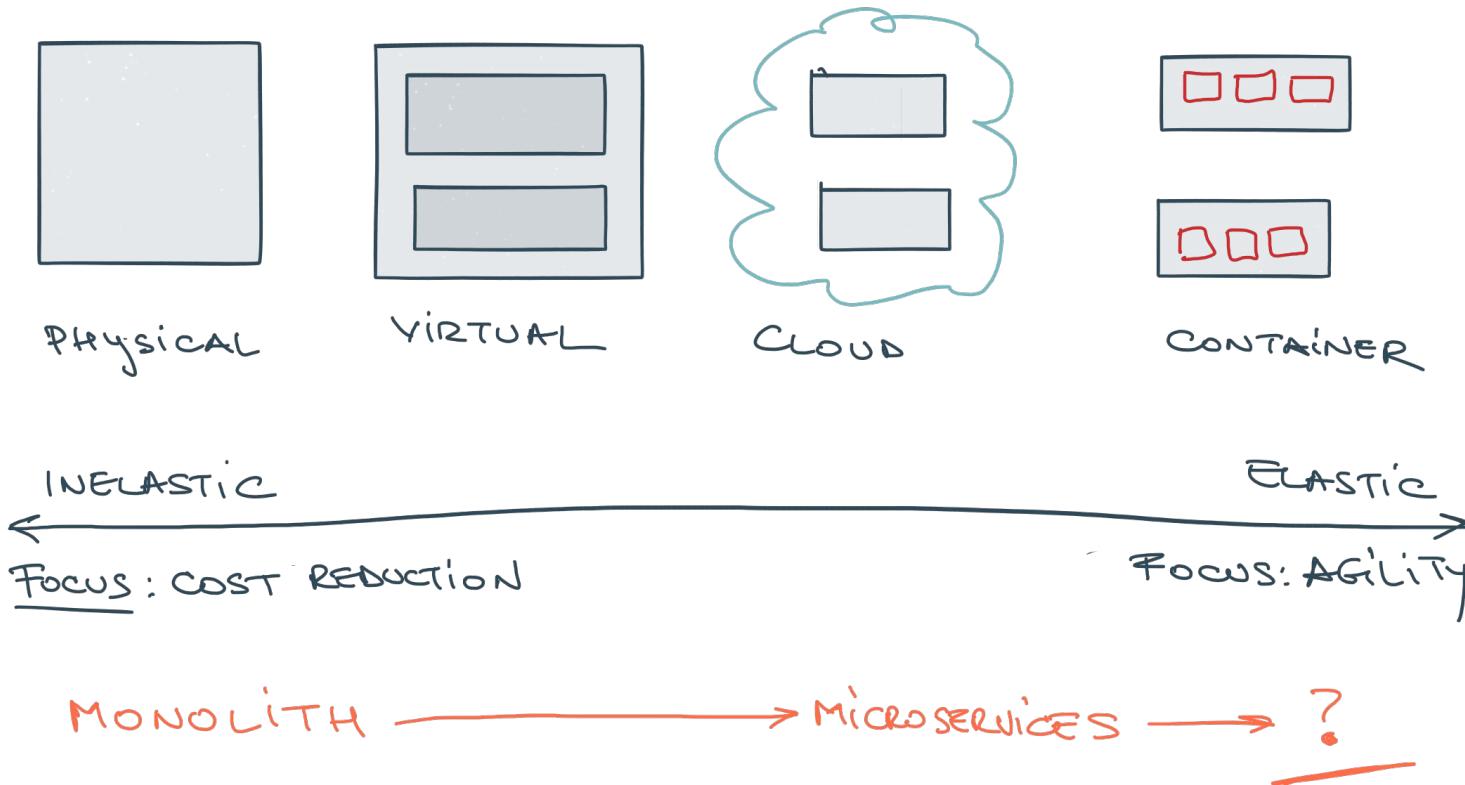


We cannot build complex systems from complex parts.

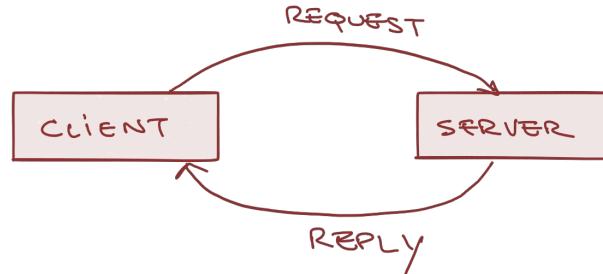
**We must keep our components as simple and
understandable as possible.**



As hardware evolves, we have more options



Request-reply vs. event-driven



Synchronous & ephemeral
Low composability
Simplified model
Low tolerance to failure
Best practices evolved as REST



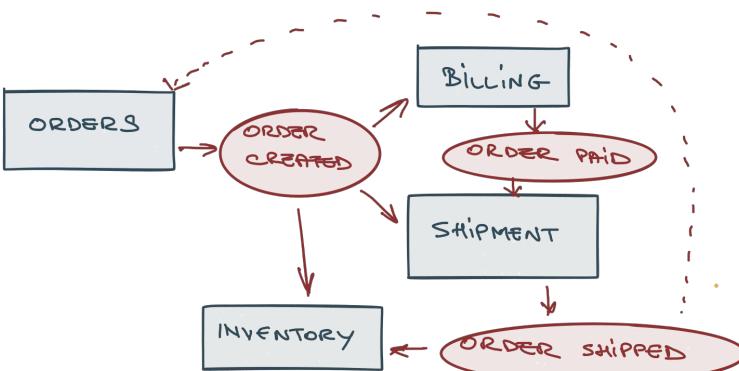
Asynchronous and persistent
Decoupled
Highly composable
Complex model
High tolerance to failure
Best practices are still evolving

What is an event?

