Sq

What traditional enterprise applications can take away from the microservices approach

The Microservices approach: an architectural prerequisite for Agile methods in enterprise applications?

Niek de Greef
ndegreef@seneqa.nl

Topics

- What is a microservices architecture
- How is it discussed contrasting “traditional” application architecture
- Microservices and traditional application architectures
- The benefits of the microservices approach for traditional applications
- Wrap up

Summary

- Microservices thinking brings new concepts to the table that are just as applicable to the traditional application space
- Is a reaction to legacy issues in new application space (Java etc)
- Microservices is not (only) about technology
- Mostly microservices is about discipline and organisation
- Some technological concepts are not mature yet, some are not addressed
- Automation of IT is everything

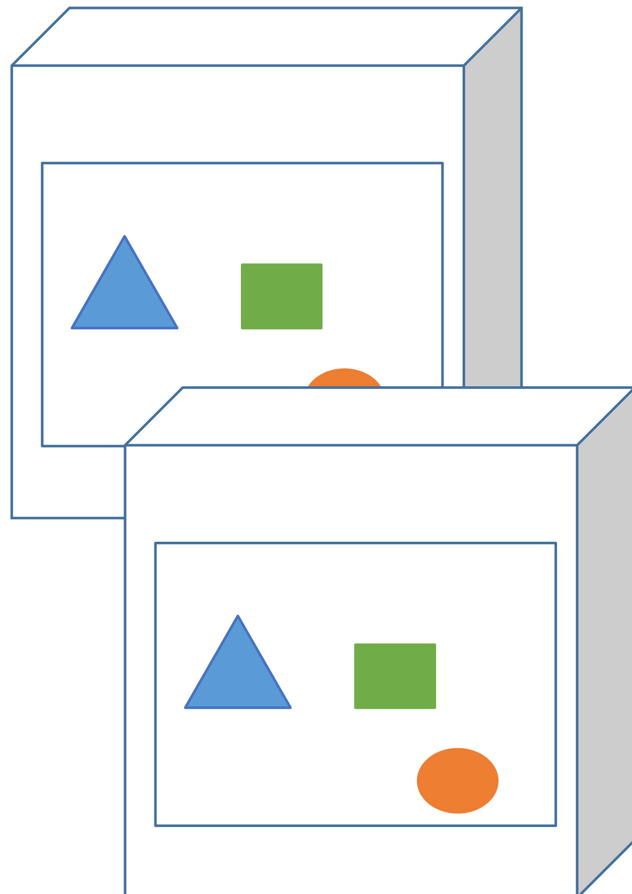
Microservices and the monolith

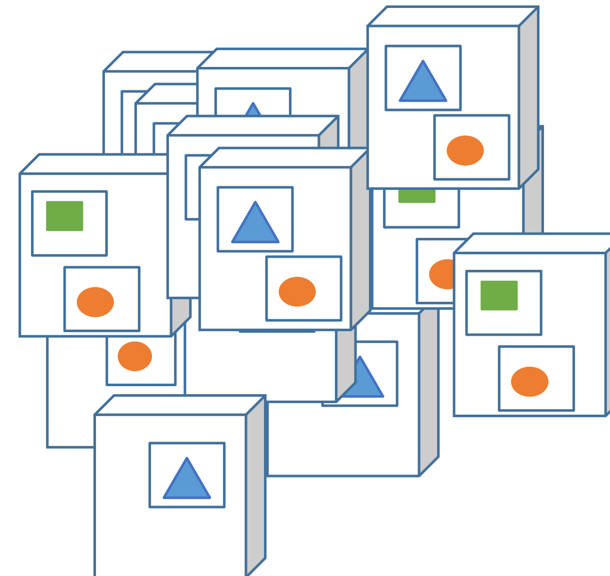
The MONOLITH

microservices

All functionality is in a single application

Scaled as a whole on multiple servers





Functionality is divided into small independent pieces

Scaled independently on many many servers

Microservices – everything hot?

