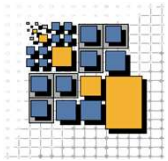
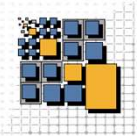


DSG-SOA-M 2024:
- Microservices -

Dr. Andreas Schönberger

Distributed Systems Group
Faculty Information Systems and Applied Computer Science
University of Bamberg

What are Microservices?



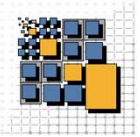
- The term *"Microservices"* refers to an architectural style that organizes applications as loosley coupled, independently manageable services
- Microservices are not new, first discussions date to 2012
- No single document or specification of Microservices
  - Diverse sources claim to describe the Microservices architectural style
  - The probably best description is provided by Martin Fowler and James Lewis:
    "Microservices - a definition of this new architectural term",
    http://martinfowler.com/articles/microservices.html

[FL14] in the following

- Target application area are *enterprise applications*
- The word "micro" in Microservices is misleading and does not mean creating as small services as possible
  → instead services are aligned with business capabilities

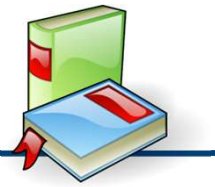
sounds like SOA?

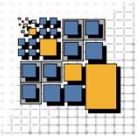
Why Microservices?



- Get a modular, evolvable and robust architecture
- Decouple development
  - Speed up dev / build / release
  - Enable individual infrastructure and code upgrades per service
- Flexibly choose between programming frameworks / persistence
  → Polyglot programming, persistence
- Tailor non-functional requirements to individual services
  - availability
  - scalability
  - performance
  - …

Essential principles of Microservices

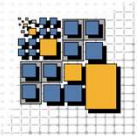




[FL14]:

- Componentization via Services
- Organized around Business Capabilities
- Products not Projects
- Smart endpoints and dumb pipes
- Decentralized Governance / Polyglot X
- Decentralized Data Management
- Infrastructure Automation
- Design for failure
- Evolutionary Design

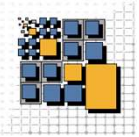
Componentization via Services



- ‘Services’ are one answer to the ‘software reuse’ problem on a coarse-grained level
  - Services encapsulate a coherent piece of functionality
  - Services require an explicit interface

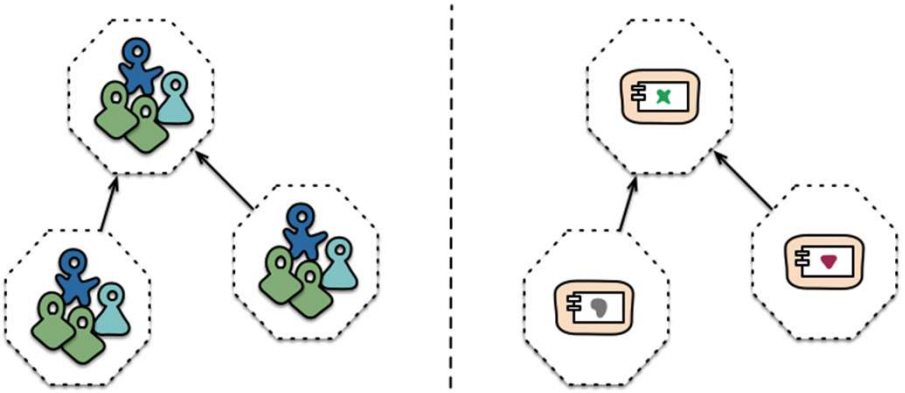
    not necessarily WSDL!
  - Services are independently replaceable, upgradeable and deployable
    → suggests services are reachable via the network
  - Services are different from libraries that are linked into a program Changing libraries may require redeploying surrounding application
- [FL14]:
  “At a first approximation, we can observe that services map to runtime processes, but that is only a first approximation. A service may consist of multiple processes that will always be developed and deployed together, such as an application process and a database that's only used by that service.”

