Building Microservice Architectures

@neal4d
nealford.com
what problem

characteristics

engineering

AGENDA
Service-oriented Architecture

abstract enterprise-level coarse-grained services owned and defined by business users

no implementation - only name, input, and output data represented as wsdl, bpel, xml, etc.

ExecuteTrade  PlaceOrder  ProcessClaim
Service-oriented Architecture

Concrete enterprise-level coarse-grained services owned by shared services teams

custom or vendor implementations that are one-to-one or one-to-many relationship with business services

Enterprise Services

CreateCustomer  CalcQuote  ValidateTrade
Service-oriented Architecture

congrete application-level fine-grained services owned by application teams bound to a specific application context

AddDriver  UpdateAddress  CalcSalesTax

application services  AS
Service-oriented Architecture

Concrete enterprise-level fine-grained services owned by infrastructure or shared services teams

Implements non-business functionality to support both enterprise and business services
Service-oriented Architecture

message bus

process choreographer

service orchestrator

mediation and routing
process choreography
service orchestration

message enhancement
message transformation
protocol transformation
Service-oriented Architecture

- **business services**
- **message bus**
  - process choreographer
  - service orchestrator
- **enterprise services**
- **application services**
  - infrastructure services

- **maximize reuse**
- **maximize canonicality**
Service-oriented Architecture

- Business services
- Message bus
- Process choreographer
- Service orchestrator
- Enterprise services
- Application services
- Infrastructure services

- Incremental change
- Operationally complex
Yesterday’s best practice is tomorrow’s anti-pattern.

We inadvertently build architectures to solve outdated problems.
Architecture is abstract until operationalized.

nealford.com/memeagora/2015/03/30/architecture_is_abstract_until_operationalized.html
Architecture is abstract until operationalized.

nealford.com/memeagora/2015/03/30/architecture_is_abstract_until_operationalized.html
what problem

characteristics

ingineering

AGENDA
Domain Driven Design
Bounded Context

Maintaining Model Integrity

- Keep model unified by Continuous Integration
- Bounded Context
- Names enter
- Overlap allied contexts through
- Relate allied contexts as
- Overlap unilaterally as
- Support multiple clients through
- Free teams to go
- Translate and insulate unilaterally with
- Big Ball of Mud
- Anticorruption layer
- Separate ways

Ubiquitous Language

Context Map

Continuous Delivery
Reliable Software Releases through Build, Test, and Deployment Automation

Jez Humble, David Farley
Microservices Architecture

distributed architecture

client requests

api layer

client requests

client requests
Microservices Architecture

separately deployed components
Microservices Architecture

Service component

Client requests

API layer

Service component

Module

Module

Module

Module

Module
Microservices Architecture

bounded context

client requests

api layer

service component
module
module

service component
module
module

service component
module
module

service component
module
module

service component
module
module

service component
module
module

service component
module
module

service component
module
module

service component
module
module

service component
module
module
Microservices Architecture

service orchestration
Monoliths vs. Microservices
Products, not Projects

projects:

products: .amazon.com’s  “You build it, you run it”
Conway’s Law

“organizations which design systems ... are constrained to produce designs which are copies of the communication structures of these organizations”

—Melvin Conway
user interface
server-side
DBA
Monoliths vs. Microservices
Smart Endpoints, Dumb Pipes
Standardize on integration, not platform

embrace polyglot solutions where sensible

Standardize in the gaps between services - be flexible about what happens inside the boxes

Have one, two or maybe three ways of integrating, not 20.

Pick some sensible conventions, and stick with them.
Decentralized Data Management

ACID versus BASE

monolith - single database

microservices - application databases
Avoid distributed transactions if at all possible
Decentralized Governance
Decentralized Governance
Decentralized Governance

Enterprise architects suffer from less pressure to make the correct choice(s) in microservice architectures.
Infrastructure Automation

compile, unit and functional test
acceptance test
integration test
user acceptance test
performance test

deploy to production

run on build machine
deployed on build machine
deployed to integration environment
deployed to UAT environment
deployed to performance environment

BUILD

TEST & RELEASE

Source Code

Build 1

Ace 1

Build 2

Ace 2

FAU OUT

FAU IN

Build 3

Ace 3

Build 4

Ace 4

Package

Integration

Test Env

Staging

Production

Env & app Config

Test Env
Small, Single Responsibility

small enough to fit in your head

rewrite over maintain

(10—1000 LOC)-ish / service

single responsibility
What problem

characteristics

engineering

AGENDA
Microservice

client requests → api layer → service components

maximize easy evolution
Microservice is the first architectural style developed post-Continuous Delivery.
Benefits
Microservice Implementation

http://www.infoq.com/presentations/Micro-Services
Asynchronicity

return optimized for ranking/aggregation, not display

Prefer timely partial over slow complete
Integration & Disintegration
Complected Deployments

complect, transitive verb: intertwine, embrace, especially to plait together
Evolutionary Architecture

Components are deployed.