An architecture style, supported by technology
Though vendors of software may tell you different

Many sources mix microservices with so much other stuff you can't see the wood through the microservices trees

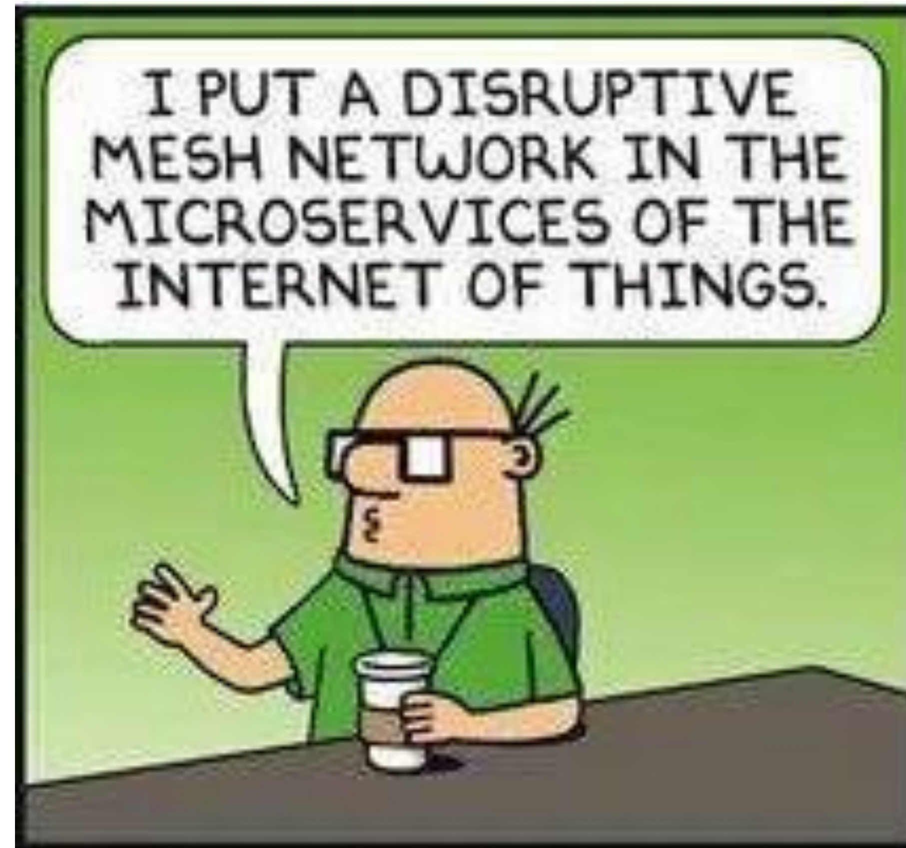
Agile

API Management

"Conway's Law"

Coordination is key
Eventual consistency
Closed source / open source
Cloud
Fine grained SOA
Containers
Servers on demand
DevOps
More semantics
Infrastructure to the application layer
Developer driver infrastructure
Fintech
Ephemeral services
Continuous change
Marketplace from cathedral to bazar
A function as a company, Service as a company, company as a service, a microservice as a product
A developer a service

Kubernetes
Node.js
Cassandra
Kafka
Swagger
NoSQL
Docker (Swarm mode)
OpenStack
WildFly
Mesos
NGINX
Many more


I PUT A DISRUPTIVE MESH NETWORK IN THE MICROSERVICES OF THE INTERNET OF THINGS.

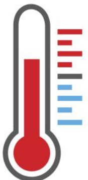
Microservices – everything hot?

