AN EVENTFUL TOUR FROM ENTERPRISE INTEGRATION TO SERVERLESS

Marius Bogoevici (@mariusbogoevici)
Christian Posta (@christianposta)
9 May, 2018

About Us

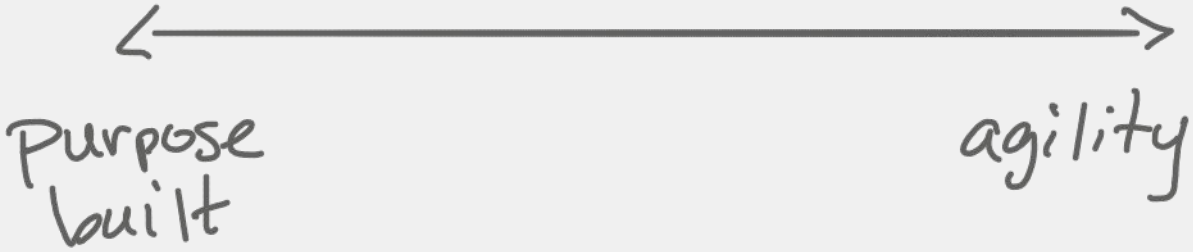
Marius Bogoevici
@mariusbogoevici
Chief Architect - Red Hat

Christian Posta
@christianposta
Chief Architect - Red Hat

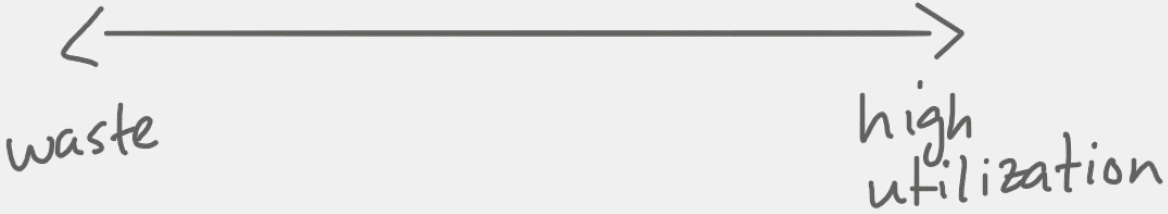
What we're going to talk about

- Event-driven architectures have been around for a bit; what are they, why are they powerful, and why are they back "en-vogue"
- Messaging is often used as a backbone for event-based distributed systems; what options do we have for cloud-native event-driven architectures?
- Integration is a necessary capability for any organization; how does streaming, cloud-native architectures and microservices fit in?
- Is Functions the next utopian architecture? Where does functions fit in a world of microservices?

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.



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.

We live in an event-driven world

Pop quiz!

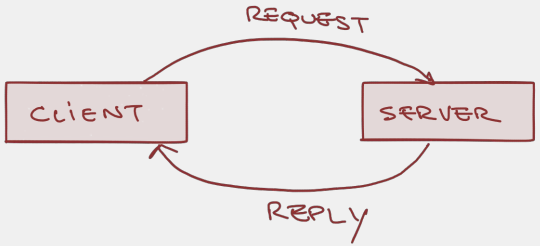
Event-Driven Architecture is a ...

Universal panacea

Solution to

- technical challenges
- business complexity
- bring agility, utilization, clarity into balance

Request-reply and event-driven interaction



Interaction: ephemeral and synchronous
Highly coupled
Low composability
Simplified model
Low tolerance to failure

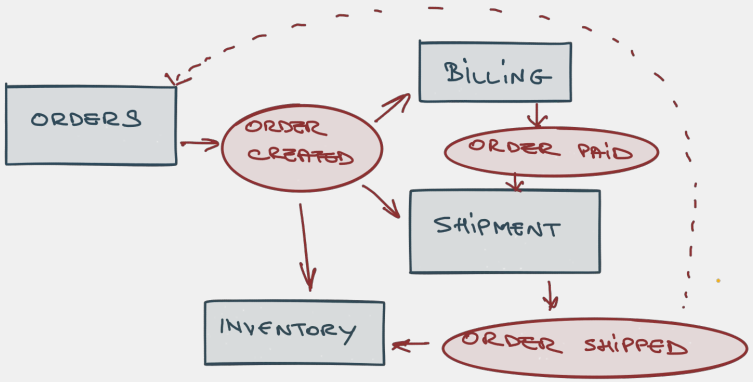
Interaction: persistent and asynchronous
Decoupled
Highly composable
Complex model
High tolerance to failure