Organized around Business Capabilities



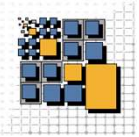
- Organization around business capabilities translates into
  - shaping cohesive sets of functionality that add value (=services)
    → Requires good domain know-how and enforces aware decision making
  - aligning team structures with services to avoid breaking the defined services cut (Conway's Law)
- Requires a well-defined and somewhat stable scope

[FL14]: *"The microservice approach to division is […] splitting up into services organized around business capability. Such services take a broad-stack implementation of software for that business area, including user-interface, persistent storage, and any external collaborations."*



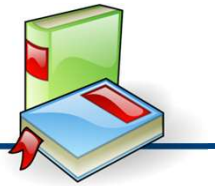
Src: [FL14]

Products not Projects

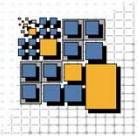


[FL14]:

- Project Model:
  Development team develops application that is later operated by a different team
- Product Model:
  “You build it, you run it” *(Werner Vogels)*
- Amazon reports that assigning responsibility of the software in production to development teams greatly improves quality and increases understanding of the domain
  ➔ Clearly calls for the product model, nowadays *DevOps*
  (May be hard to realize in embedded / automation settings!)



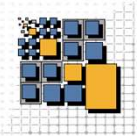
Smart endpoints and dumb pipes



- Communication between services can be provided by
  - dedicated communication / integration technologies that provide
    - stateless notification / request-response interactions (Internet protocols) or
    - reliable, asynchronous communication (Messaging technologies)
  - elaborate routing and orchestration entities that allow for message transformation, business rules application, long-running processes realization, interface mediation, brokerage etc. , i.e., EAI hubs or ESBs
- Following the latter approach
  - tends to distribute application logic between services and the hub
  - ignores that adapting services is better done inside services, not outside

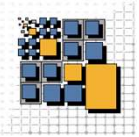
[FL14]:
"The two protocols used most commonly are HTTP request-response with resource APIs and lightweight messaging"

Decentralized Governance / Polyglot X



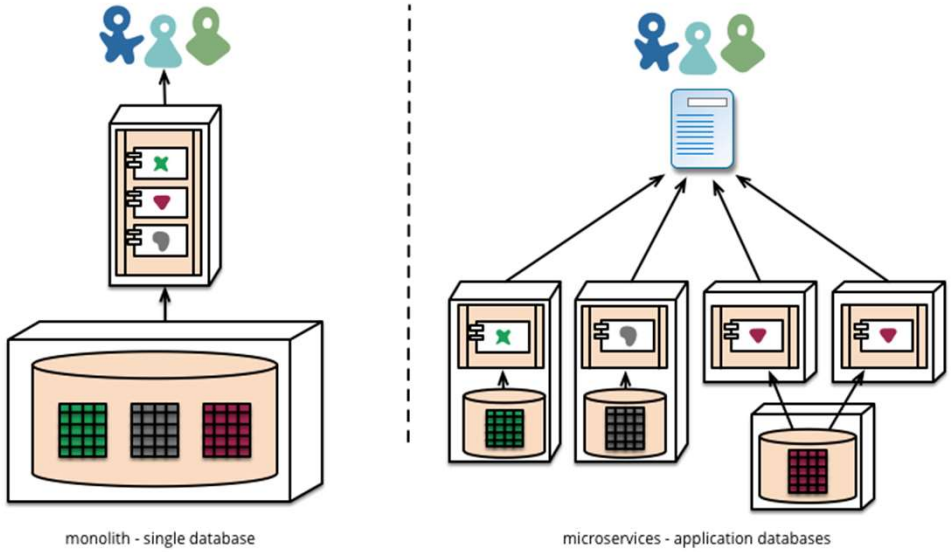
- The goal of services governance is ensuring that communication partners *"behave well"*
- What is subject to Governance in services-based systems?
  - Functionality of common/shared services
  - Service interface styles and SLAs
  - Programming language/platform
  - Persistence mechanism
  - Tool-chains
- Centralized governance, e.g., by means of services repositories and registries, tends to disregard individual needs
- Conversely, decentralized governance gives more freedom
  - Let consumers drive service contracts by adding assertions on functionality
  - Polyglot X: Choose the programming language / persistence mechanism you want (the service encapsulates it anyway)

Decentralized Data Management

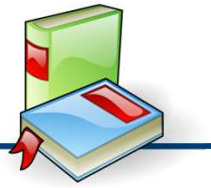


- Each service is responsible for its own data and data storage
- Allows for different views on the same data
  (for example sales, system engineering, software engineering)
- Greatly enhances testability

