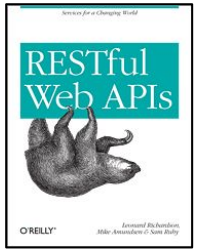
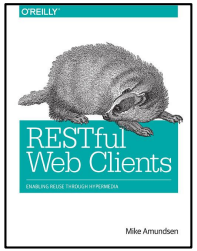
RESTful Microservices
from the Ground Up

2018 SACon London

Mike Amundsen
API Academy
@mamund






Ctrl
PgUp
Mike Amundsen
@mamund

http://g.mamund.com/msabook

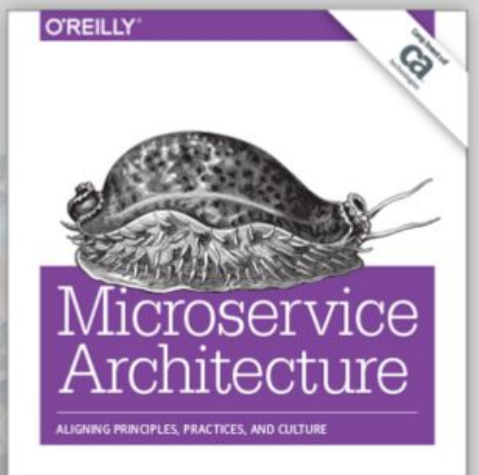
ca technologies

Microservice Architecture: Aligning Principles, Practices, and Culture

Microservices is the next evolution in software architecture designed to help organizations embrace continual change in the digital economy. But how do you design and apply an effective microservice architecture?

This new book from O'Reilly provides comprehensive guidance through seven valuable chapters that give you a deep-dive into:

- The benefits and principles of microservices
- A design-based approach to microservice architecture
- Lessons for applying microservices in practice

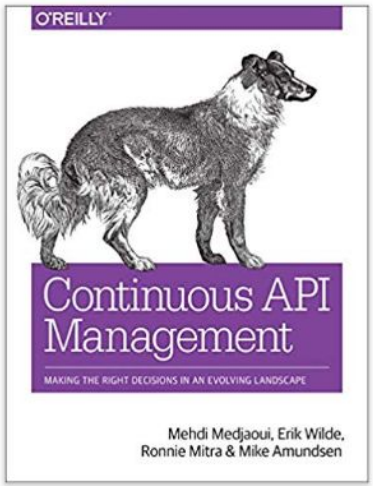


http://g.mamund.com/cambook

Continuous API Management: Making the Right Decisions in an Evolving Landscape

1st Edition

by Mehdi Medjaoui (Author), Erik Wilde (Author), Ronnie Mitra (Author), Mike Amundsen (Author)



ISBN-13: 978-1492043553
ISBN-10: 1492043559
Why is ISBN important?

Paperback $49.99

Other Sellers from $49.99

Pre-order

This title will be released on January 4, 2019.
Ships from and sold by Amazon.com. Gift-wrap available.
prime

prime $49.99

Pre-order Price Guarantee.

1 New from $49.99

Deliver to MICHAEL - Erlanger 41018

Qty: 1

Pre-order: Add to Cart

Pre-order now

Turn on 1-Click ordering

More Buying Choices

1 New from $49.99

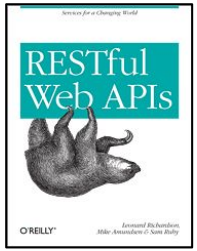
1 New from $49.99

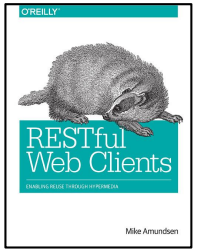
See All Buying Options

Agenda

- What are RESTful Microservices?
- Models, Messages, and Vocabularies
- Three Types of Microservices
- BREAK
- Six Stability Patterns
- Runtime Service Infrastructure
- The Adaptable System
- Summary

API ACADEMY

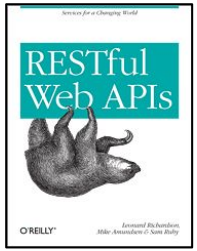


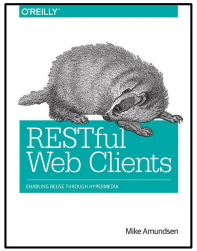


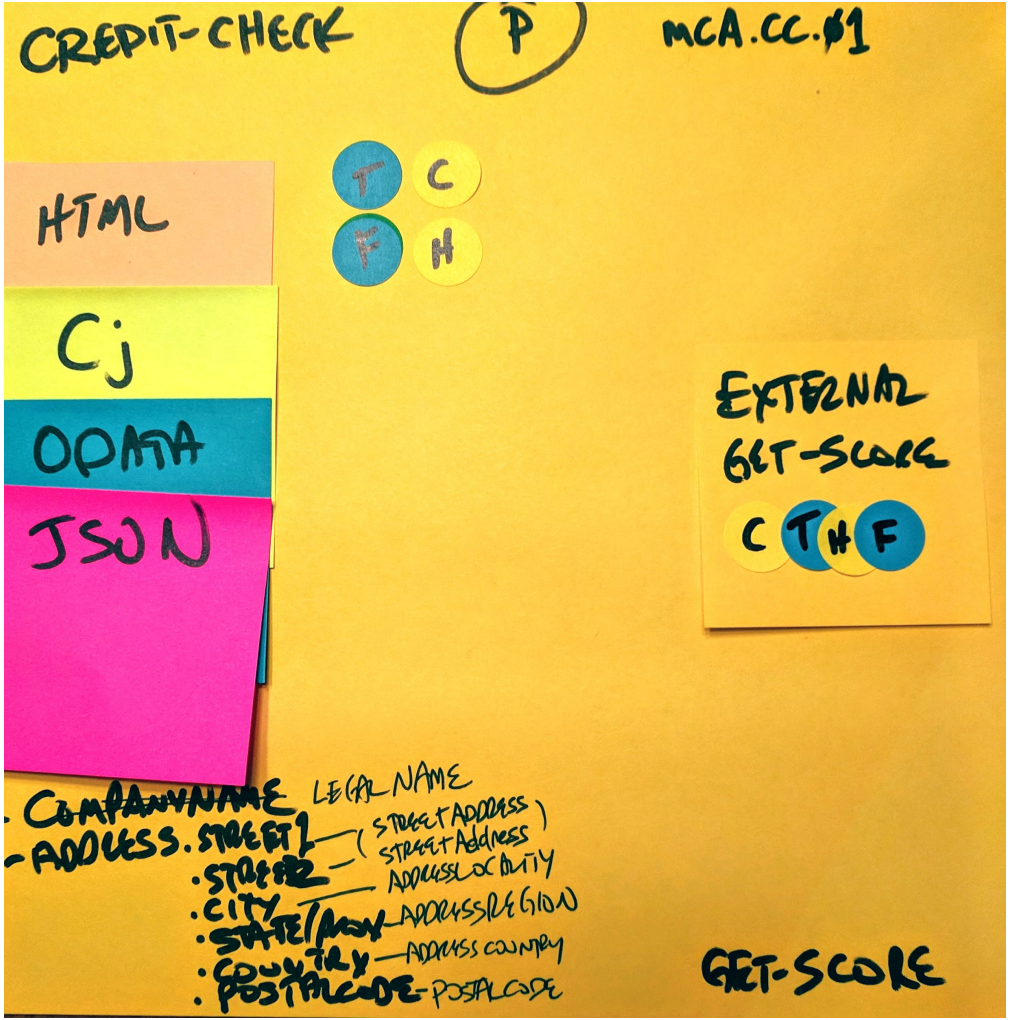
About the Exercises

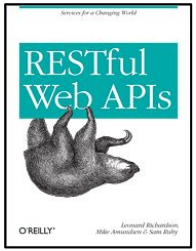
- We'll be working in groups
- All the exercises today are "table top" style
- You'll analyze, design, implement, stabilize, advertise, discover, and update your team's service
- You'll compute your *ServiceCost* score at the end

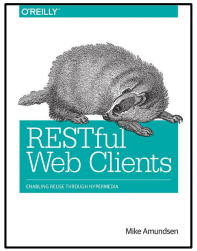
API ACADEMY



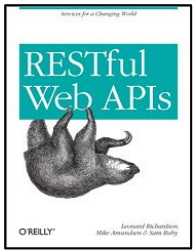


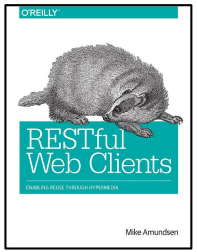

CREDIT-CHECK
P
MCA.CC.#1
HTML
Cj
ODATA
JSON
T
C
F
H
EXTERNAL
GET-SCORE
C T H F
COMPANYNAME
LEGAL NAME
ADDRESS.STREET1
(STREET ADDRESS
STREET ADDRESS)
.STREET2
.CITY
ADDRESSLOCALITY
.STATE/PROV
ADDRESSREGION
.COUNTRY
ADDRESS COUNTRY
.POSTALCODE
POSTALCODE
GET-SCORE


RESTful
Web APIs
O'REILLY


O'REILLY
RESTful
Web Clients
Mike Amundsen

What are RESTful Microservices?

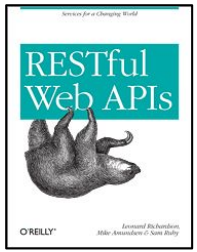


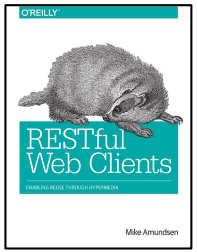


What are RESTful Microservices?

- Microservices
- RESTful-ness
- Microservice Constraints
- *Analysis Exercise*

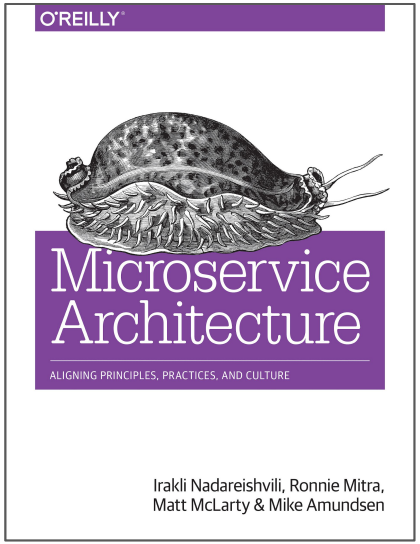
API ACADEMY

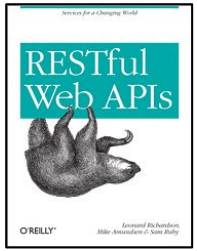


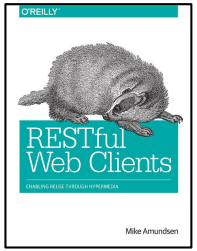


Microservices

"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."





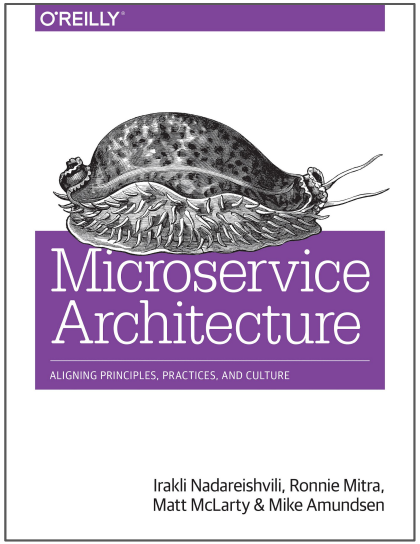


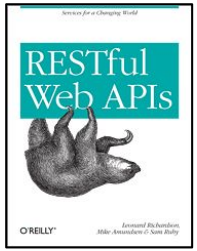
API ACADEMY

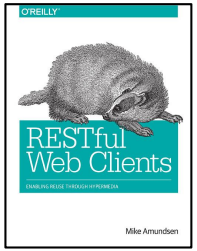
http://g.mamund.com/msabook

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."*





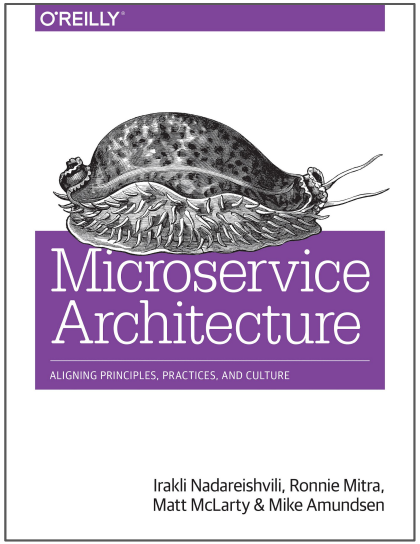


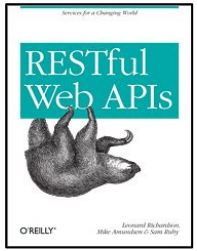
API ACADEMY

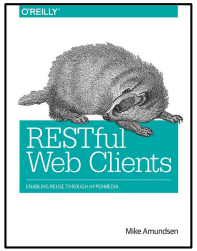
http://g.mamund.com/msabook

Microservices

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







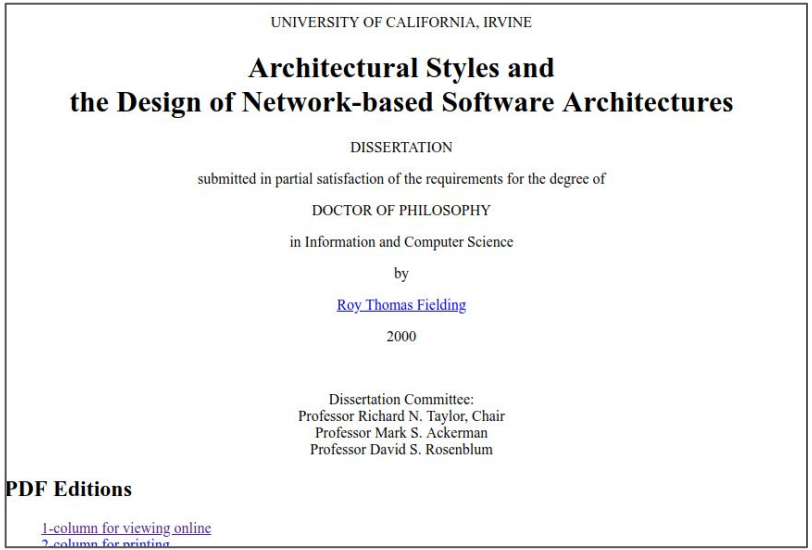
API ACADEMY

http://g.mamund.com/msabook

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

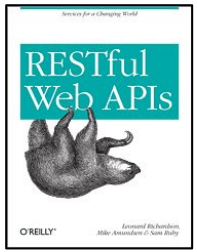
Dissertation Committee:
Professor Richard N. Taylor, Chair
Professor Mark S. Ackerman
Professor David S. Rosenblum

