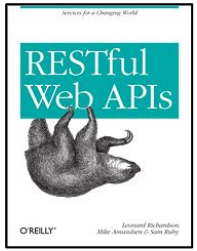
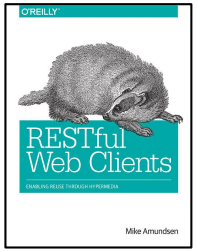
RESTful Microservices
from the Ground Up

Mike Amundsen
API Academy
@mamund

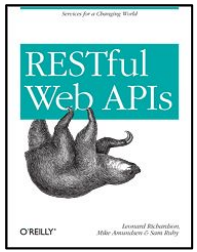


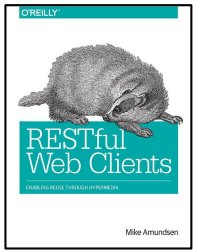


Agenda

- 9:00 - 9:45 : What are RESTful Microservices?
- 9:45 - 10:30 : Models, Messages, and Vocabularies
- 10:30 - 10:45 : BREAK
- 10:45 - 11:30 : Runtime Service Infrastructure
- 11:30 - 12:15 : The Adaptable System
- 12:15 - 12:30 : Summary

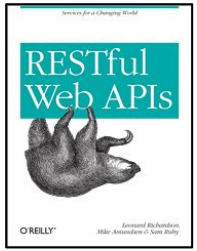
API ACADEMY

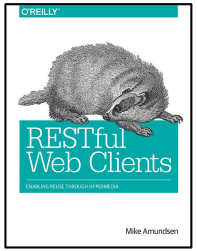




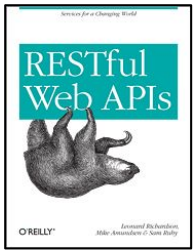
Materials

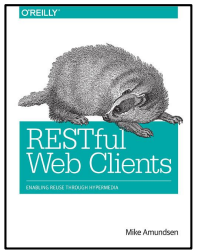
- Laptop w/ wifi
- NodeJS
- Browser and cURL
- Your favorite editor
- Github and Heroku
- Pen and Paper





What are RESTful Microservices?

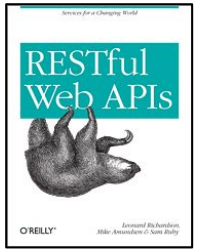


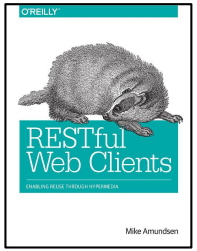


What are RESTful Microservices?

- Microservices
- RESTful-ness
- A New Kind of Service
- *Analysis Exercise*

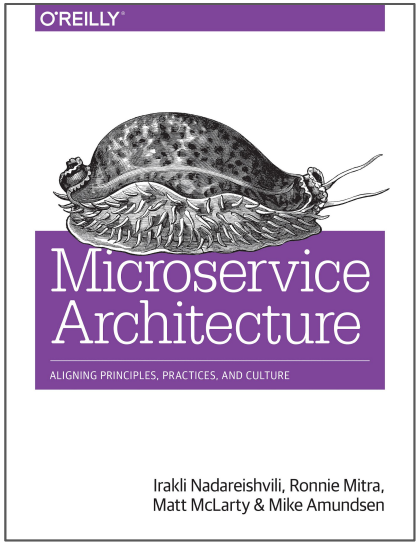
API ACADEMY

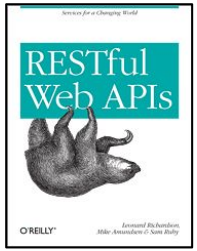


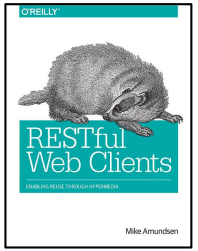


Microservices

"A microservice is an independently deployable component of bounded scope that supports interoperability through message-based communication. Microservice architecture is a style of engineering highly automated, evolvable software systems made up of capability-aligned microservices."



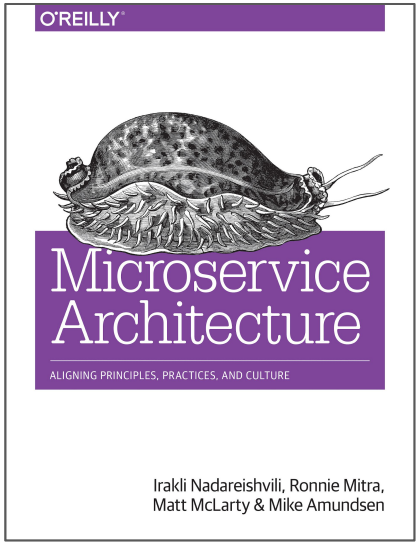


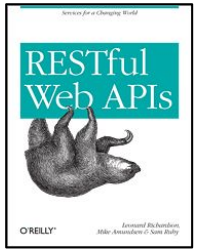


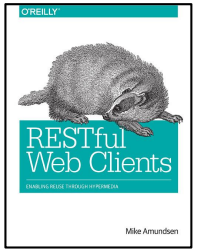
API ACADEMY

Microservices

*"A microservice is an **independently deployable** component of bounded scope that supports interoperability through message-based communication. Microservice architecture is a style of engineering highly automated, evolvable software systems made up of capability-aligned microservices."*



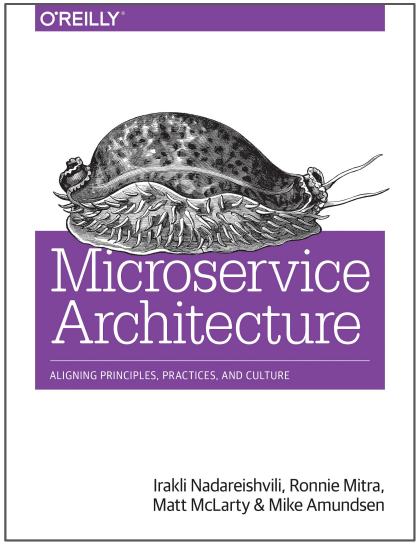


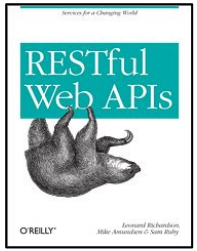


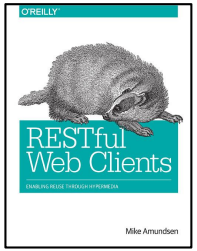
API ACADEMY

Microservices

*"A microservice is an **independently deployable** component of **bounded scope** that supports interoperability through message-based communication. Microservice architecture is a style of engineering highly automated, evolvable software systems made up of capability-aligned microservices."*



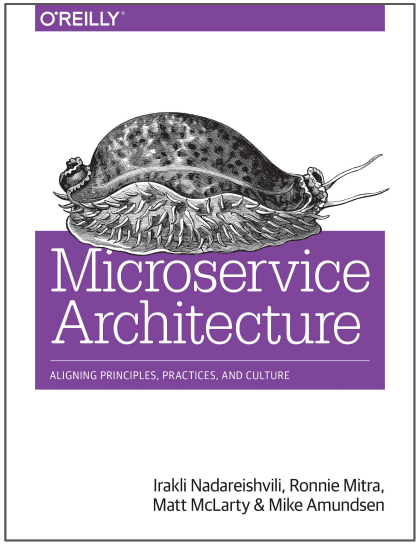


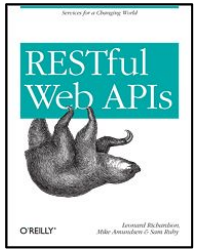


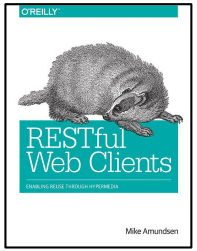
API ACADEMY

Microservices

*"A microservice is an **independently deployable** component of **bounded scope** that supports interoperability through **message-based** communication. Microservice architecture is a style of engineering highly automated, evolvable software systems made up of capability-aligned microservices."*