Coordination is key
Eventual consistency
Closed source / open source
Cloud
Fine grained SOA
Containers
Servers on demand
DevOps
More semantics
Infrastructure to the application layer
Developer driver infrastructure
Fintech
Ephemeral services
Continuous change
Marketplace from cathedral to bazar
A function as a company, Service as a company, company as a service, a microservice as a product
A developer a service

Agile
API Management
"Conway's Law"
Kubernetes
Node.js
Cassandra
Kafka
Swagger
NoSQL
Docker (Swarm mode)
OpenStack
WildFly
Mesos
NGINX
Many more

An architecture style, supported by technology
Though vendors of software may tell you different

Many sources mix microservices with so much other stuff
you can't see the wood through the microservices trees



These are ALL technologies and concepts to relate to challenges around a microservices architecture
They do not **define** a microservices architecture

What defines a microservices architecture (Martin Fowler)

Components independently replaceable / upgradeable
Organised around business capabilities
Products, not projects
Smart endpoints and dumb pipes
Decentralised governance
Decentralised data management
Infrastructure automation
Design for failure
Evolutionary design

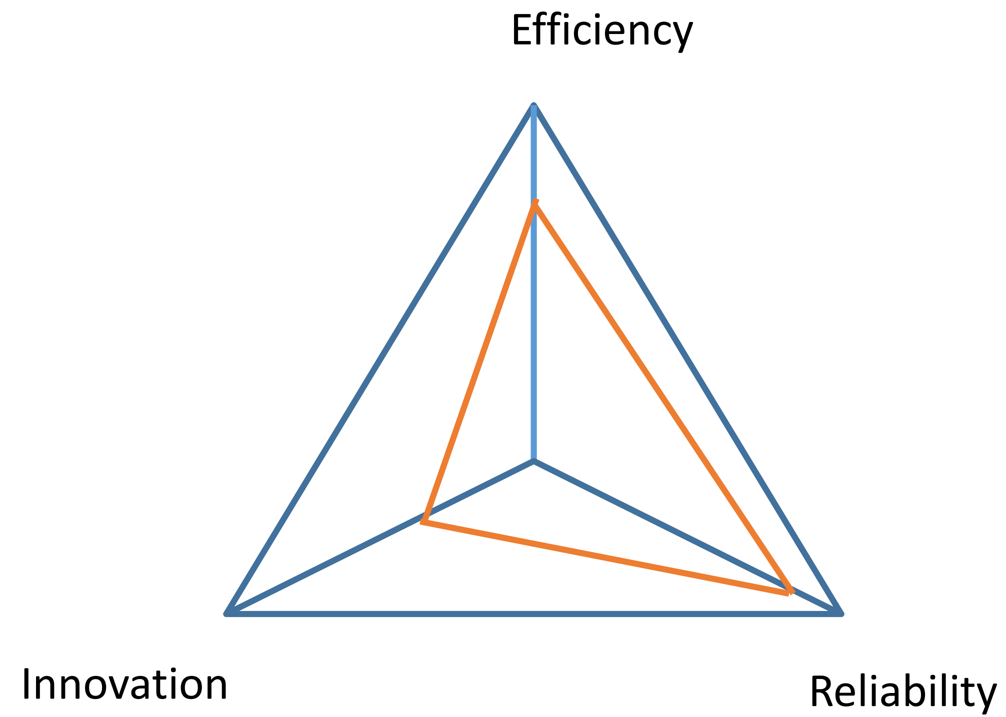
Microservices and traditional applications

- So how does microservices relate to our job: running traditional applications in a traditional environment
- Imagine a large ledger, such as a Current Account, or a Hotel Reservation application as a microservice
- Can we imagine these as microservices – e.g. working with 50 TB of data - and still call it micro?

- Microservices aims for flexilitity
- Traditionally we have optimized for robustness and efficiency

It’s a tradeoff

Sq



The monolith - a sidestep

Fowler's model monolithic application– used as the opposite of microservices architecture

This "reference" monolithic application

- a very large blob of code
- multitude of responsibilities
- operating all on the same set of shared data
- partial deployments
- bi-weekly check-in off ALL code, …

This monolith houses a large number of bad design practices

Existing best practices, especially w.r.t. decoupling, would have been alternatives.

