Cubes, Hexagons, Triangles, and More: Understanding the Microservice Architecture Through Shapes

Chris Richardson

Founder of Eventuate.io

Founder of the original CloudFoundry.com

Author of POJOs in Action and Microservices Patterns

@crichardson

chris@chrisrichardson.net

http://adopt.microservices.io

JFOKUS2020

Presentation goal

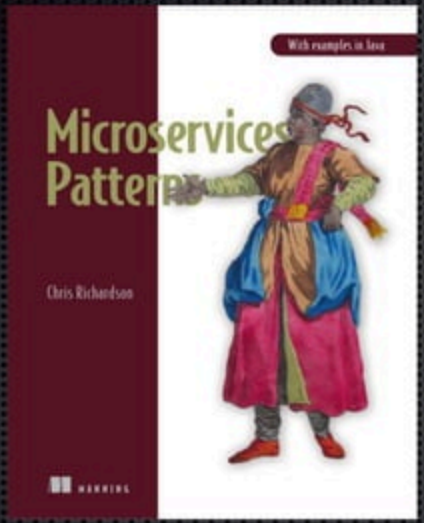
The essential characteristics of the Microservice Architecture

About Chris

eventuate

http://adopt.microservices.io

About Chris

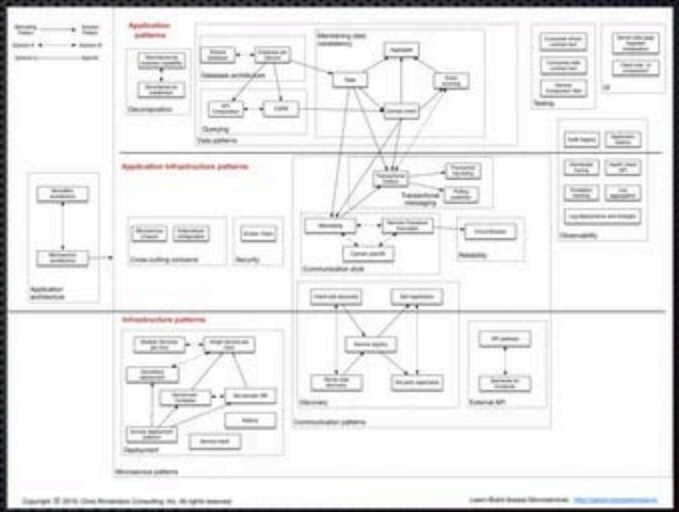


40% discount with code
ctwjfokus20

https://microservices.io/book

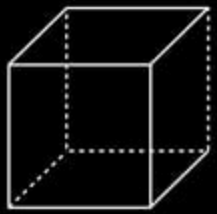
About Chris: microservices.io

- Microservices pattern language
- Articles
- Example code
- Microservices Assessment Platform

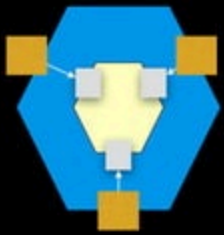


Agenda

Success Triangle



Scale Cube



Hexagonal Architecture

Iceberg services

Messaging

Testing Pyramid

+

Marketplace is volatile, uncertain, complex and ambiguous

+

Businesses must innovate faster

⇒

Deliver software rapidly, frequently and reliably

Quantifying rapid, frequent and reliable delivery

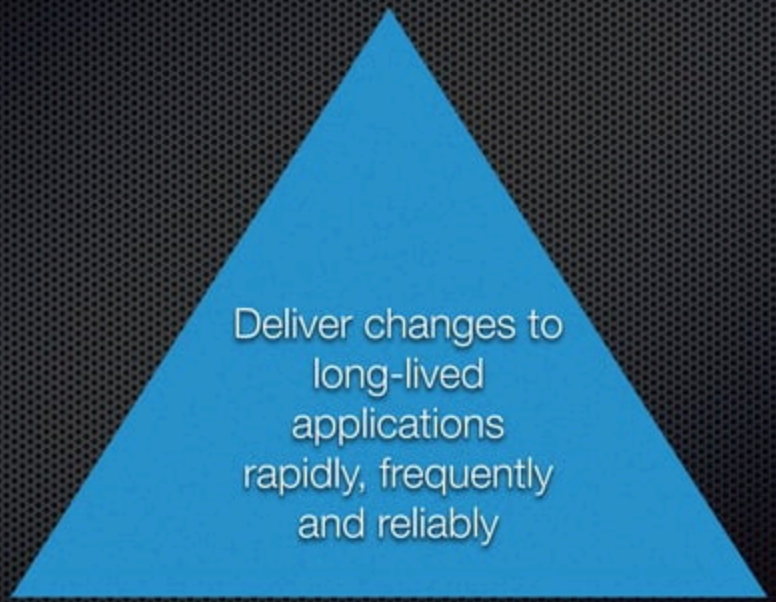
- Velocity
  - **Lead time** - time from commit to deploy
  - **Deployment frequency** - deploys per developer per day
- Reliability
  - **Change failure rate** - % of deployments that cause an outage
  - **Mean time to recover from a deployment failure**

Successful applications often live a very long time

Technology changes

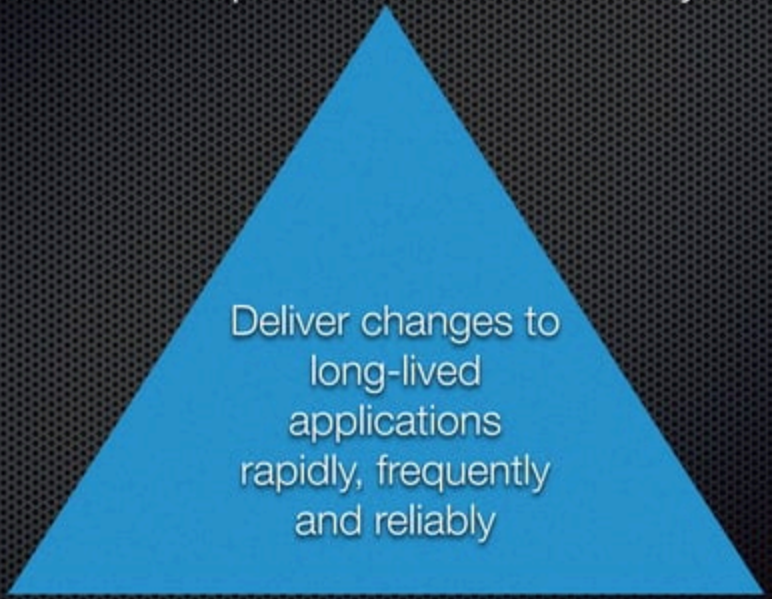
Need to be able to easily modernize applications

Success Triangle



Success Triangle

Process: Lean + DevOps/Continuous Delivery & Deployment

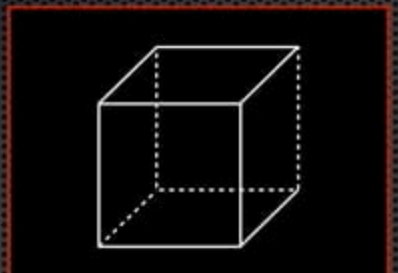


Organization:
Small, autonomous teams

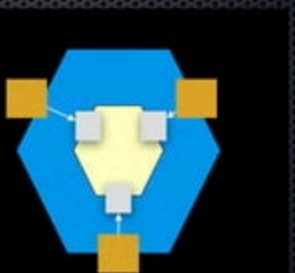
Architecture: ???