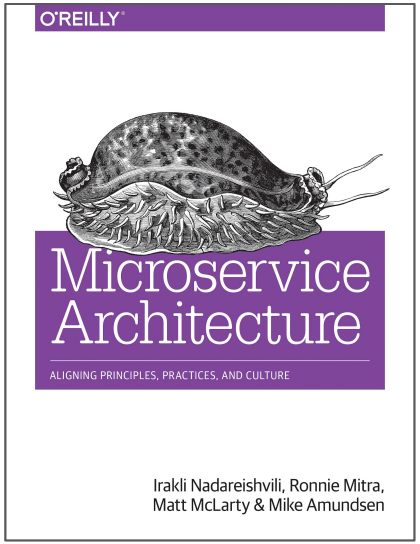


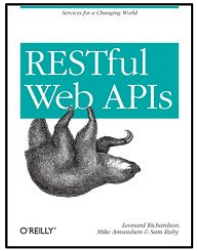


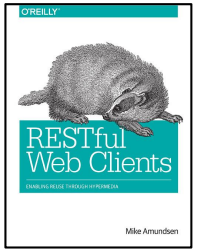
API ACADEMY

Microservices

*"A microservice is an **independently deployable** component of **bounded scope** that supports interoperability through **message-based** communication. Microservice architecture is a style of engineering **highly automated**, evolvable software systems made up of capability-aligned microservices."*



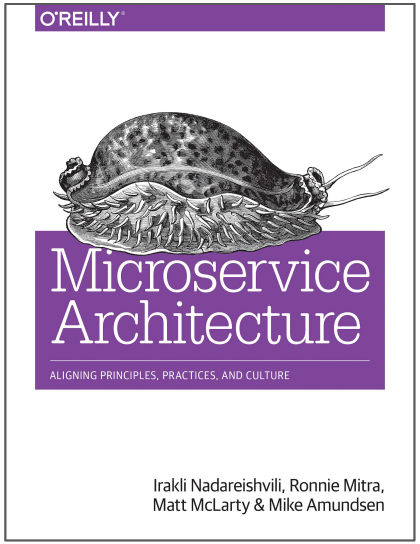


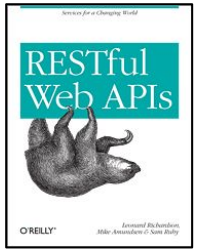


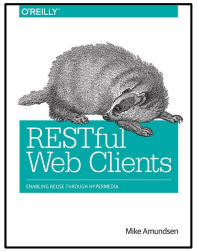
API ACADEMY

Microservices

*"A microservice is an **independently deployable** component of **bounded scope** that supports interoperability through **message-based** communication. Microservice architecture is a style of engineering **highly automated**, **evolvable** software systems made up of capability-aligned microservices."*



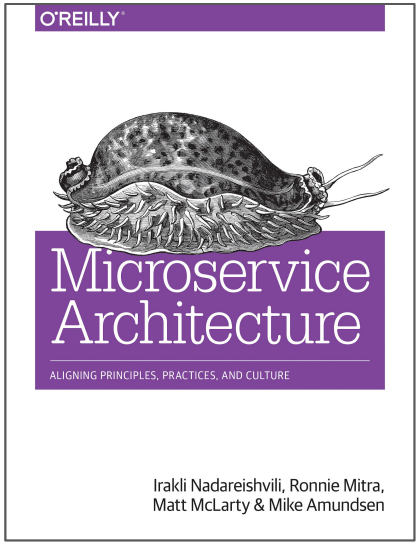


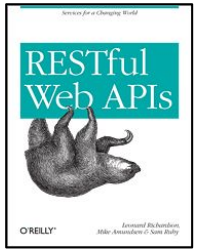


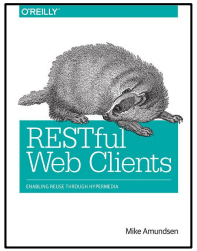
API ACADEMY

Microservices

- Independently deployable
- Bounded scope
- Message-based
- Highly automated
- Evolvable





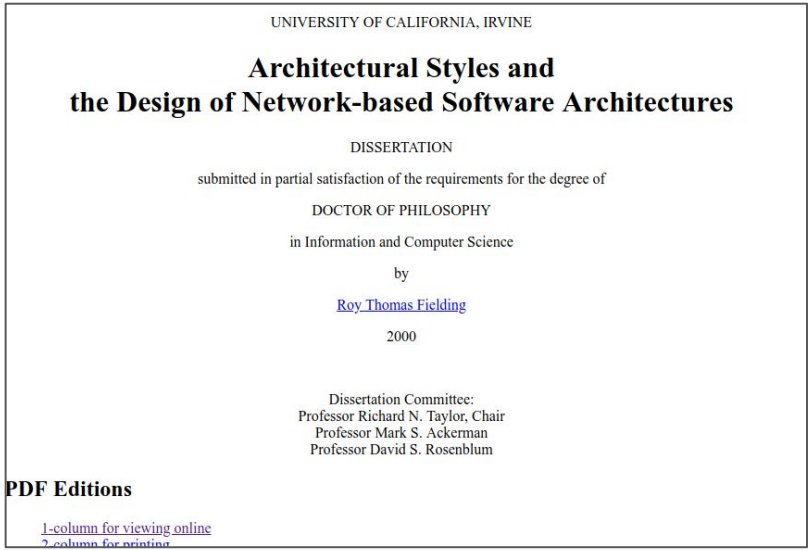


API ACADEMY

RESTful-ness

"This dissertation defines a framework for understanding software architecture via architectural styles and demonstrates how styles can be used to guide the architectural design of network-based application software."

- Fielding, 2000



UNIVERSITY OF CALIFORNIA, IRVINE

Architectural Styles and the Design of Network-based Software Architectures

DISSERTATION

submitted in partial satisfaction of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in Information and Computer Science

by

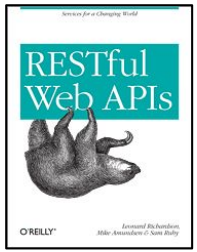
Roy Thomas Fielding

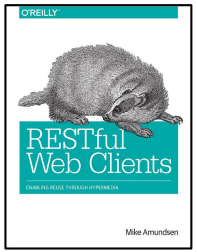
2000

Dissertation Committee:
Professor Richard N. Taylor, Chair
Professor Mark S. Ackerman
Professor David S. Rosenblum

PDF Editions

1-column for viewing online
2-column for printing

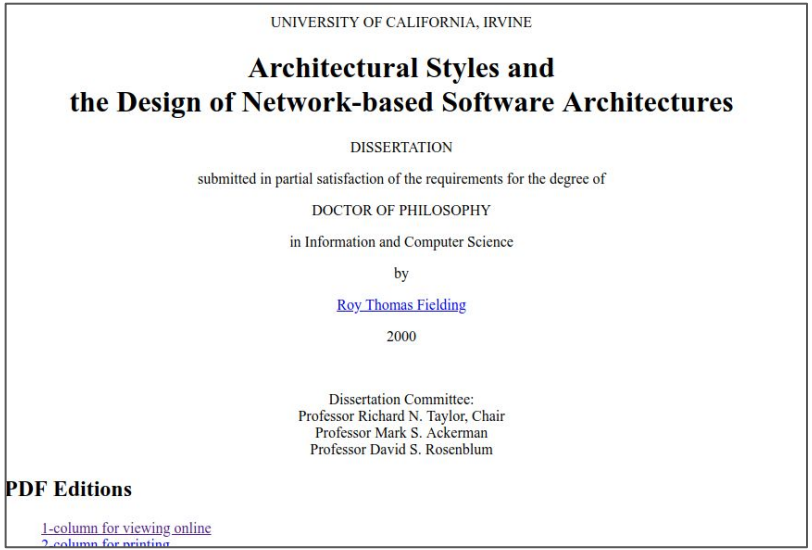




RESTful-ness

"This dissertation defines a framework for ***understanding software architecture*** *via architectural styles and demonstrates how styles can be used to guide the architectural design of network-based application software."*

- Fielding, 2000



UNIVERSITY OF CALIFORNIA, IRVINE

Architectural Styles and the Design of Network-based Software Architectures

DISSERTATION

submitted in partial satisfaction of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in Information and Computer Science

by

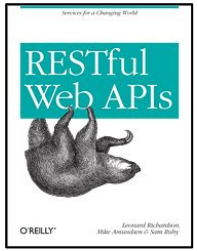
Roy Thomas Fielding

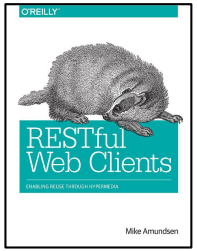
2000

Dissertation Committee:
Professor Richard N. Taylor, Chair
Professor Mark S. Ackerman
Professor David S. Rosenblum