The danger of this ultrabad example: any non-microservices architecture is a monolithic architecture that leads to big unmanageable blobs of code

Fowler is best source and most nuanced

The reference traditional application architecture

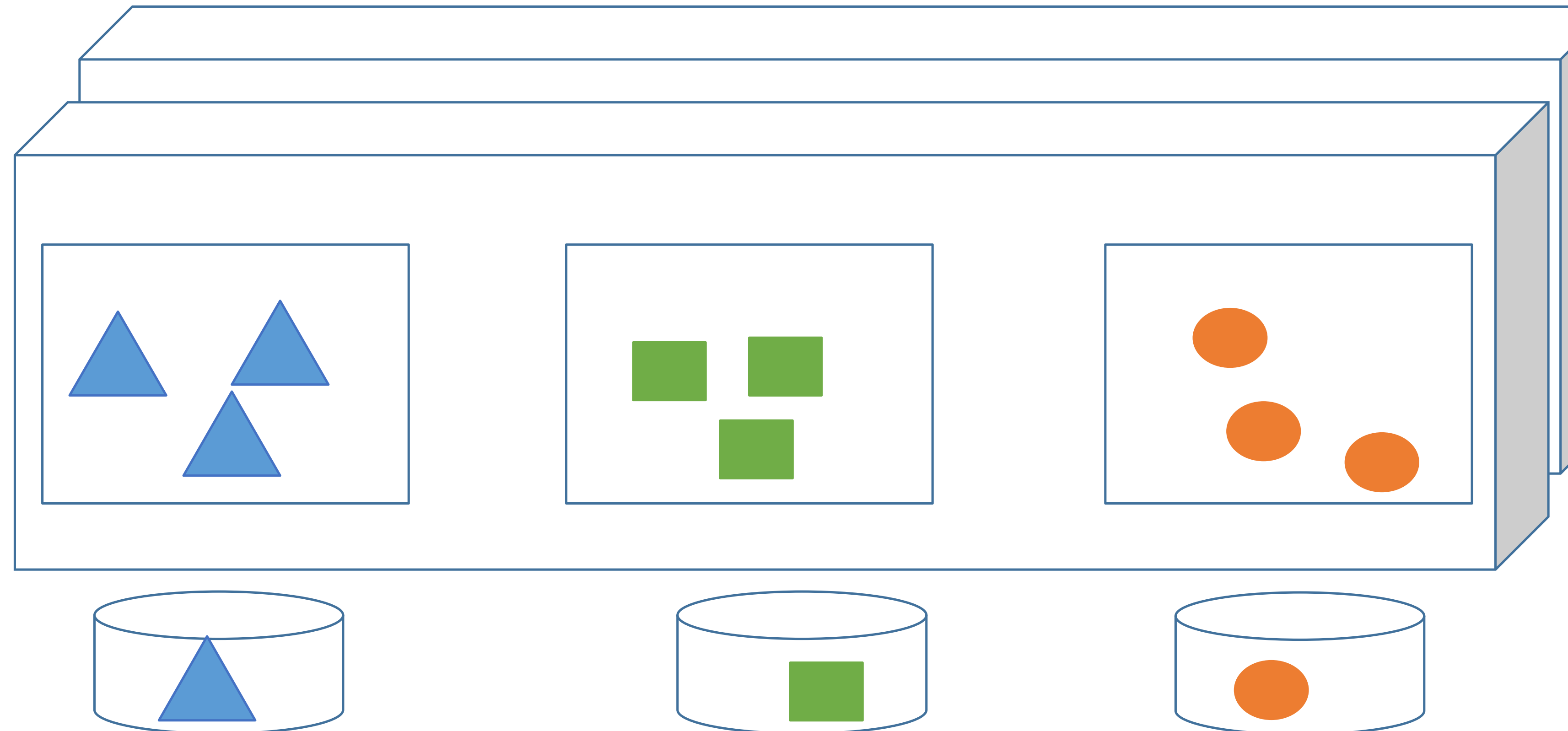
Sq

We (should) have adopted decoupling (since 1998 or so, if not ealrier)

Decoupled into application (components)