Agenda

Success Triangle



Scale Cube



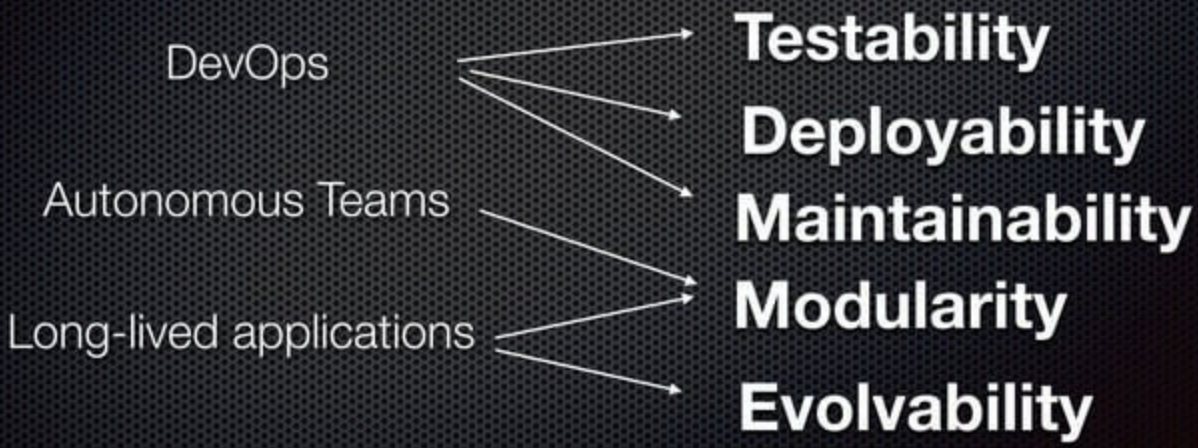
Hexagonal Architecture

Iceberg services

Messaging

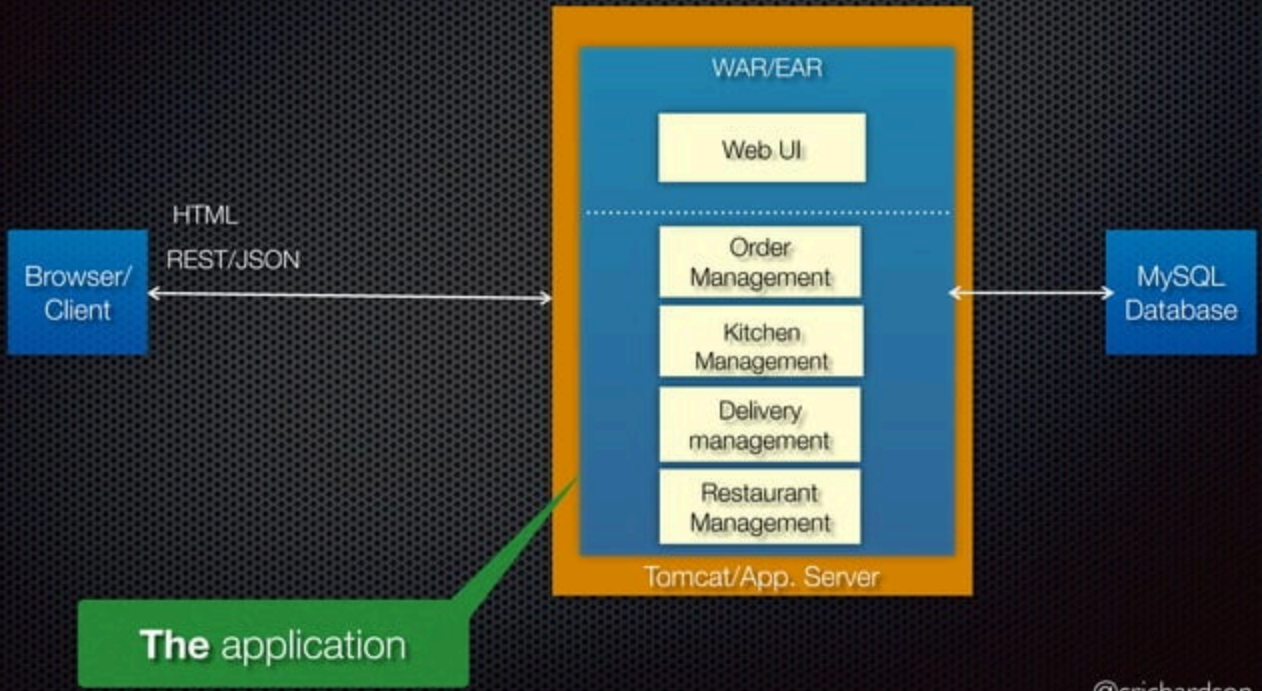
Testing Pyramid

Required architectural quality attributes (.a.k.a. -ilities)



https://en.wikipedia.org/wiki/Non-functional_requirement

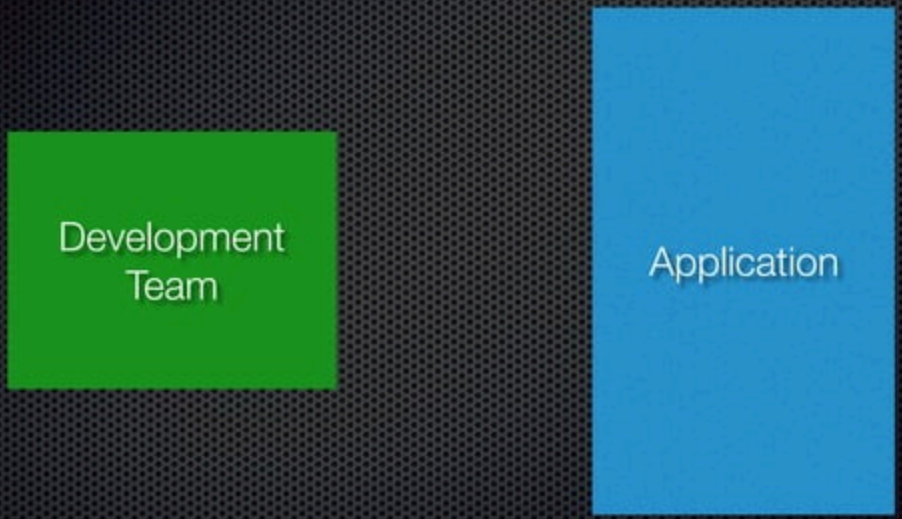
Food To Go: Monolithic architecture



-ilities of small monoliths

Testability ✅
Deployability ✅
Maintainability ✅
Modularity ✅
Evolvability ✅

But successful applications keep growing….



… and growing

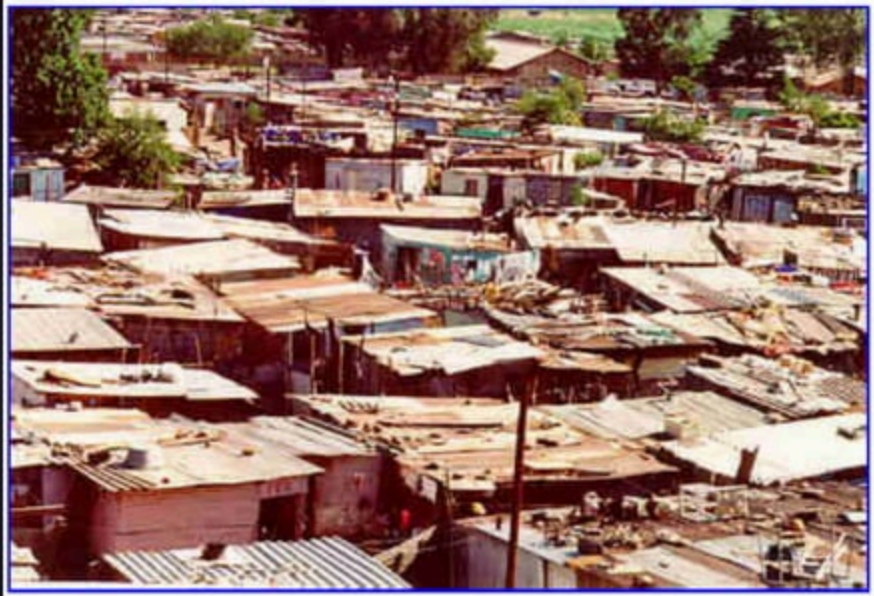

Development Team A
Development Team B
Development Team C
Application

… and modularity breaks down…

BIG BALL OF MUD

alias

SHANTYTOWN
SPAGHETTI CODE

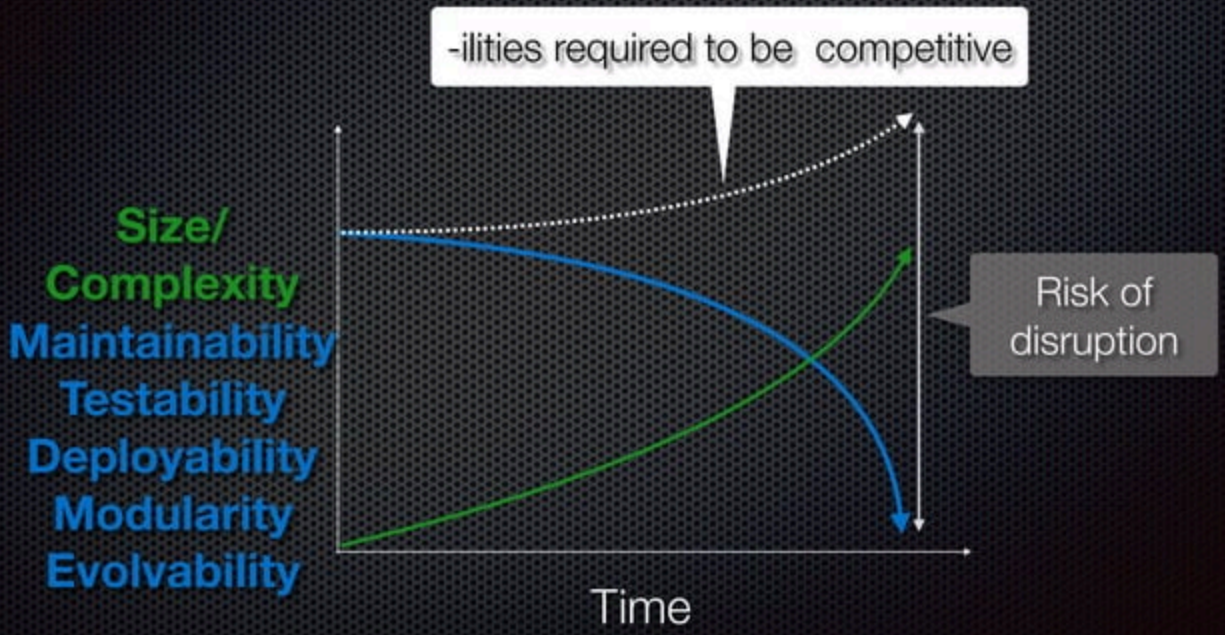


"haphazardly structured, sprawling, sloppy, duct-tape and bailing wire, spaghetti code jungle."

http://www.laputan.org/mud/
@crichardson

Technology stack becomes
increasingly obsolete
BUT a rewrite is not feasible

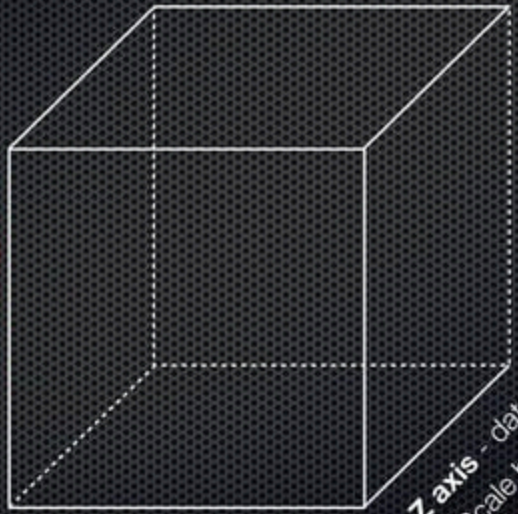
Rapid, frequent and reliable delivery **eventually** becomes impossible



The scale cube

Y axis - functional decomposition

Scale by splitting different things



Z axis - data partitioning
Scale by splitting similar things

X axis
- horizontal duplication

The microservice architecture is
an architectural style
that structures an application as a
set of services

Each ~~micro~~**service** is:

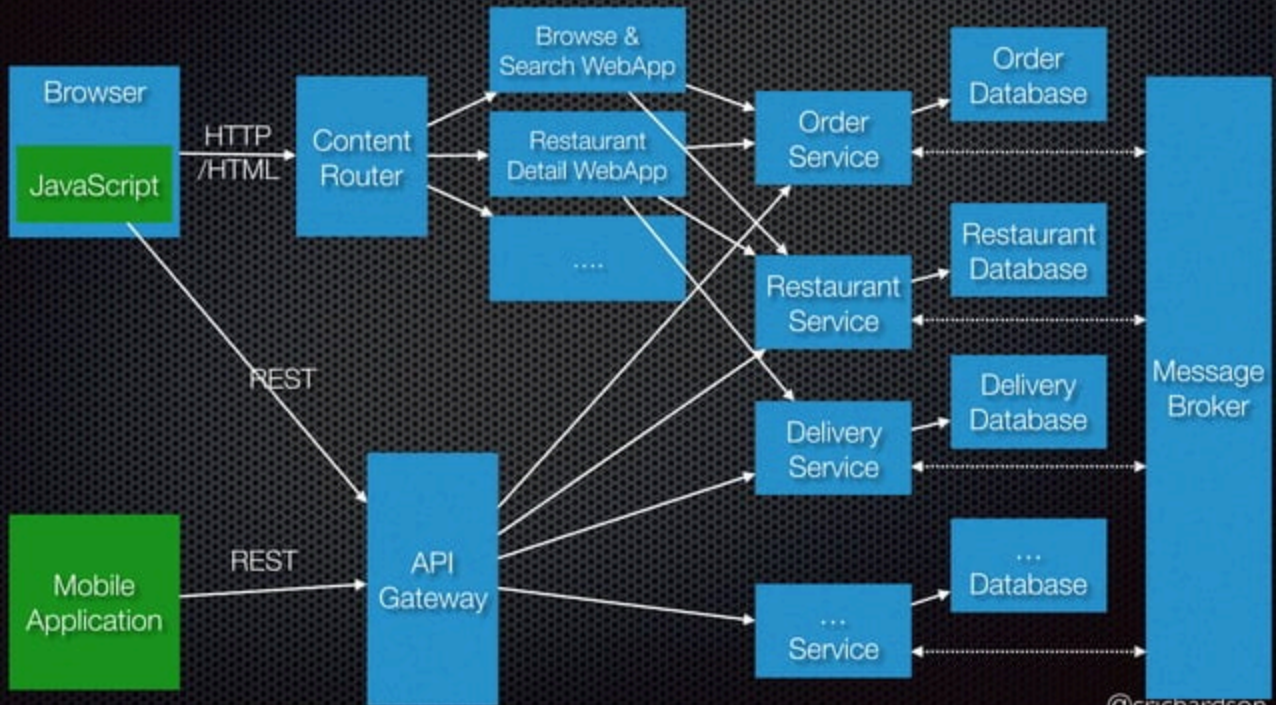
- highly maintainable and testable
- loosely coupled
- independently deployable
- organized around business capabilities
- owned by a small team

Start with one service per team

Small, autonomous team

Split service only to solve a problem
e.g. accelerate development and testing

Food to Go: Microservice architecture

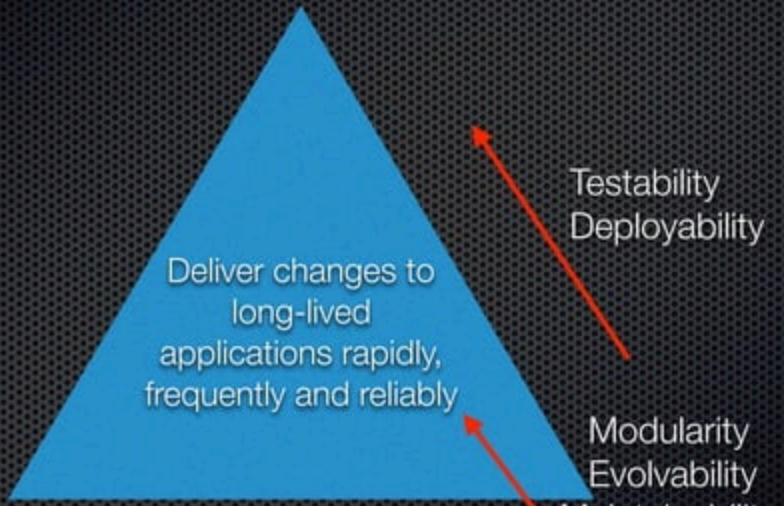


Benefits of microservices

- **Maintainability** - small service ⇒ easier to understand and change
- **Modularity** - a service API is impermeable ⇒ enforces modularity
- **Evolvability** - evolve each service's technology stack independently
- **Testability** - small service ⇒ easier/faster to test
- **Deployability** - each service is independently deployable

*Improved scalability and fault tolerance too

Process: Lean + DevOps/Continuous Delivery & Deployment



Organization: Small, autonomous, **product** teams

Modularity

Architecture: microservices

Drawbacks of microservices…

Complexity

Development: IPC, partial failure, distributed data

Testing: Integration, end to end, …

Deployment

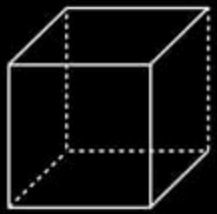
…

…Drawbacks of microservices

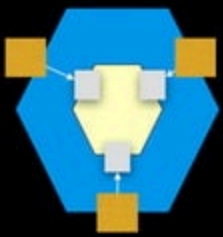
- Correctly identifying service boundaries and avoiding the distributed monolith anti-pattern
- Refactoring a monolithic application to a microservice architecture