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

Events in the digital business

- We live in an event-driven world (literally), and that impacts how we do business
- Next-generation digital business is about agility and experimentation
  - Shifting focus from analyzing the status quo to understanding the change in progress
  - Blurring the distinction between events and data
  - Architectural focus shifting from data-centric to event-driven
- Increased importance of bottom-up approaches in business event design
  - Complex event processing driven by experimentation, analytics, machine learning
  - Emphasis on readiness to observe and collect events before ascribing them a business meaning

Event-driven architectures reduce friction

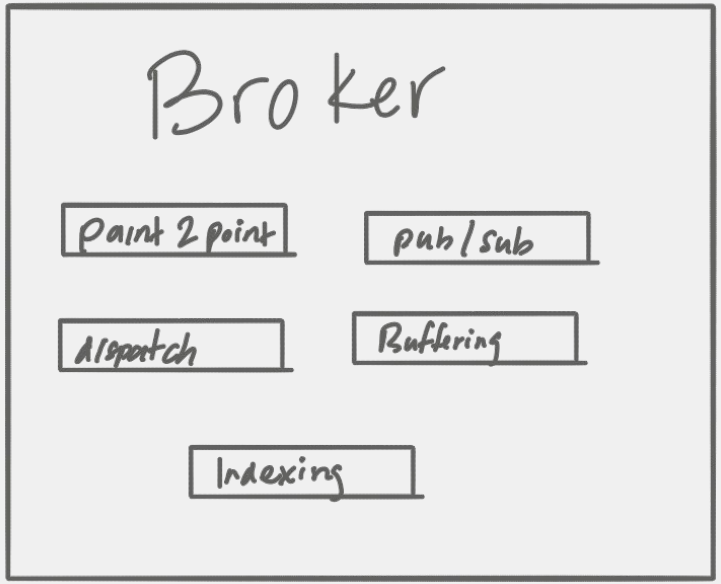
- From a technical standpoint:
  - Building robust and resilient distributed architectures
- From a development process standpoint
  - High composability encourage agility and experimentation
- From a business standpoint:
  - Aligning digital business with the real world

Delivering events through infrastructure

Event distribution infrastructure with message brokers

- Publish subscribe semantics (vs queuing)
- Subscribers receive events at their own pace
- High utilization of consumers, regardless of event publish
- Persistent vs non-persistent
- Example: ActiveMQ, RabbitMQ, etc

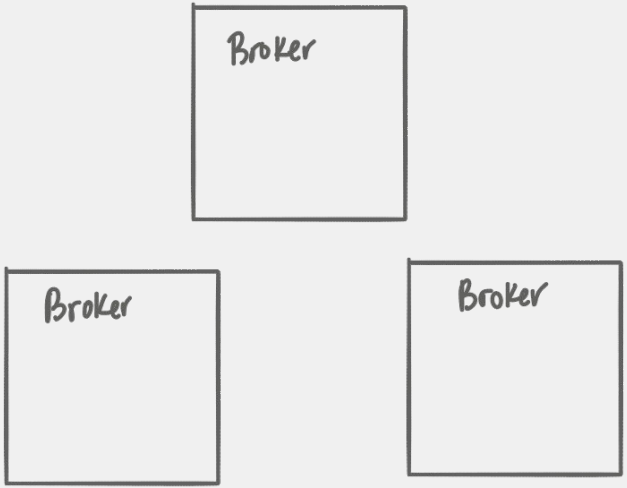
Event distribution infrastructure with message brokers