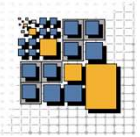
- Allows for Polyglot Persistence
- Suggests transaction-less coordination between services which requires error handling
  - Write-Off
  - Retry
  - Compensate



Src: [FL14]



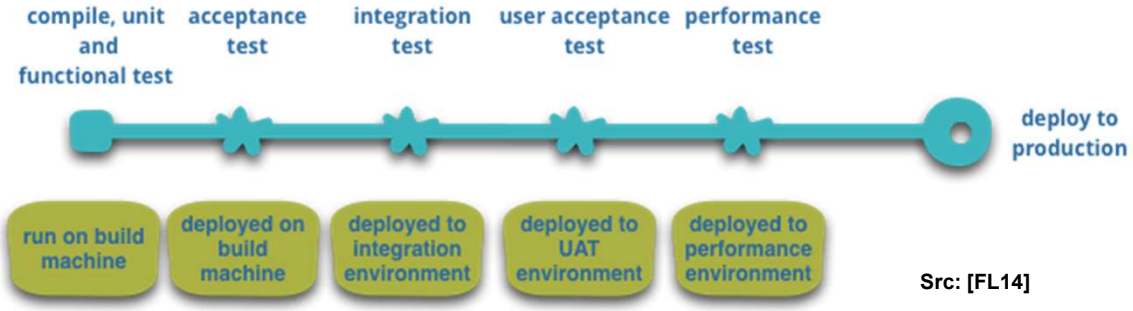
Infrastructure Automation



- Handling a growing number of microservices forbids lengthy and error-prone manual assembly, deployment and testing

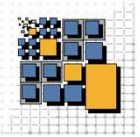
Instead:

- Use automated tests to discover implementation flaws and unintended side effects quickly and easily → gain control over what you do
- Use automated deployment to deliver new functionality quickly and to allow for frequent integration across services
- Keep development and production environments as similar as possible (barely possible without automation)



Src: [FL14]

Design for failure

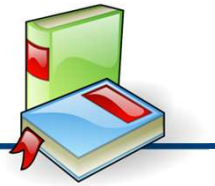


[FL14]:
“Since services can fail at any time, it's important to be able to detect the failures quickly and, if possible, automatically restore service.”

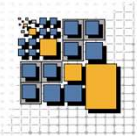
- Accepting somewhat independent management of individual microservices means managing interdependencies of services and dealing with emergent behavior is more important
- Real-time monitoring much more important
  - Concerning classical monitoring targets such as hardware usage
  - Concerning semantic monitoring targets such as business transactions processed

  →should be used as an early warning system
- Think about challenging the resilience of the system, think about the chaos monkey!
- Motivates redundant system design ☺

https://blog.codinghorror.com/working-with-the-chaos-monkey/



Evolutionary Design

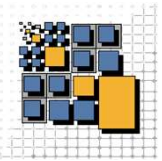


The world is changing, so are requirements

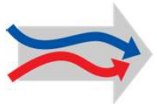
- Assumption: service interface changes are inevitable over time
  - → Well-defined interfaces that do not need to be changed are a myth!
  - → Make it as easy as possible to change interface definitions
- [FL14] make *"[…] frequent, fast, and well-controlled changes"*

Domain-Driven Design!

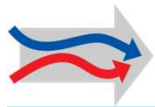
  - Cohesion and loose coupling foster replaceability and upgradeability
    - things that change together shall be kept together, i.e., within one service boundary
    - Conversely, things that do not change together should be kept separate to foster independent release cycles
  - Use test automation, fault-tolerant clients and consumer-driven contracts to enable partial upgrade of service landscapes
  - Use code repository analysis to investigate change behavior over time
  - Use APIs of core components to enable easy development of services
  - Consider the possibility to throw away services / write one-time services



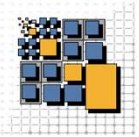
- Usage Guidelines for Microservices -



Distributed Systems Group
Faculty Information Systems and Applied Computer Science
University of Bamberg

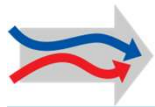


You are doing it wrong if …

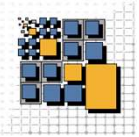


- you always have to deploy a release chain of services
- deploy multiple Services (war, ear) to the same JEE container
- share a database between services
- use EJBs or other non-lightweight protocol for service-to-service communication
- enforce centralized standards/libraries/stack/governance across all services