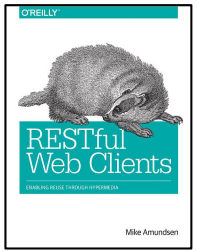
PDF Editions

1-column for viewing online
2-column for printing

API ACADEMY

https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

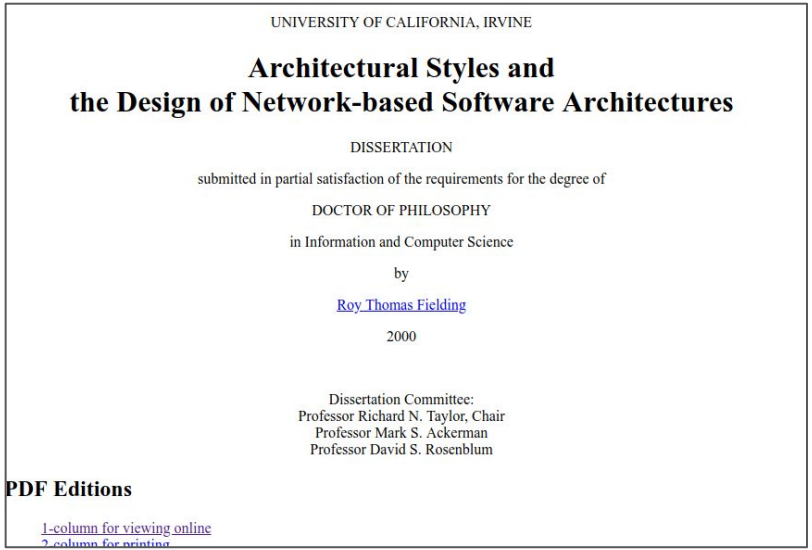




RESTful-ness

"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

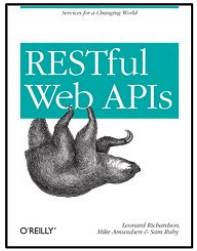
2000

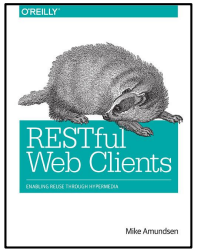
Dissertation Committee:
Professor Richard N. Taylor, Chair
Professor Mark S. Ackerman
Professor David S. Rosenblum

PDF Editions

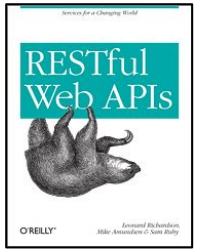
1-column for viewing online
2-column for printing

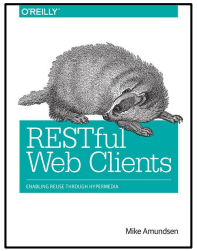
https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm





RESTful-ness



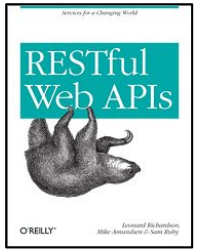


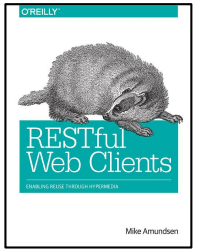
RESTful-ness

Properties

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

API ACADEMY





RESTful-ness

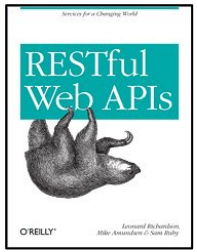
Properties

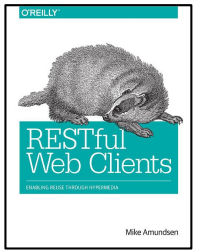
- Performance
- Scalability
- Simplicity
- Modifiability
- Visibility
- Portability
- Reliability

+ Requirements

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

API ACADEMY

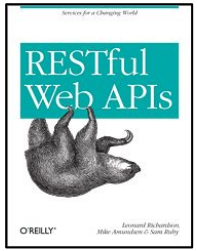


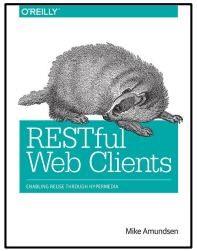


RESTful-ness

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

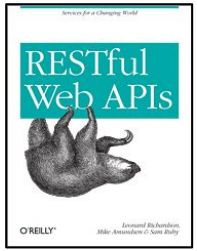


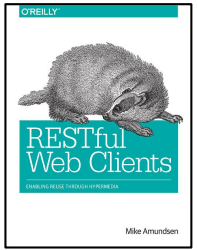


RESTful-ness

"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





http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven

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

Home About Archives Links

Untangled

musings of Roy T. Fielding

Mon 20 Oct 2008

REST APIs must be hypertext-driven

Posted by Roy T. Fielding under software architecture, web architecture

[51] Comments

I am getting frustrated by the number of people calling any HTTP-based interface a REST API. Today's example is the SocialSite REST API. That is RPC. It screams RPC. There is so much coupling on display that it should be given an X rating.

What needs to be done to make the REST architectural style clear on the notion that hypertext is a constraint? In other words, if the engine of application state (and hence the API) is not being driven by hypertext, then it cannot be RESTful and cannot be a REST API. Period. Is there some broken manual somewhere that needs to be fixed?

API designers, please note the following rules before calling your creation a REST API:

- A REST API should not be dependent on any single communication protocol, though its successful mapping to a given protocol may be dependent on the availability of metadata, choice of methods, etc. In general, any protocol element that uses a URI for identification must allow any URI scheme to be used

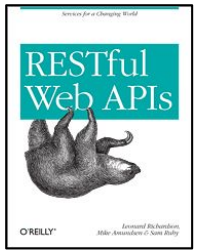
Archived Entry

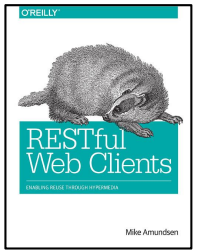
Post Date : Monday, Oct 20th, 2008 at 5:20 am
Category : software architecture and web architecture
Tags : hypertext, REST
Do More : Both comments and pings are currently closed.

Search

Categories

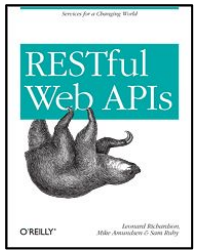
blogging (4)
Family (1)
open source (3)
software architecture (7)
standards (2)
systems engineering (2)
» system dynamics (1)
web architecture (11)

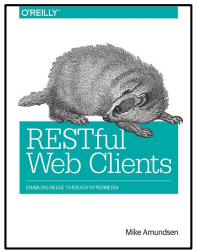




http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven

Fielding's REST ticks many of the boxes for Microservices

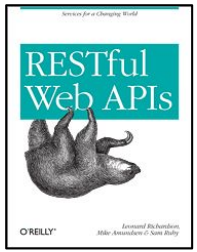


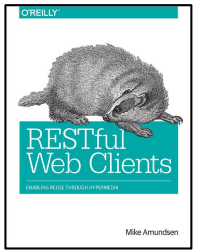


Microservice Constraints

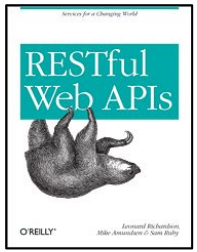
- Manage only service-state, not client state (no persistent sessions)
- Rely on Uniform Interface protocols (HTTP, MQTT, CoAP, etc.)
- Communicate in Structured Formats (HTML, Atom, Cj, HAL, etc.)
- Support Shared Vocabularies (ALPS, DCAP, etc.)
- Support Advertising, Discovery, and Health-Check

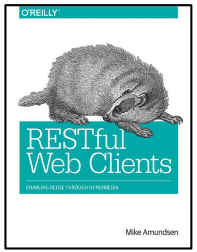
API ACADEMY





Analysis Exercise



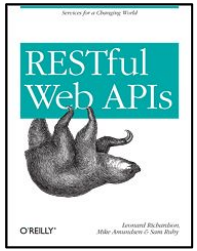


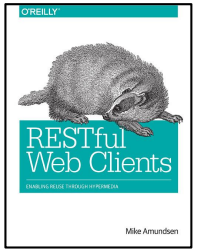
CREDIT-CHECK

MCA.CC.#1

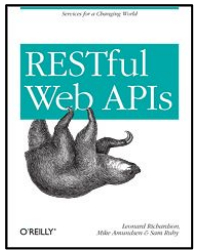
- COMPANYNAME
- ADDRESS.STREET1
  .STREET2
  .CITY
  .STATE/PROV
  .COUNTRY
  .POSTALCODE

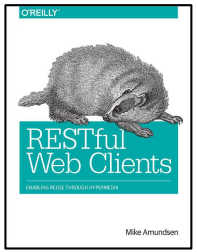
GET-SCORE





Models, Messages, and Vocabularies

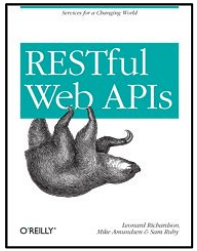


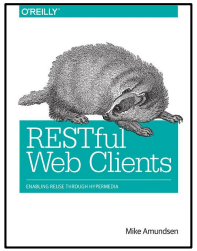


Models, Messages, and Vocabularies

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

API ACADEMY

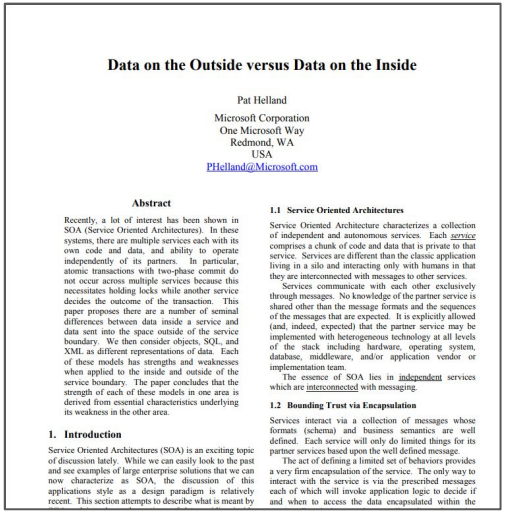




Data on the Inside vs. Data on the Outside

"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.

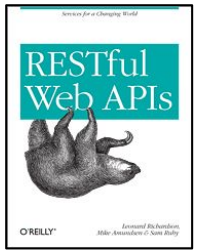
1.2 Bounding Trust via Encapsulation

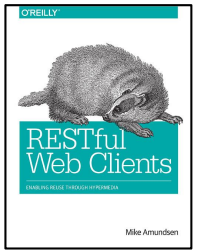
Services interact via a collection of messages whose formats (schema) and business semantics are well defined. Each service will only do limited things for its partner services based upon the well defined message.

