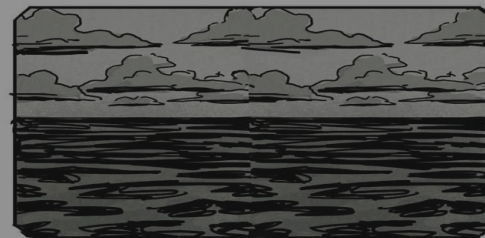
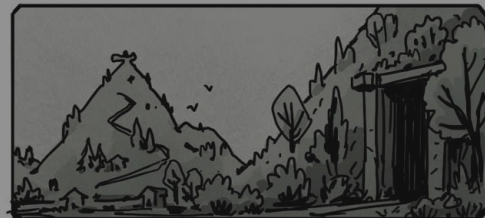


OMNI TERRA

Discovering RESTful Web Microservices

Mike Amundsen
@mamund
training.amundsen.com





Mike Amundsen
@mamund

training.amundsen.com

Amundsen Training

Home ▾ 🔍

Learning with Mike Amundsen



[Technical](#)



[Transformation](#)



[Teams](#)



[Inspiration](#)

<http://g.mamund.com/msabook>

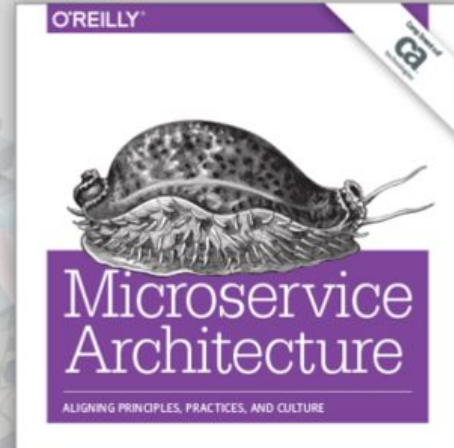


Microservice Architecture: Aligning Principles, Practices, and Culture

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:

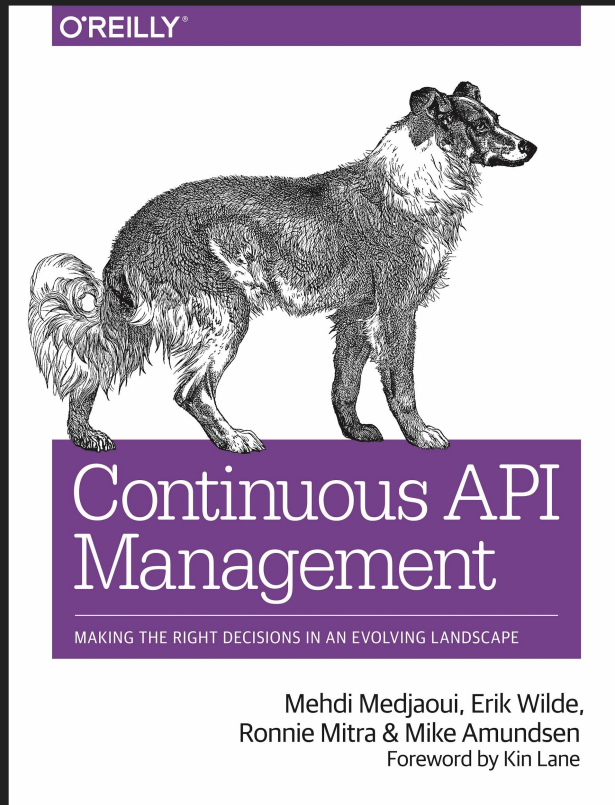
- The benefits and principles of microservices
- A design-based approach to microservice architecture
- Lessons for applying microservices in practice



<http://g.mamund.com/cambook>

*"A reusable guide to the technology,
business, and politics of doing APIs
at scale within the enterprise."*

-- Kin Lane, API Evangelist



<http://g.mamund.com/great-apis>

"I wish I had this book 20 years ago."

"A great classroom text or web guide."

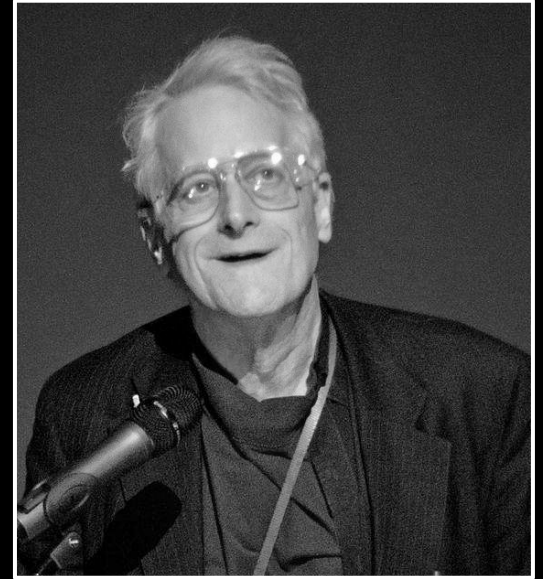
*"Useful in a way that doesn't tie it to
specific technologies."*



RESTful Web Microservices?