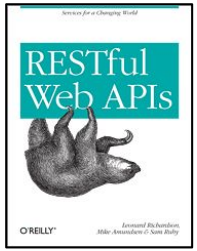
PDF Editions

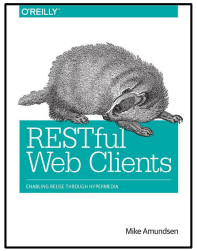
1-column for viewing online
2-column for printing





RESTful-ness



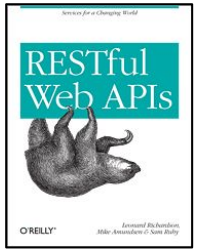


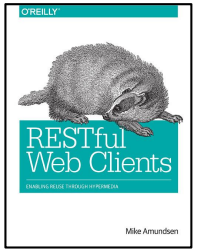
RESTful-ness

Properties

- Performance
- Scalability
- Simplicity
- Modifiability
- Visibility
- Portability
- Reliability

API ACADEMY





RESTful-ness

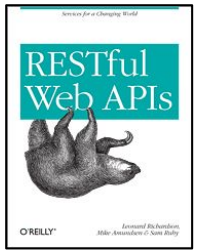
Properties

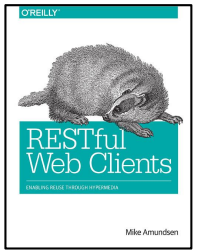
- Performance
- Scalability
- Simplicity
- Modifiability
- Visibility
- Portability
- Reliability

+ Requirements

- Low-Entry Barrier
- Extensibility
- Distributed Hypermedia
- Internet Scale

API ACADEMY

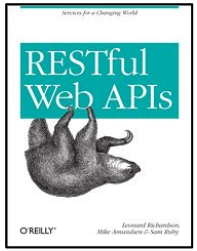


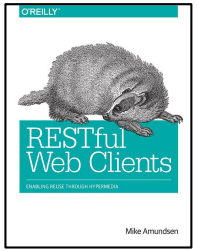


RESTful-ness

Properties	+ Requirements	= Constraints
• Performance	• Low-Entry Barrier	• Client-Server
• Scalability	• Extensibility	• Stateless
• Simplicity	• Distributed Hypermedia	• Cache
• Modifiability	• Internet Scale	• Uniform Interface
• Visibility		• Layered System
• Portability		• Code on Demand
• Reliability		

API ACADEMY

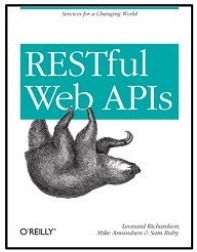


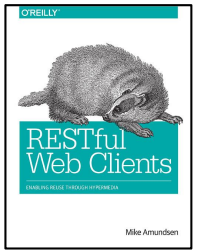


RESTful-ness

"When I say hypertext, I mean the simultaneous presentation of information and controls such that the information becomes the affordance through which the user (or automaton) obtains choices and selects actions."

- Fielding, 2008



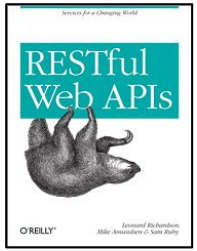


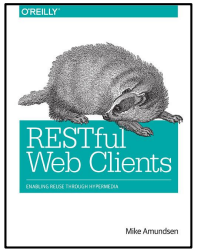
API ACADEMY

RESTful-ness

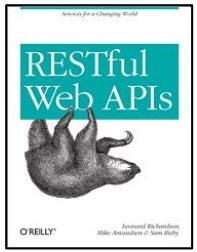
"When I say hypertext, I mean the simultaneous presentation of information and controls such that the ***information becomes the affordance*** *through which the user (or automaton) obtains choices and selects actions."*

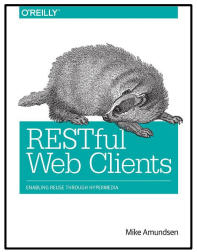
- Fielding, 2008





Fielding's REST ticks many of the boxes for Microservices

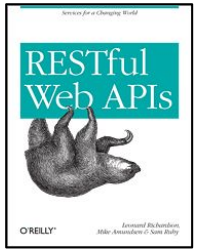


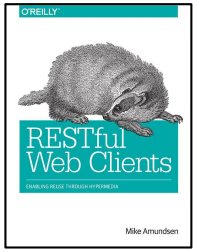


A New Kind of Service Constraints

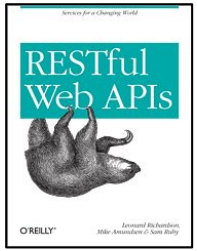
- Manage only service-state, not client state (no persistent sessions)
- Rely on Uniform Interface protocols (HTTP, MQTT, CoAP, etc.)
- Communicate in Structured Formats (HTML, Atom, Cj, HAL, etc.)
- Support Shared Vocabularies (ALPS, DCAP, etc.)
- Support Advertising, Discovery, and Health-Check

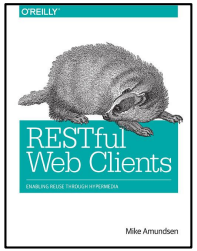
API ACADEMY



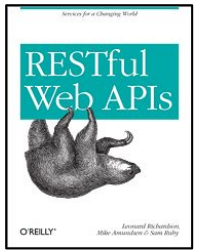


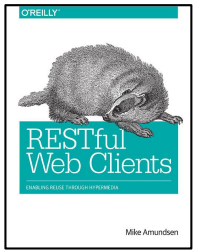
Analysis Exercise





Models, Messages, and Vocabularies

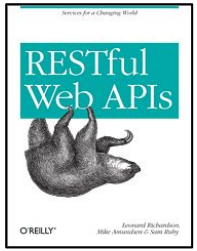


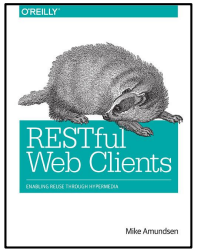


Models, Messages, and Vocabularies

- Models on the Inside
- Messages on the Outside
- Vocabularies Everywhere
- *Design Exercise*

API ACADEMY

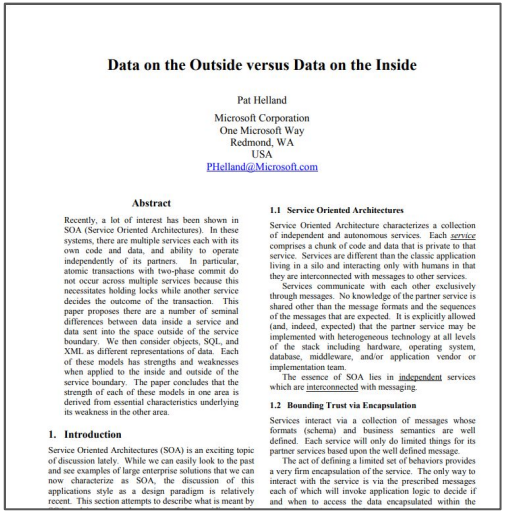




Data on the Inside vs. Data on the Outside

"This paper proposes there are a number of seminal differences between data inside a service and data sent into the space outside of the service boundary."

-- Pat Helland, 2005



Data on the Outside versus Data on the Inside

Pat Helland

Microsoft Corporation
One Microsoft Way
Redmond, WA
USA
PHelland@Microsoft.com

Abstract

Recently, a lot of interest has been shown in SOA (Service Oriented Architectures). In these systems, there are multiple services each with its own code and data, and ability to operate independently of its partners. In particular, atomic transactions with two-phase commit do not occur across multiple services because this necessitates holding locks while another service decides the outcome of the transaction. This paper proposes there are a number of seminal differences between data inside a service and data sent into the space outside of the service boundary. We then consider objects, SQL, and XML as different representations of data. Each of these models has strengths and weaknesses when applied to the inside and outside of the service boundary. The paper concludes that the strength of each of these models in one area is derived from essential characteristics underlying its weakness in the other area.