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

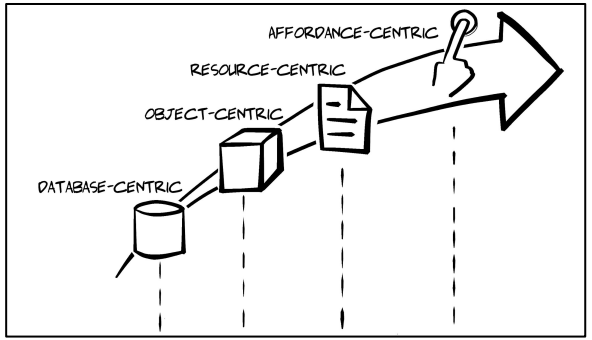
cidrdb.org/cidr2005/papers/P12.pdf





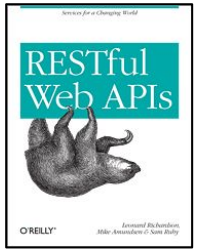
Models on the Inside

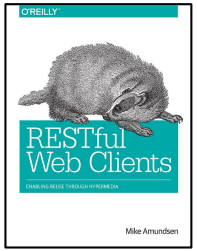
- Inside is immediate, transactional
- Data storage models (**`customers.db, orders.db`**)
- Programming object models (**`objCustomer`**)
- Inside is local, controllable
- Inside relies on a shared "now"



API ACADEMY

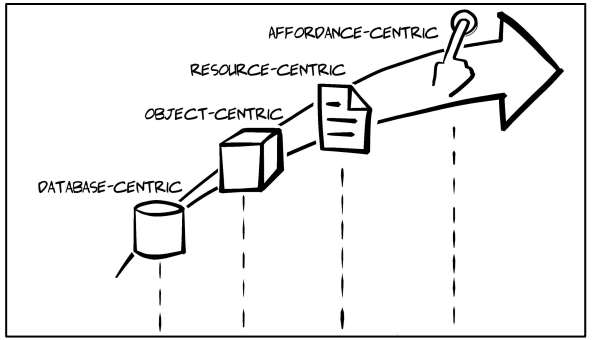
http://amundsen.com/talks/2017-07-chattanooga/





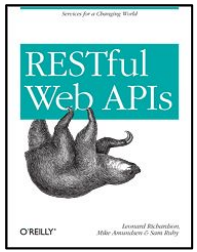
Messages on the Outside

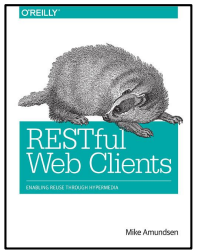
- Outside is always in the past, non-transactional
- Resource models (**`/customers/, /orders/`**)
- Message models (**`customer.html, order.hal`**)
- Outside is remote, uncontrollable
- There is no shared "now"



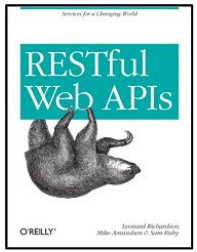
API ACADEMY

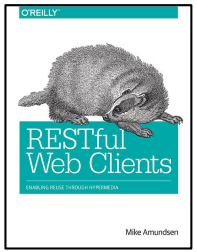
http://amundsen.com/talks/2017-07-chattanooga/





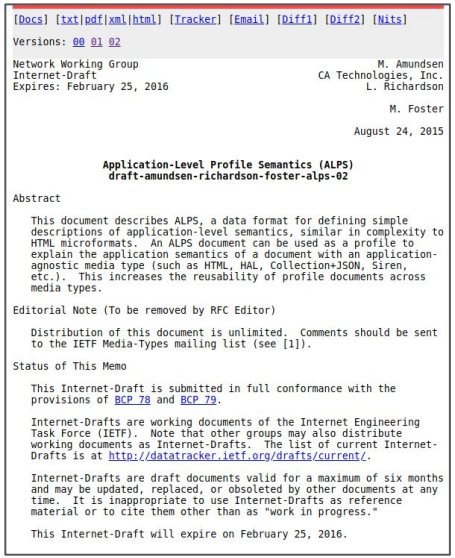
If the models are different inside and out, what is shared?





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 — M. Amundsen
Internet-Draft — CA Technologies, Inc.
Expires: February 25, 2016 — 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]).

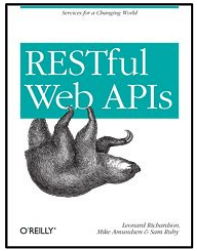
Status of This Memo

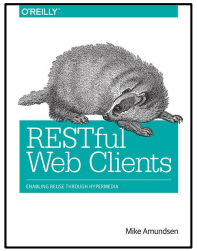
This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

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.

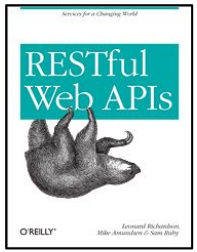


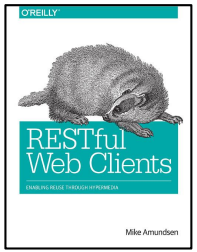


https://tools.ietf.org/html/draft-amundsen-richardson-foster-alps-02

Design Exercise

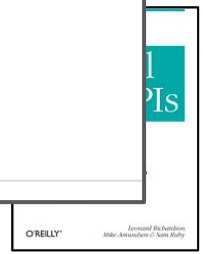
API ACADEMY

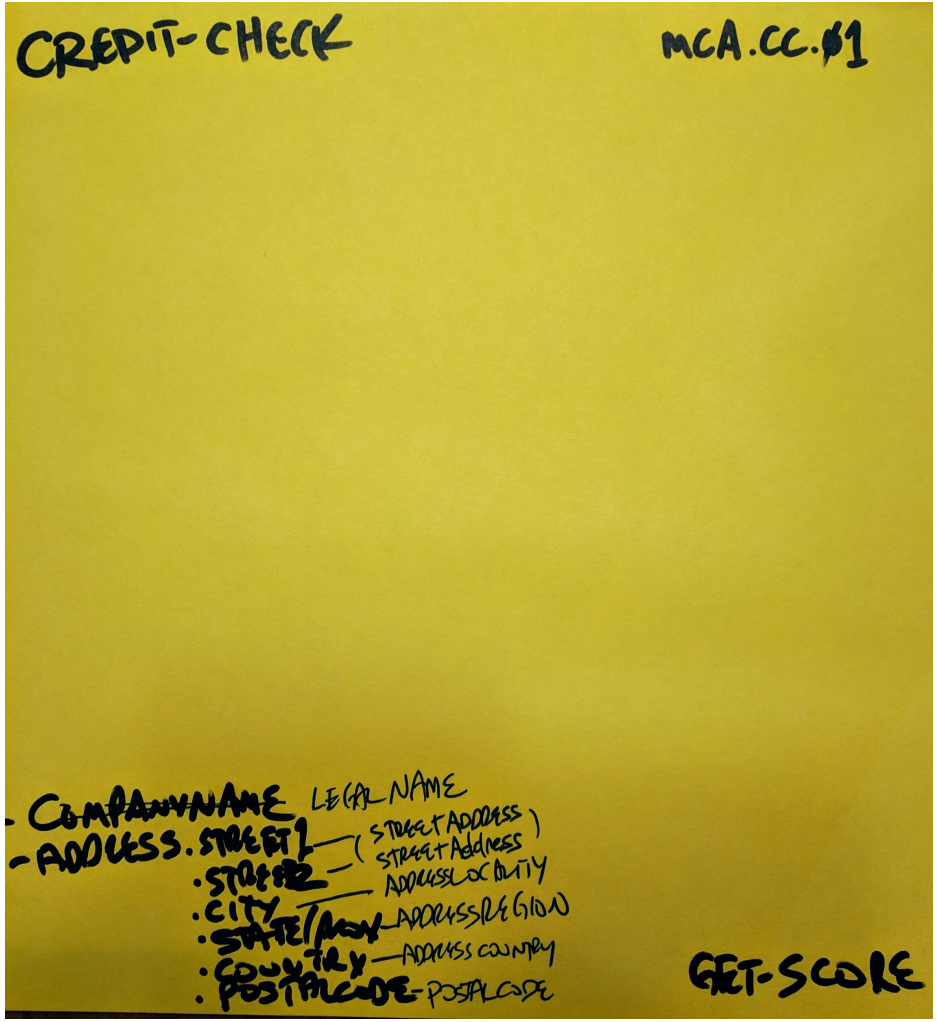


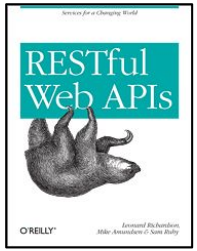


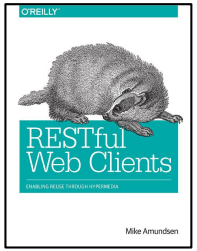
```
<alps version="1.0">
  <doc>Service description for CreditCheck Service for BigCo, Inc.</doc>

  <!-- data points -->
  <descriptor id="legalName" type="semantic" />
  <descriptor id="streetAddress" type="semantic" text="steet" />
  <descriptor id="addressLocality" type="semantic" text="city"/>
  <descriptor id="addresRegion" type="semantic" text="state/province/region" />
  <descriptor id="addressCountry" type="semantic" />
  <descriptor id="postalCode" type="semantic" />
  <descriptor id="score" type="semantic" text="value from 0 to 10"/>
  <!-- actions -->
  <descriptor id="getScore" type="safe" text="call service" returns="score">
    <descriptor href="#legalName" />
    <descriptor href="streeAddress" />
    <descriptor href="addressLocality" />
    <descriptor href="addressRegion" />
    <descriptor href="addressCountry" />
    <descriptor href="postalCode" />
  </descriptor>

</alps>
```

API ACADEMY

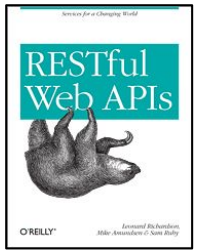


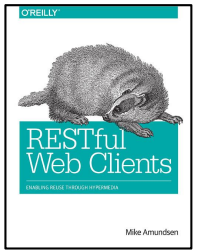

CREDIT-CHECK
MCA.CC.#1
- COMPANYNAME LEGAL_NAME
- ADDRESS.STREET1 (STREET ADDRESS)
·STREET2 STREET ADDRESS
·CITY ADDRESSLOCALITY
·STATE/PROV ADDRESSREGION
·COUNTRY ADDRESS COUNTRY
·POSTALCODE POSTALCODE
GET-SCORE


RESTful
Web APIs
O'REILLY


O'REILLY
RESTful
Web Clients
Mike Amundsen

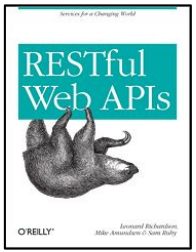
Three Types of Microservices





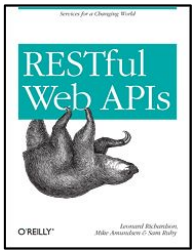
Three Types of Microservices

- Stateless
- Persistence
- Aggregator



Stateless


API ACADEMY

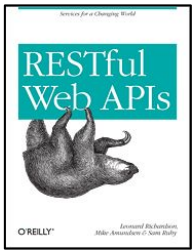

RESTful
Web APIs
O'REILLY


O'REILLY
RESTful
Web Clients
Mike Amundsen

Stateless Microservices

- Simple processors (converters, translators, etc.)
- No dependence on other microservices
- No local data storage (disk I/O)

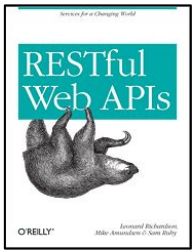
The most common MSC example, but the least useful!



Stateless Microservices

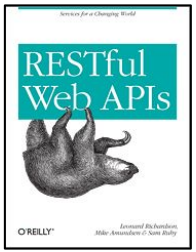
```
// http server handling data conversions
function conversionServer(request, response) {
  response = convertValue(request);
  return response;
}
```

WARNING: NOT REAL CODE!

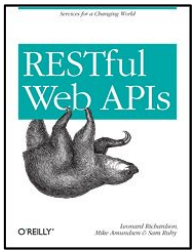


Stateless Microservices

- No shared state
- Easy to replace
- Easy to scale up



Persistence

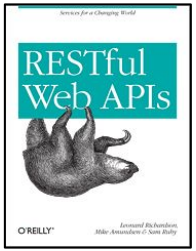


Persistence Microservices

- Simple (local) storage (reads and/or writes)
- Disk I/O dependent
- Possibly VM, one-U, dependent

Commonly needed MSC, not the easiest to implement.

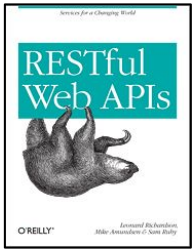
API ACADEMY



Persistence Microservices

```
function updateOrders(request, response) {
  response = localStorage.write(request);
  return response;
}
```

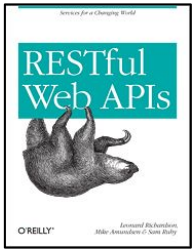
WARNING: NOT REAL CODE!



Persistence Microservices

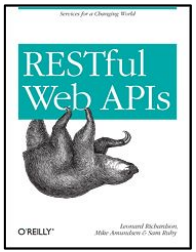
- System of Record/Source of Truth
- Relatively easy to scale for reads (CQRS)
- No cross-service two-phase commits (Saga)
- See Aggregator...

API ACADEMY



Aggregator


API ACADEMY

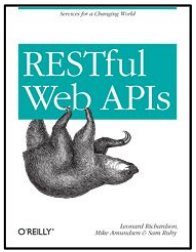

RESTful
Web APIs
O'REILLY


O'REILLY
RESTful
Web Clients
Mike Amundsen

Aggregator Microservices

- Depends on other ("distant") microservices
- Network dependent
- Usually Disk I/O dependence, too

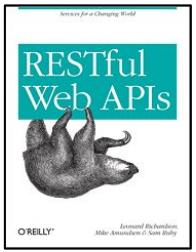
The most often-needed; most challenging, too.