Types of events

- User activity tracking
- Infrastructure monitoring
- Business activity events
- Domain events

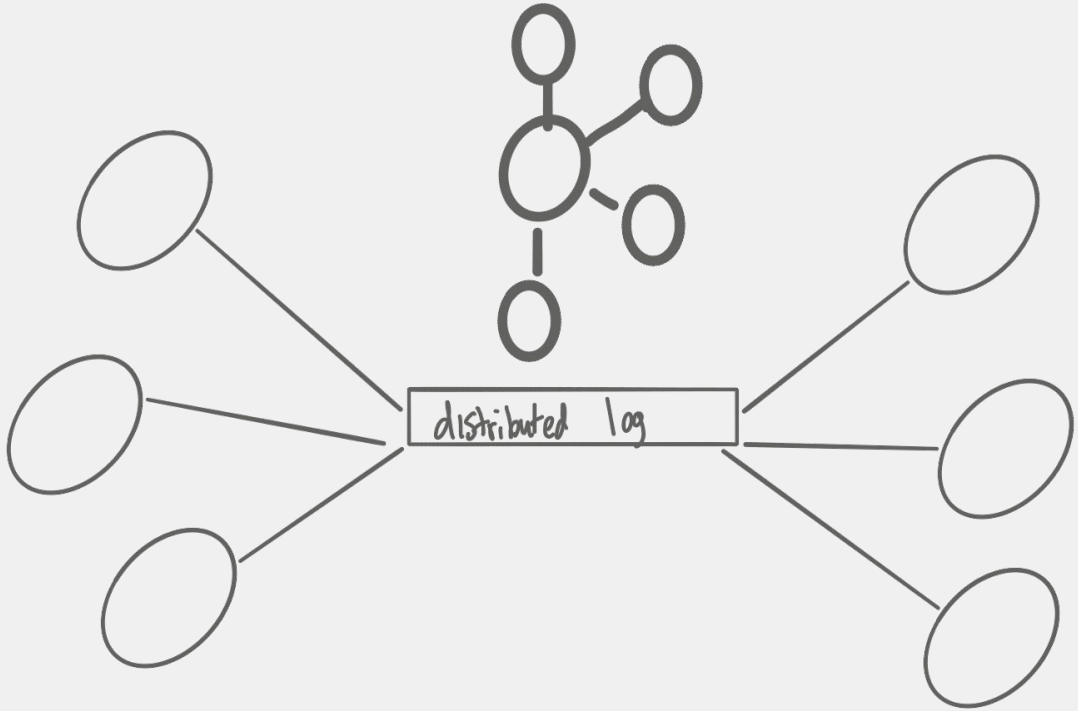
Event distribution infrastructure with message brokers



Handling large explosion in event sources, requires optimization in broker technology

- Decentralized processing
- Move indexing and bookkeeping to consumers
- Make fundamental data structure first class citizen (log data structure)
- Replication and failover part of the protocol
- Example: Apache Kafka, Kinesis, etc