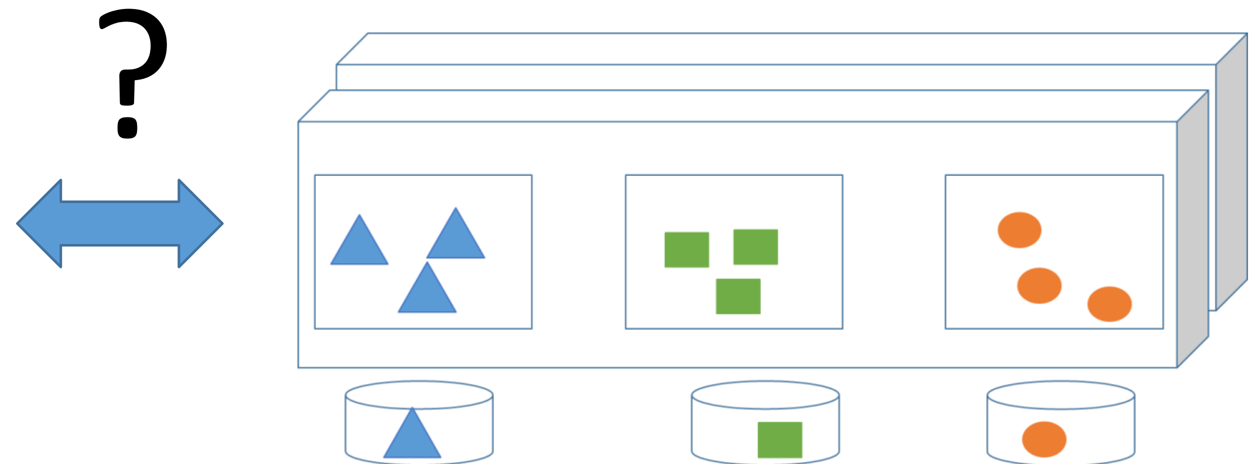
Application components responsible for their own data

Functionality exposed through services



What if we map the microservices characteristics to our traditional applications

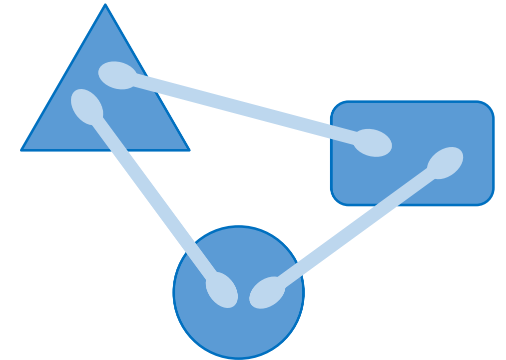
Components independently replaceable / upgradeable
Organised around business capabilities
Products, not projects
Smart endpoints and dumb pipes
Decentralised governance
Decentralised data management
Infrastructure automation
Design for failure
Evolutionary design



Discussion summary

Componentization via Services

Component: a unit of software that is independently replaceable and upgradeable.)



Take aways

- Long existing best practice: decoupling, also in traditional IT
- For optimization, local or networked interfaces can be chosen side by side
- But: we do need to put some guidelines in place to assure we cut up the application modules, in the right way, but that is a topic we'll address next.

Set up around business capabilities

Components are setup around business capabilities

Take away

- Is this already a practice in your development organisations?
- Does it still exist, seperate develop and maintain organisations?

Products, not projects

Product-oriented development leads to focused on business value, better quality, delivered with more flexibility

Take aways

- Yes we want it and again a prerequisite is a small granularity of services, so we need to assure decoupling
- No technical limitation to apply this in traditional applications space
- Adopted with the roll-out of agile methods and DevOps approached – so here microservice and agile converge