- Action or occurrence, something that happened in the past
 - ‘Order created’, ‘user logged in’, ‘
- Event characteristics:
 - Immutable
 - Optionally persistent
 - Shareable
- Event types: [1]
 - Notification
 - State Transfer (Command)
 - Event-Sourcing/CQRS

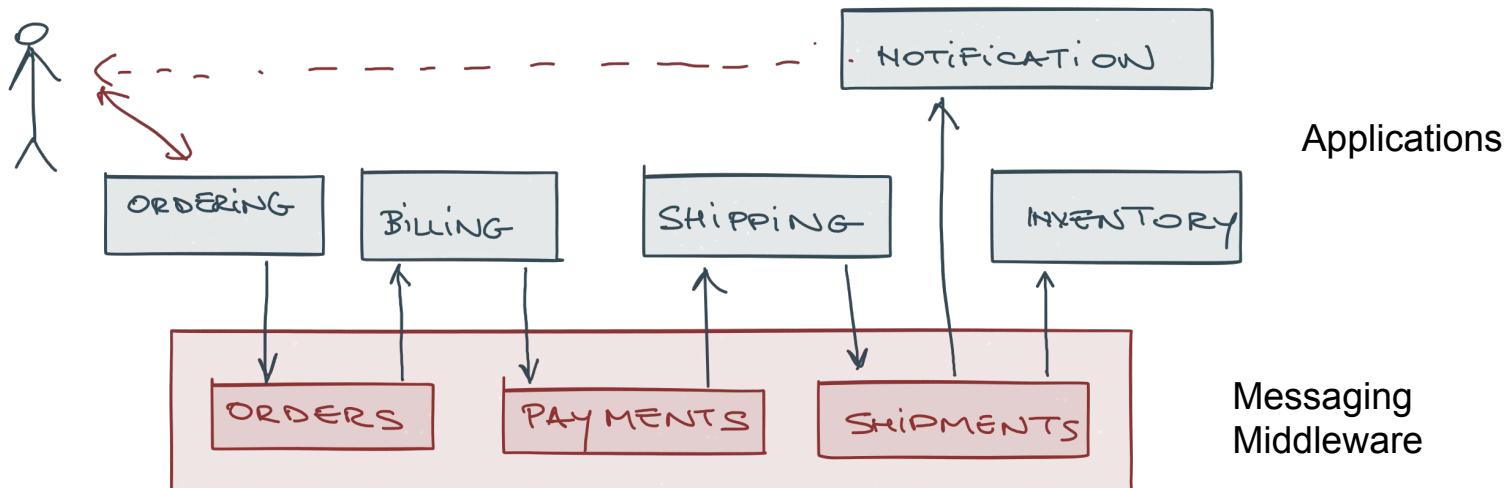
[1] <https://martinfowler.com/articles/201701-event-driven.html>

Designing systems with events

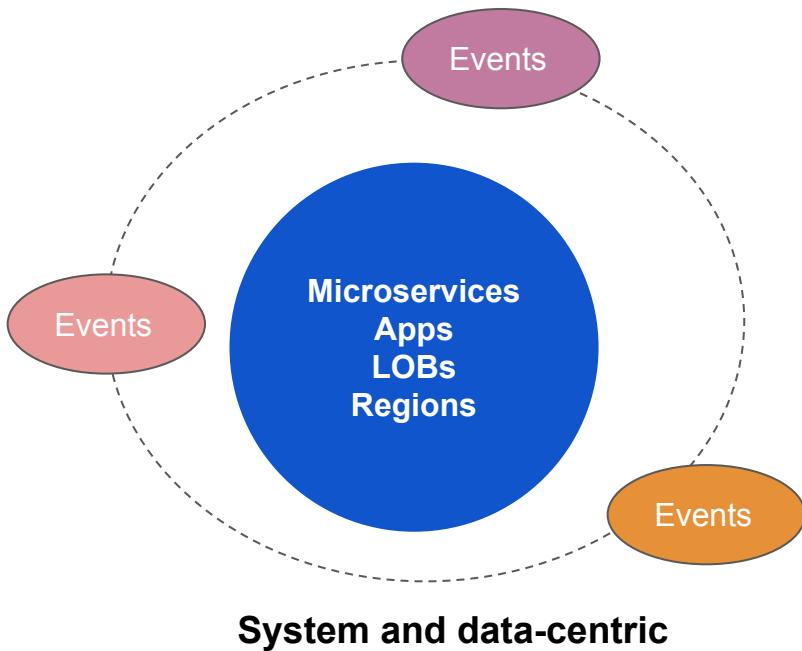


- EDA: event-centric approach in system design
 - Treating events as part of your domain model
 - Designing components as event handlers and emitters
- EDA is aligned with the goals of domain-driven design
 - Enforce isolation and decoupling between bounded contexts
 - Properly designed events can create an expressive ubiquitous language
- EDA creates highly observable and extensible systems
- Event storming: events-first design