Event distribution infrastructure with message brokers

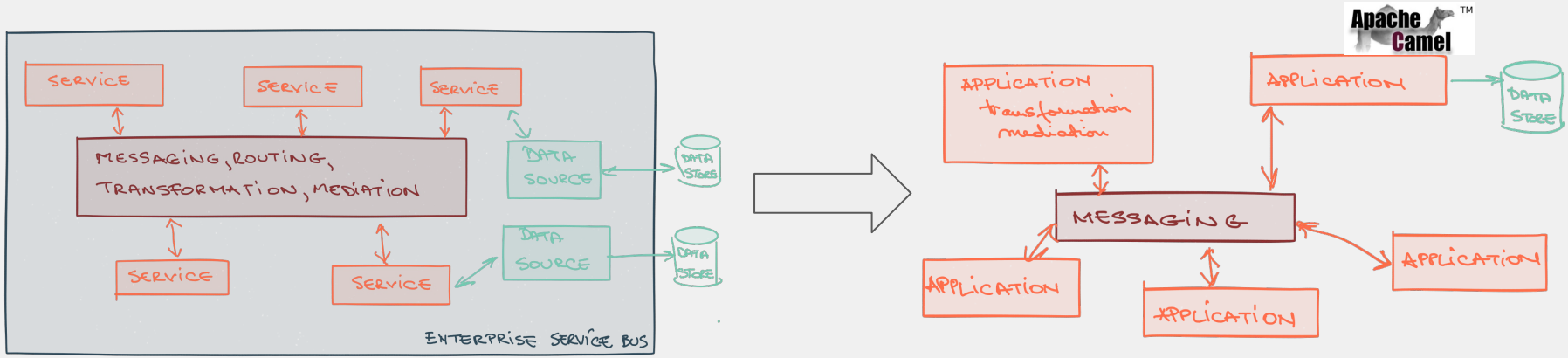


APACHE kafka®

STRIMZI

Using events for integration

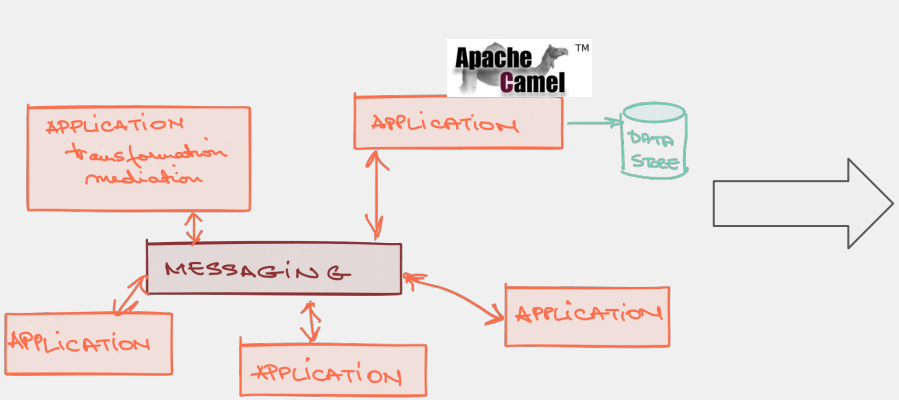
From ESBs to agile integration

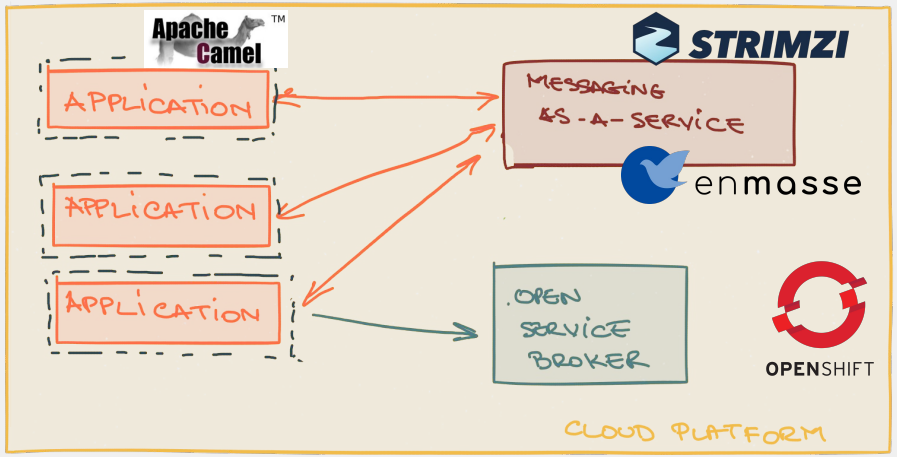


Optimized for utilization
Centralized, tightly coupled
Mixing logic with infrastructure

Optimized for agility
Decentralized, decoupled
Separate messaging middleware from logic

Modern enterprise integration: agile, decentralized, cloud-native





Optimized for agility
Decentralized, decoupled
Separate messaging middleware from logic

