Microservices Guide

In short, the microservice architectural style is an approach to developing a single application as a **suite of small services**, each **running in its own process** and communicating with lightweight mechanisms, often an HTTP resource API. These services are **built around business capabilities** and **independently deployable** by fully automated deployment machinery. There is a **bare minimum of centralized management** of these services, which may be written in different programming languages and use different data storage technologies.

-- James Lewis and Martin Fowler (2014)

A guide to material on martinfowler.com about microservices.

Martin Fowler

21 Aug 2019

Late in 2013, hearing all the discussion in my circles about microservices, I became concerned that there was no clear definition of microservices (a fate that caused many problems for SOA). So I got together with my colleague James Lewis, who was one of the more experienced practitioners of this style. Together we wrote

Microservices
common characteristics of this architectural style
by James Lewis and Martin Fowler

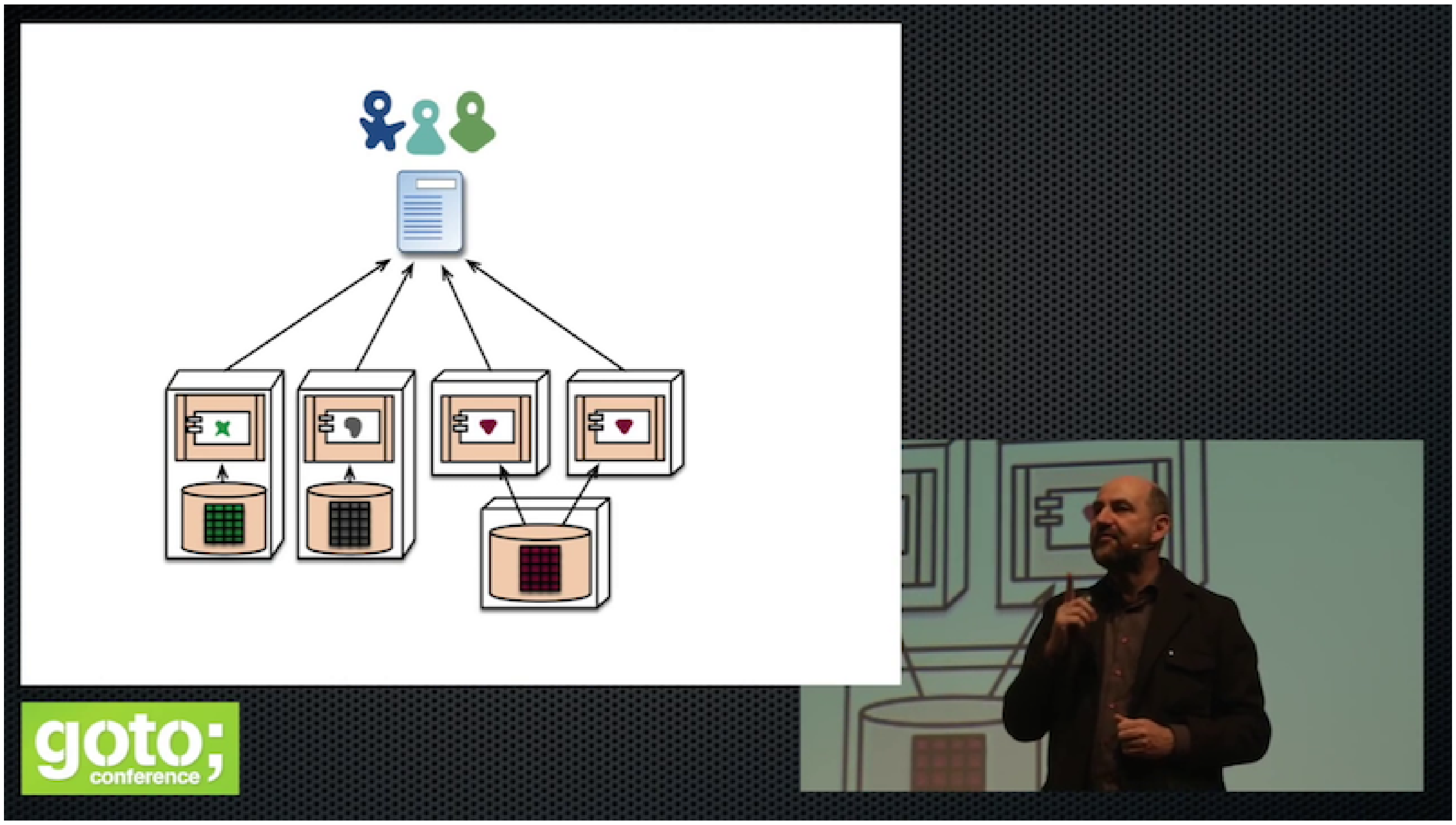
We wrote the article to provide a firm definition for the microservices style, which we did by listing the common characteristics of microservice architectures that we saw in the field.