The Web

The words hypertext, hyperlink, and hypermedia were coined by Ted Nelson around 1965.

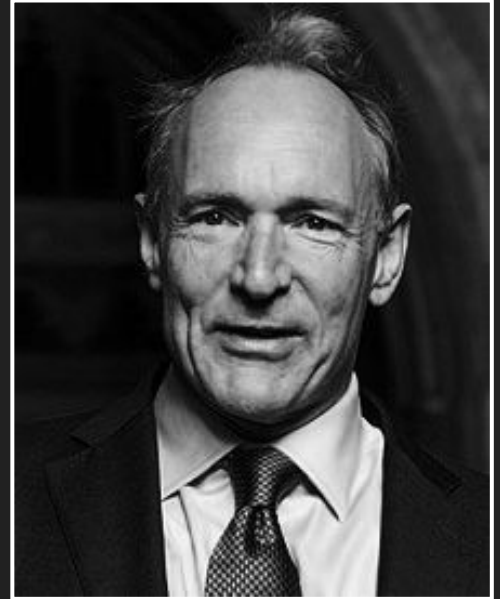


Ted Nelson

HTTP and HTML

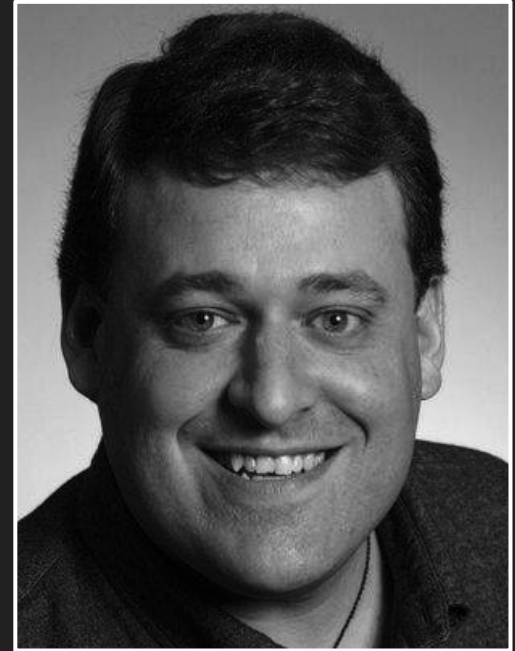
"We should work toward a universal linked information system, in which generality and portability are [most] important."

-- Tim Berners-Lee, 1989



REpresentational State Transfer (REST)

"REST emphasizes scalability of component interactions, generality of interfaces, independent deployment of components, and intermediary components."



-- Roy Fielding, 2000

Microservices

"An approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms."