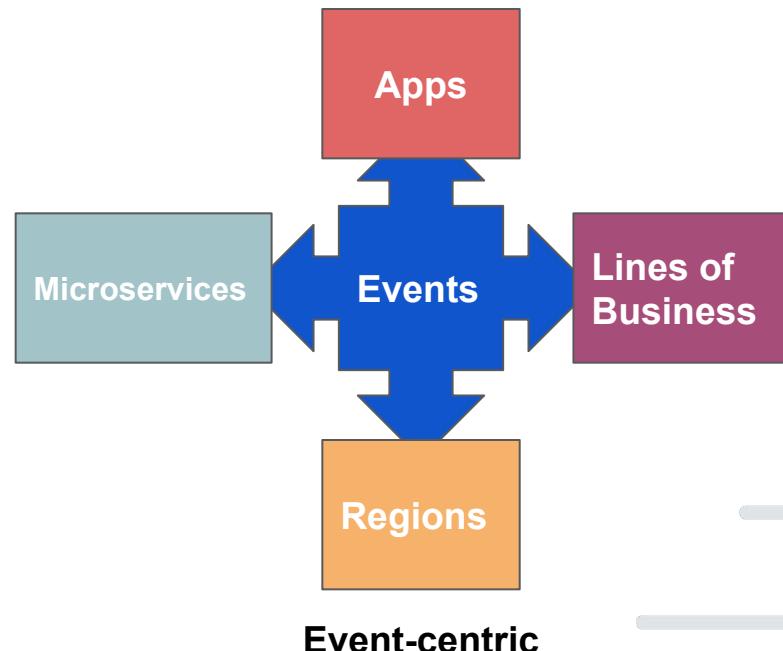
Event-driven microservices



RETHINKING EVENT-DRIVEN ARCHITECTURE

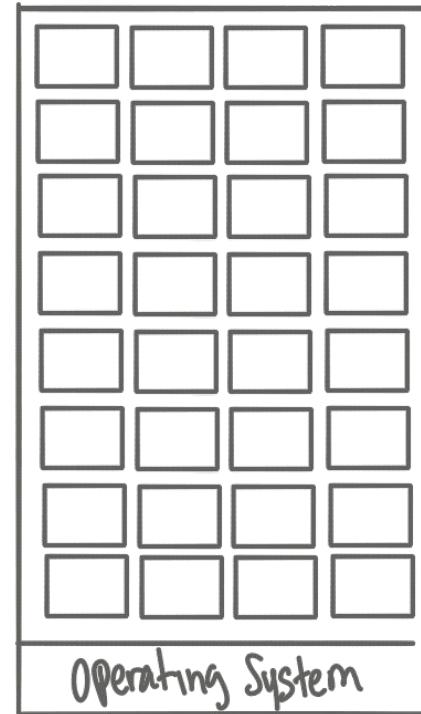
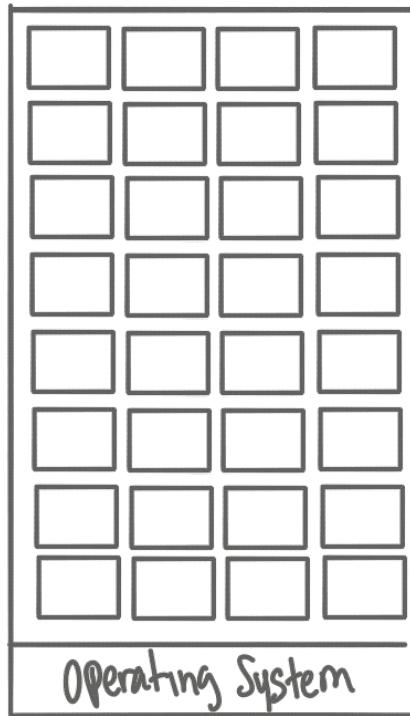
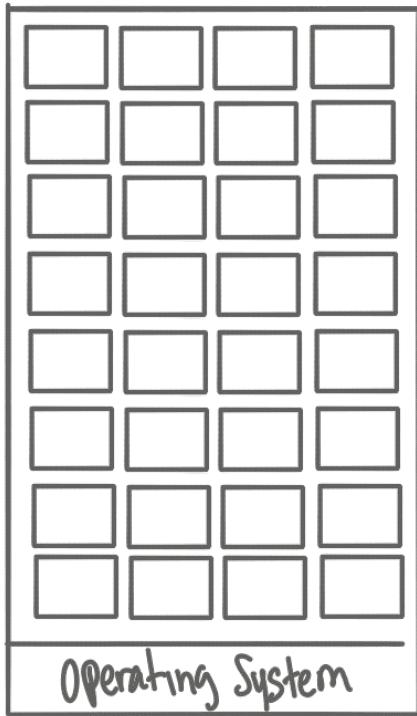


Events are designed to respond to ad-hoc connectivity needs



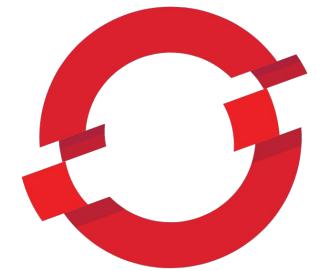
Events are first class citizens that describe the interactions in the enterprise

Microservices in containers: Increasing agility, isolation, utilization



Orchestrating containers on cloud native platforms

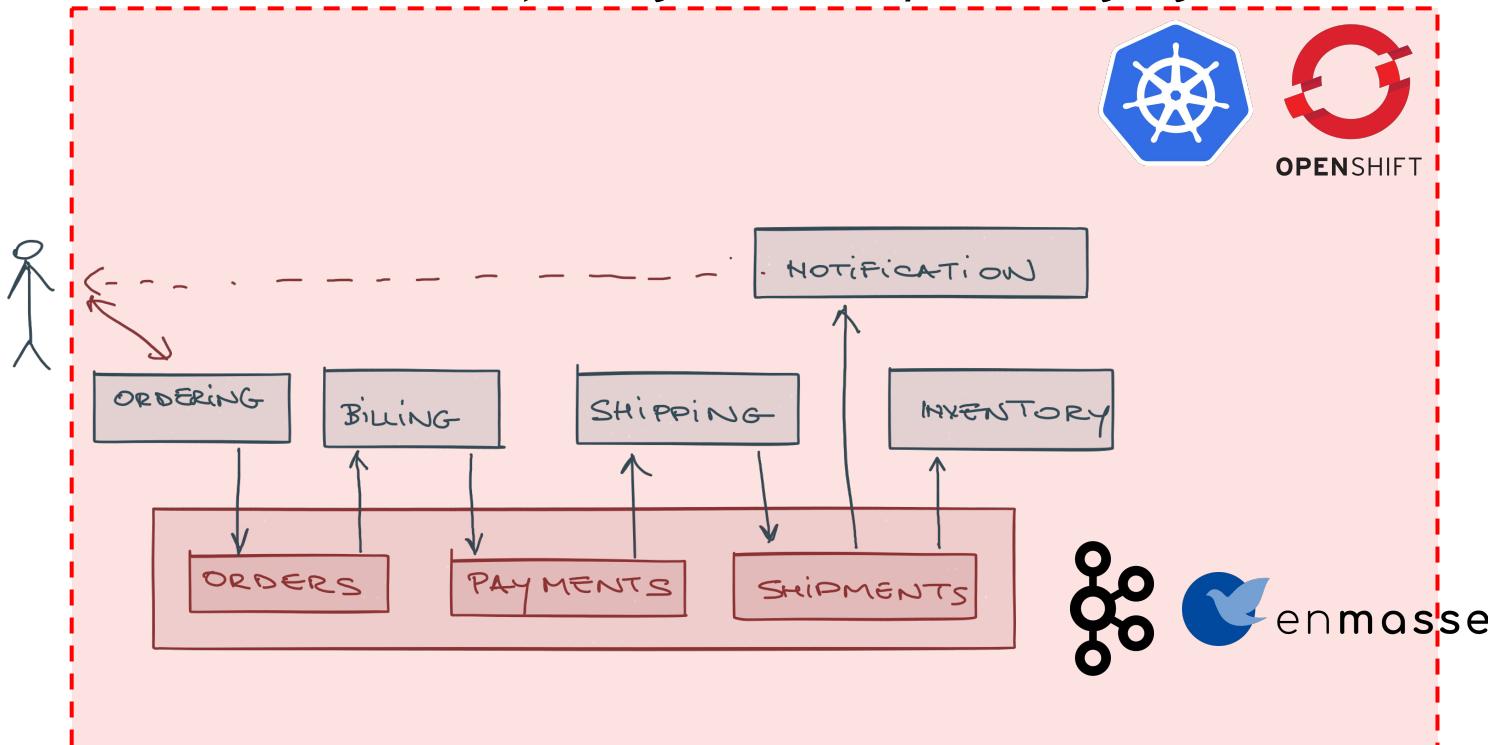
- Use a **platform** that makes running apps reliable, transparent and boring
- In-built resource management
 - Memory, CPU, disk
- Elastic scaling
- Monitoring and failover
 - Health, logging, metrics
- Routing and load balancing
- Rolling upgrades and CI/CD
- Namespacing