Aggregator Microservices

```
function writeOrders(request, response) {
  var resourceList = ["customerDB", "orderDB", "salesDB"]'
  var serviceList = gatherResources(resourceList);
  response = serviceList(request)

  return response;
}
```

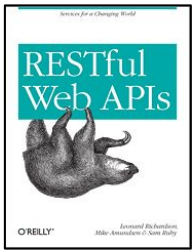
WARNING: NOT REAL CODE!



Aggregator Microservices

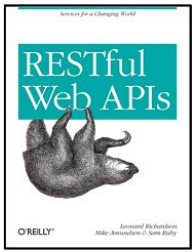
- Sequence vs. Parallel calls
- Timing is everything
- Easy to scale (should be…)

API ACADEMY



So...

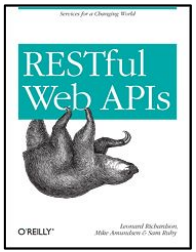

API ACADEMY


RESTful
Web APIs
O'REILLY

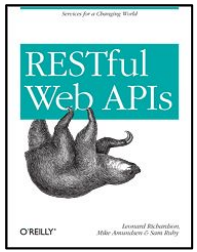

O'REILLY
RESTful
Web Clients
Mike Amundsen

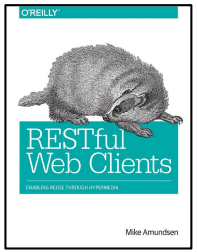
Three Types of Microservices

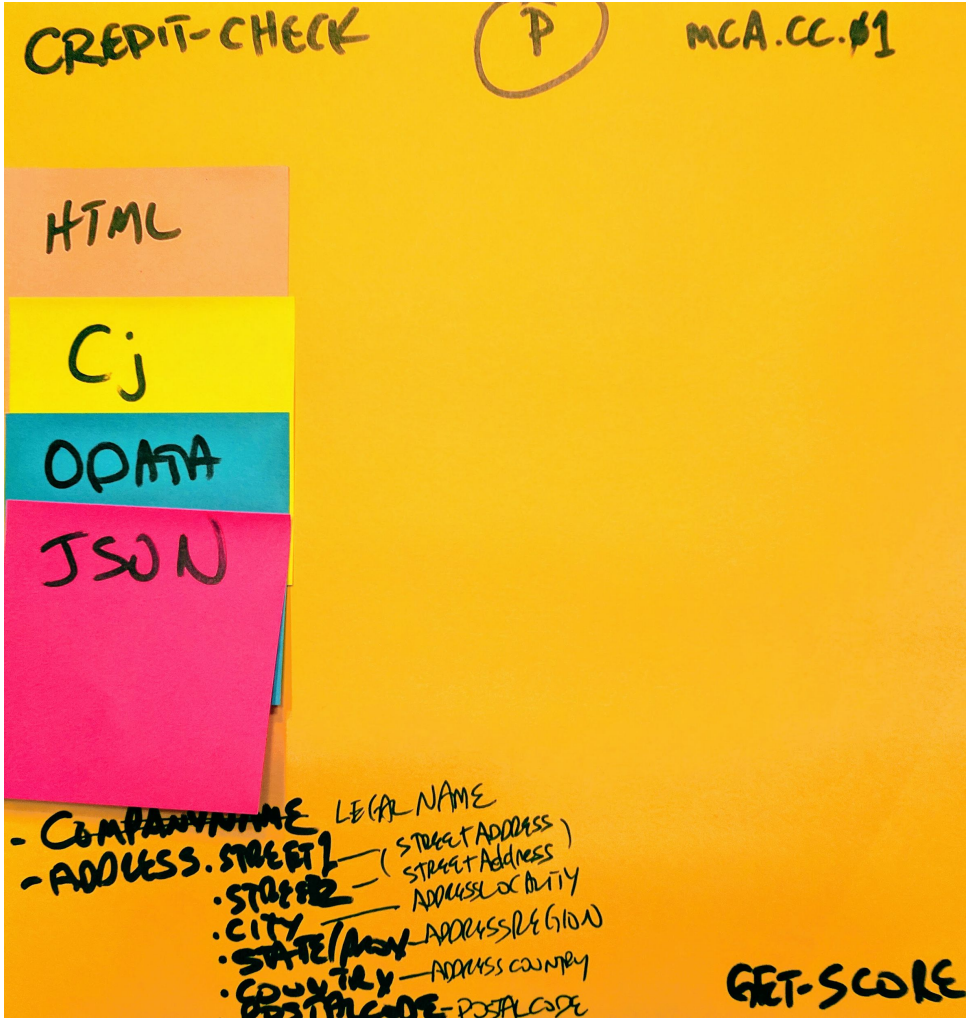
- Stateless for computing
- Persistence for storage
- Aggregator for solutions

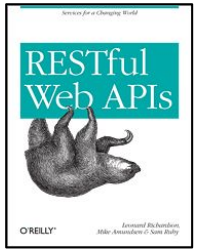


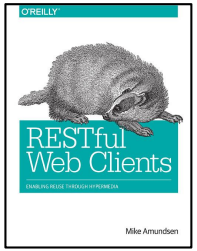
Implementation Exercise





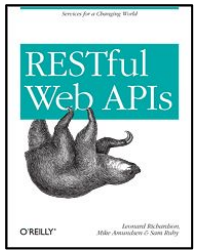

CREDIT-CHECK
P
MCA.CC.#1
HTML
Cj
ODATA
JSON
- COMPANYNAME
LEGAL NAME
- ADDRESS.STREET1
(STREET ADDRESS)
STREET ADDRESS
.STREET2
ADDRESSLOCALITY
.CITY
.STATE/PROV
ADDRESSREGION
ADDRESS COUNTRY
.POSTALCODE
GET-SCORE

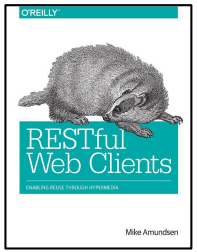

RESTful
Web APIs
O'REILLY


O'REILLY
RESTful
Web Clients
Mike Amundsen

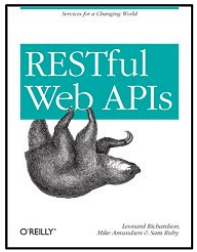
API ACADEMY

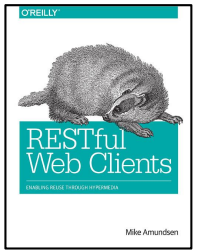
BREAK


RESTful
Web APIs
O'REILLY


O'REILLY
RESTful
Web Clients
Mike Amundsen

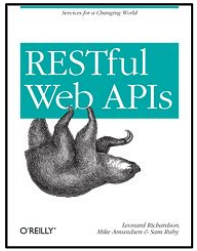
Six Stability Patterns

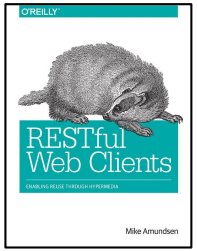





Release It!
Second Edition
Design and Deploy
Production-Ready Software
Michael T. Nygard
BETA

API ACADEMY

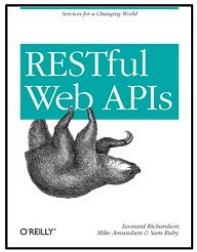

RESTful
Web APIs
O'REILLY

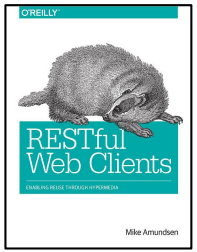

O'REILLY
RESTful
Web Clients
Mike Amundsen

“Bugs will happen. They cannot be eliminated, so they must be survived instead.”

-- Michael T. Nygard

API ACADEMY





Nygard Stability Patterns

- **Timeout**
- **Circuit Breaker**
- **Bulkhead**
- **Steady State**
- **Fail Fast**
- **Handshaking**

- Caching : A capacity pattern referenced here, too

"Nygard Stability Patterns" -- **Timeout**

"The timeout is a simple mechanism allowing you to stop waiting for an answer once you think it will not come."

-- Chapter 5.1

"Nygard Stability Patterns" -- **Timeout**

"The timeout is a simple mechanism allowing you to stop waiting for an answer once you think it will not come."

-- Ch 5.1

```
// set up proper shutdown
process.on('SIGTERM', function () {
  discovery.unregister(null, function(response) {
    try {
      uuidGenerator.close(function() {
      console.log('gracefully shutting down');
        process.exit(0);
      });
    } catch(e){}
  });
  setTimeout(function() {
    console.error('forcefully shutting down');
    process.exit(1);
  }, 10000);
});
```

WARNING: NOT REAL CODE!

"Nygard Stability Patterns" -- **Circuit Breaker**

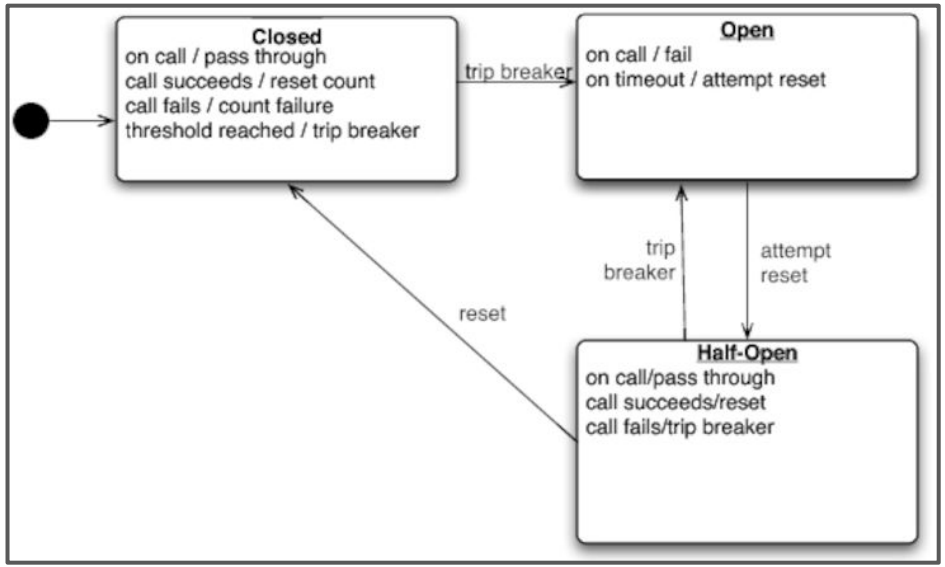
"Circuit breakers are a way to automatically degrade functionality when the system is under stress."

-- Chapter 5.2

"Nygard Stability Patterns" -- **Circuit Breaker**

"Circuit breakers are a way to automatically degrade functionality when the system is under stress."

-- Chapter 5.2



"Nygard Stability Patterns" -- **Bulkhead**

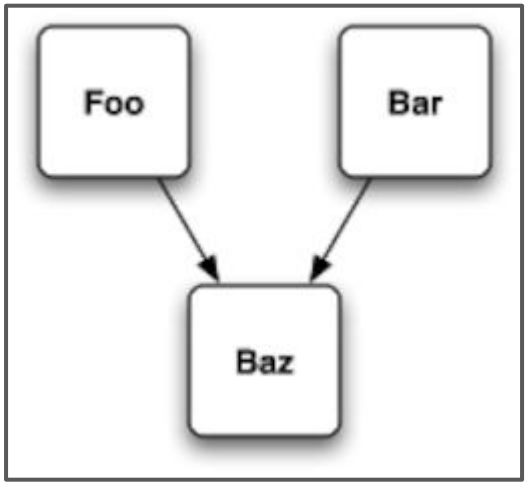
"The bulkhead enforces a principle of damage containment."

-- Chapter 5.3

"Nygard Stability Patterns" -- **Bulkhead**

"The bulkhead enforces a principle of damage containment."

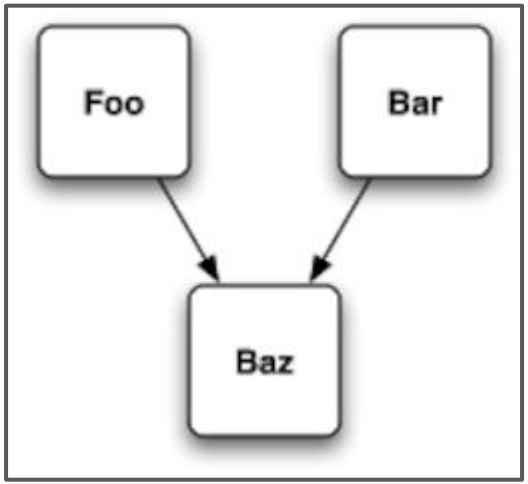
-- Chapter 5.3

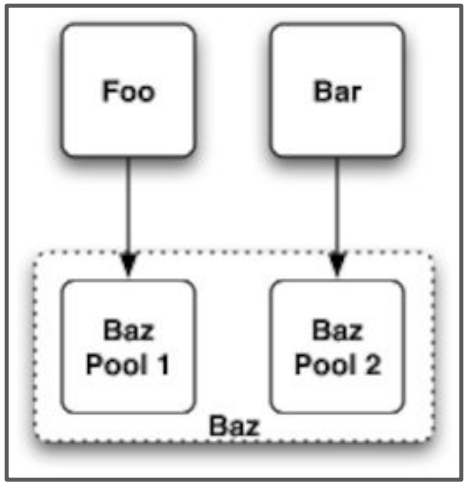


"Nygard Stability Patterns" -- **Bulkhead**

"The bulkhead enforces a principle of damage containment."