Preserves benefits of agility while optimizing resource utilization
Clear separation of concerns between compute and data infrastructure and application logic

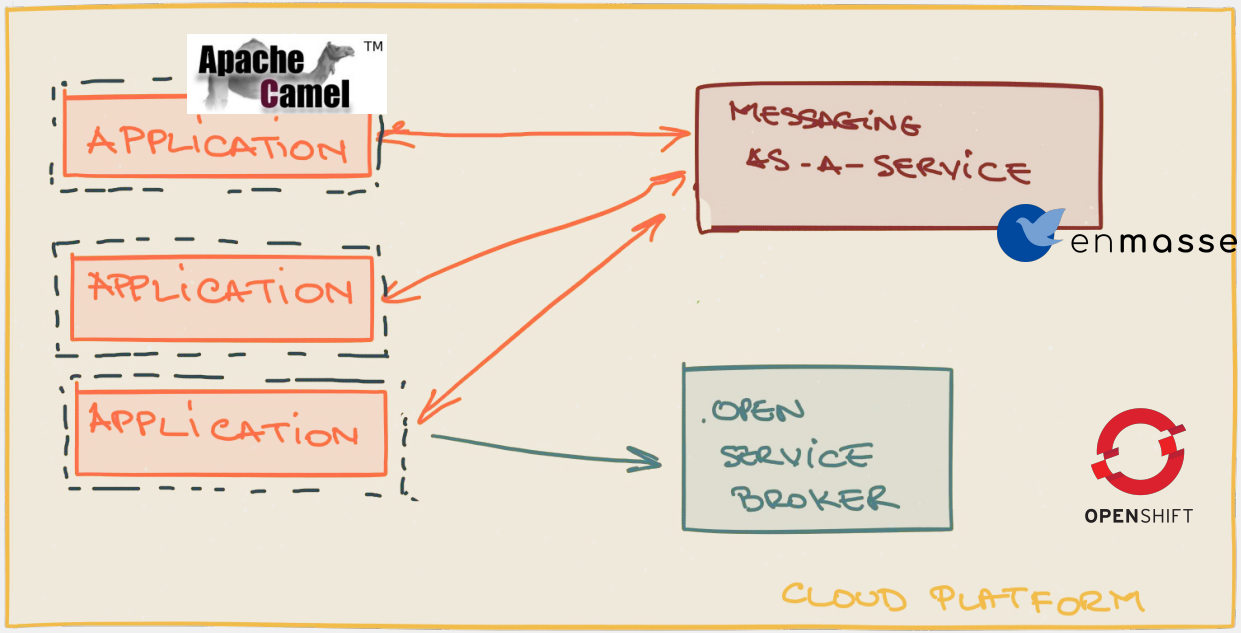
Enterprise integration patterns for microservices

- Originally designed for building integrated solutions out of siloed enterprise systems
- Applicable to general-purpose event-driven interaction
- Very well suited for building event-oriented distributed systems (aka event-driven microservices) - e.g. with Apache Camel

Enterprise Integration and Streaming

- Perpetual data and event “streams” as a first class citizen
- Data in aggregate vs individual messages
- Small services working together to interpret large numbers of streams
- Data in perpetual motion
- Eventual consistency as data synchronization pattern
- Examples: Apache Camel, Kafka Streams, stream-processing frameworks