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

We also looked at common questions such as "how big is a microservice" and "what's the difference between microservices and Service-Oriented Architecture". The article catalyzed interest in microservices.

"Do we use it, do we not use it?

... and what on earth is it in the first place?"



In my short introductory talk (~25 minutes) I pick out the most important defining characteristics, compare microservices to monoliths, and outline the vital things do before putting a first microservice system into production.

When should we use Microservices?

Any architectural style has trade-offs: strengths and weaknesses that we must evaluate according to the context that it's used. This is certainly the case with microservices. While it's a useful architecture - many, indeed most, situations would do better with a monolith.

Microservices provide benefits…

- ✓ Strong Module Boundaries: Microservices reinforce modular structure, which is particularly important for larger teams.
- ✓ Independent Deployment: Simple services are easier to deploy, and since they are autonomous, are less likely to cause system failures when they go wrong.
- ✓ Technology Diversity: With microservices you can mix multiple languages, development frameworks and data-storage technologies.

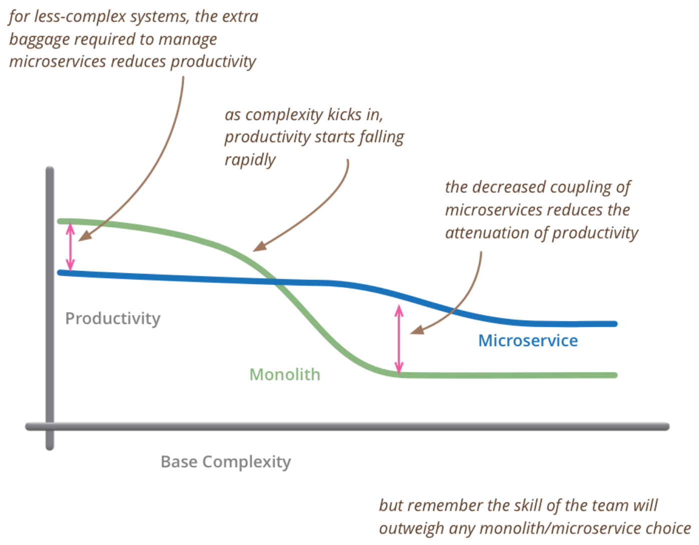
…but come with costs

- ✖ Distribution: Distributed systems are harder to program, since remote calls are slow and are always at risk of failure.
- ✖ Eventual Consistency: Maintaining strong consistency is extremely difficult for a distributed system, which means everyone has to manage eventual consistency.
- ✖ Operational Complexity: You need a mature operations team to manage lots of services, which are being redeployed regularly.