1. Introduction

Service Oriented Architectures (SOA) is an exciting topic of discussion lately. While we can easily look to the past and see examples of large enterprise solutions that we can now characterize as SOA, the discussion of this applications style as a design paradigm is relatively recent. This section attempts to describe what is meant by

1.1 Service Oriented Architectures

Service Oriented Architecture characterizes a collection of independent and autonomous services. Each service comprises a chunk of code and data that is private to that service. Services are different than the classic application living in a silo and interacting only with humans in that they are interconnected with messages to other services.

Services communicate with each other exclusively through messages. No knowledge of the partner service is shared other than the message formats and the sequences of the messages that are expected. It is explicitly allowed (and, indeed, expected) that the partner service may be implemented with heterogeneous technology at all levels of the stack including hardware, operating system, database, middleware, and/or application vendor or implementation team.

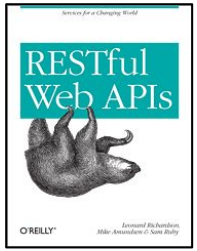
The essence of SOA lies in independent services which are interconnected with messaging.

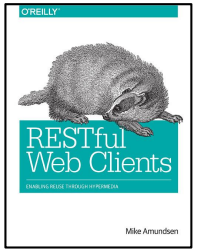
1.2 Bounding Trust via Encapsulation

Services interact via a collection of messages whose formats (schema) and business semantics are well defined. Each service will only do limited things for its partner services based upon the well defined message.

The act of defining a limited set of behaviors provides a very firm encapsulation of the service. The only way to interact with the service is via the prescribed messages each of which will invoke application logic to decide if and when to access the data encapsulated within the

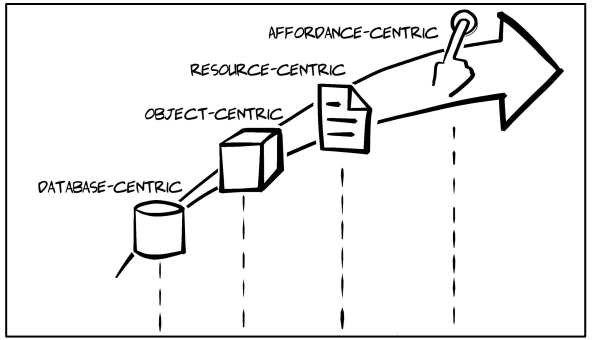
API ACADEMY



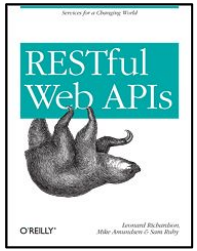


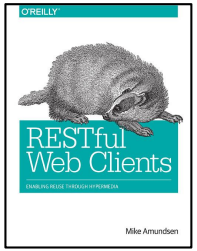
Models on the Inside

- Inside is immediate, transactional
- Data storage models (**`customers.db, orders.db`**)
- Programming object models (**`objCustomer`**)
- Inside is local, controllable
- Inside relies on a shared "now"



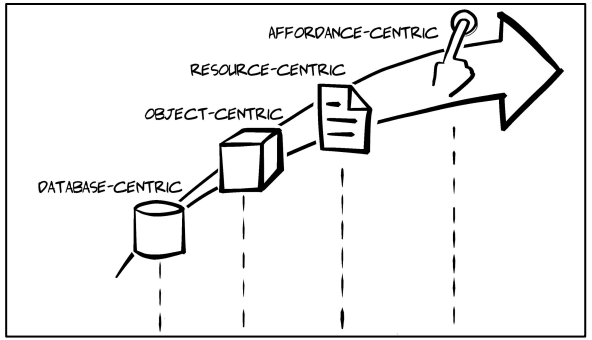
API ACADEMY

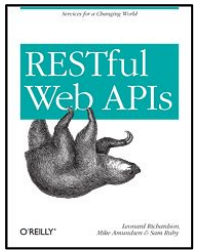


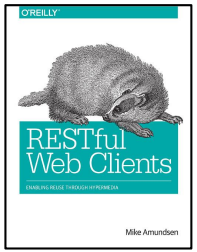


Messages on the Outside

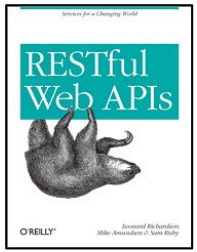
- Outside is always in the past, non-transactional
- Resource models (**`/customers/, /orders/`**)
- Message models (**`customer.html, order.hal`**)
- Outside is remote, uncontrollable
- There is no shared "now"

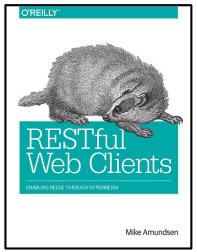






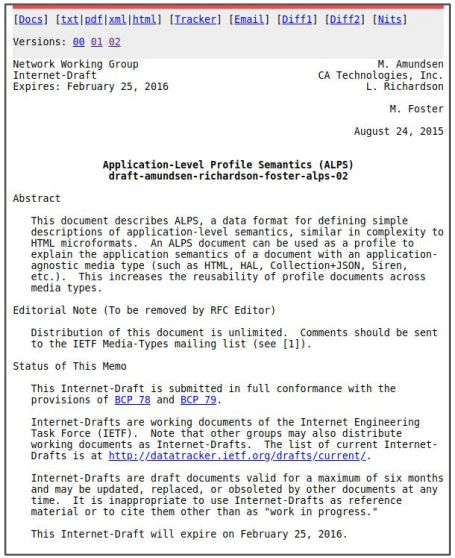
If the models are different inside and out, what is shared?





Vocabularies Everywhere

- Vocabulary is how humans share (language, slang, etc.)
- We use the same vocabulary for many models
- Vocabularies delineate domains (medicine, IT, etc.)
- IT vocabularies already exist:
  - Dublin Core
  - schema.org
  - microformats
  - IANA Link Relation Values
- ALPS is a media-type and protocol independent description format



[Docs] [txt|pdf|xml|html] [Tracker] [Email] [Diff1] [Diff2] [Nits]

Versions: 00 01 02

Network Working Group
Internet-Draft
Expires: February 25, 2016

M. Amundsen
CA Technologies, Inc.
L. Richardson

M. Foster

August 24, 2015

Application-Level Profile Semantics (ALPS)
draft-amundsen-richardson-foster-alps-02

Abstract

This document describes ALPS, a data format for defining simple descriptions of application-level semantics, similar in complexity to HTML microformats. An ALPS document can be used as a profile to explain the application semantics of a document with an application-agnostic media type (such as HTML, HAL, Collection+JSON, Siren, etc.). This increases the reusability of profile documents across media types.

Editorial Note (To be removed by RFC Editor)

Distribution of this document is unlimited. Comments should be sent to the IETF Media-Types mailing list (see [1]).

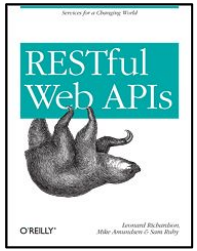
Status of This Memo

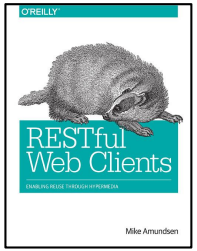
This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

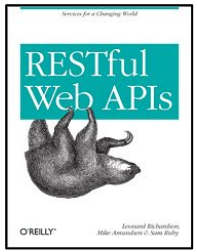
This Internet-Draft will expire on February 25, 2016.