Modern enterprise integration: agile, decentralized, cloud-native

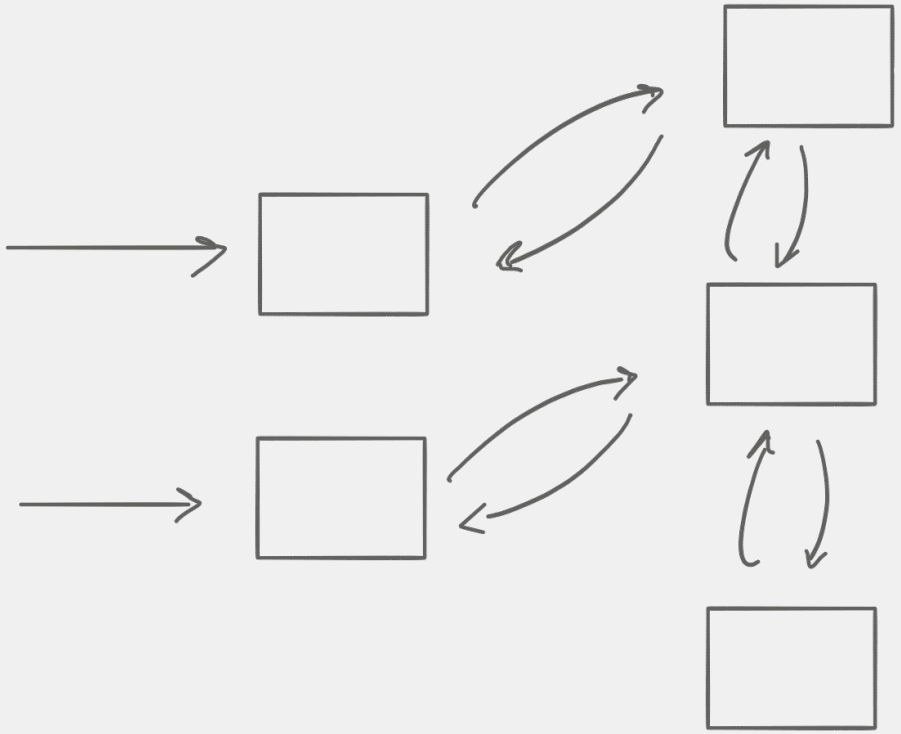


Microservices and Functions

Microservices

- Reduce the responsibility of a service to a specific business functionality
- Allow parallelization of work
- Independently deployable (infra) and independently releasable (business)
- Can optimize for increased utilization (separating out parts of the code base that exhibit different i/o, throughput, latency needs)
- Re-use services where applicable