(from Microservice Trade-Offs)

Microservice Premium

Monolith First

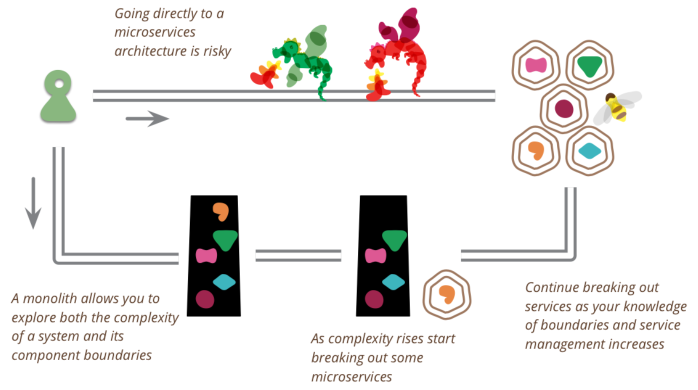


The microservices architectural style has been the hot topic over the last year. At the recent O'Reilly software architecture conference, it seemed like every session talked about microservices. Enough to get everyone's over-hyped-bullshit detector up and flashing. One of the consequences of this is that we've seen teams be too eager to embrace microservices, not realizing that microservices introduce complexity on their own account. This adds a premium to a project's cost and risk - one that often gets projects into serious trouble.

by Martin Fowler BLIKI 13 May 2015

Read more...

MICROSERVICES


Going directly to a microservices architecture is risky
A monolith allows you to explore both the complexity of a system and its component boundaries
As complexity rises start breaking out some microservices
Continue breaking out services as your knowledge of boundaries and service management increases

As I hear stories about teams using a microservices architecture, I've noticed a common pattern.

1. Almost all the successful microservice stories have started with a monolith that got too big and was broken up
2. Almost all the cases where I've heard of a system that was built as a microservice system from scratch, it has ended up in serious trouble.

This pattern has led many of my colleagues to argue that **you shouldn't start a new project with microservices, even if you're sure your application will be big enough to make it worthwhile.** .

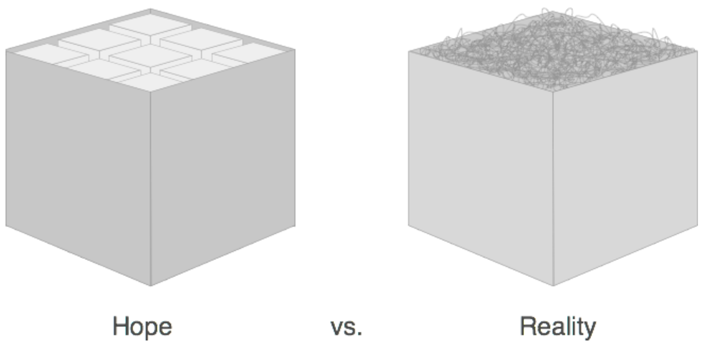
by Martin Fowler BLIKI 3 Jun 2015

Read more...

EVOLUTIONARY DESIGN

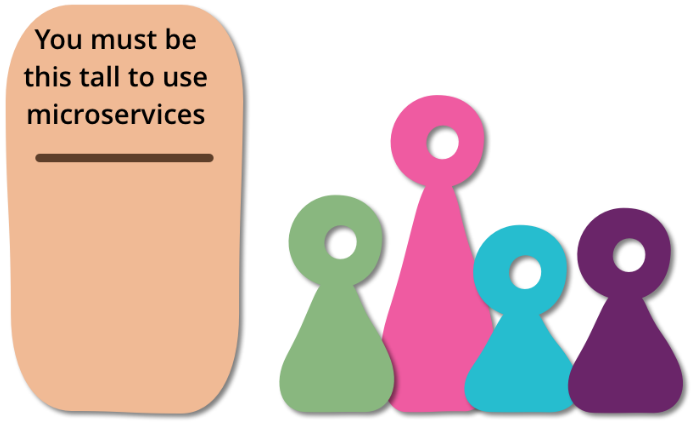
MICROSERVICES

Don’t start with a monolith



In the last few months, I’ve heard repeatedly that the only way to get to a successful microservices architecture is by starting with a monolith first. To paraphrase Simon Brown: If you can’t build a well-structured monolith, what makes you think you can build a well-structured set of microservices? The most recent – and, as usual, very convincing – rendering of this argument comes from Martin Fowler on this very site. As I had a chance to comment on an earlier draft, I had some time to think about this. And I did, especially because I usually find myself in agreement with him, and some others whose views I typically share seemed to agree with him, too