Features are released.

Applications consist of routing.
Evolutionary Architecture

Dis-integrate services that monitoring shows are no longer used
How Big?

- Release risk vs. number of services

Graph showing a decrease in release risk as the number of services increases.
Backends for Frontends

https://www.thoughtworks.com/insights/blog/bff-soundcloud
Backends for Frontends
BFF as Migration Path
what problem

characteristics

engineering

AGENDA
Design For Failure

clients must respond gracefully to provider failure

aggressive monitoring:
- business relevant
- architectural
- semantic
You have to get *much* better at monitoring.
Capture metrics, and logs, for each node, and aggregate them to get a rolled up picture.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>numberOfApplicationErrors</td>
<td>57</td>
</tr>
<tr>
<td>numberOfServicedRequestsWithResponse200</td>
<td>136711</td>
</tr>
<tr>
<td>numberOfServicedRequestsWithResponse304</td>
<td>27782</td>
</tr>
<tr>
<td>numberOfServicedRequestsWithResponse404</td>
<td>303</td>
</tr>
<tr>
<td>numberOfServicedRequestsWithResponse500</td>
<td>141</td>
</tr>
<tr>
<td>totalNumberOfServicedRequests</td>
<td>172383</td>
</tr>
</tbody>
</table>
Use synthetic transactions to test production systems.
Correlation IDs

Use correlation IDs to track down nasty bugs
Use timeouts, circuit breakers and bulk-heads to avoid cascading failure.
Engineering Consistency

Consider Service Templates to make it easy to do the right thing!
Orchestration describes the automated arrangement, coordination, and management of complex computer systems, middleware, and services.

http://en.wikipedia.org/wiki/Orchestration_(computing)
choreography vs. orchestration in microservices
Orchestration

- recalc
- quote
- update
- claims
- change
- address
- notify
- insured
Choreography

mediator versus broker topology

http://shop.oreilly.com/product/110000195.do
Testing Microservices

http://martinfowler.com/articles/microservice-testing/
Test Pyramid for Microservices

- Exploratory
- End-to-end
- Component
- Integration
- Unit
Integration Testing
Integration Testing

Resources
- Service Layer
- Domain
- Repositories
  - Data Mappers / ORM
- External Datastore
- Gateway
- HTTP Client
- External Service

Boundary Types:
- Network Boundary
- Logical Boundary
- External
- Persistence
- Connection
- Integration Test Boundary
Component Testing
Component Testing

shims:

inproctester
github.com/aharin/inproctester

Plasma
github.com/jennifersmith/plasma
Component Testing

- Resources
  - Service Layer
  - Domain
  - Gateways
  - HTTP Client
- Repositories
- Data Mappers / ORM
- External Datastore

- Protocols
- Domains
- Network Boundary
- Logical Boundary
- External
- Persistence
- Communication
- Component Test Boundary
Consumer Driven Contracts

http://martinfowler.com/articles/consumerDrivenContracts.html
Contract Testing

Pact
github.com/realestate-com-au/pact

Pacto
github.com/thoughtworks/pacto

Janus
github.com/gga/janus
End-to-End Testing

- focus on personas & user journeys
- make tests data-independent
- as few as possible
- choose endpoints wisely
- rely on infrastructure as code for repeatability
Deployment
Abstract out underlying platform differences to provide a uniform deployment mechanism.
Don’t Let Changes Build Up

staging

production
Don’t Let Changes Build Up

Don’t let changes build up - release as soon as you can, and preferably one at a time!
Service Discovery
Dynamic Service Registries

https://consul.io/

http://zookeeper.apache.org/

https://coreos.com/etcd/
Service Visualization

adrianco / spigo
https://github.com/adrianco/spigo
# Tools

## Ansible
A versatile orchestration engine that can automate systems and apps. Instead of a custom scripting language or code, it is very simple and shell based. It is also agent-less, so you can just start using it right away and get things done.

- linux, open-source, provisioning, config-mgmt, orchestration, python

## Dokku Alt
Dokku on Steroids. The smallest PaaS implementation you’ve ever seen. It’s fork of original dokku. The idea behind this fork is to provide complete solution with plugins covering most of use-cases which are stable and well tested.

- linux, open-source, virt, cloud-paas, provisioning, shell?

## Batou
Batou makes it easy to perform automated deployments. It combines Fabric’s simplicity and SSH automation, with Puppet’s declarative syntax and idempotence

- linux, open-source, provisioning, python

## Dokku
It uses docker, git-receive and a few other lightweight and clever libraries to build a quick PaaS, all around just 100 lines of code! An excellent small tool to get started with PaaS systems. The same developer is creating a larger scale, production quality system called Flynn.