OPENSHIFT



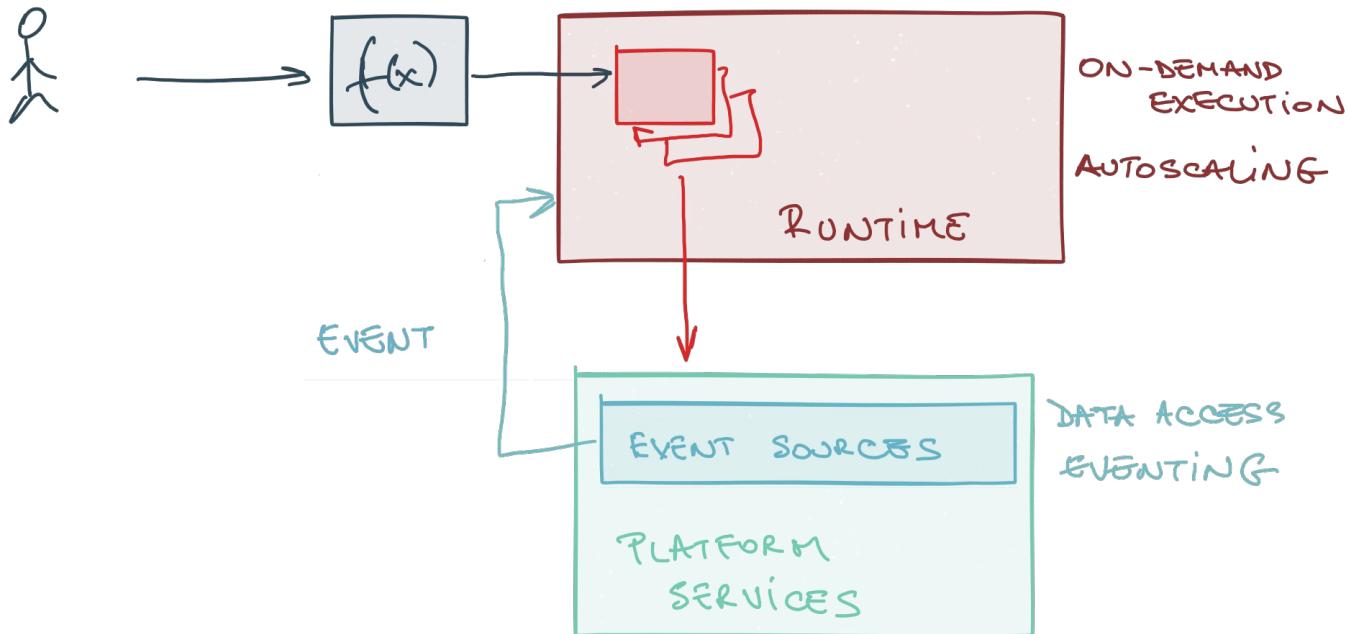
Event-driven microservices in the cloud native age: Elastic computing and utility messaging



Some challenges with microservices ...

- Utilization
 - Idling under low traffic
 - High resource consumption - memory, disk
- Connectivity
 - Must know broker location
 - Integration with event sources
- Abstraction
 - Must know broker type
 - Dependence on data/payload formats
- Observability
- Security

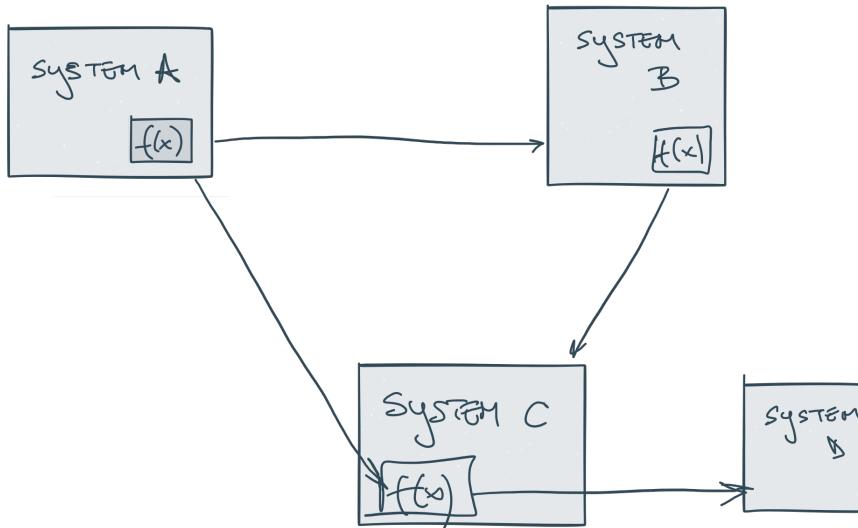
Serverless model: event-based and elastic