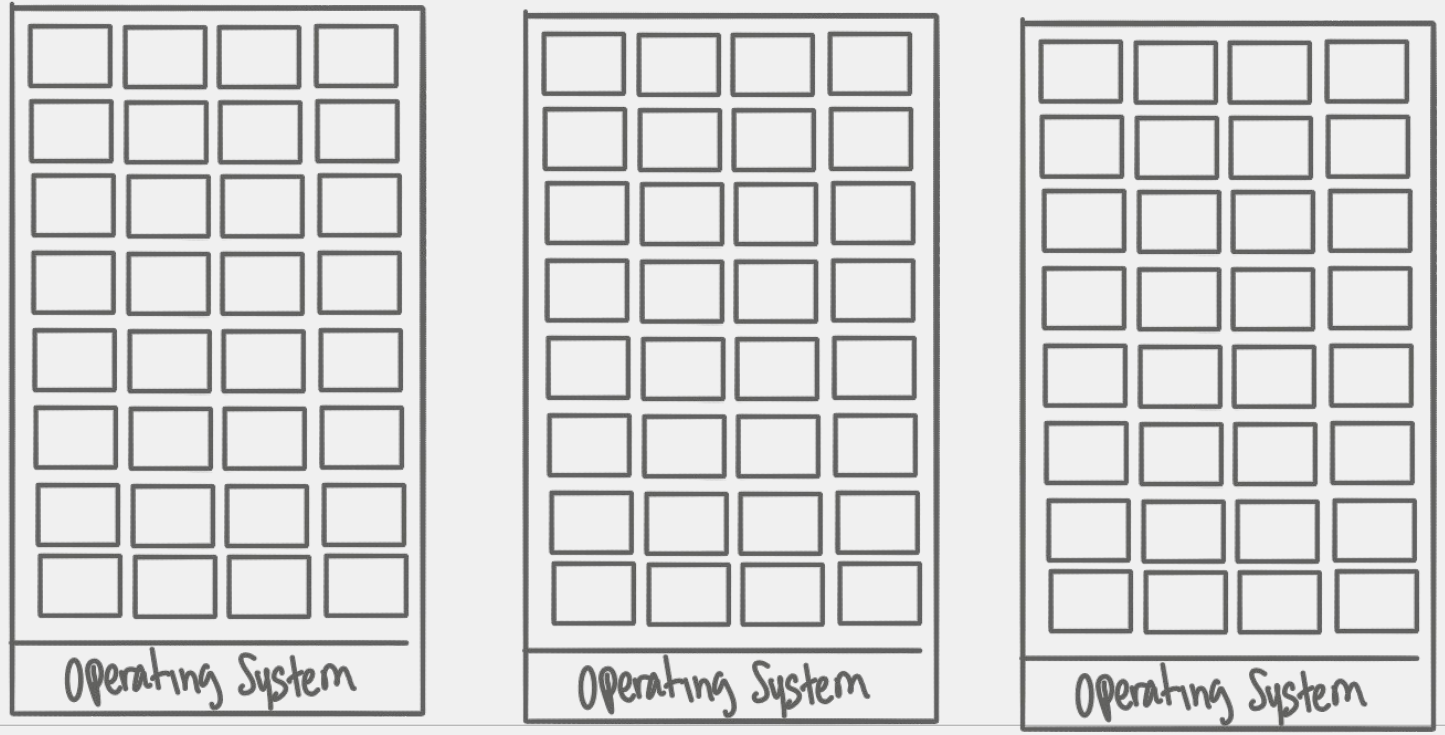
Microservices



Containerization

- Reduce overhead in running services
- Higher density/utilization gains
- Portable across deployment platforms
- Rich ecosystem (see Kubernetes!)

Containers and microservices



What’s your usecase?

- Microservices great at enabling agility from existing systems
- Well understood business, well understood boundaries
- Don’t optimize for microservices unless you have problems with your application architecture
- Don’t complicate experimentation / value discovery with complex architecture

Exploratory use cases

- Usually not well understood
- MVPs are throwaway
- Usage patterns unknown
- Adoption unpredictable

Under-utilization use cases

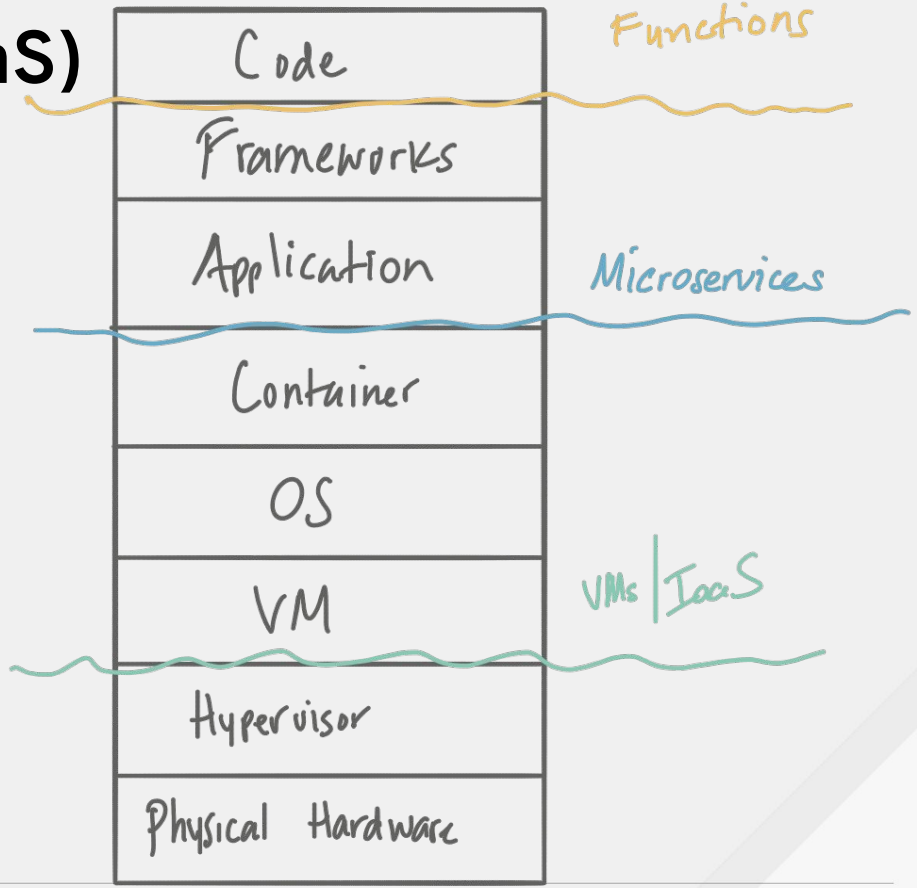
- Low number of hours/minutes of use
- Event-driven, spikey utilization
- Lots of compute for very short period of time