Smart endpoints and dumb pipes

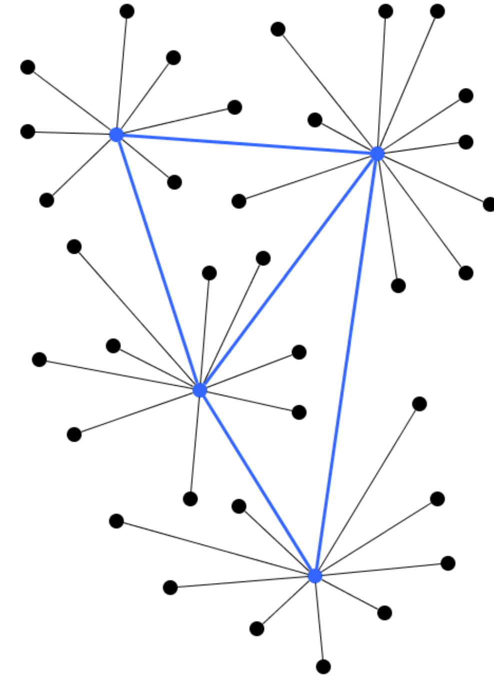
Lightweight communication, either though simple http request/response methods, or lightweight messaging for an assured-delivery asynchronous protocol.

Take aways

- Forget about ESBs (?)
- Clear service definitions, and discipline!
- We have a great legacy in stacking applications too rigidly – often historically driven by “cost” (or some other short term goal)
- Trade-off: distributed computing is extremely complex especially w.r.t. transactionality; local interfaces keeping transaction coordination ease can be preferred (but wisely)
- We must move away from integrating applications through tight coupling and only use that very wisely

Decentralised governance

Give teams power to decentrally develop applications, using tools for choice, interacting through open, published interfaces

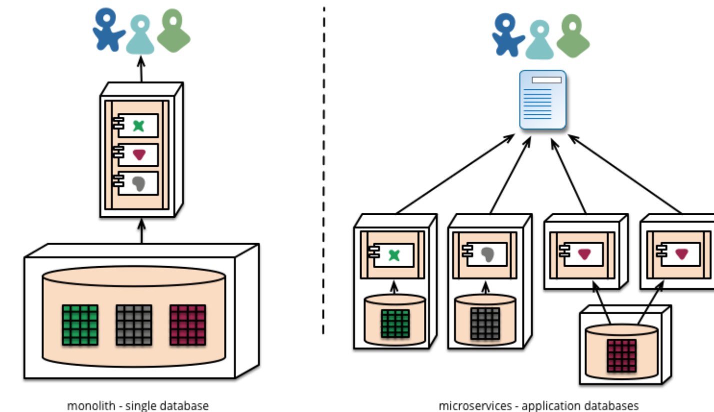


Take aways

- Traditional application are historically under centralized control. Culture, not technology, prevents adopting a decentralized governance model
- A technical prerequisite however is decoupling
- Totally decentralized governance? There are benefits in cost and operational efficiency to be gained from centralized governance.
- Also in the case of Netflix, there is a central team that puts in place frameworks for all DevOps teams to use for their development efforts – as a central governance example
- We want decentralised teams to work highly independent
- But rules and guidelines to assure pieces will work together. In other words: we need architecture
- And centralised service lifecycle management to prevent proliferation of service versions

Decentralized data management

Decentralized data management means each component is responsible for its own data



Take aways

- Very very complex: keeping data consistent in a distributed model
- Even the most fanatic microservices adepts warn for this one
- Even Netflix has separated Billing for this reason, where more care is required
- The virtues of a centralized data management should not be thrown overboard while the votes are still out on the use cases for distributed data models

Infrastructure automation

Automate provisioning of infrastructure to give developers maximum flexibility

Take aways

- This is a big need and also challenge in traditional space!
- Current approach to maintain large test environments is failing!
- In traditional applications (infrastructure) automation was never deeply invested in
- Technologies are emerging now that brings these traditional technologies to a state where they can benefit for (ultra) fast provisioning of infrastucture
- The maturity of the technologies supporting this however, is way beyond the technologies available for microservices.
- Suppliers and customers must be expected to significantly invest in this area.