Microservice Prerequisites



As I talk to people about using a microservices architectural style I hear a lot of optimism. Developers enjoy working with smaller units and have expectations of better modularity than with monoliths. But as with any architectural decision there are trade-offs. In particular with microservices there are serious consequences for operations, who now have to handle an ecosystem of small services rather than a single, well-defined monolith. Consequently if you don't have certain baseline competencies, you shouldn't consider using the microservice style.

by Martin Fowler BLIKI

share seemed to agree with him, too.

I'm firmly convinced that starting with a monolith is usually exactly the wrong thing to do.

by Stefan Tilkov ARTICLE
9 Jun 2015

Read more…

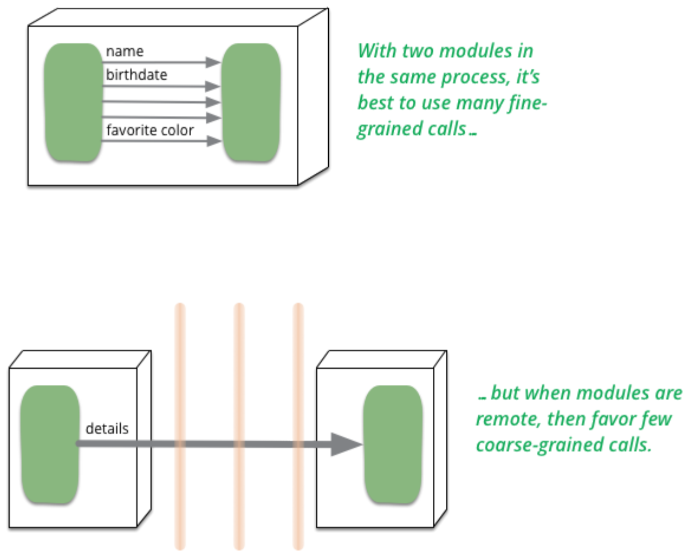
MICROSERVICES

by Martin Fowler BLIKI
28 Aug 2014

Read more…

MICROSERVICES

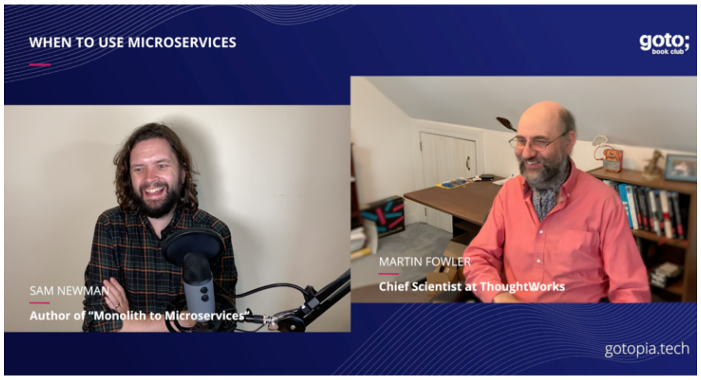
Microservices and the First Law of Distributed Objects


name
birthdate
favorite color
With two modules in the same process, it's best to use many fine-grained calls…
details
…but when modules are remote, then favor few coarse-grained calls.