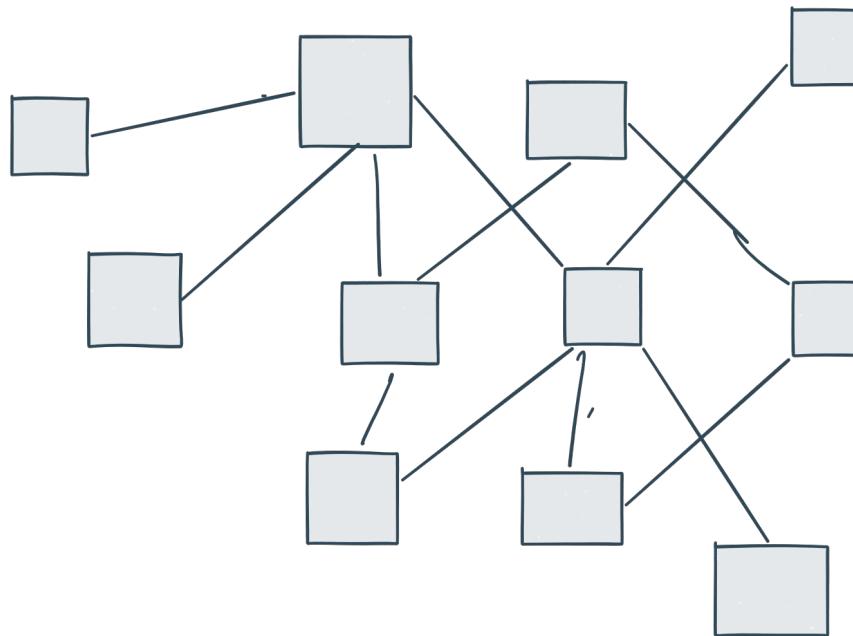
Enriching microservice architecture with serverless facilities

- Elastic execution and avoidance of idling (autoscale)
- Utility data and messaging infrastructure
 - Provided via platform services
- Decoupling of business logic from messaging infrastructure

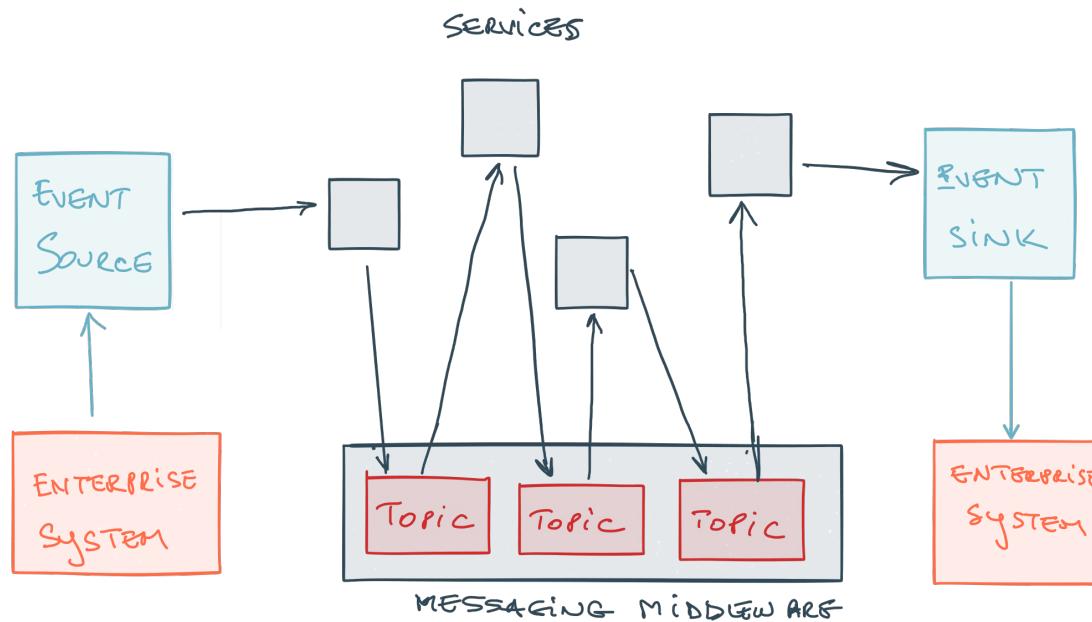
Architectural risks with serverless: loss of domain perspective ...



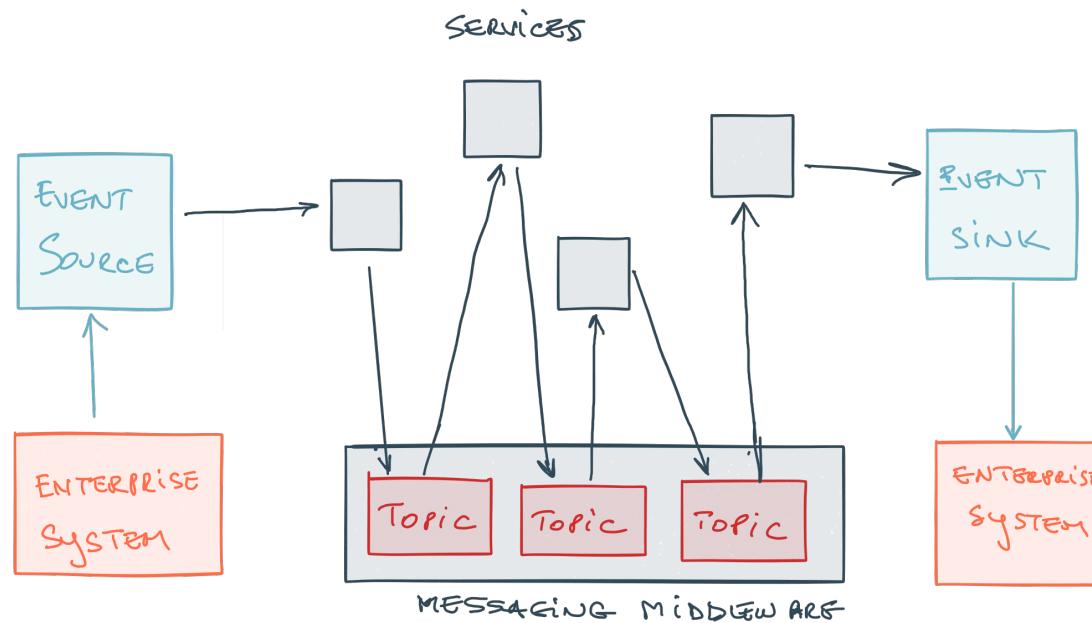
Architectural risks with serverless: ... point-to-point integrations and sprawl