Limited integration use cases

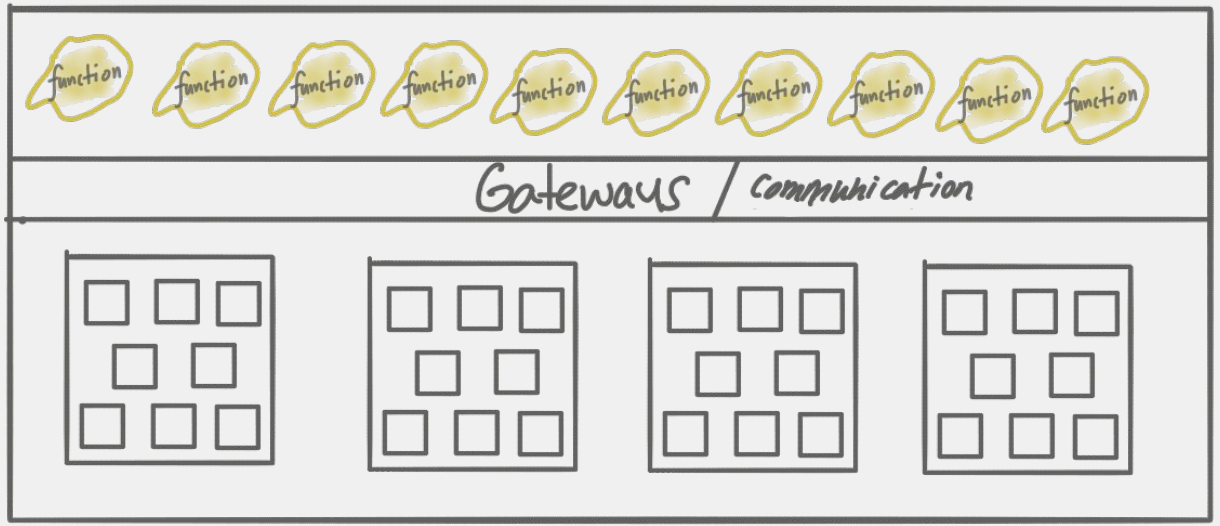
- Webhook callbacks
- Scheduled tasks
- File processing
- Reacting to database changes
- Limited stream processing

Functions as a Service (FaaS)

- Pay only for usage without regard for topology (Serverless)
- Event driven by nature
- On demand
- Write only code, heavy lifting is handled for you
- High parallelization
- High utilization



Functions as a Service (FaaS)



As you move to cloud native, you have *options*.

- Event-driven microservices
- Containers (on prem, in the cloud)
- Functions
- Serverless (databases, message queues, caches, etc)

Thanks for coming to our talk!

Marius Bogoevici
@mariusbogoevici
Platform Architect - Red Hat

Christian Posta
@christianposta
Chief Architect - Red Hat

THANK YOU


plus.google.com/+RedHat


facebook.com/redhatinc


linkedin.com/company/red-hat


twitter.com/RedHat

youtube.com/user/RedHatVideos