-- Chapter 5.3





"Nygard Stability Patterns" -- **Steady State**

"The system should be able to run indefinitely without human intervention."

-- Chapter 5.4

- Avoid fiddling
- Purge data w/ app logic
- Limit caching
- Roll the logs

"Nygard Stability Patterns" -- **Steady State**

"The system should be able to run indefinitely without human intervention."

-- Chapter 5.4

- Avoid fiddling
- Purge data w/ app logic
- Limit caching
- Roll the logs

"Nygard Stability Patterns" -- **Fail Fast**

"If the system can determine in advance that it will fail at an operation, it’s always better to fail fast."

-- Chapter 5.5

"Nygard Stability Patterns" -- **Fail Fast**

"If the system can determine in advance that it will fail at an operation, it's always better to fail fast."

-- Chapter 5.5

```
function bookOrders(orderList, timeBudget) {
  var status = false;
  var resources = ["customerdata","orderdata","salesdata"];
  setTimeout(function(resources) {
    var status = confirmResourceAvailability(resources);
    if(status===true && timeBudget>500) {
      try {
        status = writeOrders(orderList,resources);
      }
      catch (ex) {
        error("failed to write orders : {errordcode}",ex);
      }
    }
    else {
      error("failed to acquire resources : FAILFAST");
    }
  }, timeBudget);
}
```

WARNING: NOT REAL CODE!

"Nygard Stability Patterns" -- **Handshaking**

"Handshaking is all about letting the server protect itself by throttling its own workload."

-- Chapter 5.6

"Nygard Stability Patterns" -- **Handshaking**

"Handshaking is all about letting the server protect itself by throttling its own workload."

-- Chapter 5.6

```
function sendOrders(orderList, timeBudget) {
  if(
      (health.responseMS+health.latencyMS) < timeBudget
    ) {
      bookOrders.send(orderList,timeBudget);
    }
    else {
      error("failed to send orders: HEALTHCHECK");
    }
  }
```

WARNING: NOT REAL CODE!

"Nygard Stability Patterns" -- **Cache**

"Caching can reduce the load on the server and cut response times to a fraction of what they would be without caching."

-- Chapter 10.2

"Nygard Stability Patterns" -- **Cache**

"Caching can reduce the load on the server and cut response times to a fraction of what they would be without caching."

-- Chapter 10.2

```
// server marks data cache-able
function sendResults(response) {
  response.writeHead(status,
    { 'Content-Type' : 'text/plain',
      'Cache-Control': 'public,max-age=108000'}
  );
  response.end(value+'\n');
}
```

WARNING: NOT REAL CODE!

"Nygard Stability Patterns" -- **Cache**

"Caching can reduce the load on the server and cut response times to a fraction of what they would be without caching."

-- Chapter 10.2

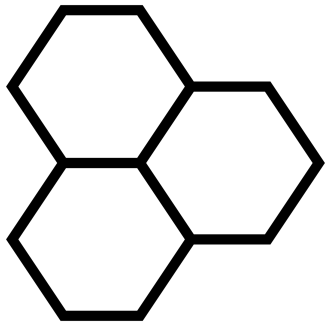
```
// server marks data cache-able
function sendResults(response) {
  response.writeHead(status,
    { 'Content-Type' : 'text/plain',
      'Cache-Control': 'public,max-ag
  );
  response.end(value+'\n');
}
```

```
// client manages local cache
function getData(URL) {
  data = null;
  data = cache.read(URL);
  if(!data) {
    data = requestResults(URL);
    cache.write(URL,data);
  }
  return data;
}
```

WARNING: NOT REAL CODE!

Three Types of Microservice Components

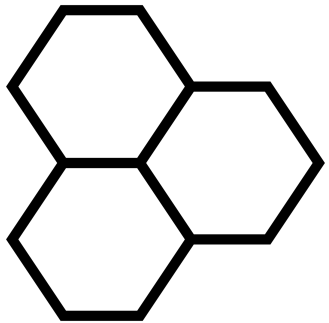
- Stateless (compute)
- Persistence (storage)
- Aggregation (choreography)



Stateless Microservice

- Simple processors (converters, translators, etc.)
- No dependence on other microservices
- No local data storage (disk I/O)

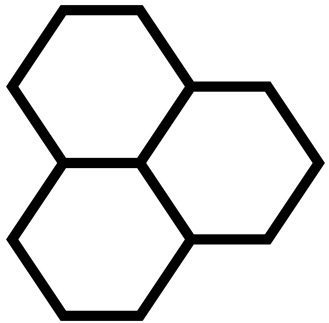
- *Caching*
- Fail Fast



Persistence Microservice

- Simple (local) storage (reads and/or writes)
- Disk I/O dependent
- Possibly VM, one-U, dependent

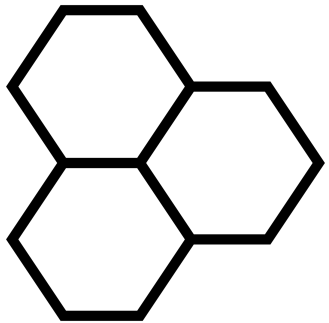
- *Caching*
- Fail Fast
- Timeout
- Circuit Breaker
- Steady State



Aggregation Microservice

- Depends on other ("distant") microservices
- Network dependent
- Usually Disk I/O dependence, too

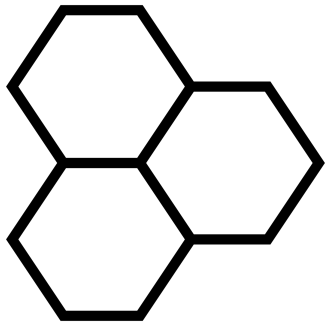
- *Caching*
- Fail Fast
- Timeout
- Circuit Breaker
- Steady State
- Handshaking
- Bulkhead



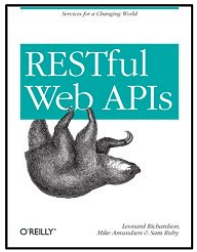
Three Types of Microservice Components

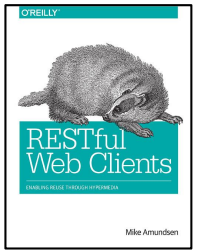
- Stateless
- Persistence
- Aggregation

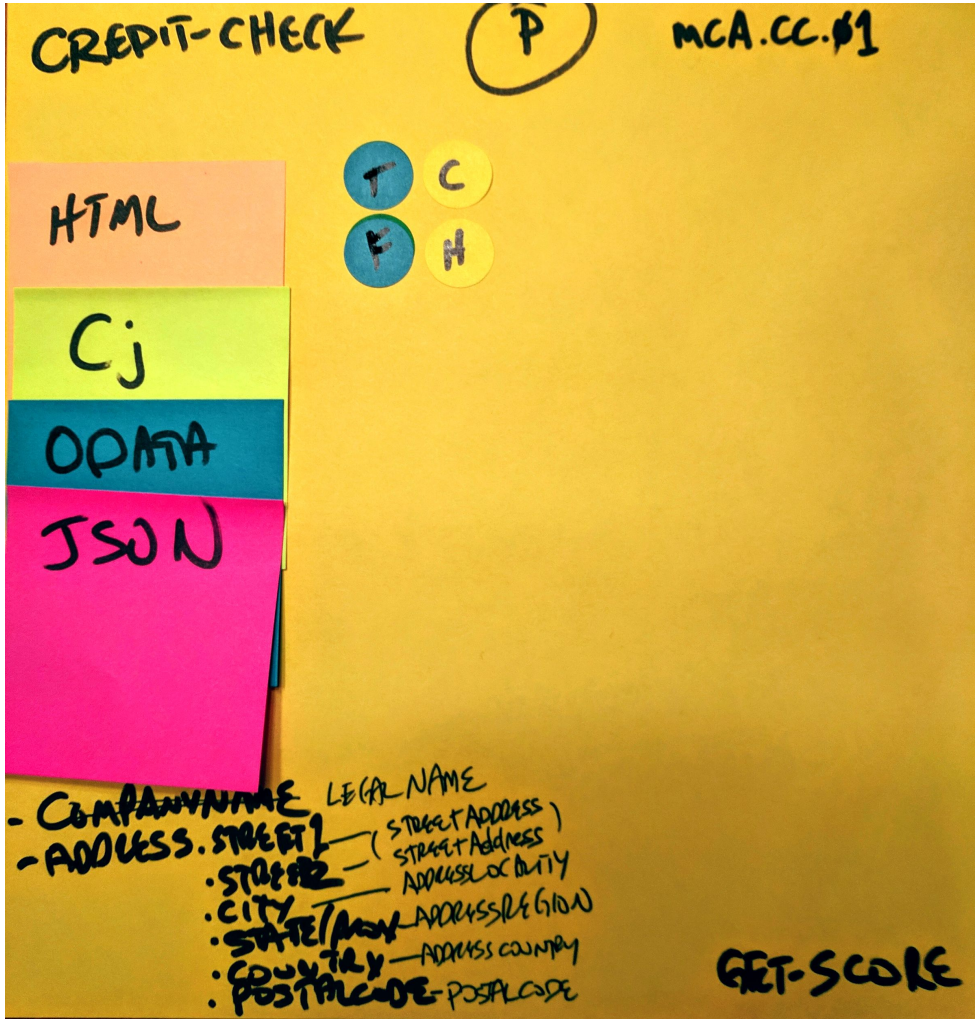
Apply Nygaard's Stability Patterns to improve the health of your components and your system.



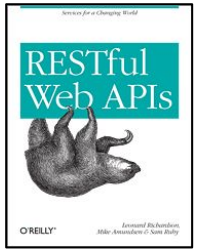
Stability Exercise

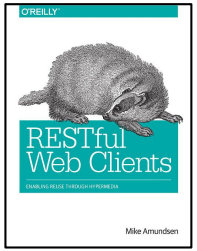




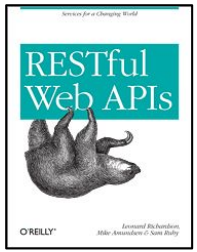

CREDIT-CHECK
P
MCA.CC.#1
HTML
Cj
ODATA
JSON
T
C
F
H
- COMPANYNAME LEGAL NAME
- ADDRESS. STREET1 (STREET ADDRESS)
STREET ADDRESS
.STREET2
ADDRESSLOCALITY
.CITY
.STATE/PROV ADDRESSREGION
.COUNTRY ADDRESS COUNTRY
.POSTALCODE POSTALCODE
GET-SCORE

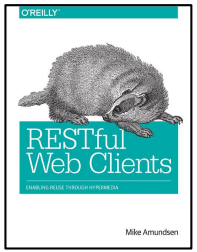
API ACADEMY


RESTful
Web APIs
O'REILLY


O'REILLY
RESTful
Web Clients
Mike Amundsen

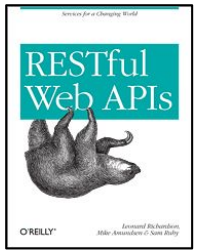
Runtime Service Infrastructure

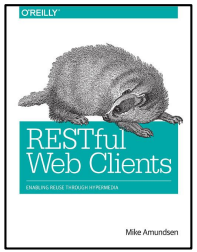




Runtime Service Infrastructure

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

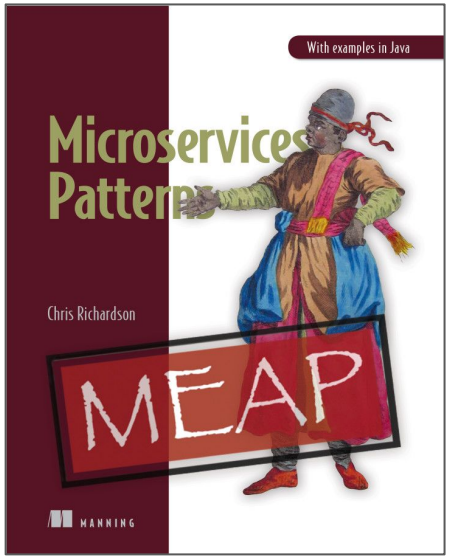


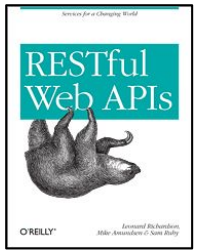


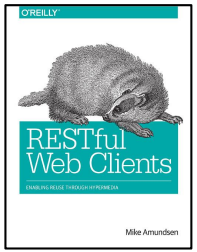
Advertising Services

"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







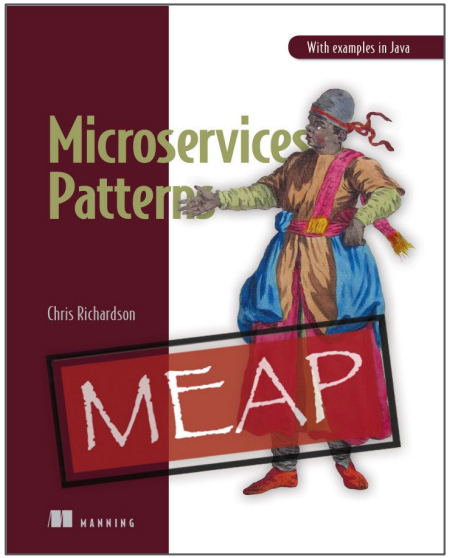
API ACADEMY

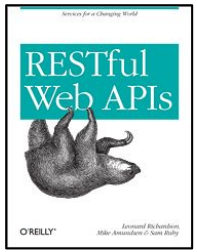
http://microservices.io/patterns/self-registration.html