Agenda

Success Triangle



Scale Cube



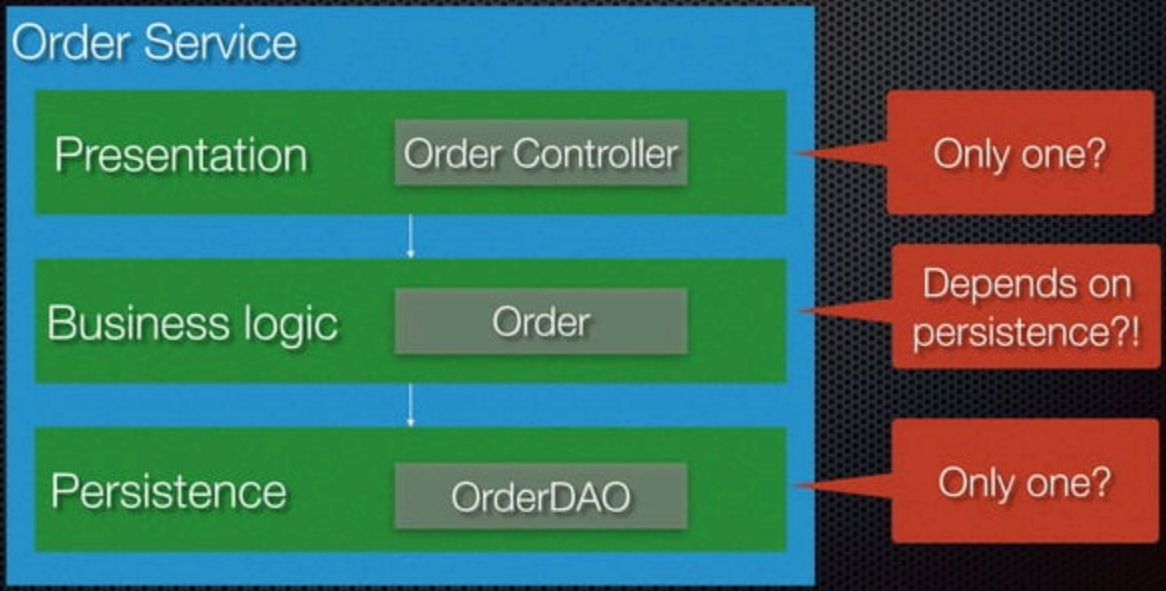
Hexagonal Architecture

Iceberg services

Messaging

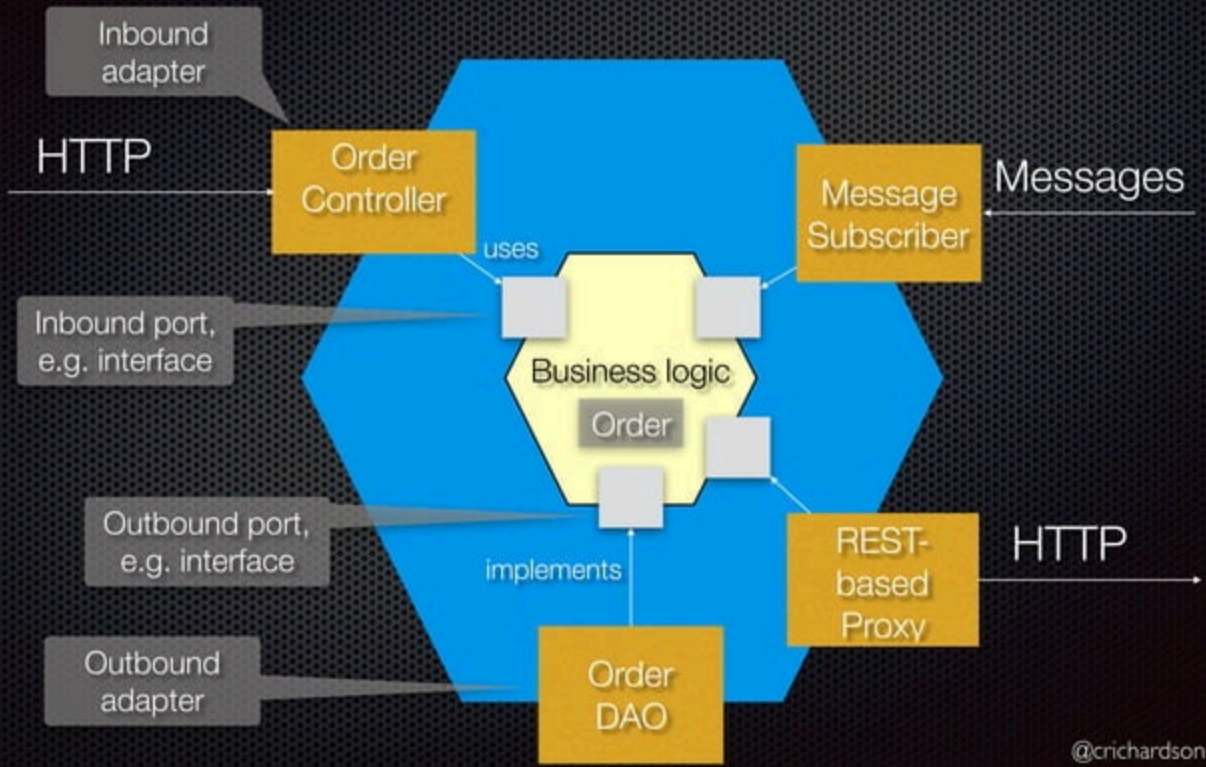
Testing Pyramid

The traditional 3-tier/layered architecture

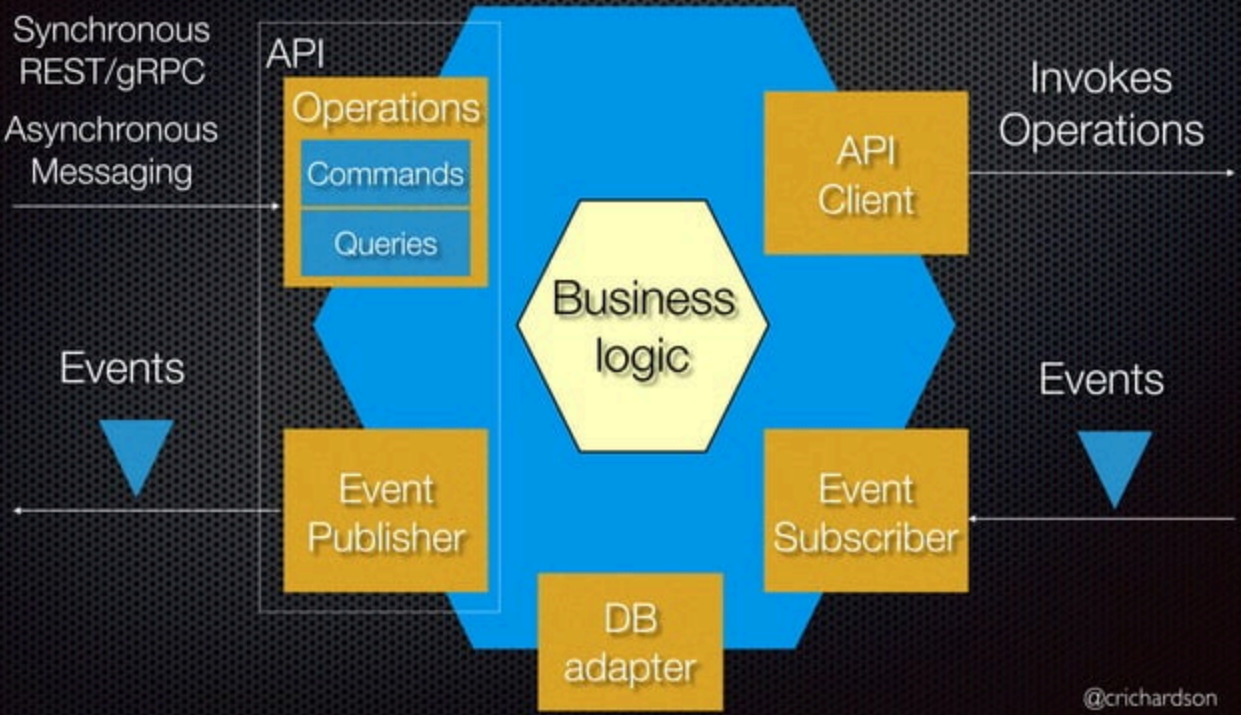


Doesn't reflect reality!

The Hexagonal architecture, a.k.a. ports and adapters

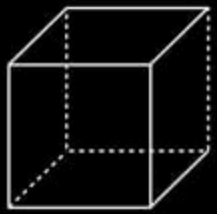


The structure of a service…

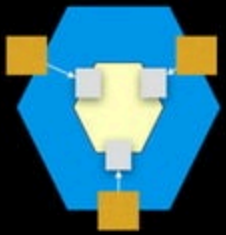


Agenda

Success Triangle



Scale Cube



Hexagonal Architecture

Iceberg services

Messaging

Testing Pyramid

Loose coupling is essential

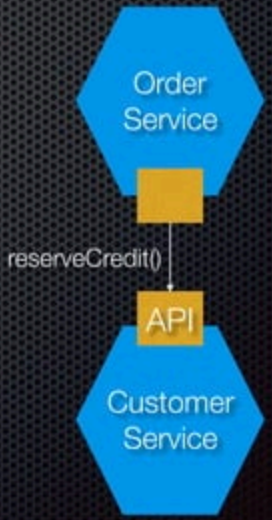
Services collaborate, e.g. Order Service must reserve customer credit

⇒

Coupling is inevitable

BUT

Services must be loosely coupled



Runtime coupling

- Order Service cannot respond to a synchronous request (e.g. HTTP POST) until Customer Service responds

vs

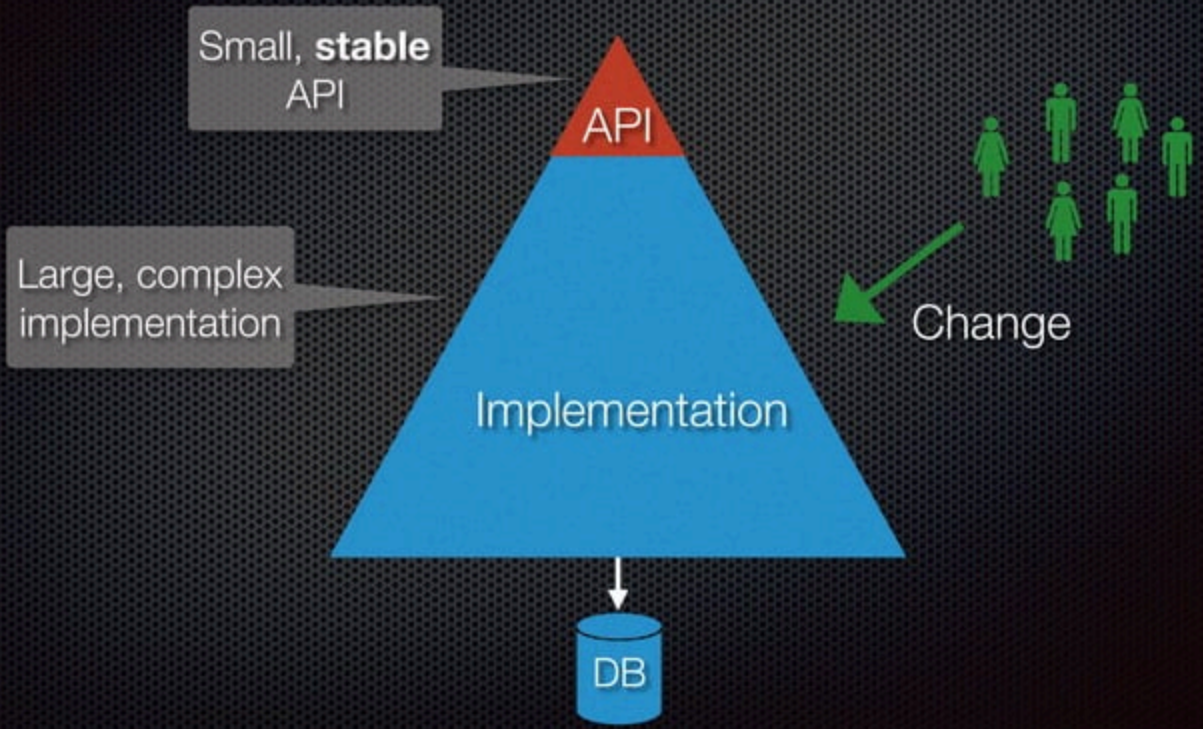
Design time coupling

- Change Customer Service ⇒ change Order Service

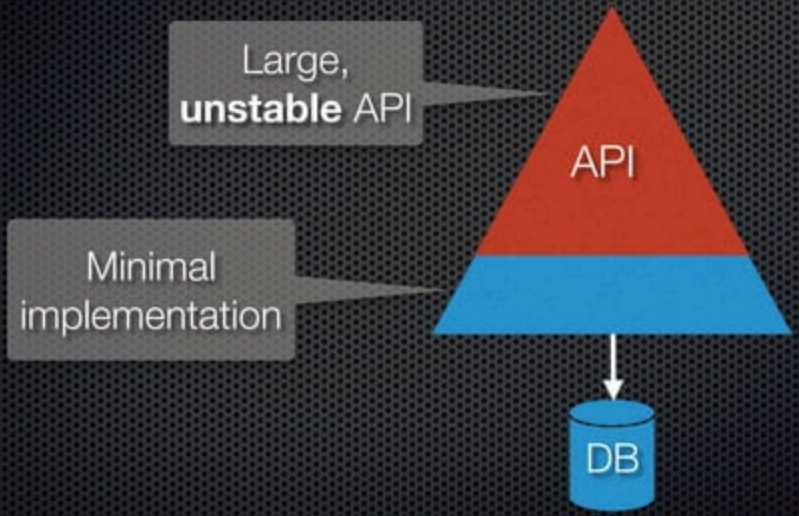
Loose coupling - design time

- Design-time coupling requires coordination between teams:
  - e.g. Meetings to discuss API changes
  - Slows down development
- Essential to minimize design time coupling:
  - Use well-designed, stable APIs
  - Be careful with shared libraries - best for utilities