In P of EAA I said "don't distribute your objects". Does this advice contradict my interest in Microservices?

by Martin Fowler ARTICLE

Interview with Sam Newman about Microservices


WHEN TO USE MICROSERVICES
goto; book club
SAM NEWMAN
Author of "Monolith to Microservices"
MARTIN FOWLER
Chief Scientist at ThoughtWorks
gotopia.tech

goto conferences asked me to do an interview with Sam Newman on his book: "Monoliths to Microservices". This turned into a general conversation about microservices and when to use them. Sam considers the three main reasons for them to be independent deployability, isolation of

13 Aug 2014

Read more…

API DESIGN MICROSERVICES

data, and reflecting the organizational structure. I'm more skeptical of the first, but consider data and people to be complicated parts of software development.

by Martin Fowler VIDEO

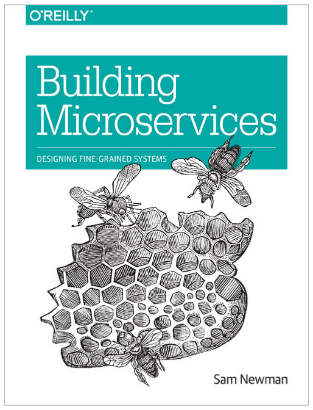
4 Sep 2020

More…

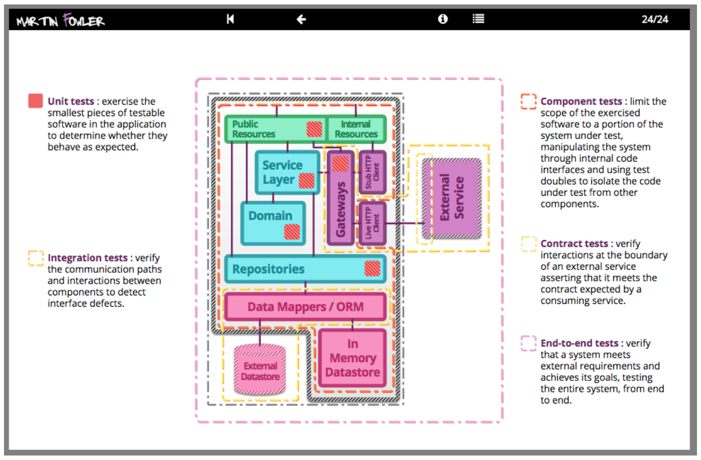
INTERVIEWS MICROSERVICES

Building Microservices

Microservice architectures are fairly new, but I've been fortunate that we've worked with them at Thoughtworks since their earliest appearances. For a cohesive description of how best to work with them, the best introduction is Sam Newman's book Building Microservices which he wrote based on our experiences and other published accounts.



Testing Strategies in a Microservice Architecture



🟥 **Unit tests** : exercise the smallest pieces of testable software in the application to determine whether they behave as expected.

Integration tests : verify the communication paths and interactions between components to detect interface defects.

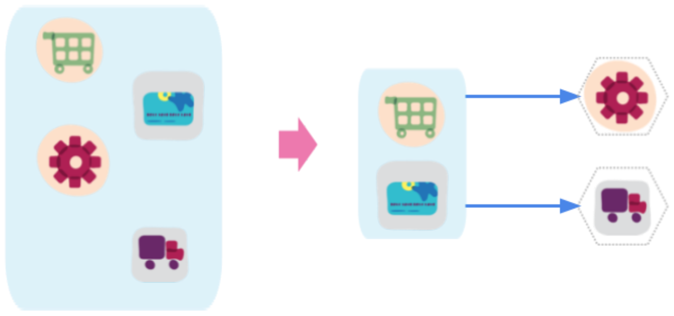
Component tests : limit the scope of the exercised software to a portion of the system under test, manipulating the system through internal code interfaces and using test doubles to isolate the code under test from other components.

Contract tests : verify interactions at the boundary of an external service asserting that it meets the contract expected by a consuming service.

End-to-end tests : verify that a system meets external requirements and achieves its goals, testing the entire system, from end to end.