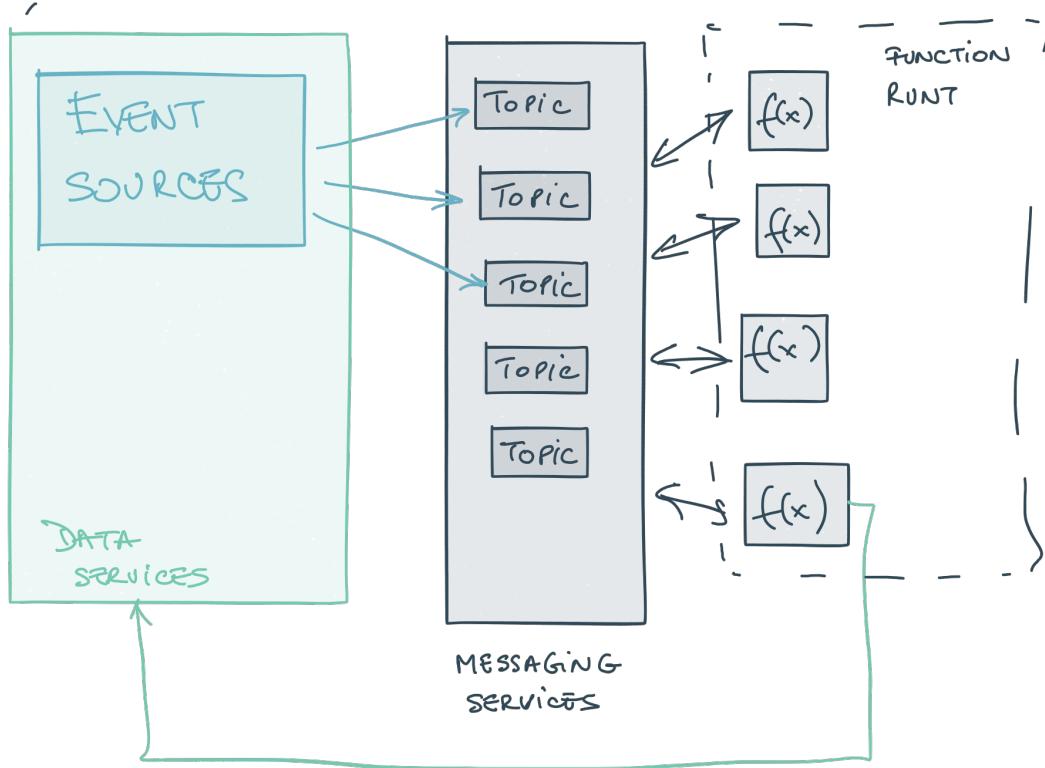
Recap: event hubs in microservice architecture



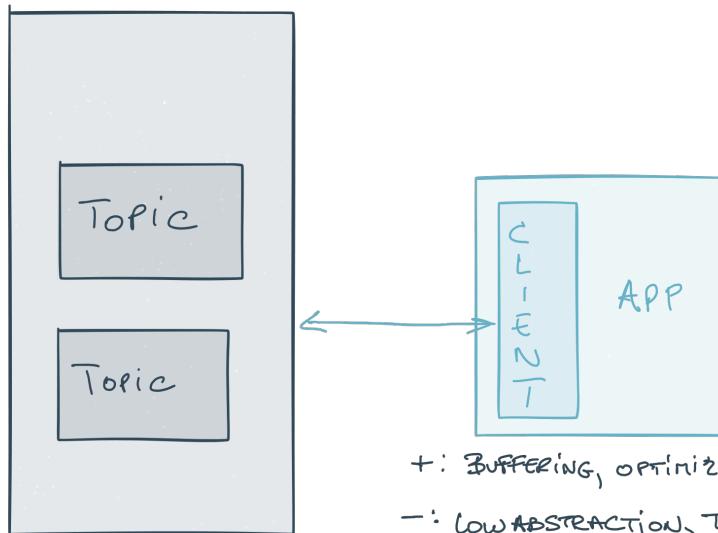
Event-centric microservice architectures...