Design for failure

Robots are configured that go about killing processes in the application stack to validate if applications can survive such breakages

“All great changes are preceded by chaos”

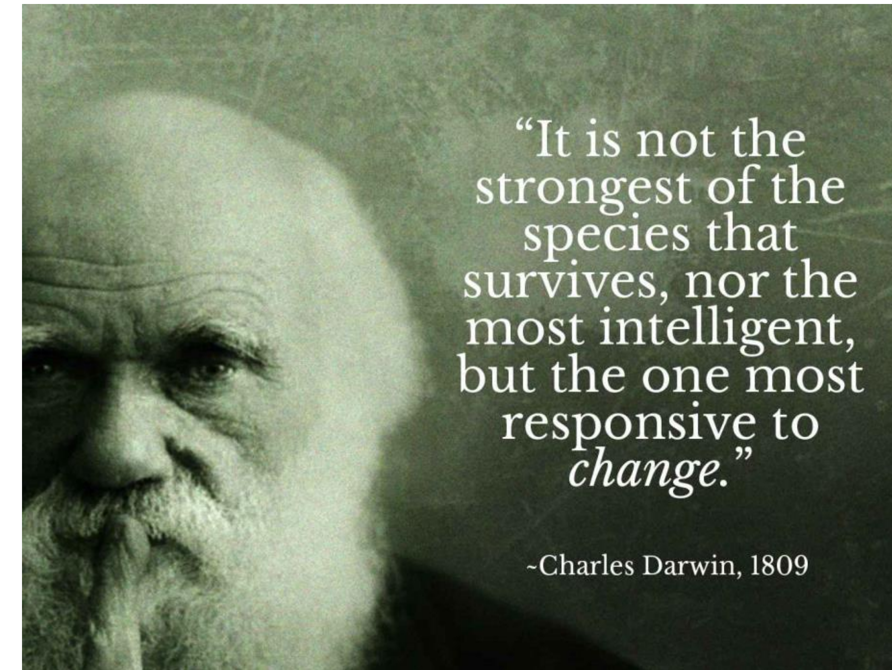
Anonymous

Take aways

- This ChaosMonkey is a great new concept
- To be explored for any business critical application
- Automates resilience testing to a high degree
- Test automation has become mainstream now, but this approach to also test-automate resilience testing could be called revolutionary

Evolutionary design

Service decomposition as a tool to enable application developers to control changes in their application without slowing down change capacity



Take aways

- I would rephrase this to evolutionary realization: the option to redo the internal structure of the service implementation (the design) as well as enhance the functionality of the service
- Again decoupling opens up possibilities for evolutionary design
- Hide service implementations from service consumers, allowing for an agile evolution of service functionality
- An agile organisation of the DevOps teams (decentralised governance) provides an evolutionary realisation of our services and business functionality

Unexplored territories

Sq

- Testing
  - How to set up test environment in such a distributed model. How to ensure testing with the right versions of all these flexible services.
  - Data management: the distributed data model. How to get a consistent test set. Automation is part of the answer. Stubbing might be another. Both apply to traditional applications as well.
- Security
- Batch
- Other?

Conclusions from this evaluation

- Things we should continue doing
  - Centralized data
  - Central high level architecture (but decentral design and realisation)
- Things we should start doing immediately in traditional application space
  - Radical application componentisation
  - Componentisation facilitates many other things
    - Agile organisation and delievry, evolutionary design, …
  - Automation of everything: Infrastructure, application, data, (failure) testing, …
  - Facilitate continuous testing
  - Can we do a ChoasMonkey like thing??

Some references

- https://www.youtube.com/watch?v=wgdBVIX9ifA
- https://martinfowler.com/articles/microservices.html#AreMicroservicesTheFuture
- https://martinfowler.com/articles/microservices.html#CharacteristicsOfAMicroserviceArchitecture
- http://obverseconsulting.com/2017/08/25/what-traditional-architectures-can-learn-from-microservices/
- https://www.infoq.com/news/2016/02/not-just-microservices
- https://www.youtube.com/watch?v=_dSA71NjVFE
- https://www.youtube.com/watch?v=57UK46qfBLY