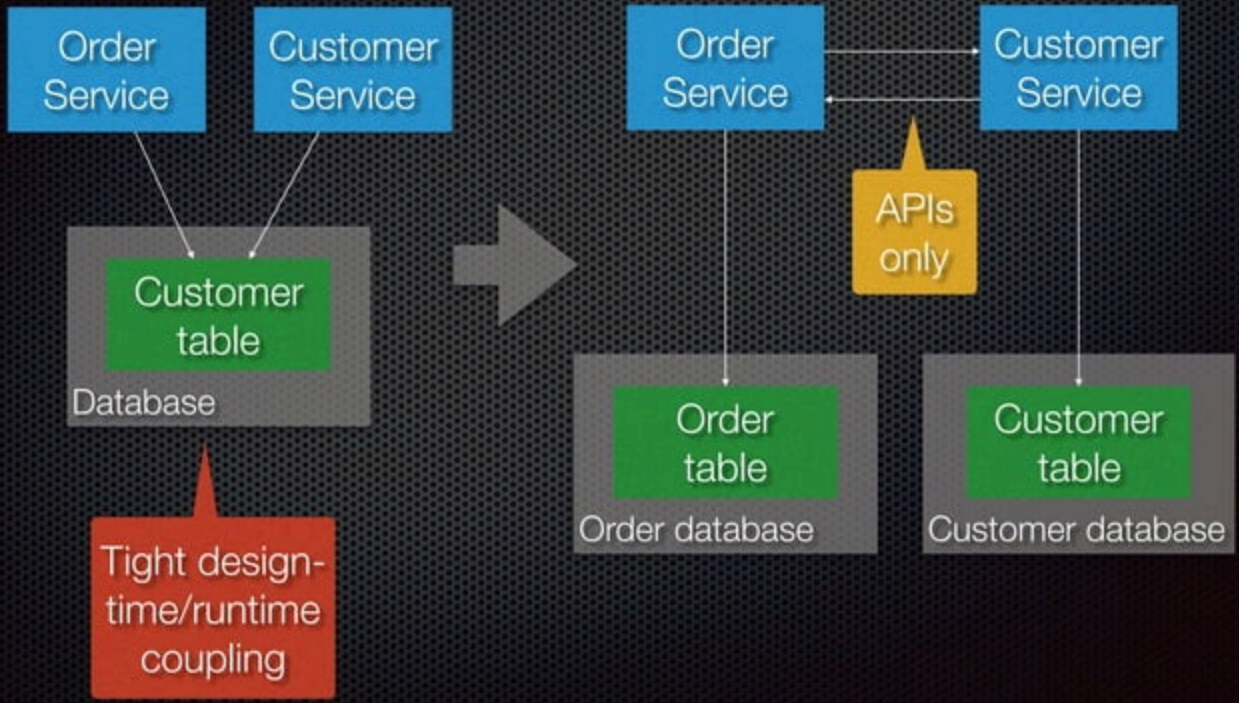
Design iceberg services



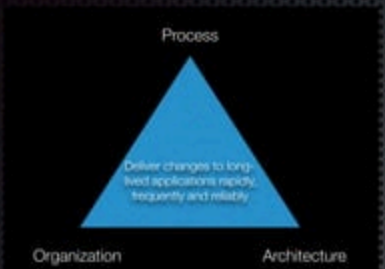
Avoid CRUD service anti-pattern: database wrapper



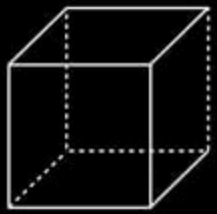
Avoid shared database tables



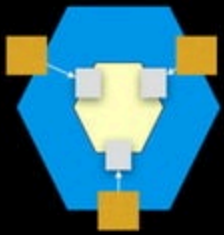
Agenda



Success Triangle



Scale Cube



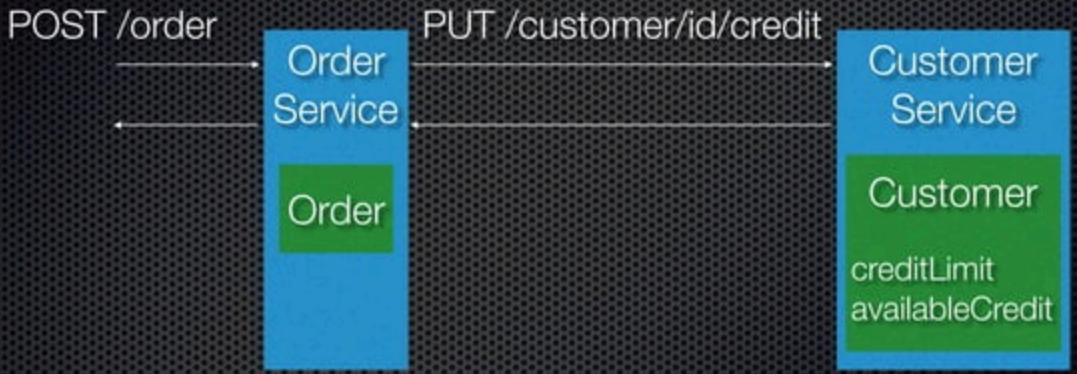
Hexagonal Architecture

Iceberg services

Messaging

Testing Pyramid

The trouble with synchronous IPC : runtime coupling => reduced availability



availability(createOrder) =
availability(OrderService) x
availability(CustomerService) 😢

Anti-pattern: Distribution is free

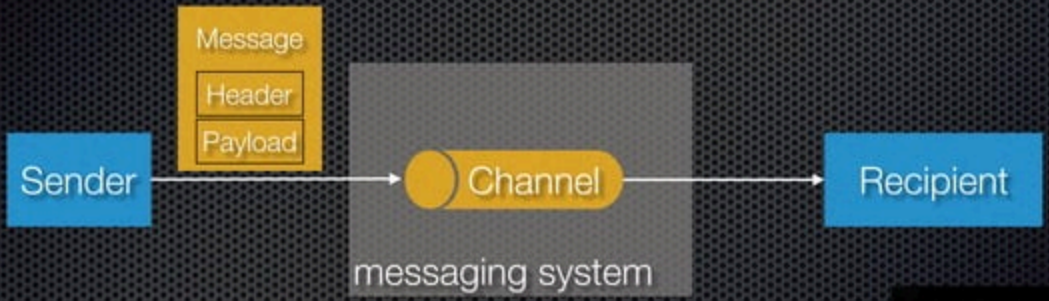
- Problem:
  - Developers treat services as if they are programming language-level modules (that communicate via HTTP)
- Consequences:
  - IPC is relatively expensive ⇒ high latency
  - Synchronous communication ⇒ temporal coupling ⇒ reduced availability - $serviceAvailability^{number\ of\ services}$

Self-contained service:

Can handle a synchronous request without waiting for a response from another service