Src.: Michael Omann, Microservices, Senacor Technologies

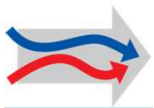


There is nothing wrong with …

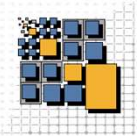


- services with more than XXX classes, XXX LoC
- blocking calls to a database or services
- not using asynchronous calls
- not deploying to a container
- not using a service discovery framework
- using relational databases
- not using JSON/HTTP

Adapted from: Michael Omann, Microservices, Senacor Technologies

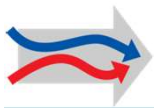


Best Practices for Cloud (Web) Development I
Implementation Guidelines for Microservices?

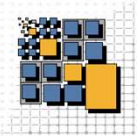


1. **Codebase**
   One codebase tracked in revision control, many deploys
2. **Dependencies**
   Explicitly declare and isolate dependencies, no system-wide packages
3. **Config**
   Store config in the environment
4. **Backing Services**
   Treat backing services as attached resources
5. **Build, release, run**
   Strictly separate build and run stages
6. **Processes**
   Execute the app as one or more stateless processes

Cf. The Twelve-Factor App,
https://12factor.net/

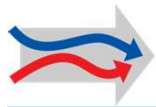


Best Practices for Cloud (Web) Development II Implementation Guidelines for Microservices?

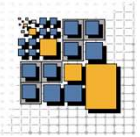


7. **Port binding**
   Export services via port binding
8. **Concurrency**
   Scale out via the process model
9. **Disposability**
   Maximize robustness with fast startup and graceful shutdown
10. **Dev/prod parity**
    Keep development, staging, and production as similar as possible
11. **Logs**
    Treat logs as event streams
12. **Admin processes**
    Run admin/management tasks as one-off processes

Cf. The Twelve-Factor App,
https://12factor.net/



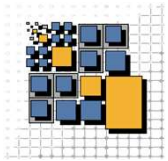
Liabilities of using Microservices



- Replacing in-memory calls of monoliths by remote calls of microservices may create inhibitive latency

Service interface != class interface

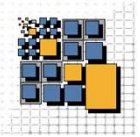
- Distributed Transactions
  - Not simply done as in Java EE
  - Alternative solutions such as building XA on top of HTTP, using message queuing, some event-channel between datastore, compensations or simply ignoring failures may not be a good match
- Service issues may propagate itself due to the latency on top and overload scenarios may emerge
  → Distributed nature may aggravate debugging
- End-2-End Testing may be significantly harder
- Infrastructure automation, design for failure and DevOps re-quire significant investments and probably process changes



- Microservices and Related Architectural Styles -

Distributed Systems Group
Faculty Information Systems and Applied Computer Science
University of Bamberg

Microservices vs SOA



Let's check core characteristics of Microservices against SOA and vice versa!

Src.: Pixabay

In the red corner:

Martin Fowler and James Lewis, describing the Microservices architectural style, MF's blog, 2014

In the blue corner:

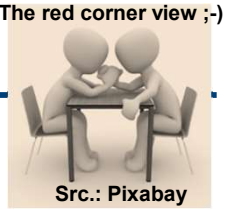
Hess, Humm, Voss, describing guidelines for creating SOAs, sd&m Research, 2006

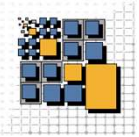
More authors! SOA obviously heavyweighted ;-)

[HHV 06] in the following

My favourite source for SOA principles

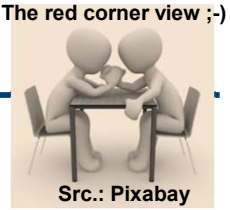
Does SOA comply with Microservices Characteristics? – Round I





Microservices principle [FL14]	SOA Answer
Componentization via Services	Same in SOA; WSDL-SOAP realization typical, but SOA never dictated WSDL!
Organized around Business Capabilities	Same in SOA
Products not Projects	Not in scope; DevOps emerged rather after SOA
Smart endpoints and dumb pipes	Same thing in early SOA; Later choreography and orchestration, service mediation and ESBs were added
Decentralized Governance / Polyglot X	Service-orientation implies decentralization and independent choice of language; However, many SOA approaches did a lot to reestablish centralized governance (SLAs, registries, global data models)