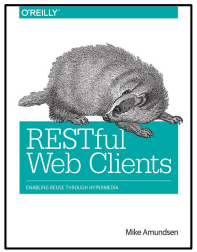
Advertising Services

"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





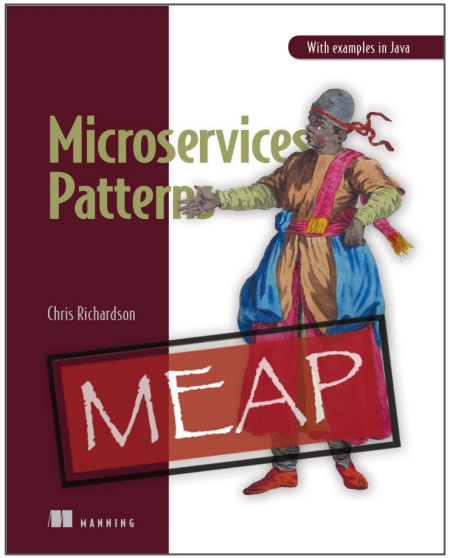


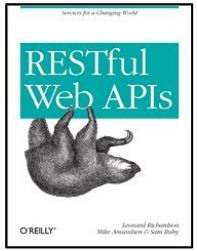
API ACADEMY

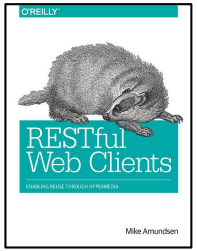
http://microservices.io/patterns/self-registration.html

Advertising Services

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







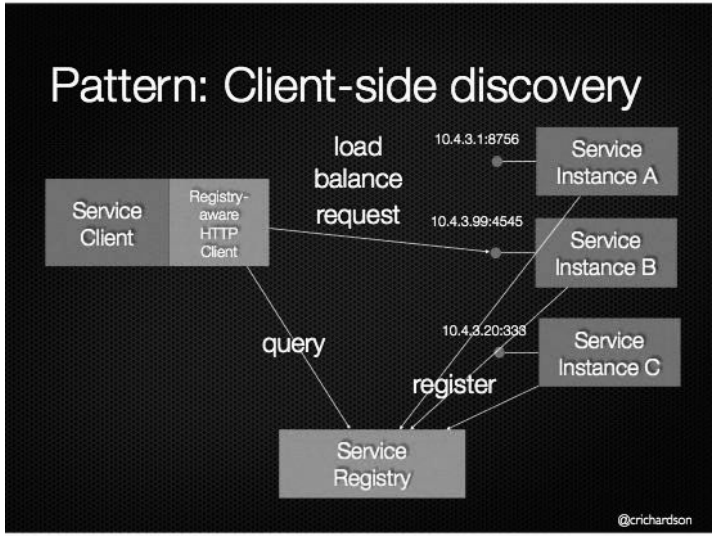
API ACADEMY

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

Discovering Services

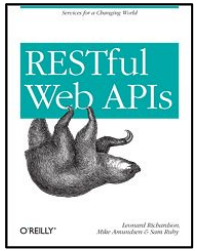
"When making a request to a service, the client obtains the location of a service instance by querying a Service Registry, which knows the locations of all service instances."

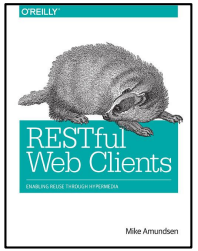
-- microservices.io



API ACADEMY

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

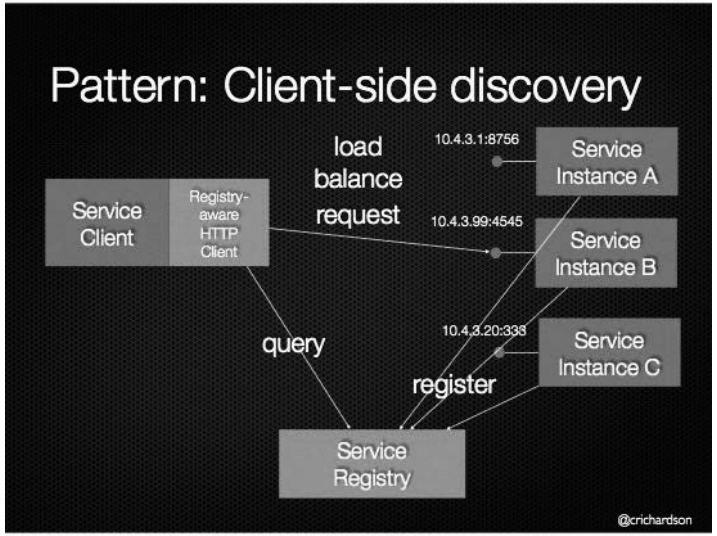




Discovering Services

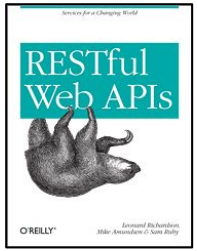
*"When making a request to a service, the client obtains the location of a service instance by **querying a Service Registry**, which knows the locations of all service instances."*

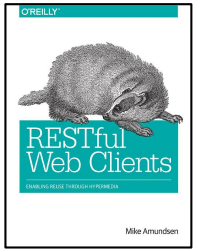
-- microservices.io



API ACADEMY

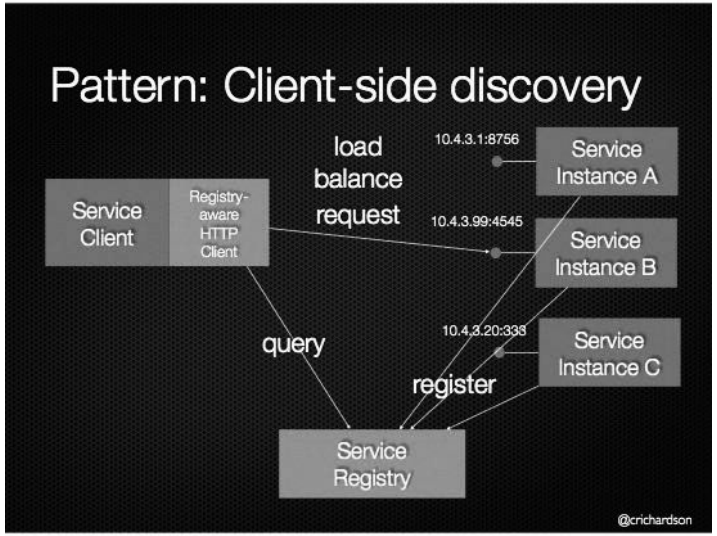
http://microservices.io/patterns/client-side-discovery.html

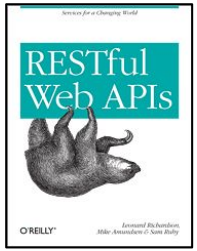


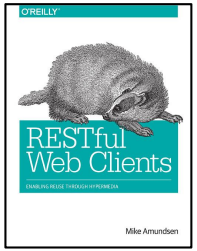


Discovering Services

- Configure client w/ **`registryURL`**
- Query Registry w/ **`serviceURI`**
- Registry returns **`serviceURL`**
- Client uses **`serviceURL`**
- Renewal optional





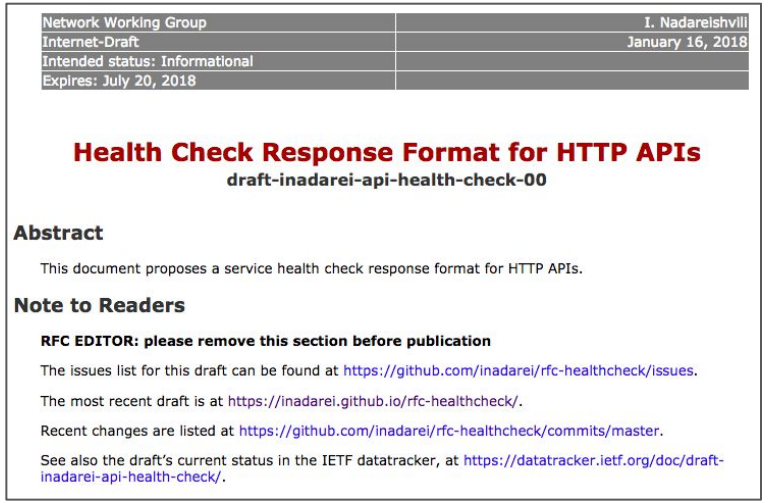


http://microservices.io/patterns/client-side-discovery.html

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.

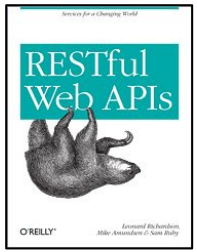
The most recent draft is at https://inadarei.github.io/rfc-healthcheck/.

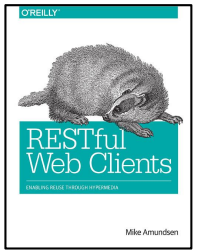
Recent changes are listed at https://github.com/inadarei/rfc-healthcheck/commits/master.

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/

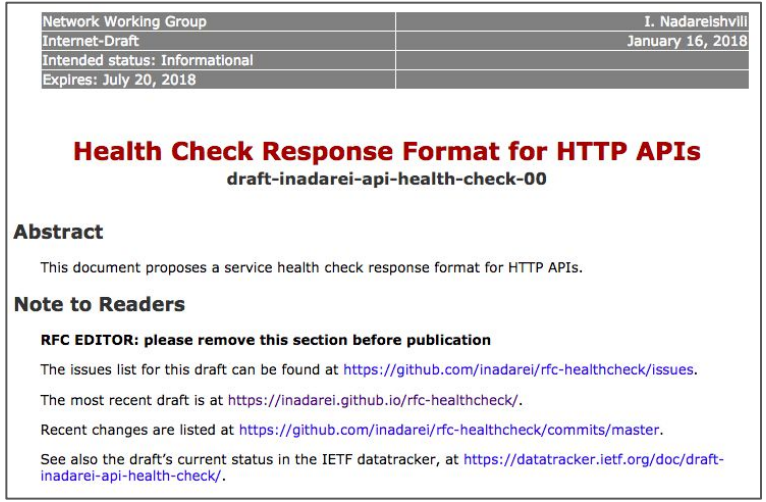




Health Checking

*"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.

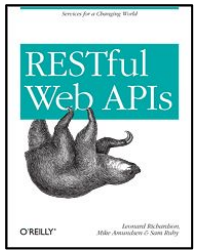
The most recent draft is at https://inadarei.github.io/rfc-healthcheck/.

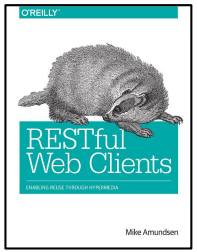
Recent changes are listed at https://github.com/inadarei/rfc-healthcheck/commits/master.

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

API ACADEMY

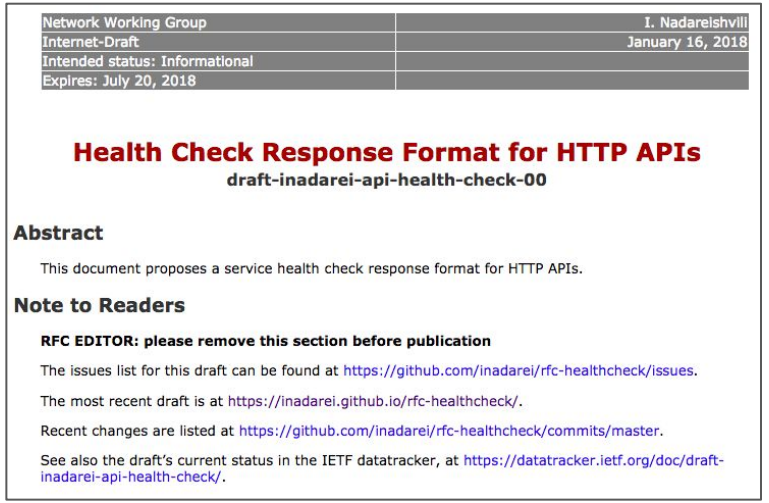
https://inadarei.github.io/rfc-healthcheck/





Health Checking

- Services support health-checks
- Services renew with the registry
- Registry drops service on failed checks
- Registry drops 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.

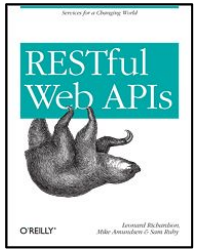
The most recent draft is at https://inadarei.github.io/rfc-healthcheck/.

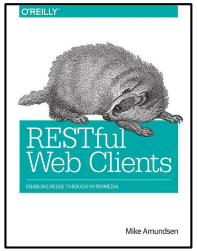
Recent changes are listed at https://github.com/inadarei/rfc-healthcheck/commits/master.