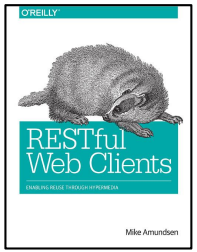




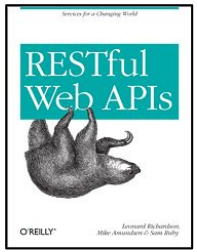
API ACADEMY

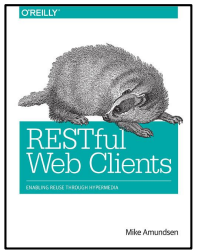
Design Exercise



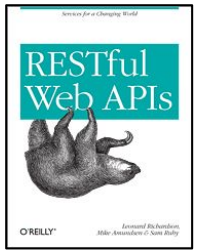


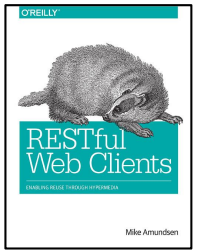
BREAK


RESTful
Web APIs
O'REILLY


O'REILLY
RESTful
Web Clients
Mike Amundsen

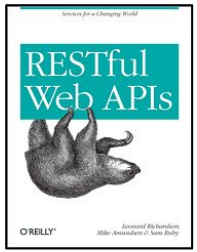
Runtime Service Infrastructure

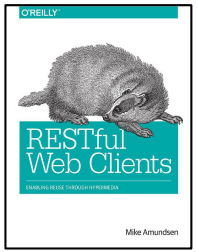




Runtime Service Infrastructure

- Advertising Services
- Discovering Services
- Health Checking
- *Discovery Exercise*

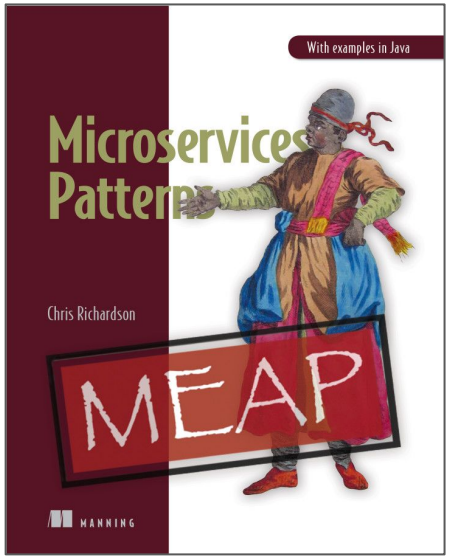


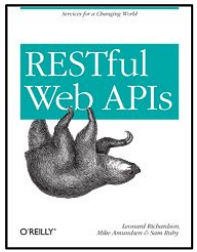


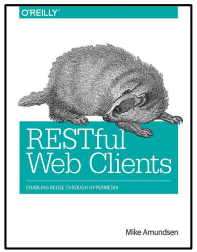
Advertising Services

"A service instance is responsible for registering itself with the service registry. On startup the service instance registers itself (host and IP address) with the service registry and makes itself available for discovery. The client must typically periodically renew its registration so that the registry knows it is still alive. On shutdown, the service instance unregisters itself from the service registry."

-- microservices.io





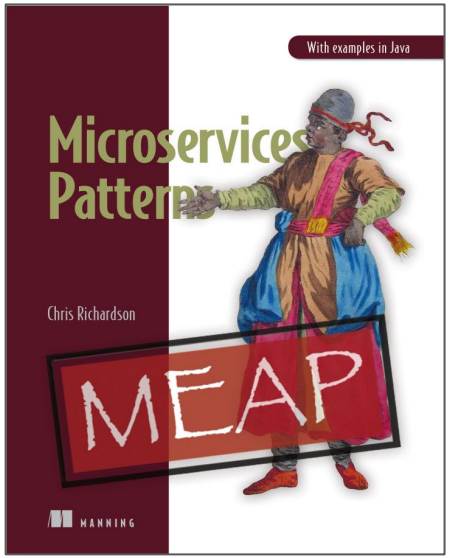


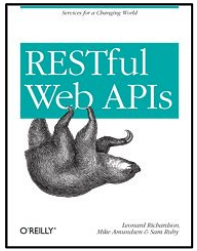
API ACADEMY

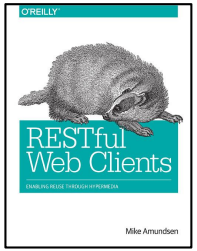
http://microservices.io/patterns/self-registration.html

Advertising Services

- Register upon startup
- De-Register at shutdown
- Renew periodically
- De-Register after crashes



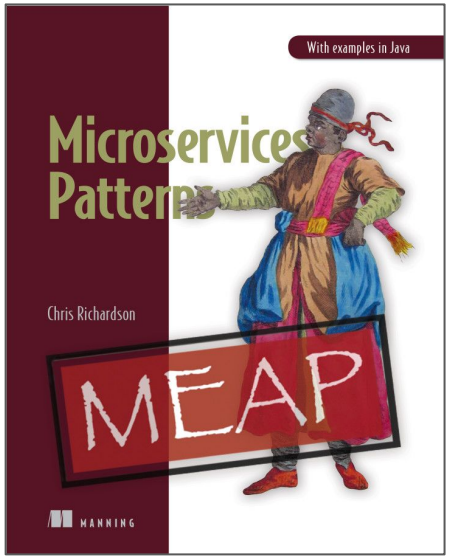


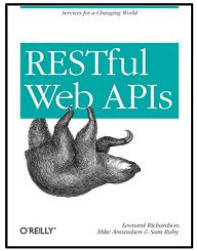


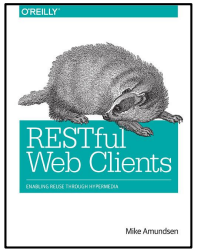
API ACADEMY

Advertising Services

CODE EXAMPLE HERE





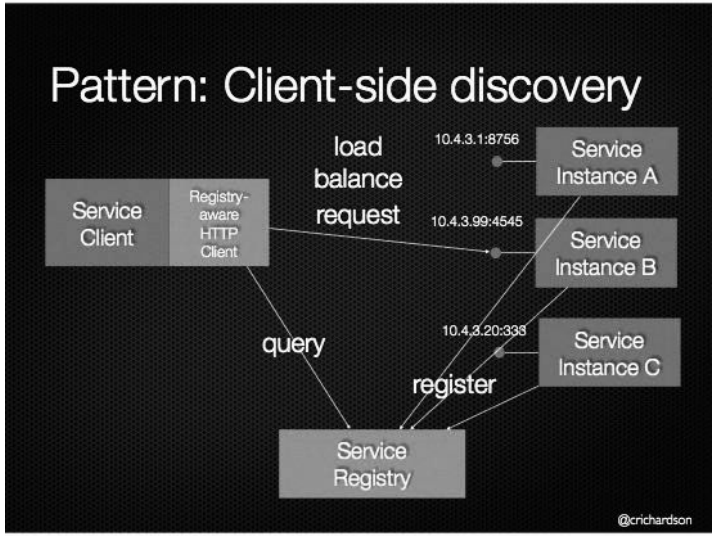


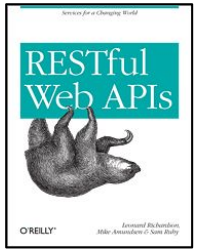
API ACADEMY

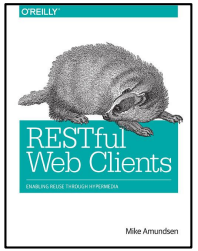
Discovering Services

When making a request to a service, the client obtains the location of a service instance by querying a Service Registry, which knows the locations of all service instances.

-- microservices.io



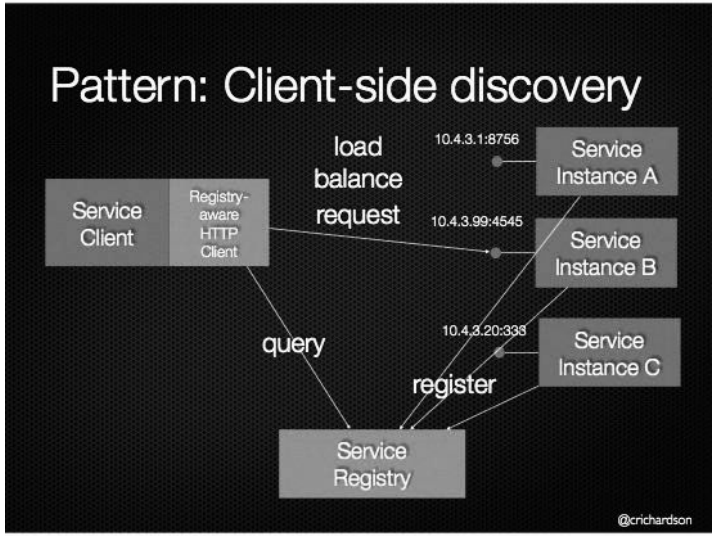


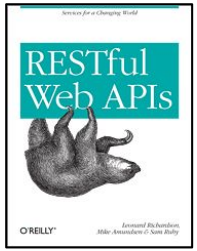


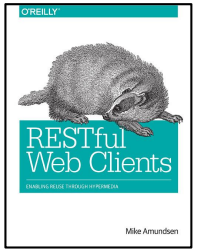
http://microservices.io/patterns/client-side-discovery.html