There has been a shift in service based architectures over the last few years towards smaller, more focussed "micro" services. There are many benefits with this approach such as the ability to independently deploy, scale and maintain each component and parallelize development across multiple teams. However, once these additional network partitions have been introduced, the testing strategies

How to break a Monolith into Microservices



As monolithic systems become too large to deal with, many enterprises are drawn to breaking them down into the microservices architectural style. It is a worthwhile journey, but not an easy one. We've learned that to do this well, we need to start with a simple service, but then draw out services that are based on vertical capabilities that are important to the business and subject to frequent change. These services should be large at first and preferably not dependent upon the that applied for monolithic in process applications need to be reconsidered. Here, we plan to discuss a number of approaches for managing the additional testing complexity of multiple independently deployable components as well as how to have tests and the application remain correct despite having multiple teams each acting as guardians for different services.

by Toby Clemson INFODECK 18 Nov 2014

Read more…

POPULAR TESTING INFODECKS MICROSERVICES

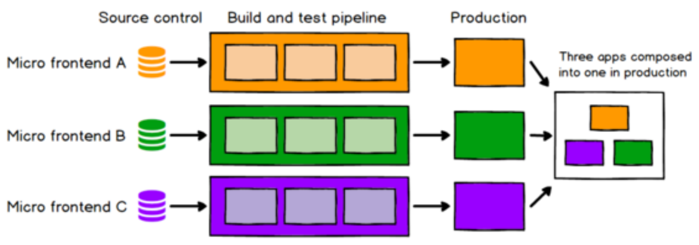
remaining monolith. We should ensure that each step of migration represents an atomic improvement to the overall architecture.

by Zhamak Dehghani ARTICLE 24 Apr 2018

Read more…

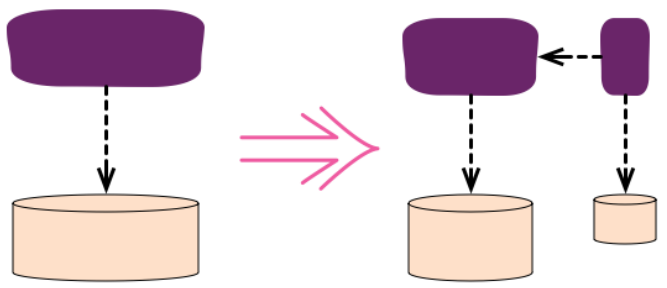
MICROSERVICES LEGACY REHAB

Micro Frontends



Good frontend development is hard. Scaling frontend development so that many teams can work simultaneously

How to extract a data-rich service from a monolith



When breaking monoliths into smaller

on a large and complex product is even harder. In this article we'll describe a recent trend of breaking up frontend monoliths into many smaller, more manageable pieces, and how this architecture can increase the effectiveness and efficiency of teams working on frontend code. As well as talking about the various benefits and costs, we'll cover some of the implementation options that are available, and we'll dive deep into a full example application that demonstrates the technique.

by Cam Jackson ARTICLE 19 Jun 2019

Read more…

APPLICATION ARCHITECTURE FRONT-END MICROSERVICES

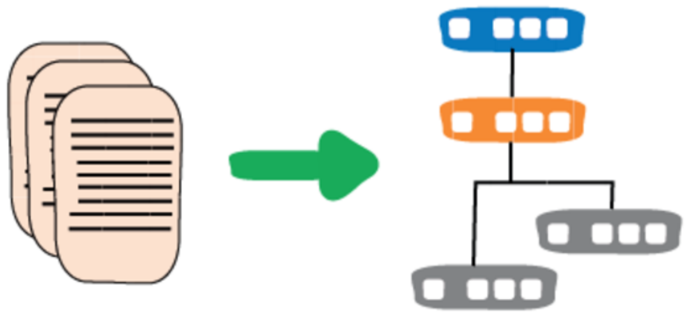
services, the hardest part is actually breaking up the data that lives in the database of the monolith. To extract a data-rich service, it is useful to follow a series of steps which retain a single write-copy of the data at all times. The steps begin by making a logical separation in the existing monolith: splitting service behavior into a separate module, then separating data into a separate table. These elements can be separately moved into a new autonomous service.