Does SOA comply with Microservices Characteristics? – Round II



Microservices principle [FL14]	SOA Answer
Decentralized Data Management	Data ownership is an implication of service-cut; Undermined by global data models in some SOAs
Infrastructure Automation	Toolchains from service definition to implementation and testing; not at the level of continuous delivery
Design for failure	Actually an implication of SOA; Monitoring and runtime analysis frameworks in scope; However, no chaos monkeys
Evolutionary Design	Modular design and monitoring are key SOA principles; Acceptance of interface changes rather low; Stringent XSD definitions tend to break easily; Advanced SOA concepts use more consumer-driven contracts

Do Microservices comply with SOA characteristics? – Round I

SOA principle [HHV06]	Microservices Answer
Business-driven component cut	Full comply
Loose coupling by means of services	Full comply
Service interfaces as contractual obligations	Yes, but with more focus on consumer-driven contracts and decentralized governance
Uniform communication mechanism and platform independence	Full comply; In addition, focus on more simple communication technologies
Simple implementation, replaceability of services, changeability of overall system	Full comply

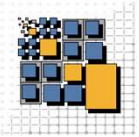
Do Microservices comply with SOA characteristics? – Round II

SOA design guideline [HHV06]	Microservices Answer
Domain-driven components	Full comply
One service category per component (master data, process data, application, interaction)	Fine with Microservices; No concern so far
Dependencies follow service categories	Fine with Microservices; general layering strategy
No cyclic dependencies	Full comply; general software engineering principle
High Cohesion, Loose coupling	Full comply; general software engineering principle
Data encapsulation	Full comply; general software engineering principle
Technology neutral	Full comply
No local references (files, complex objects)	Full comply
Normal, i.e., complete and no redundancies	Fine with Microservices; No concern so far

Do Microservices comply with SOA characteristics? – Round III

SOA design guideline [HHV06]	Microservices Answer
Coarse granularity	Full comply (despite the name); confer [FL14]
Idempotent	Yes, if possible
Context free	Fine with Microservices
Coupling mechanisms follow domain coupling	Full comply
Transaction control follows domain coupling	Fine with Microservices; in particular, consider giving up transaction control if write-off / retry / compensation is cheap and easy
Self-descriptiveness of messages follows domain coupling	Full comply

Microservices vs SOA: Summary



The one and only architectural style is for the consulting companies‘ sales guys!

Src.: Pixabay

Martin Folwer, James Lewis, [FL14]:

„[…] however, we aren't arguing that we are certain that microservices are the future direction for software architectures. While our experiences so far are positive compared to monolithic applications, we're conscious of the fact that not enough time has passed for us to make a full judgement.
Often the true consequences of your architectural decisions are only evident several years after you made them.”

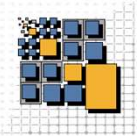
Sam Newman, Building Microservices, 2015:

“The microservice approach has emerged from real-world use, taking our better understanding of systems and architecture to do SOA well.
So you should instead think of microservices as a specific approach for SOA in the same way that XP or Scrum are specific approaches for Agile software development”

IMHO:

Microservices are an evolution of SOA with focus on practical realization!

Do Microservices dictate usage of REST?



First, please accept that Microservices and REST target at different application scenarios!

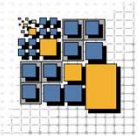
Roy Fielding (father of REST):

"The REST interface is designed to be efficient for large-grain hypermedia data transfer, optimizing for the common case of the Web, but resulting in an interface that is not optimal for other forms of architectural interaction."

So, Microservices at no means dictate using REST, but some of the REST principles fit well with Microservices!

See next slide

Do Microservices dictate usage of REST?