- linux, open-source, virt, cloud-paas, provisioning, shell?

## Bcfg2
Bee-config (Bcfg) 2 is a centralized configuration management server to configure large number of systems, built

- linux, open-source, virt, cloud-paas, provisioning, shell?

## FAI

---

[www.devopsbookmarks.com/](http://www.devopsbookmarks.com/)
Turnkey Platforms

Deploy and manage microservices with power and ease.

Vamp, or the Very Awesome Microservices Platform, savings the pain out of running complex and critical service-based architectures. Vamp’s core features are a platform-agnostic microservices DSL, powerful A/B testing/canary releasing, autoscaling and an integrated metrics & event engine.

Learn more  Quick start

Vamp 3.0.3 licensed under Apache 2.0

Manage your microservices through a single pane of glass.

Vamp provides you with a set of powerful features to manage microservices and container based architectures, all through a single pane of glass. Vamp has deeply ingrained support for canary releases and A/B testing, auto-scaling, service discovery, a live metrics & events stream.

Simple canary releasing
Testing out a new service with just your iOS users?
Vamp gives you a straight DSL and API to plan your canary releases. Blue/green deployments and A/B tests.

Platform independent auto scaling
Using SLA, Vamp allows you to scale up (and down) your services, just like on AWS, but on any platform: in the cloud or on-premises. Vamp’s SLA model is open, event driven and pluggable.

Learn more →
Turnkey Platforms

Otto

Today we announce Otto — the successor to Vagrant. Otto is the single solution to develop and deploy any application, with first class support for microservices. It is the most powerful tool we’ve built yet.

Otto automatically builds development environments without any configuration; it can detect your project type and has built-in knowledge of industry-standard tools to setup a development environment that is ready to go. When you’re ready to deploy, Otto builds and manages an infrastructure, sets up servers, builds, and deploy the application.

With the growing trend of microservices, Otto knows how to install and configure service dependencies for development and deployment. It automatically exposes these dependencies via DNS for your application to consume.

Vagrant brought simplicity and power to development, and we believe we’ve brought that same elegant user experience to both development and deployment with Otto.

Read on to learn more.

The industry has needed this kind of abstraction for a long time, and it’s finally here with Otto.

Ben McRee, Head of Technical Operations at Confluent Commence

The Successor to Vagrant

Vagrant does a lot right. Vagrant downloads have continued to accelerate and grow to this day. There are more Vagrant downloads per day today than ever before. But we think we can do even better.

We’ve learned a lot about developers and development environments over the past six years:

• Development environments are similar: All Ruby development environments look alike; all PHP development environments look alike, etc. There isn’t much deviation between two development environments for the same language or framework. The Vagrantfile requires users to configure these environments for every project, rather than having it abstracted away.

• Developers want to deploy: After developing an application, the next step is to deploy “vagrant up” to production has been a feature request for years. Production environments look very different from development: load balancers, firewalls, routing concerns, new configurations, etc. Unfortunately, the Vagrantfile isn’t a good format to describe this information.

• Microservices are difficult: More and more applications are being written in a service oriented way. Modeling these services for development and deploy is difficult. The Vagrantfile puts the burden of installing/configuring every service onto

https://www.hashicorp.com/blog/otto.html
what problem

class characteristics

eengineering

AGENDA
(Micro)service architectures provide unique benefits at the cost of increased (essential) complexity.
If you can't build a monolith, what makes you think microservices are the answer?

Distributed big balls of mud

If you can't build a monolith, what makes you think microservices are the answer?

If you want evidence that the software development industry is susceptible to fashion, just go and take a look at all of the hype around microservices. It's everywhere! For some people microservices is "the next big thing", whereas for others it's simply a lightweight evolution of the big SOAP service-oriented architectures that we saw 10 years ago "done right". I do like a lot of what the current microservice architectures are doing, but it's by no means a silver bullet. Okay, I know that sounds obvious, but I think many people are jumping on them for the wrong reason.

www.codingthearchitecture.com/2014/07/06/distributed_big_balls_of_mud.html
Service-based Architecture

is there a middle ground?

service-oriented architecture

microservices architecture

service-based architecture
Service-based Architecture

- service granularity
- database scope
- integration hub
Migration

presentation layer
business layer
persistence layer
database layer

client requests
api layer

service component
service component
service component
service component
service component
service component
service component
service component
service component

database layer
Partition Along Natural Boundaries

Build a small number of larger services first.
Inverse Conway Maneuver

Build teams that look like the architecture you want (and it will follow).
Efferent Coupling

Strive for low efferent coupling for your team.
Continuous Delivery

Teams with low efferent coupling deliver relatively independently into a common integration pipeline (without fearing breaking each others builds).