https://microservices.io/patterns/decomposition/self-contained-service.html

Use asynchronous messaging

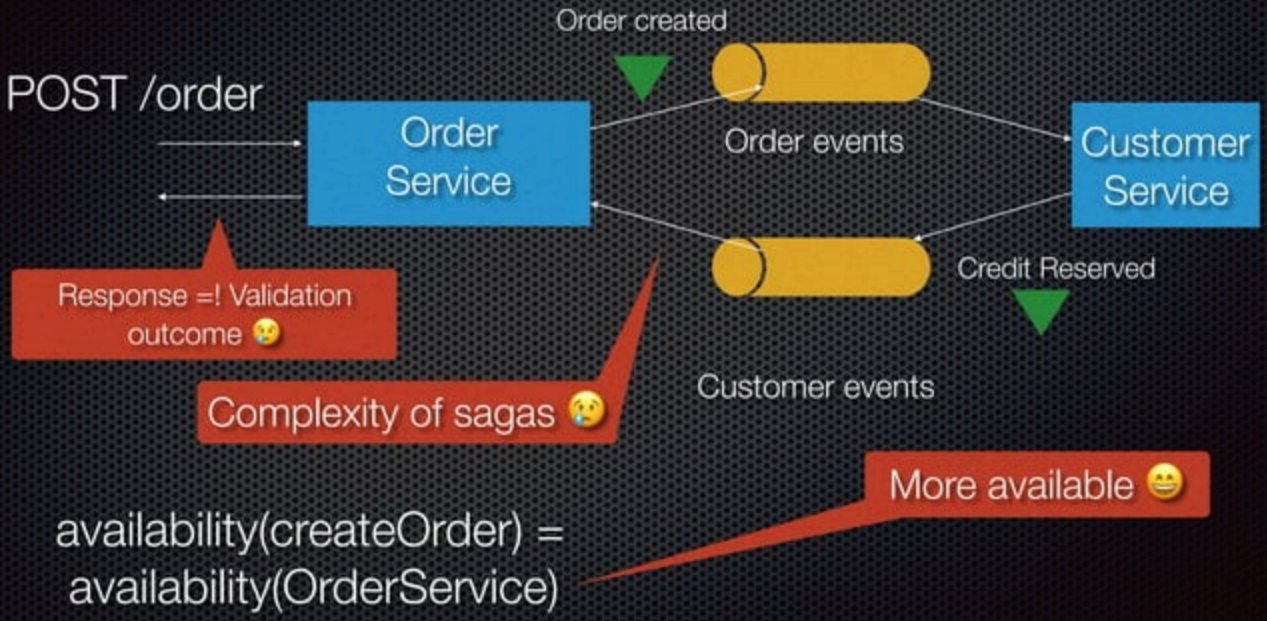


http://bit.ly/books-eip

About message channels

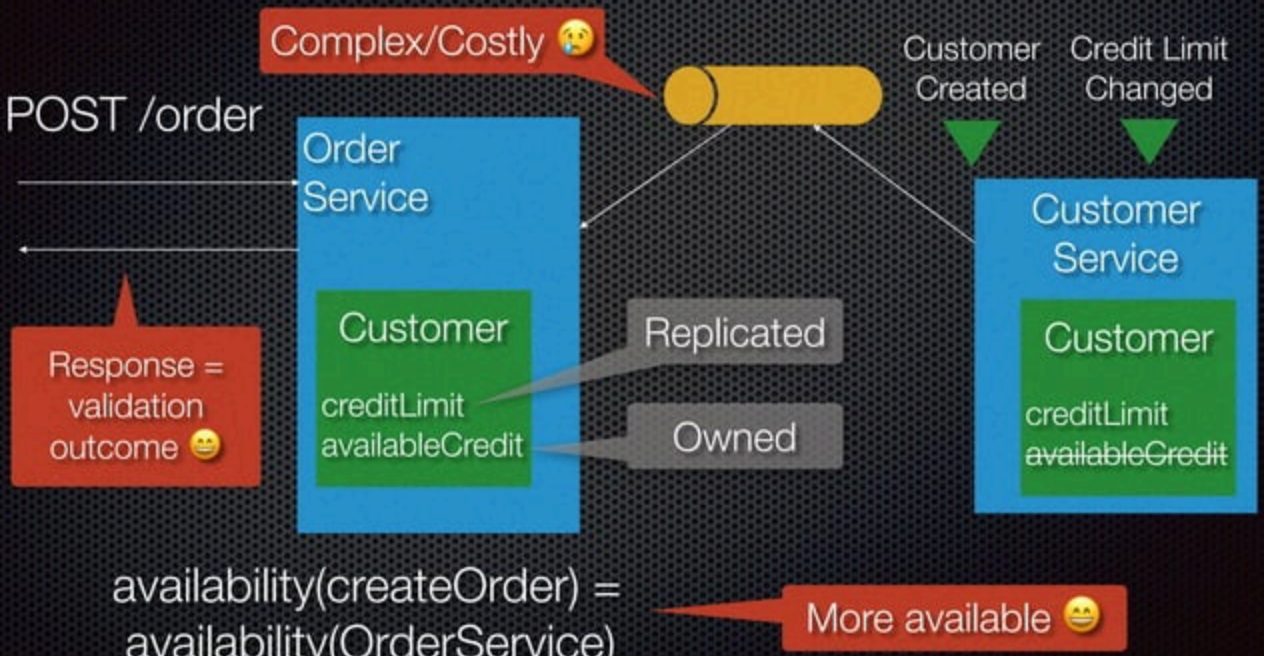
- Abstraction of message broker capabilities, e.g.
  - Apache Kafka topics
  - JMS queues and topics
  - ....
- Channel types:
  - Point-to-point - deliver to one recipient
  - Publish-subscribe - deliver to all recipients

Improving availability: sagas



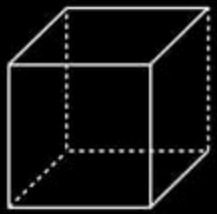
https://microservices.io/patterns/data/saga.html

Improving availability: CQRS

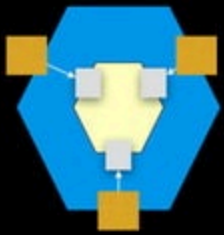


Agenda

Success Triangle



Scale Cube



Hexagonal Architecture

Iceberg services

Messaging

Testing Pyramid

Microservices enable DevOps

DevOps requires automated testing

Complexity of microservices requires good automated testing

⇒

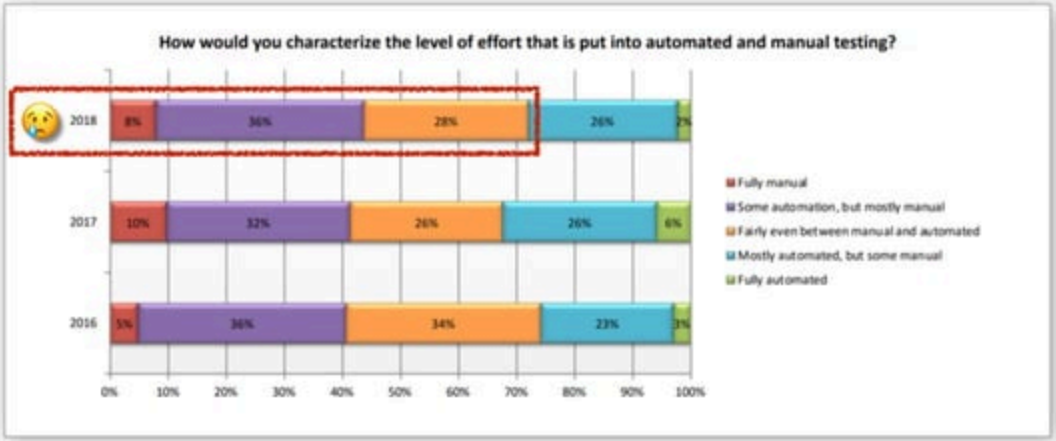
Using the Microservice Architecture without automated testing is self-defeating **AND** risky

http://bit.ly/msa-antipattern-flying-before-walking

The state of automated testing today: Oops!

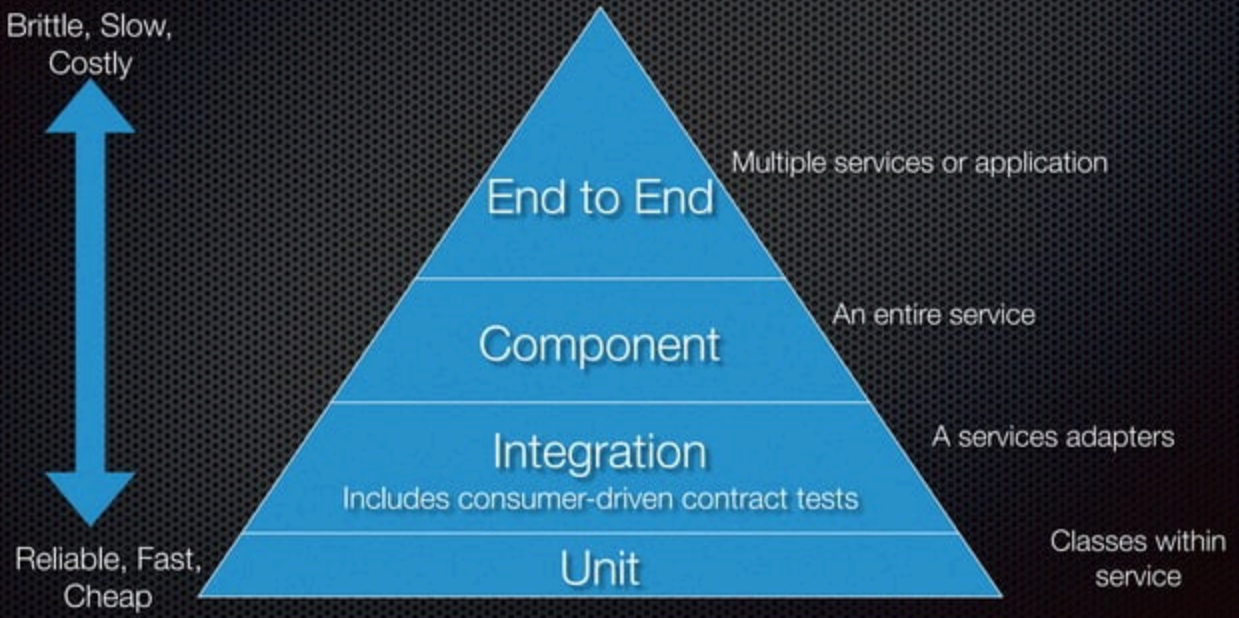
Automated testing is still not the default

Even though management is typically on board with automated testing, there is still room for improvement. Among participants, 56% do as much or more automated as manual testing, which is actually a minor decrease from 58% in 2017 and 60% in 2016. This is clearly a place where development teams can improve.

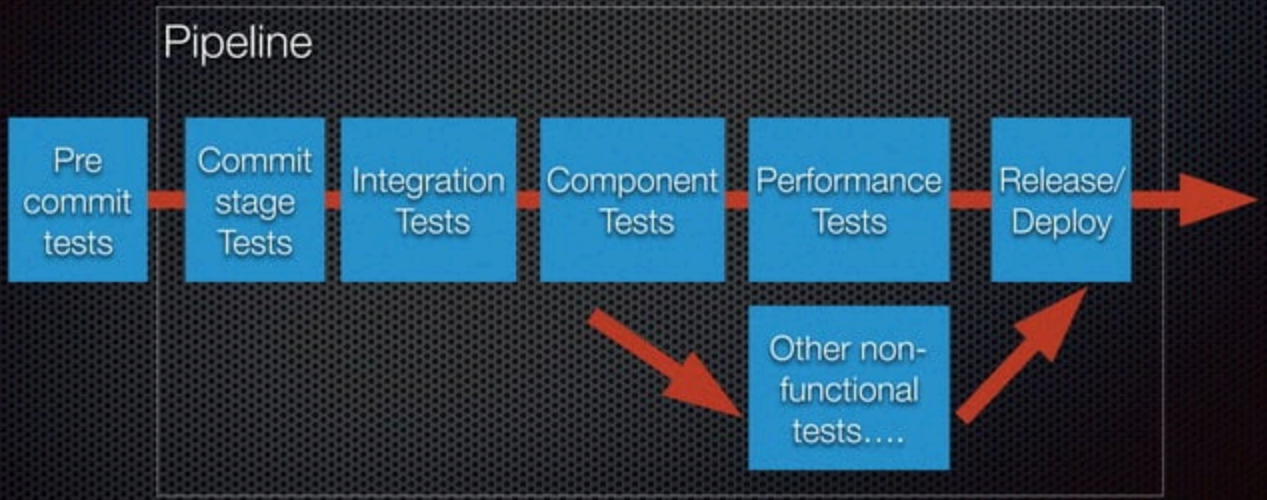


https://cdn.agilitycms.com/sauce-labs/white-papers/sauce-labs-state-of-testing-2018.pdf