Discovering Services

- Configure client w/ **`registryURL`**
- Query Registry w/ **`serviceName`**
- Registry returns **`serviceURLport`**
- Client uses **`serviceURLport`**
- Renewal optional



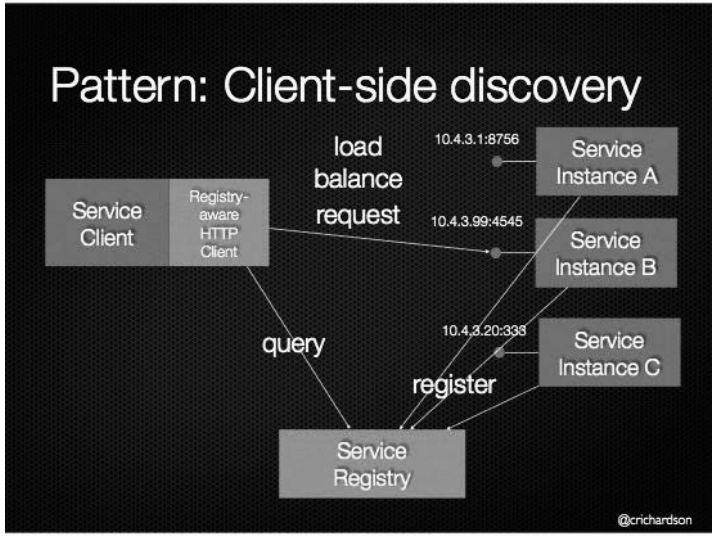


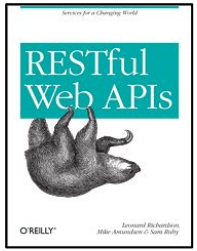


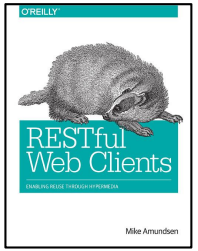
http://microservices.io/patterns/client-side-discovery.html

Discovering Services

CODE EXAMPLE





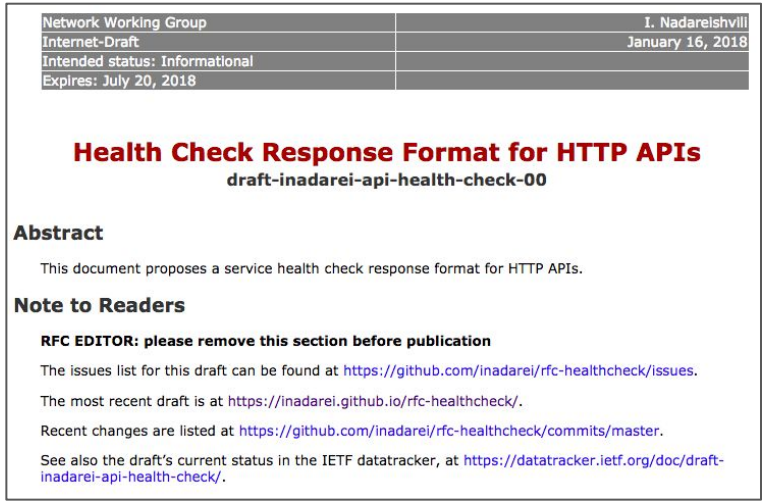


http://microservices.io/patterns/client-side-discovery.html

Health Checking

A service has an health check API endpoint (e.g. HTTP /health) that returns the health of the service. A health check client - a monitoring service, service registry or load balancer - periodically invokes the endpoint to check the health of the service instance.

-- microserivce.io



Network Working Group	I. Nadareishvili
Internet-Draft	January 16, 2018
Intended status: Informational	
Expires: July 20, 2018	

Health Check Response Format for HTTP APIs

draft-inadarei-api-health-check-00

Abstract

This document proposes a service health check response format for HTTP APIs.

Note to Readers

RFC EDITOR: please remove this section before publication

The issues list for this draft can be found at https://github.com/inadarei/rfc-healthcheck/issues.

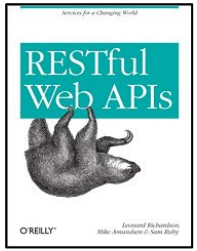
The most recent draft is at https://inadarei.github.io/rfc-healthcheck/.

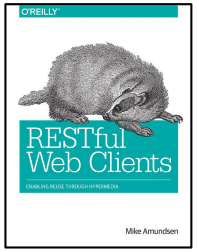
Recent changes are listed at https://github.com/inadarei/rfc-healthcheck/commits/master.

See also the draft's current status in the IETF datatracker, at https://datatracker.ietf.org/doc/draft-inadarei-api-health-check/.

API ACADEMY

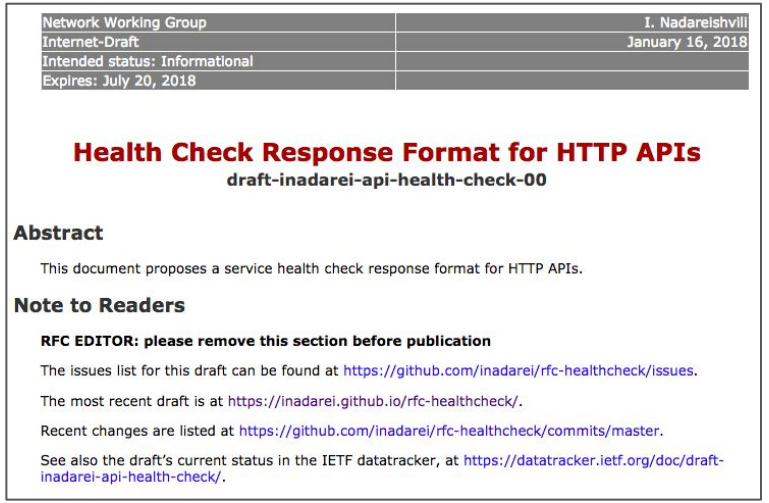
https://inadarei.github.io/rfc-healthcheck/





Health Checking

- Services support health-checks
- Services renew with the registry
- De-reg service on failed health requests
- De-reg service on expired renewals



Network Working Group	I. Nadareishvili
Internet-Draft	January 16, 2018
Intended status: Informational	
Expires: July 20, 2018	

Health Check Response Format for HTTP APIs

draft-inadarei-api-health-check-00

Abstract

This document proposes a service health check response format for HTTP APIs.

Note to Readers

RFC EDITOR: please remove this section before publication

The issues list for this draft can be found at https://github.com/inadarei/rfc-healthcheck/issues.

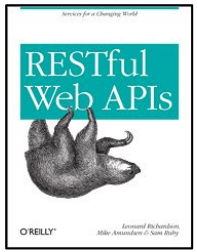
The most recent draft is at https://inadarei.github.io/rfc-healthcheck/.

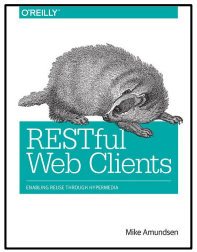
Recent changes are listed at https://github.com/inadarei/rfc-healthcheck/commits/master.

See also the draft's current status in the IETF datatracker, at https://datatracker.ietf.org/doc/draft-inadarei-api-health-check/.

API ACADEMY

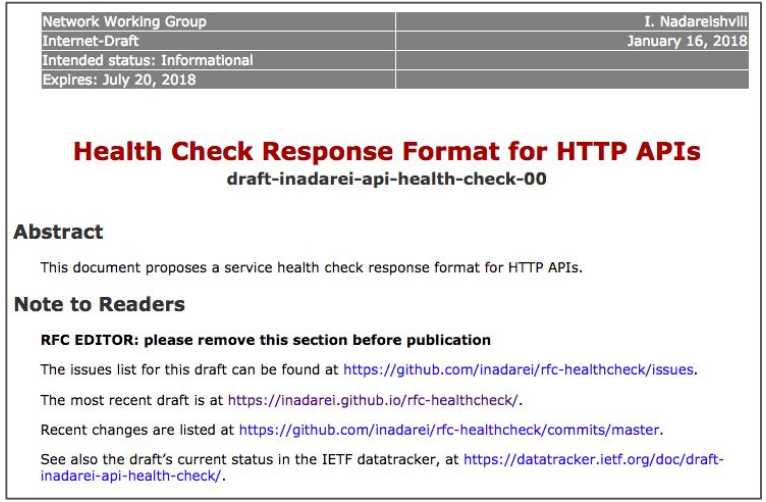
https://inadarei.github.io/rfc-healthcheck/





Health Checking

CODE EXAMPLE



Network Working Group	I. Nadareishvili
Internet-Draft	January 16, 2018
Intended status: Informational	
Expires: July 20, 2018	

Health Check Response Format for HTTP APIs

draft-inadarei-api-health-check-00

Abstract

This document proposes a service health check response format for HTTP APIs.