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

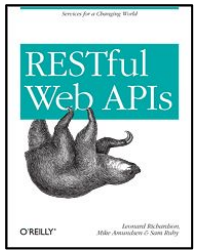
API ACADEMY

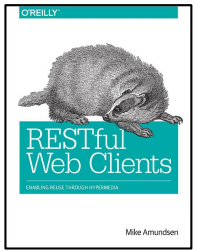
https://inadarei.github.io/rfc-healthcheck/



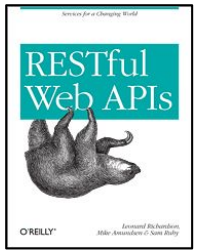


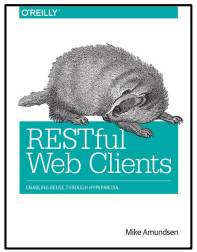
Discovery patterns are the DNS of application services.

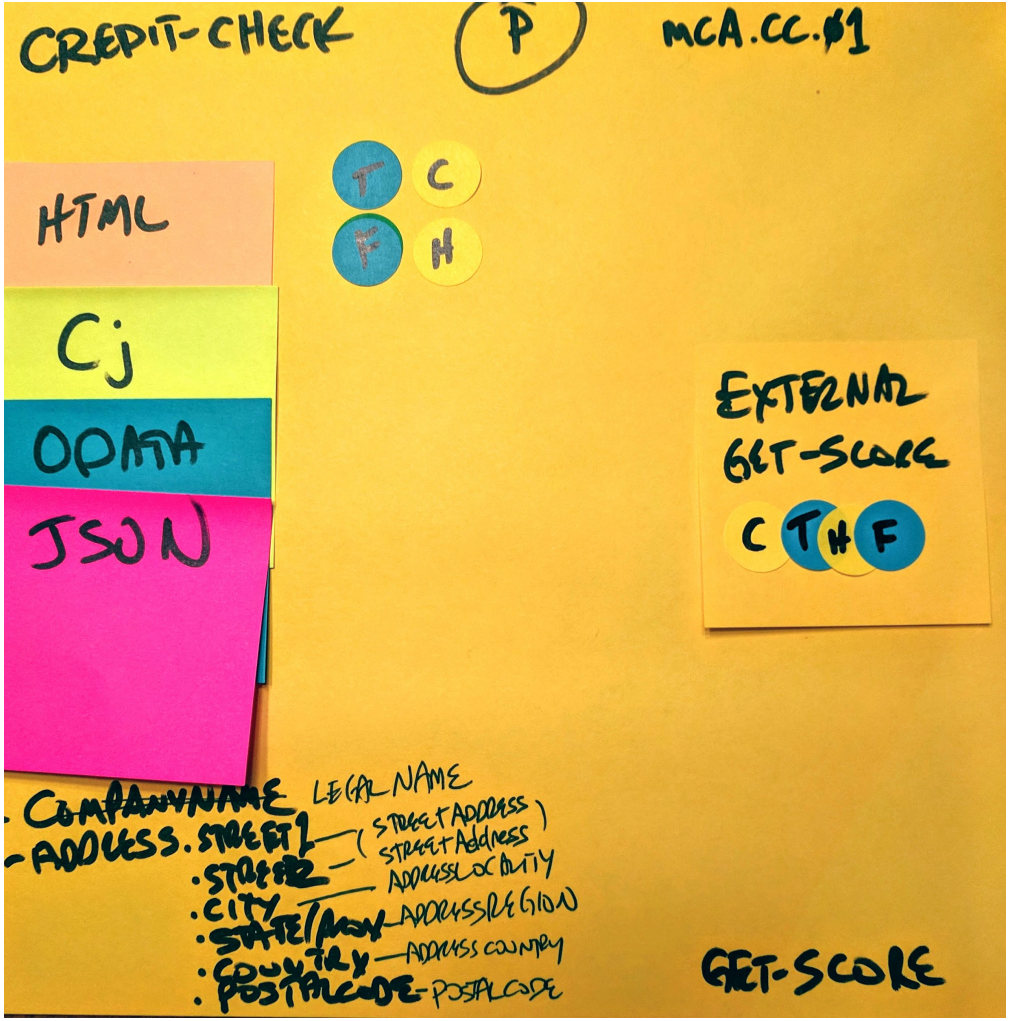


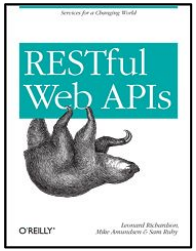


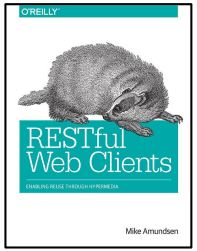
Discovery Exercise

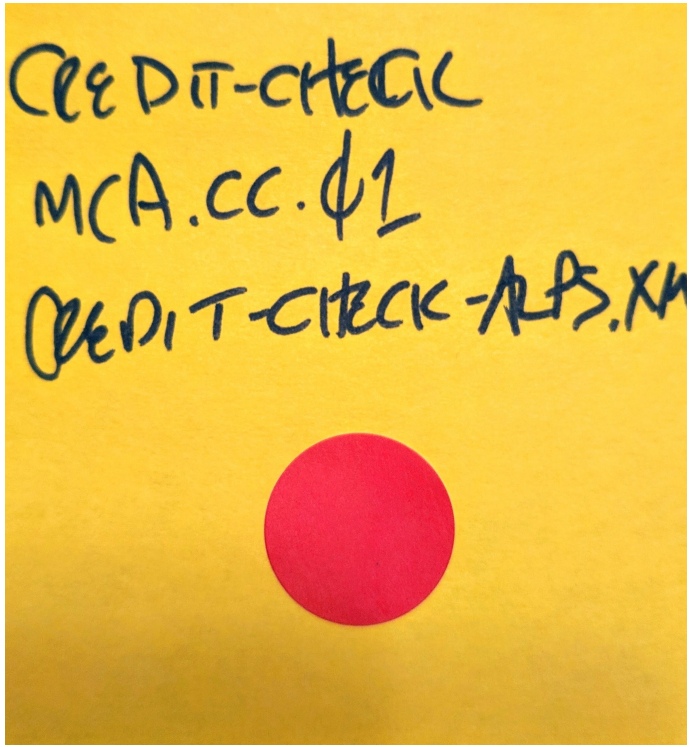




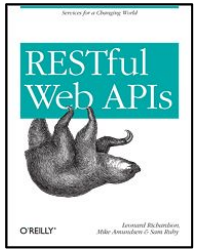

CREDIT-CHECK
P
MCA.CC.#1
HTML
Cj
ODATA
JSON
T
C
F
H
EXTERNAL
GET-SCORE
C T H F
COMPANYNAME
LEGAL NAME
ADDRESS.STREET1
(STREET ADDRESS
STREET ADDRESS)
.STREET2
.CITY
ADDRESSLOCALITY
.STATE/PROV
ADDRESSREGION
.COUNTRY
ADDRESS COUNTRY
.POSTALCODE
POSTALCODE
GET-SCORE

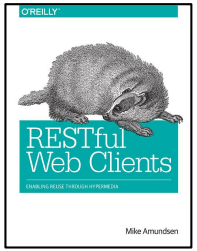

RESTful
Web APIs
O'REILLY


O'REILLY
RESTful
Web Clients
Mike Amundsen

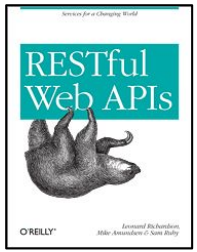

CREDIT-CHECK
MCA.CC.01
CREDIT-CHECK-ALPS.XM

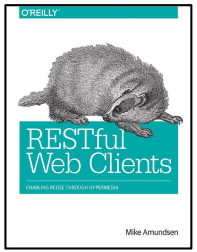
API ACADEMY


RESTful
Web APIs
O'REILLY


O'REILLY
RESTful
Web Clients
Mike Amundsen

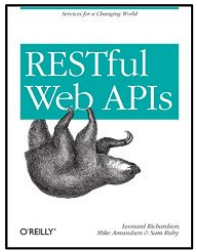
The Adaptable System

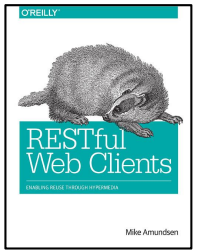




The Adaptable System

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

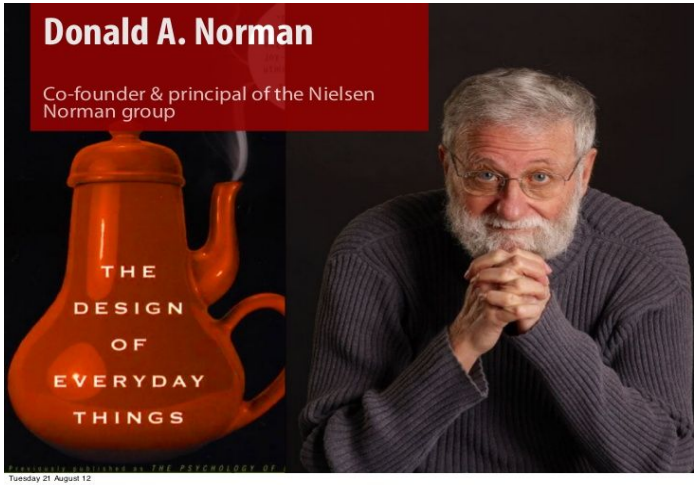




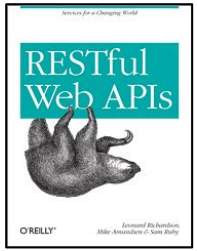
Service/API Designers

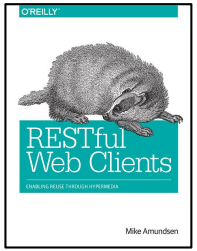
"The value of a well-designed object is when it has such a rich set of affordances that the people who use it can do things with it that the designer never imagined."

-- Donald Norman, 1994



@jnd1er

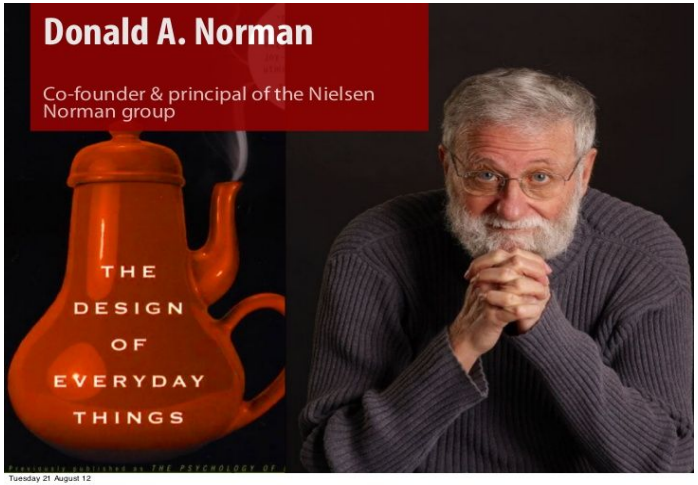




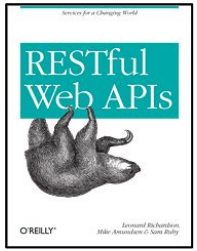
https://en.wikipedia.org/wiki/The_Design_of_Everyday_Things

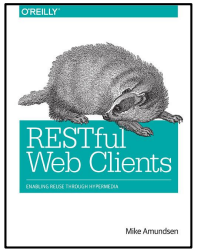
Service/API Designers

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



@jnd1er





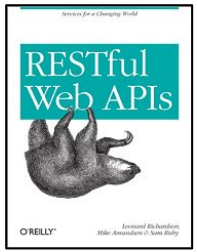
https://en.wikipedia.org/wiki/The_Design_of_Everyday_Things

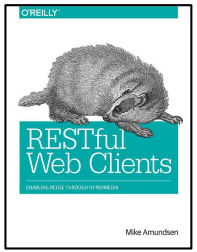
Evolvable Providers

"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

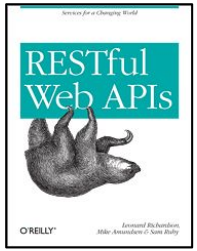
Evolvable Providers

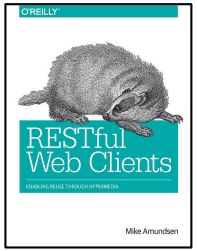
- Don't take things away
- Don't change the meaning of things
- Make all additions optional

@jasonrudolph

API ACADEMY

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

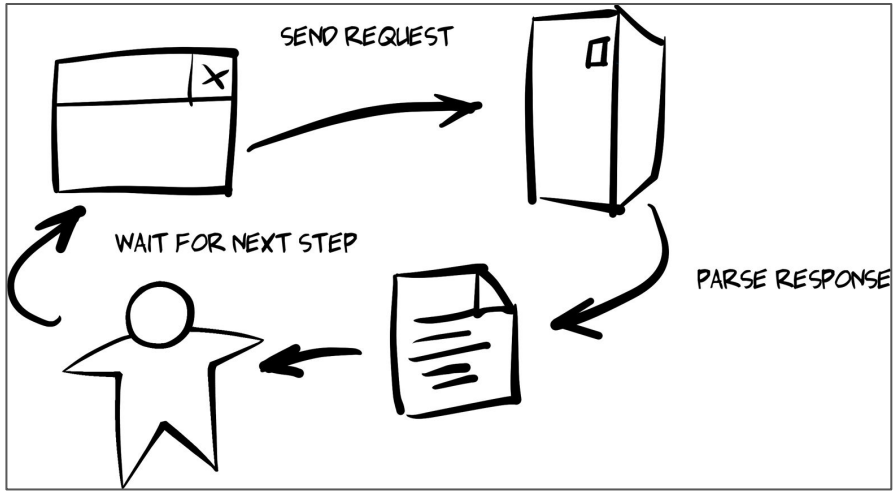




Adaptable Consumers

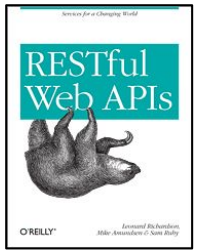
"When you can build a client that doesn't have to memorize the solution ahead of time you can start building clients who are 'smart' enough to adapt to new possibilities as the service presents them."