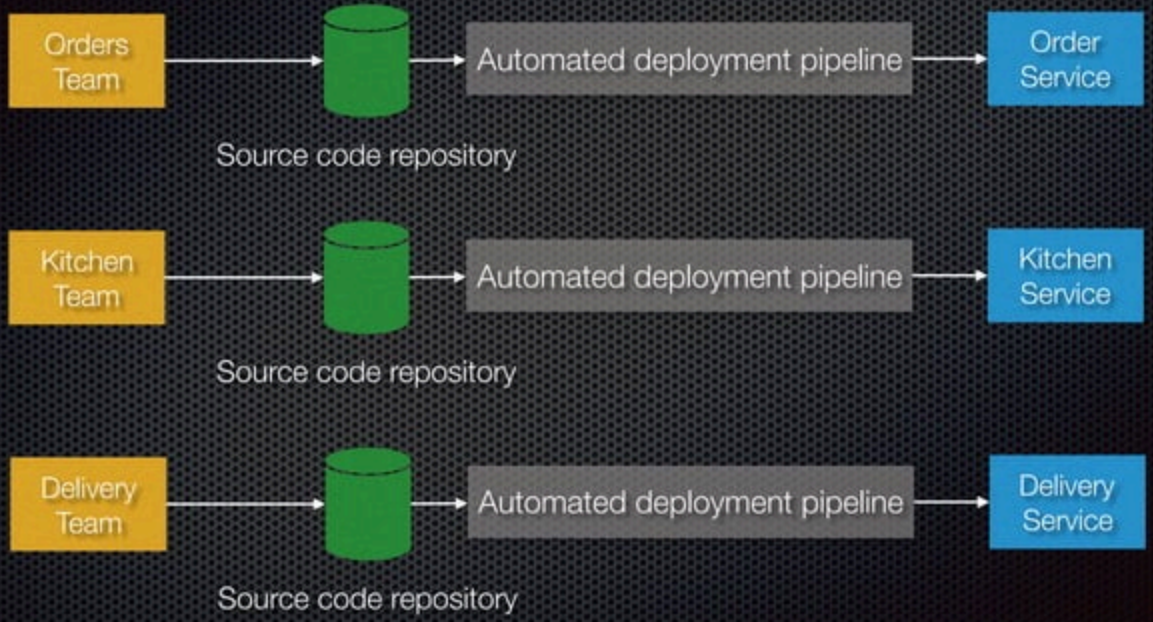
The testing pyramid and microservices



Deployment pipeline: the path from laptop to production



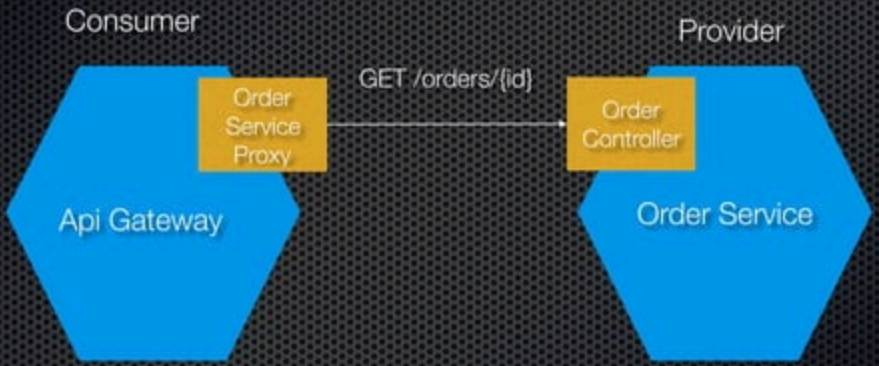
Deployment pipeline per service



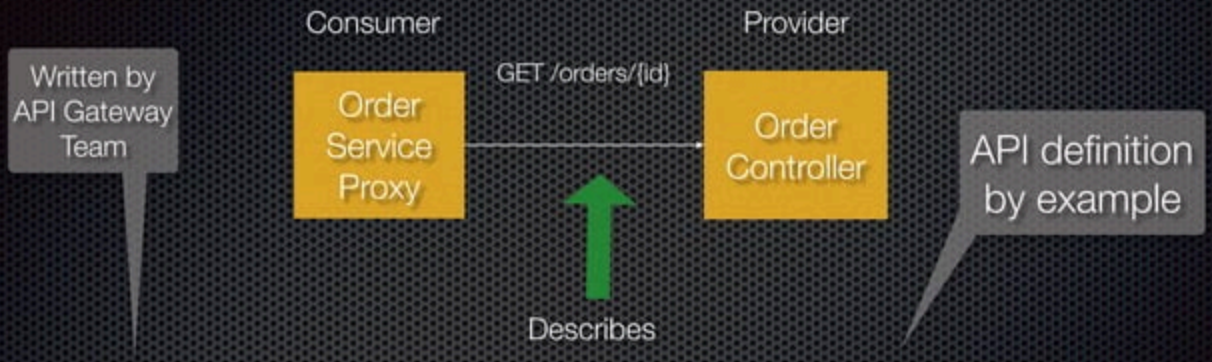
Consumer-driven contract testing

Verify that
a service (a.k.a. provider)
and its clients (a.k.a. consumers)
can communicate while **testing them in isolation**

Contract testing example



Contract testing example

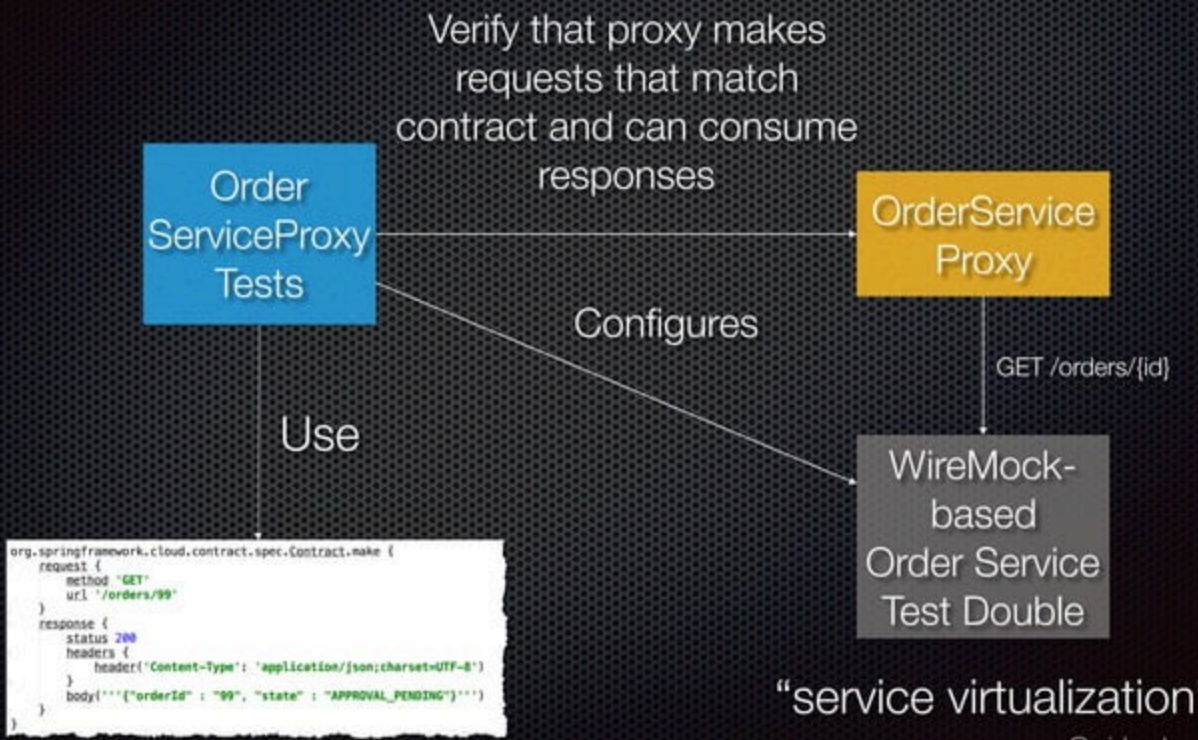



```
org.springframework.cloud.contract.spec.Contract.make {
    request {
        method 'GET'
        url '/orders/99'
    }
    response {
        status 200
        headers {
            header('Content-Type': 'application/json;charset=UTF-8')
        }
        body('''{"orderId" : "99", "state" : "APPROVAL_PENDING"}''')
    }
}
```

Request from consumer

Response from service

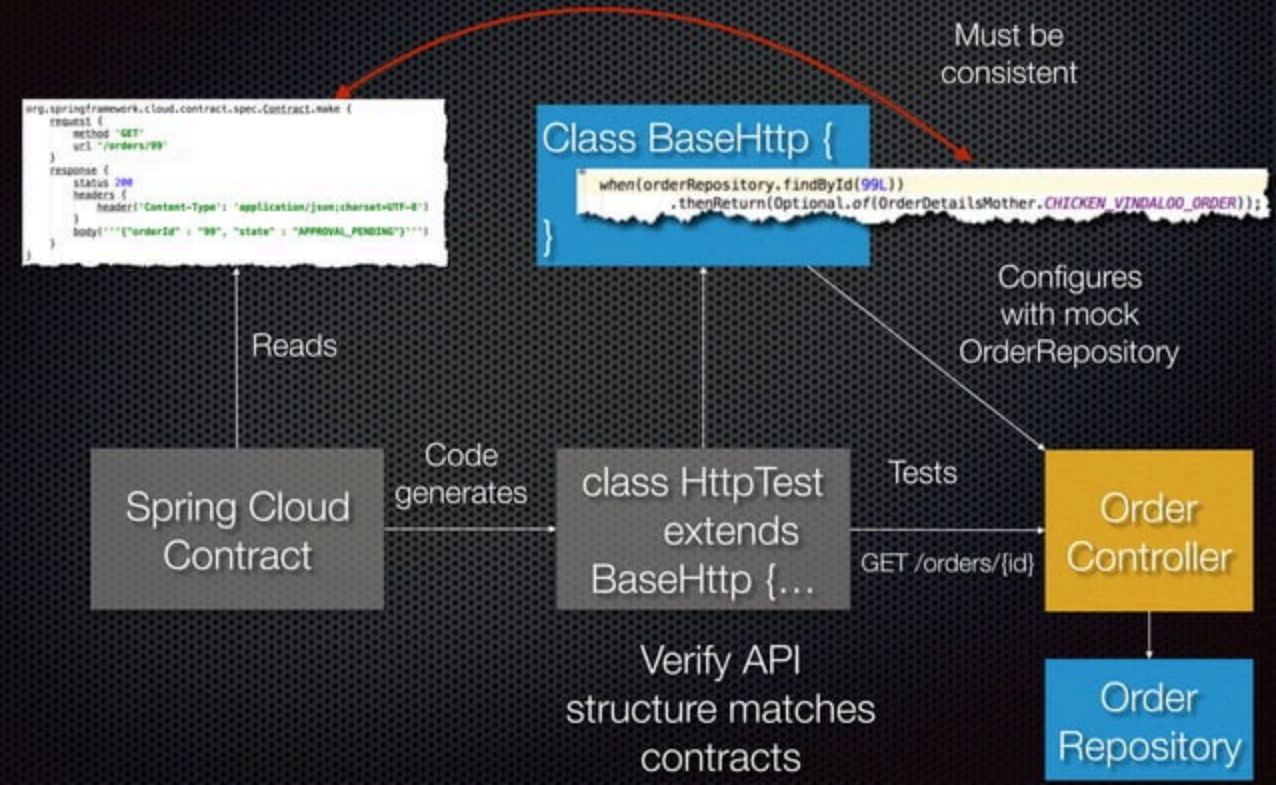
Consumer-side contract test



"service virtualization"