Note to Readers

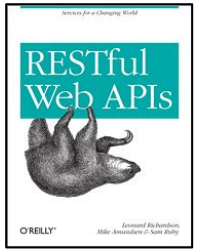
RFC EDITOR: please remove this section before publication

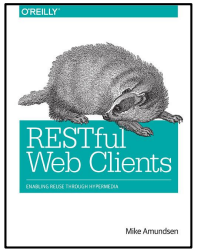
The issues list for this draft can be found at https://github.com/inadarei/rfc-healthcheck/issues.

The most recent draft is at https://inadarei.github.io/rfc-healthcheck/.

Recent changes are listed at https://github.com/inadarei/rfc-healthcheck/commits/master.

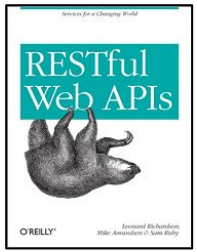
See also the draft's current status in the IETF datatracker, at https://datatracker.ietf.org/doc/draft-inadarei-api-health-check/.

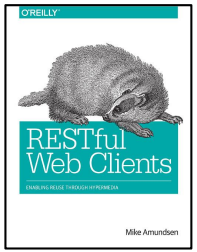




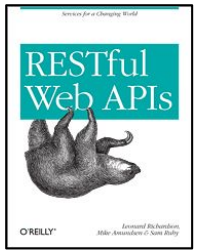
https://inadarei.github.io/rfc-healthcheck/

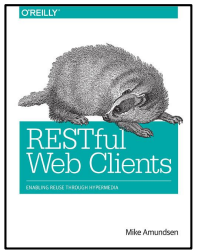
Discovery patterns are the DNS of application services.



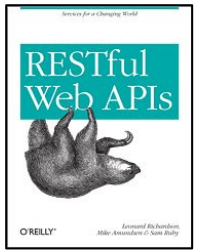


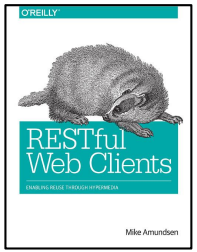
Discovery Exercise





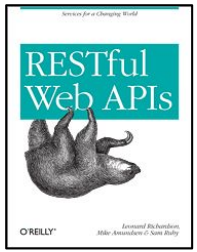
The Adaptable System

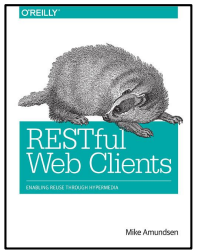




The Adaptable System

- Service/API Designers
- Evolvable Providers
- Adaptable Consumers
- *Adaptation Exercise*

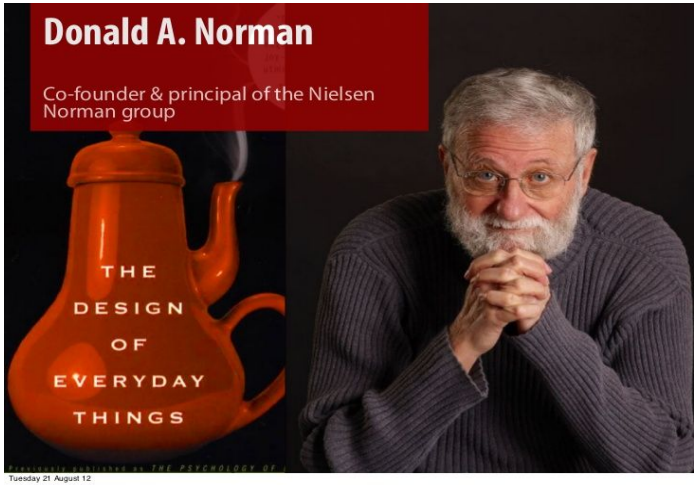




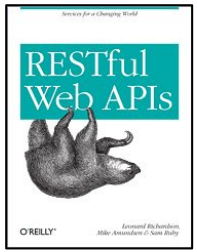
Service/API Designers

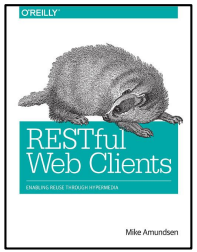
"The value of a well-designed object is when it has such a rich set of affordances that the people who use it can do things with it that the designer never imagined."

-- Donald Norman, 1994



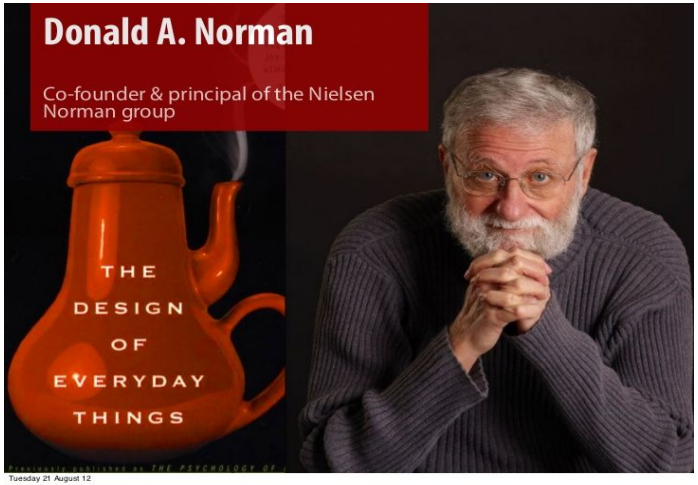
@jnd1er



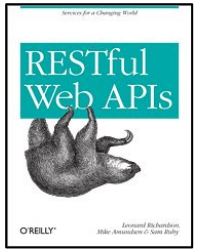


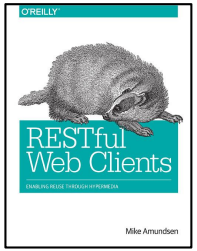
Service/API Designers

- Promise message models, not object types
- Document link identifiers, not URLs
- Publish vocabularies, not API definitions



@jnd1er



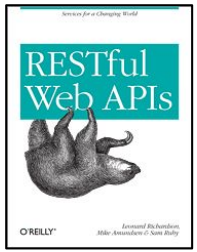


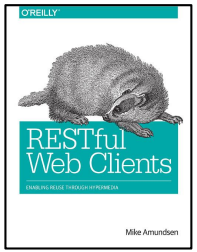
Evolvable Providers

"When people are building on top of our API, we're really asking them to trust us with the time they're investing in building their applications. And to earn that trust, we can't make changes [to the API] that would cause their code to break."