by Praful Todkar ARTICLE 30 Aug 2018

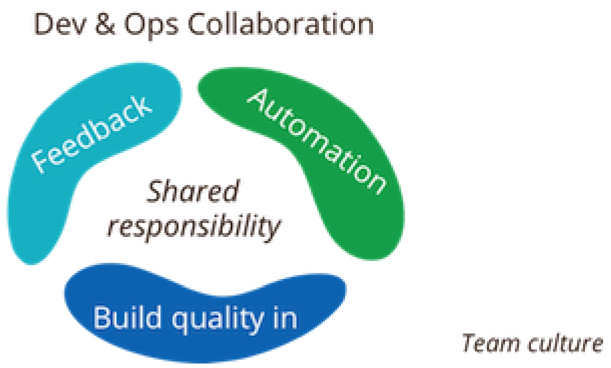
Read more…

MICROSERVICES LEGACY REHAB

Infrastructure As Code



Dev Ops Culture



Infrastructure as code is the approach to defining computing and network infrastructure through source code that can then be treated just like any software system. Such code can be kept in source control to allow auditability and ReproducibleBuilds, subject to testing practices, and the full discipline of ContinuousDelivery. It's an approach that's been used over the last decade to deal with growing CloudComputing platforms and will become the dominant way to handle computing infrastructure in the next.

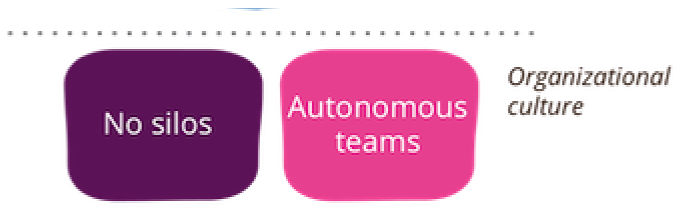
by Martin Fowler BLIKI

1 Mar 2016

Read more…

CONTINUOUS DELIVERY

MICROSERVICES



Agile software development has broken down some of the silos between requirements analysis, testing and development. Deployment, operations and maintenance are other activities which have suffered a similar separation from the rest of the software development process. The DevOps movement is aimed at removing these silos and encouraging collaboration between development and operations.

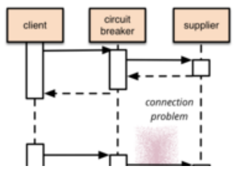
by Rouan Wilsenach BLIKI

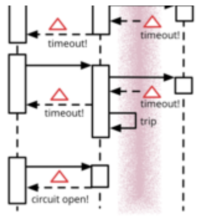
9 Jul 2015

Read more…

CONTINUOUS DELIVERY AGILE ADOPTION TEAM ORGANIZATION COLLABORATION

Circuit Breaker





It's common for software systems to make remote calls to software running in different processes, probably on different machines across a network. One of the big differences between in-memory calls and remote calls is that remote calls can fail, or hang without a response until some timeout limit is reached. What's worse if you have many callers on a unresponsive supplier, then you can run out of critical resources leading to cascading failures across multiple systems. In his excellent book Release It, Michael Nygard popularized the Circuit Breaker pattern to prevent this kind of catastrophic cascade.

The basic idea behind the circuit breaker is very simple. You wrap a protected function call in a circuit breaker object, which monitors for failures. Once the failures reach a certain threshold, the circuit breaker trips, and all further calls to the circuit breaker return with an error, without the protected call being made at all. Usually you'll also want some kind of trips.

by Martin Fowler BLIKI

6 Mar 2014

Read more…

CONTINUOUS DELIVERY
APPLICATION ARCHITECTURE