@crichardson

Provider-side contract test



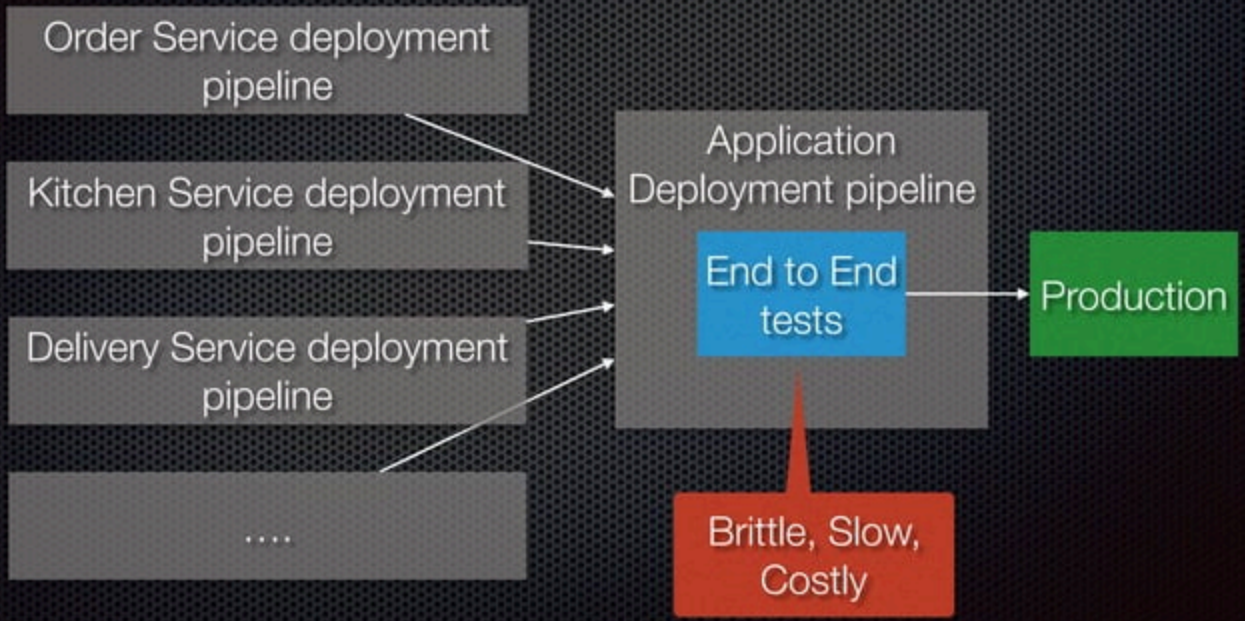
Assumption:

If consumer and provider are tested independently with the same contracts

⇒

Consumer and provider can communicate

End to end tests: a bottleneck and best avoided



https://blogs.msdn.microsoft.com/seliot/2011/04/25/i-dont-always-test-my-code-but-when-i-do-i-do-it-in-production/

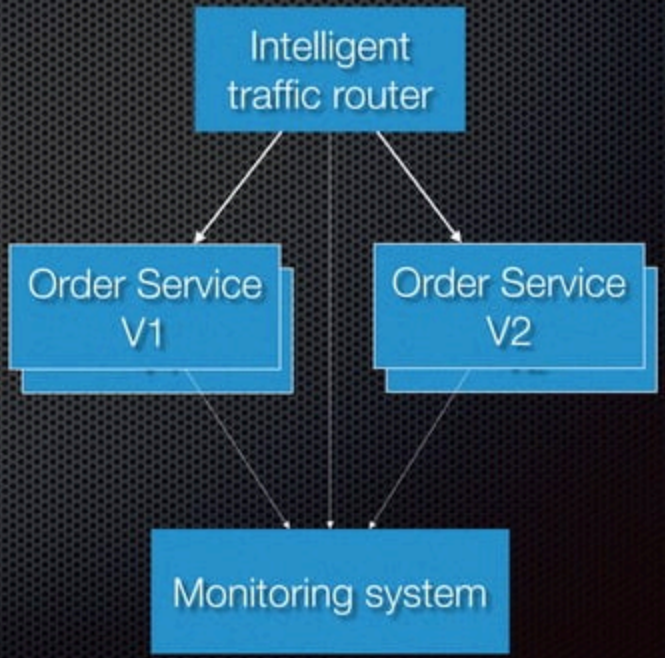
Testing in production*

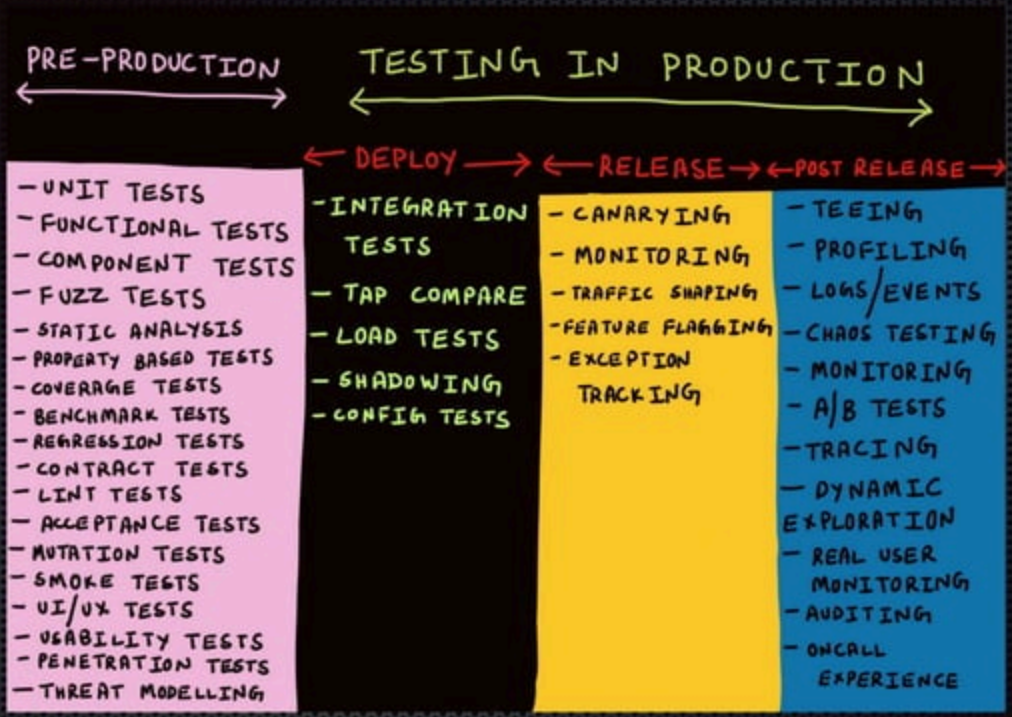
- Challenge:
  - End-to-end testing is brittle, slow, and costly
  - Your end-to-end test environment is a simulation of production
  - No matter how much you test issues will appear in production
- Therefore:
  - Separate deployment (running in production) from release (available to users)
  - Test deployed code before **releasing**
  - Automate for fast deployment, rollback and roll forward

* a.k.a. Validation in production

Test deployed code before releasing: e.g. Canary release

1. Deploy V2 alongside V1
2. Test V2
3. Route test traffic to V2
4. Release V2 to a small % of production users
5. Monitor/test (latency, errors) - undo rollout if errors
6. Increase % of production traffic going to V2
7. Repeat until 100% of traffic going to V2
8. Eventually undeploy V1



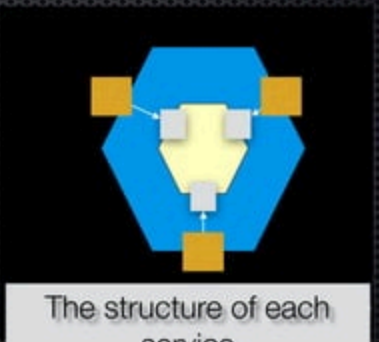


https://medium.com/@copyconstruct/testing-in-production-the-safe-way-18ca102d0ef1

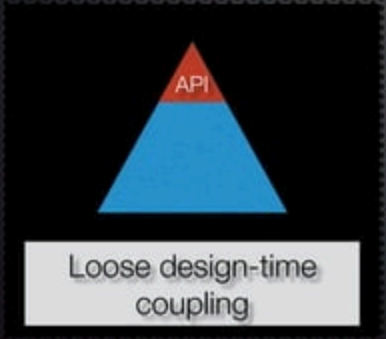
Summary

The goal

Microservice architecture has required -ilities

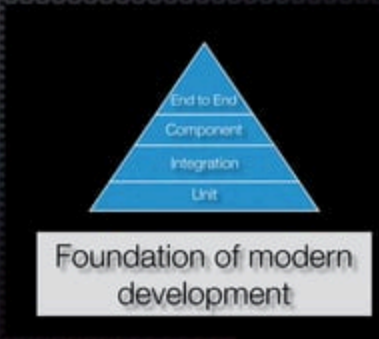


The structure of each service



Loose design-time coupling

Loose run-time coupling



Foundation of modern development

@crichardson chris@chrisrichardson.net

Questions?

40% discount with code ctwjfokus20

http://bit.ly/msa-geometry-jfokus-2020

http://adopt.microservices.io