-- Jason Rudolph, Github (2013)

@jasonrudolph





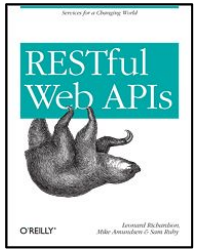
https://www.slideshare.net/yandex/api-design-at-github-jason-rudolph-github

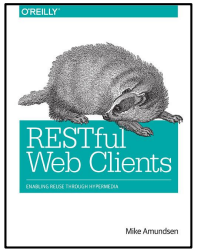
Evolvable Providers

- Don't take things away
- Don't change the meaning of things
- Make all additions optional

@jasonrudolph

API ACADEMY

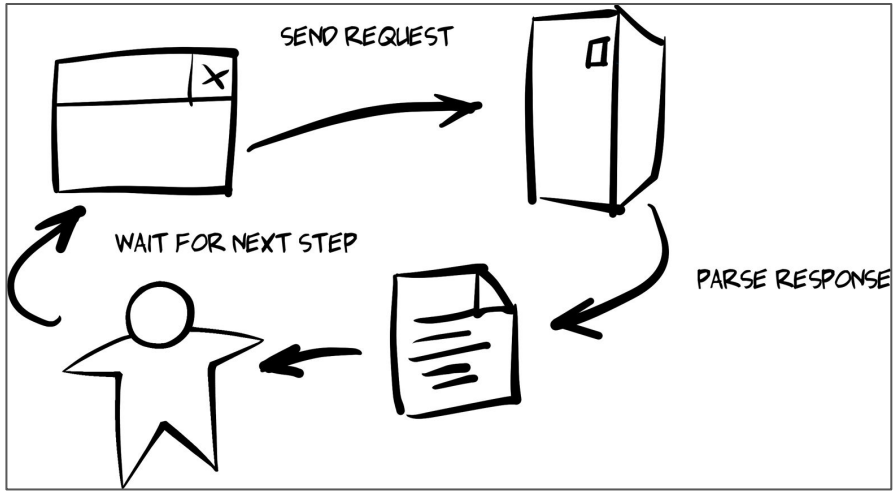




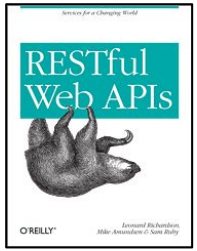
Adaptable Consumers

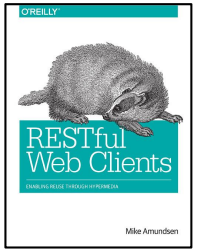
"When you can build a client that doesn't have to memorize the solution ahead of time you can start building clients who are 'smart' enough to adapt to new possible actions as the service presents them."

-- Mike Amundsen, 2016



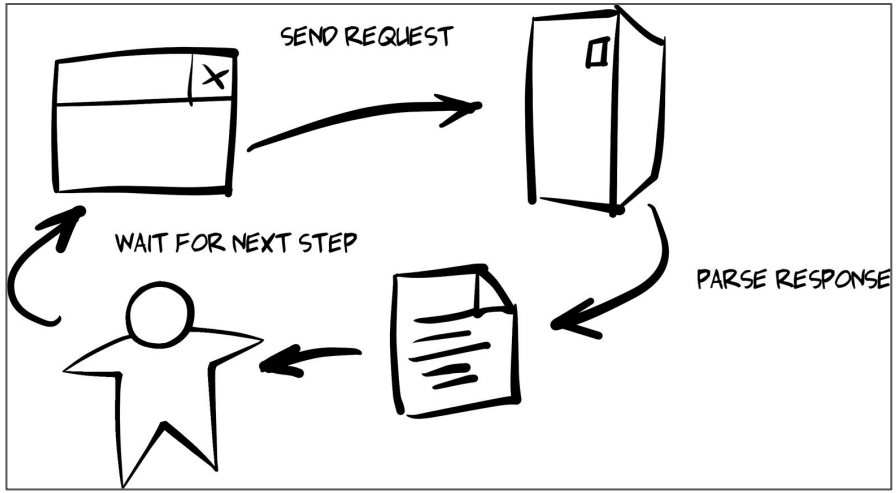
API ACADEMY



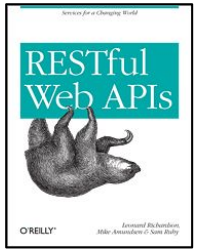


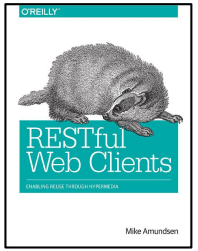
Adaptable Consumers

- Code defensively
- Code to the media type
- Leverage the API vocabulary
- React to link relations for workflow

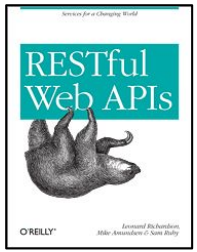


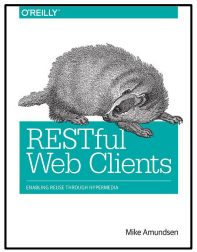
API ACADEMY



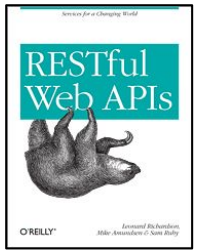


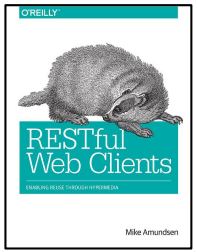
Providers evolve via humans, consumers adapt via code.



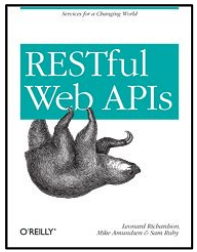


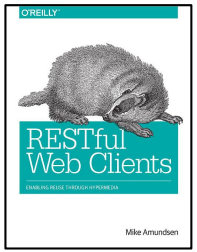
Adaptation Exercise





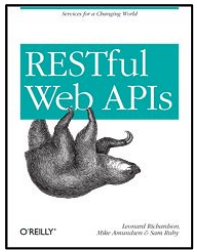
Summary

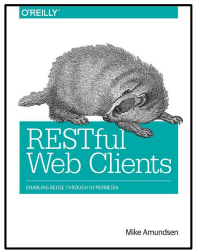

RESTful
Web APIs
O'REILLY


O'REILLY
RESTful
Web Clients
Mike Amundsen

Summary

- A RESTful Design
- Message-Oriented Implementation
- Discovery Constraints
- Emergent Adaptability



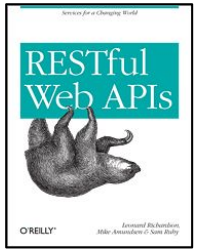


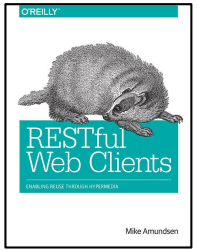
A RESTful Design

- Microservices means independent & loosely-coupled
- REST properties are close to Microservice properties
- Adopt a New Kind of Service Constraints

@Fielding

API ACADEMY

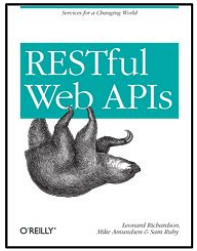


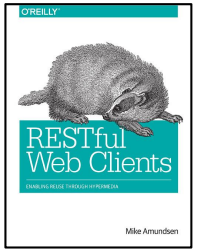


Message-Oriented Implementation

- Models on the Inside
- Messages on the Outside
- Vocabularies Everywhere

@PatHelland

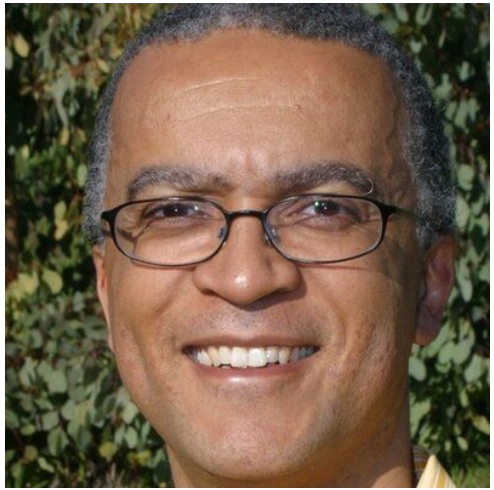




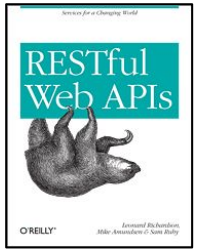
API ACADEMY

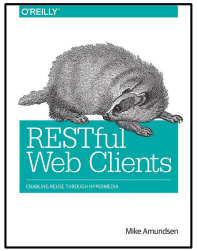
Discovery Constraints

- Advertising Services
- Discovering Services
- Health Checking/Renewals



@CRichardson



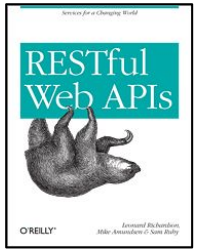


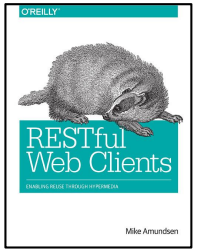
API ACADEMY

Emergent Adaptability

- Designers promise messages
- Services implement non-breaking changes
- Consumers code defensively

@mamund





API ACADEMY

RESTful Microservices
from the Ground Up

Mike Amundsen
API Academy
@mamund