... can be easily adopted for serverless design

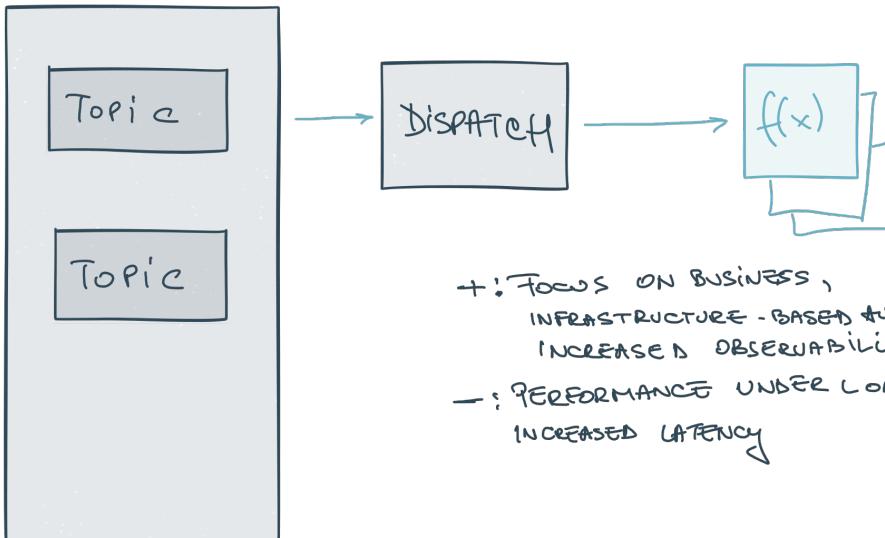


Microservices in Event-driven Architecture: Pros and cons



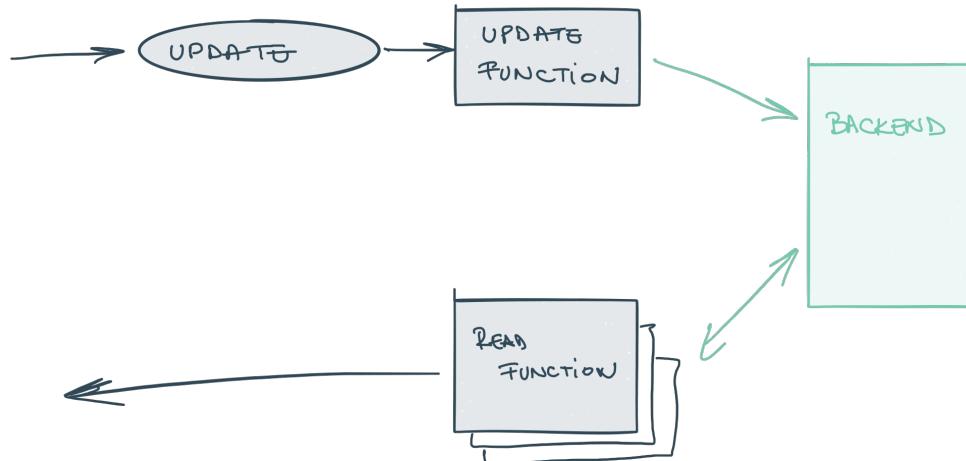
- + : BUFFERING, OPTIMIZATIONS (low latency)
- : LOW ABSTRACTION, TIED TO INFRA
POTENTIALLY LOW UTILIZATION

Functions in Event-driven Architecture: Pros and cons



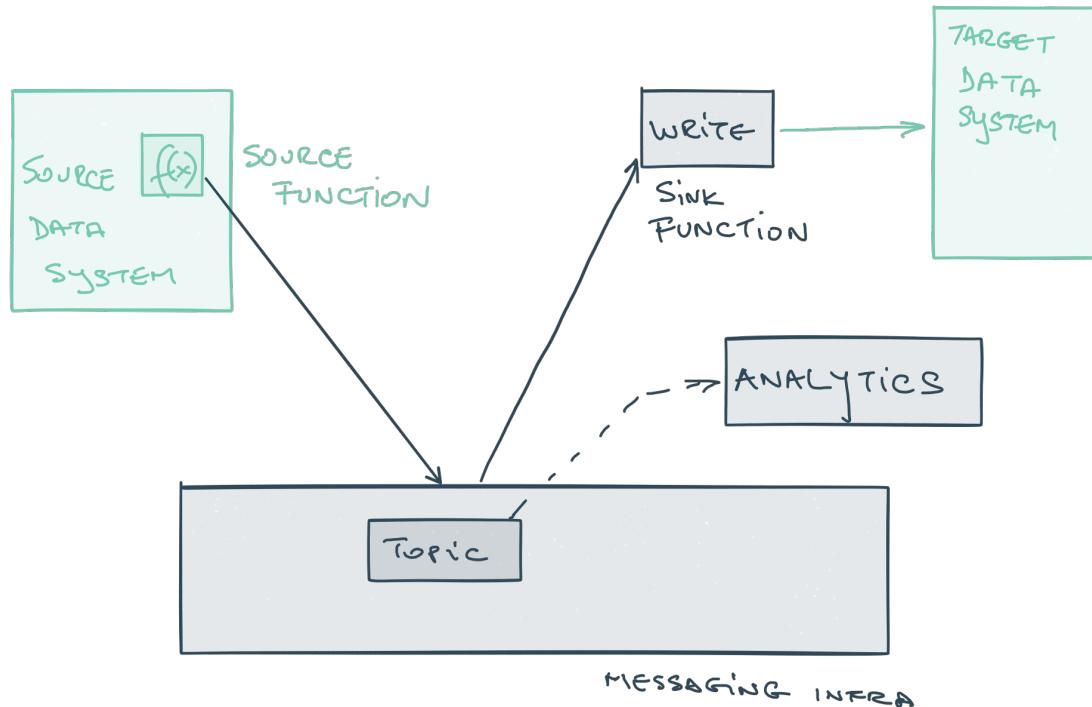
- + : FOCUS ON BUSINESS,
INFRASTRUCTURE-BASED AUTOSCALING,
INCREASED OBSERVABILITY
- : PERFORMANCE UNDER LOAD
INCREASED LATENCY