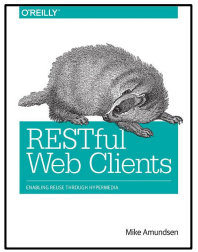
-- Mike Amundsen, 2016



API ACADEMY

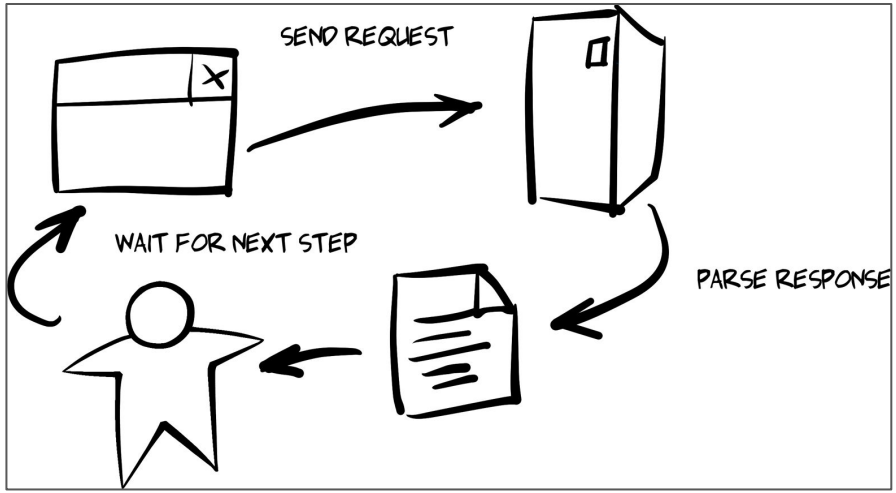
http://shop.oreilly.com/product/0636920037958.do





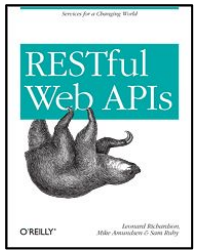
Adaptable Consumers

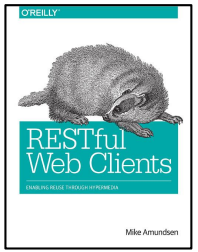
- Code defensively
- Code to the media type
- Leverage the API vocabulary
- React to link relations for workflow



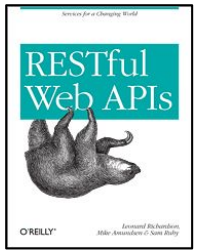
API ACADEMY

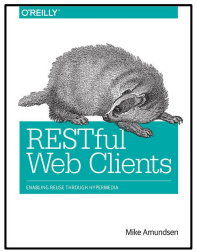
http://shop.oreilly.com/product/0636920037958.do



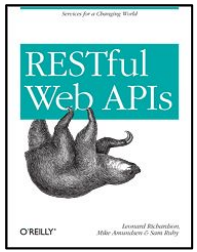


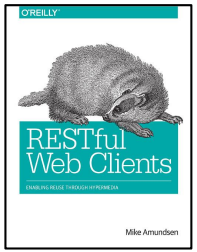
Providers evolve via humans, consumers adapt via code.

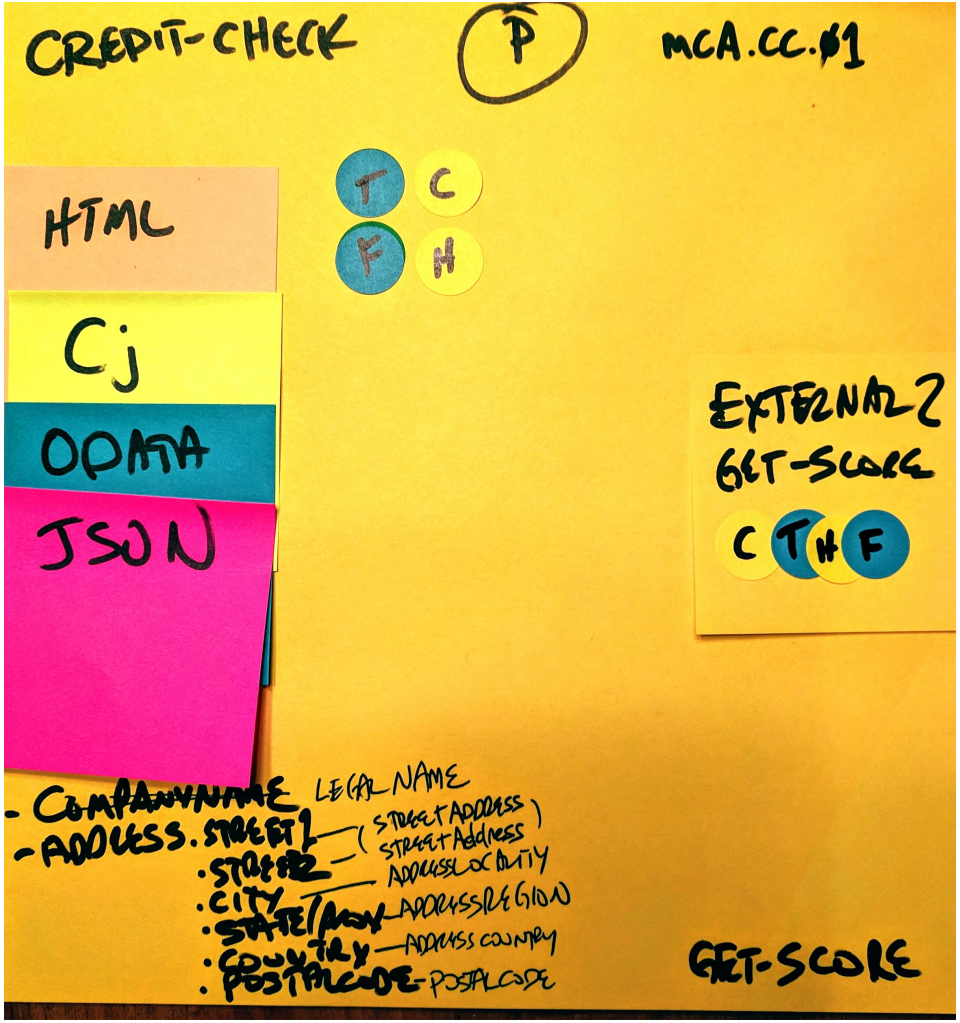


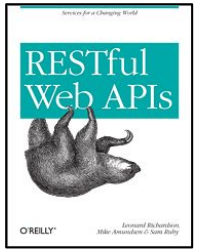


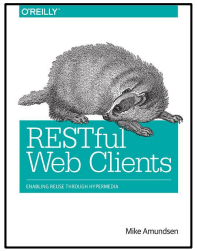
Adaptation Exercise



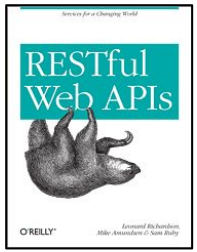


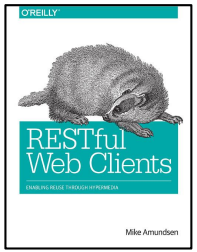

CREDIT-CHECK
P
MCA.CC.#1
T
C
F
H
HTML
Cj
ODATA
JSON
EXTERNAL?
GET-SCORE
C T H F
- COMPANYNAME
LEGAL NAME
- ADDRESS.STREET1
(STREET ADDRESS)
STREET ADDRESS
.STREET2
ADDRESSLOCALITY
.CITY
.STATE/PROV
ADDRESSREGION
.COUNTRY
ADDRESS COUNTRY
.POSTALCODE
POSTALCODE
GET-SCORE


RESTful
Web APIs
O'REILLY


O'REILLY
RESTful
Web Clients
Mike Amundsen

Summary

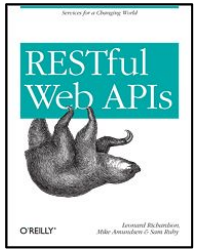

RESTful
Web APIs
O'REILLY

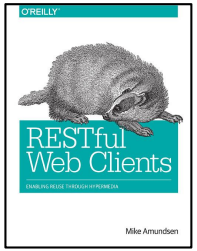

O'REILLY
RESTful
Web Clients
Mike Amundsen

Summary

- A RESTful Approach
- Message-Oriented Design
- Three Types of Microservices
- Six Stability Patterns
- Discovery Patterns
- Emergent Adaptability

API ACADEMY

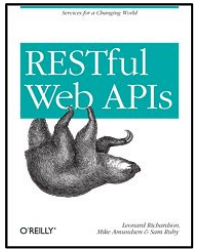


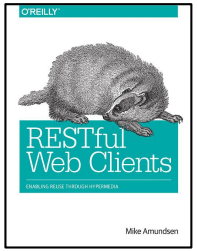


A RESTful Approach

- Microservices means independent & loosely-coupled
- REST properties are close to Microservice properties
- Adopt Microservice Constraints

@Fielding

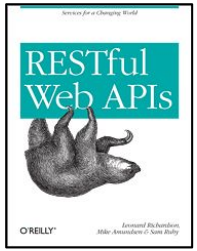


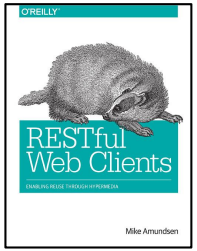


Message-Oriented Design

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

@PatHelland



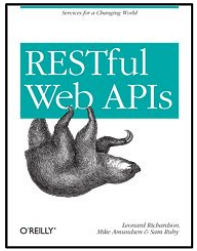


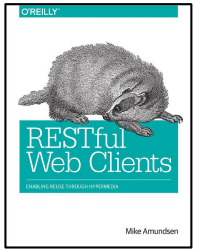
API ACADEMY

Three Microservice Types

- Stateless
- Persistence
- Aggregator

@mamund



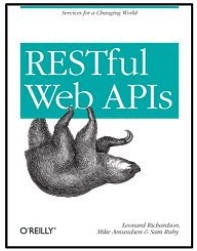


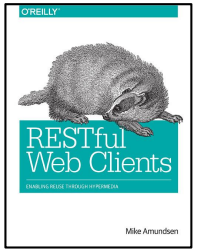
API ACADEMY

Six Stability Patterns

- Timeout
- Circuit Breaker
- Bulkhead
- Steady State
- Failfast
- Handshaking

@mtnygard

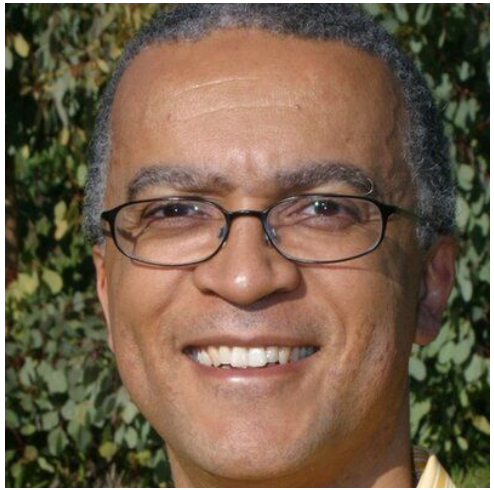




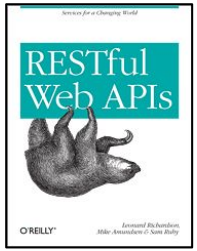
API ACADEMY

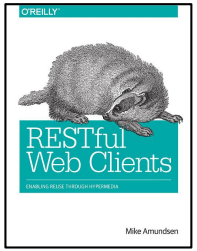
Discovery Patterns

- Advertising Services
- Discovering Services
- Health Checking/Renewals



@CRichardson



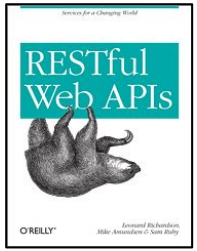


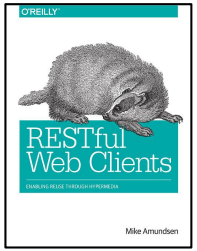
API ACADEMY

Emergent Adaptability

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

@mamund





API ACADEMY

RESTful Microservices
from the Ground Up

2018 SACon London

Mike Amundsen
API Academy
@mamund