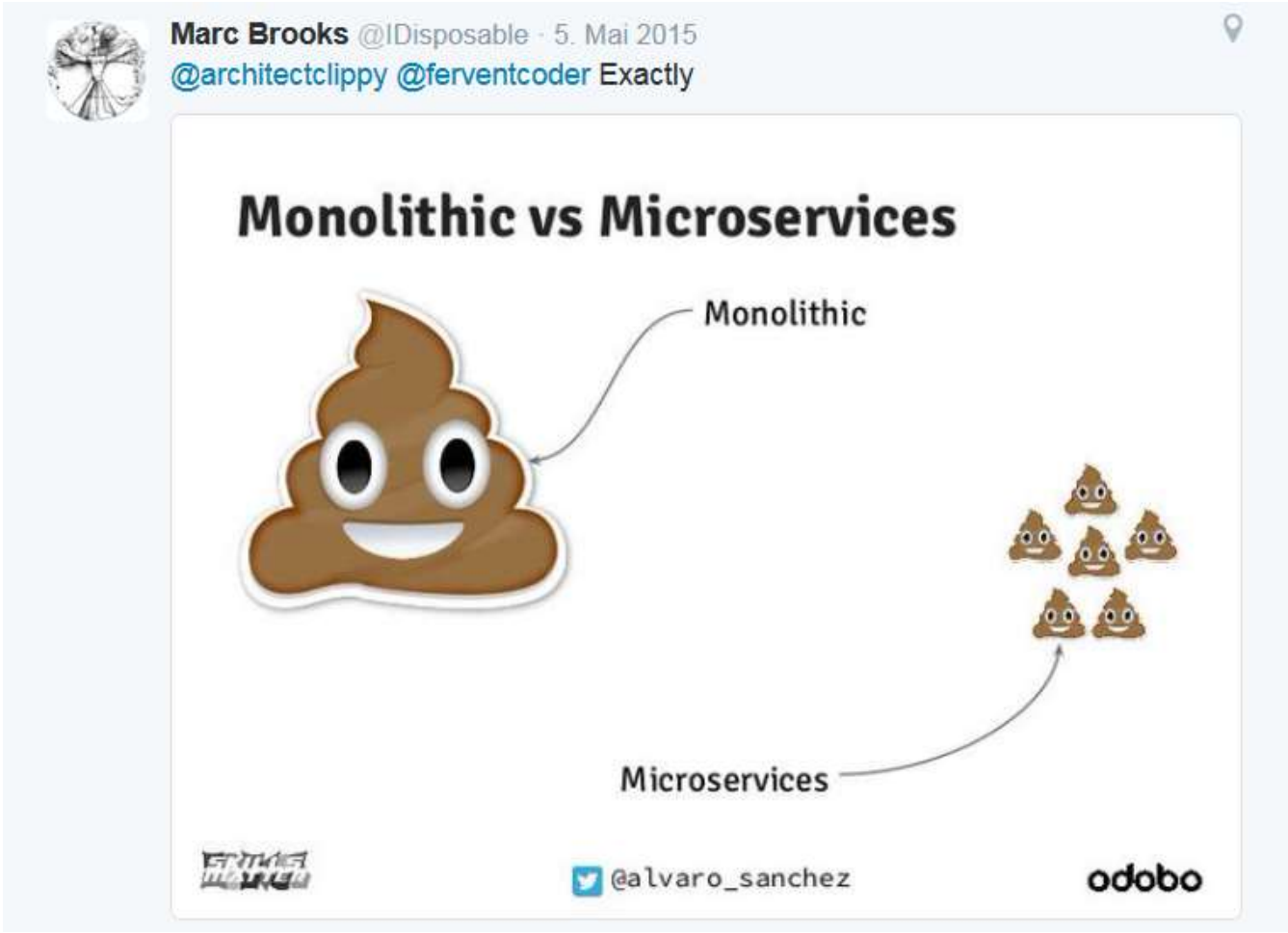


Let us examine the REST principles:

- Addressable resources
  - → Microservice resources obviously shall be addressable,
    but usage of absolute identifiers is of less interest, because
    processing by intermediaries and caching does not play a major role
- Representation-oriented manipulation of resources
  - → Increases quality of Microservices, but it is not a must
- Self-Descriptive Messages
  - → As regards application data description, increases quality of Microservices;
    As regards processing / caching of messages by intermediaries, not a concern
- Stateless communication
  - → It is hard to think of distributed systems that do not benefit from statelessness
- HATEOAS
  (Hypermedia as the engine of application state)
  - → Increases quality of Microservices, but it is not a must

Knowledge of Fielding's original writing assumed; not some *pseudo REST* source

No Comment


Marc Brooks @IDisposable · 5. Mai 2015
@architectclippy @ferventcoder Exactly
Monolithic vs Microservices
Monolithic
Microservices
@alvaro_sanchez
odobo