Event-driven microservice use cases applied to serverless Event Sourcing/CQRS



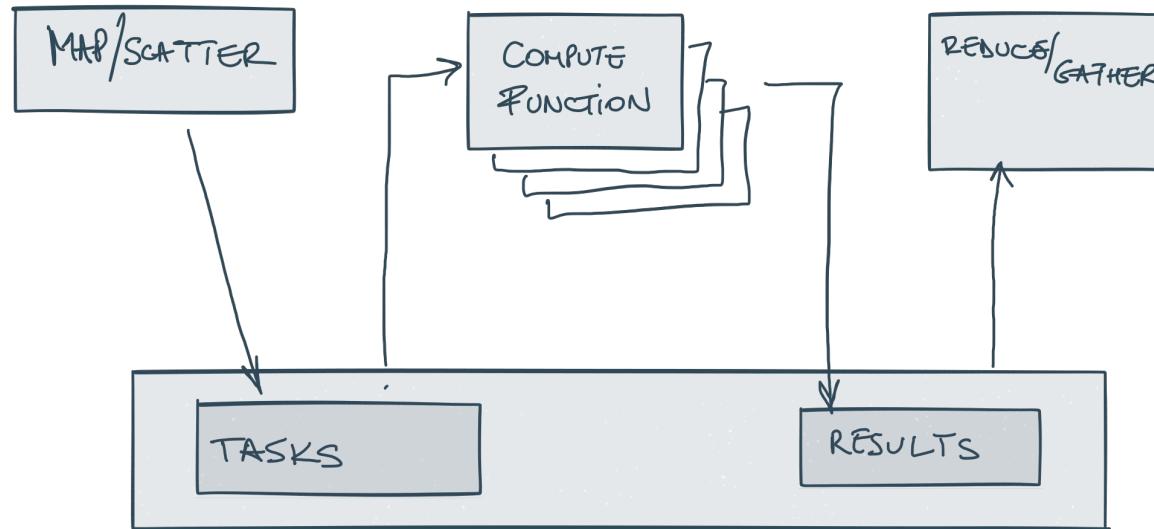
Event-driven microservice use cases applied to serverless

Streaming ETL, CDC

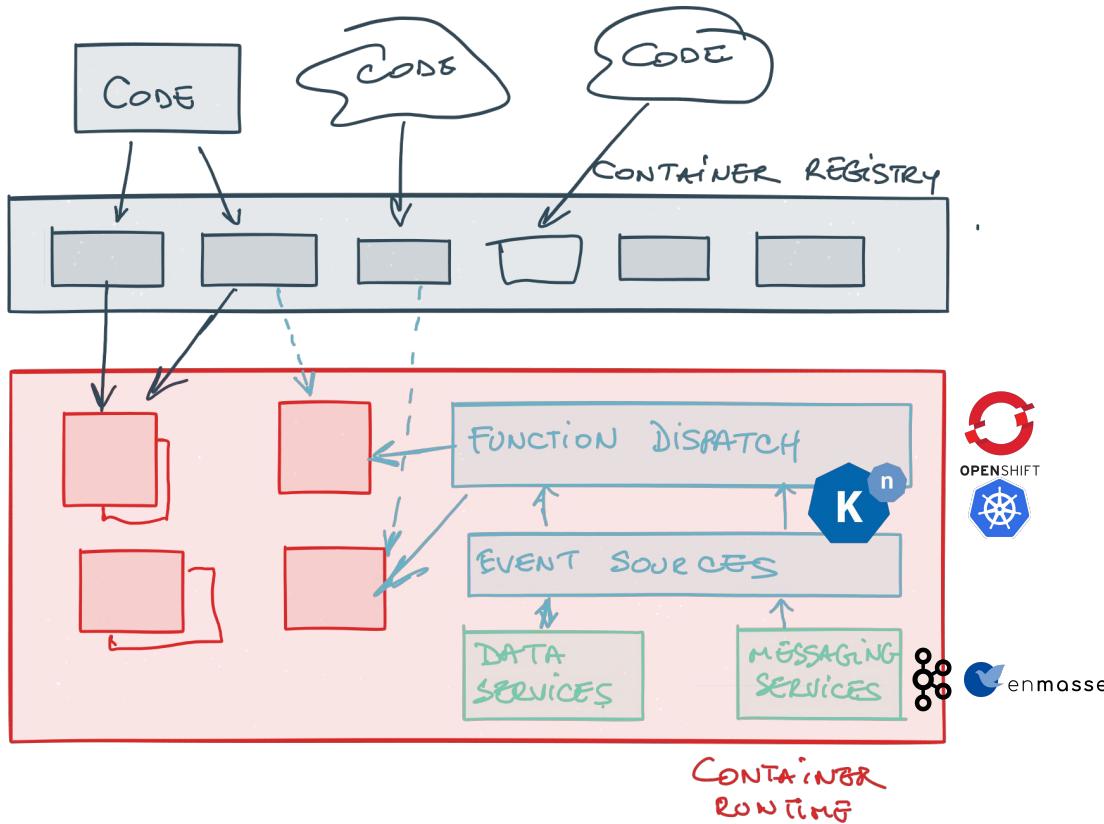


Event-driven microservice use cases applied to serverless

Load Balancing and High Scale Compute



Container-centric microservices and functions on Kubernetes

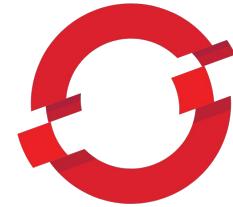


Optimizing utilization outside of autoscaling

- Impedance mismatch between runtimes optimized for physical/virtual environments and containerization
 - Large footprint (low density)
 - Long startup times - latency
- Recommendations:
 - Favor technologies with small footprint and low latency, in particular for serverless
 - Favor technologies that allow for easy change of deployment model (independent deployment vs on-demand/serverless) while preserving investment in business logic

Your technology radar

- Service Mesh (e.g. Istio):
 - Provide microservice interconnectivity
- Serverless platforms (e.g. Knative)
 - Container build and on-demand scheduling
- Container-native frameworks (e.g. Quarkus)
 - Optimize Java workloads for containerized apps
- Strimzi - Kafka operator for Kubernetes/OpenShift
- EnMasse - Messaging-as-a-Service for Kubernetes/OpenShift
- FaaS frameworks (e.g. Camel-K)
 - Schedule integration code directly on platform or via Knative



OPENSIFT



Conclusions

- Event-driven microservices are key for implementing highly distributed, extensible architectures
- Serverless platforms are a natural fit for several event-driven microservice use cases
- Serverless architectures should complement event-based execution with event-centric design
- Always consider tradeoffs:
 - Serverless vs independent deployment (aka ‘traditional’ microservice)
 - Optimized runtime vs technical investment (can you have both?)