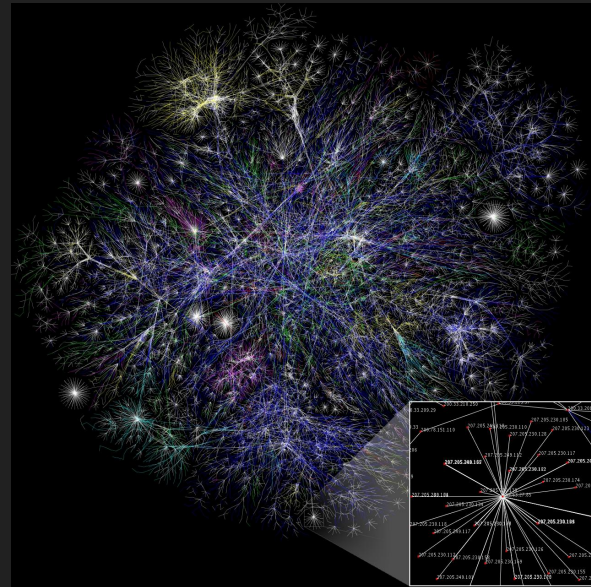
-- Martin Fowler, 2014

<https://www.thoughtworks.com/insights/blog/microservices-nutshell>

*Loosely-coupled components
running in an
engineered system.*

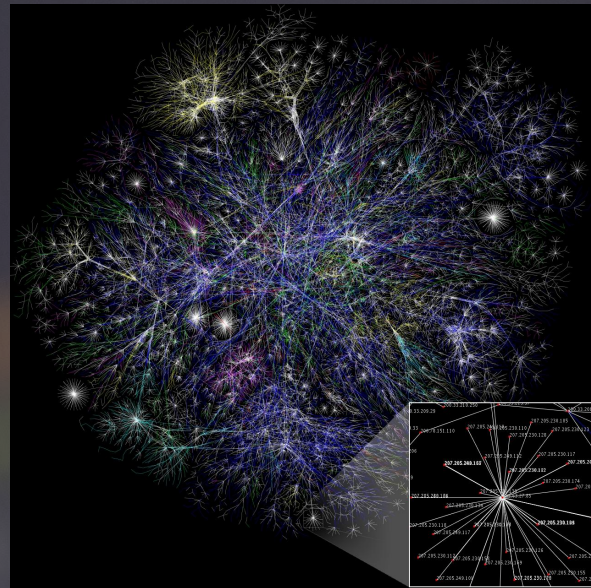


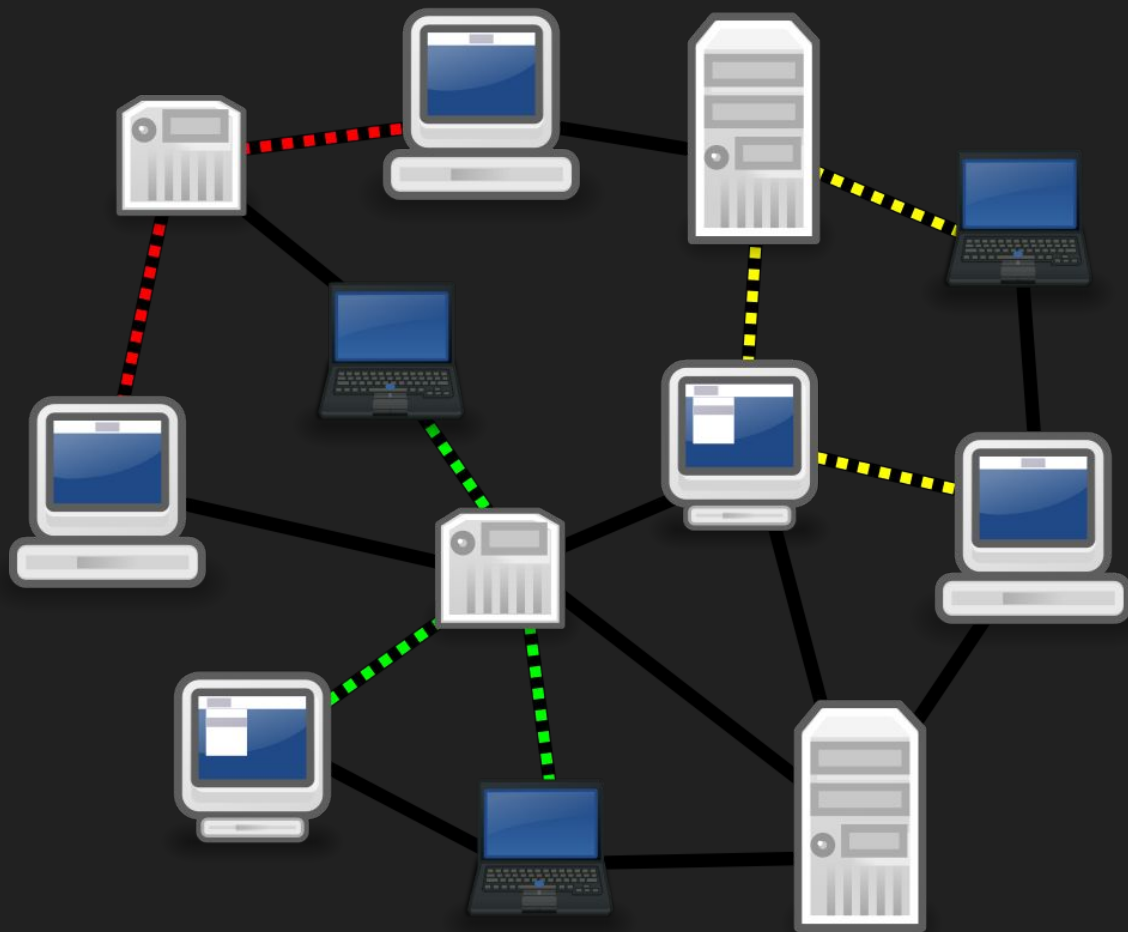
Traveling the Network

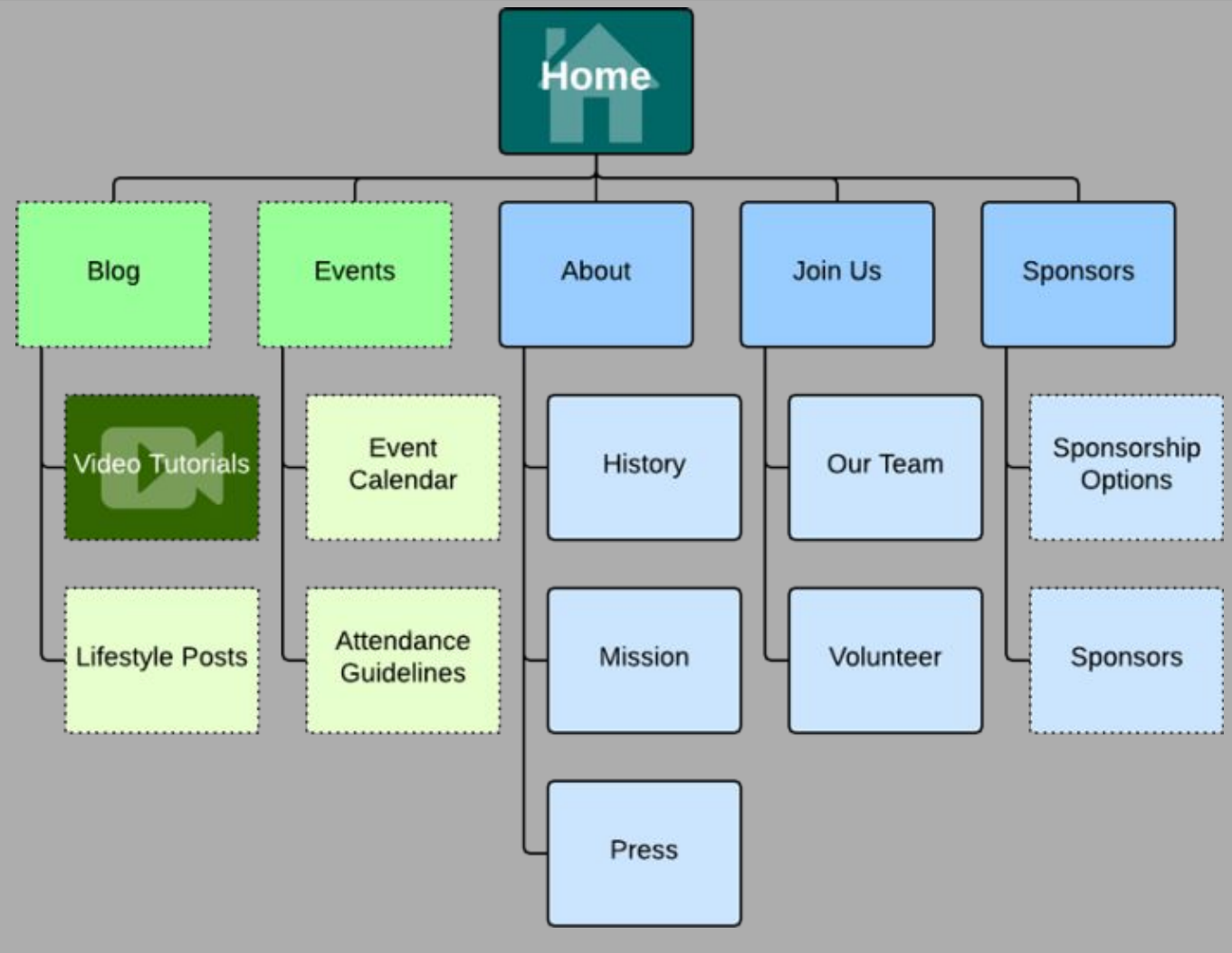




Programming the Network









/user

[Show/Hide](#)[List Operations](#)[Expand Operations](#)[Raw](#)

POST

[/user.json/createWithArray](#)

Creates list of users with given input array

POST

[/user.json](#)

Create user

POST

[/user.json/createWithList](#)

Creates list of users with given list input

PUT

[/user.json/{username}](#)

Updated user

DELETE

[/user.json/{username}](#)

Delete user

GET

[/user.json/{username}](#)

Get user by user name

GET

[/user.json/login](#)

Logs user into the system

GET

[/user.json/logout](#)

Logs out current logged in user session

/pet

[Show/Hide](#)[List Operations](#)[Expand Operations](#)[Raw](#)

GET

[/pet.json/{petId}](#)

Find pet by ID

POST

[/pet.json](#)

Add a new pet to the store

PUT

[/pet.json](#)

Update an existing pet

GET

[/pet.json/findByStatus](#)

Finds Pets by status

LEGEND to MAP SYMBOLS

COUNTRY

Feature Town



desert/wasteland



grassland



swamp



hills



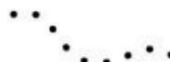
mountains



cave



political border



road



river



town/village



cities



tower/fortress



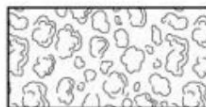
castle



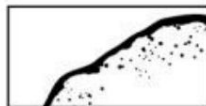
country capital



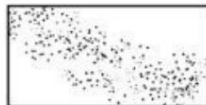
windmill



forest



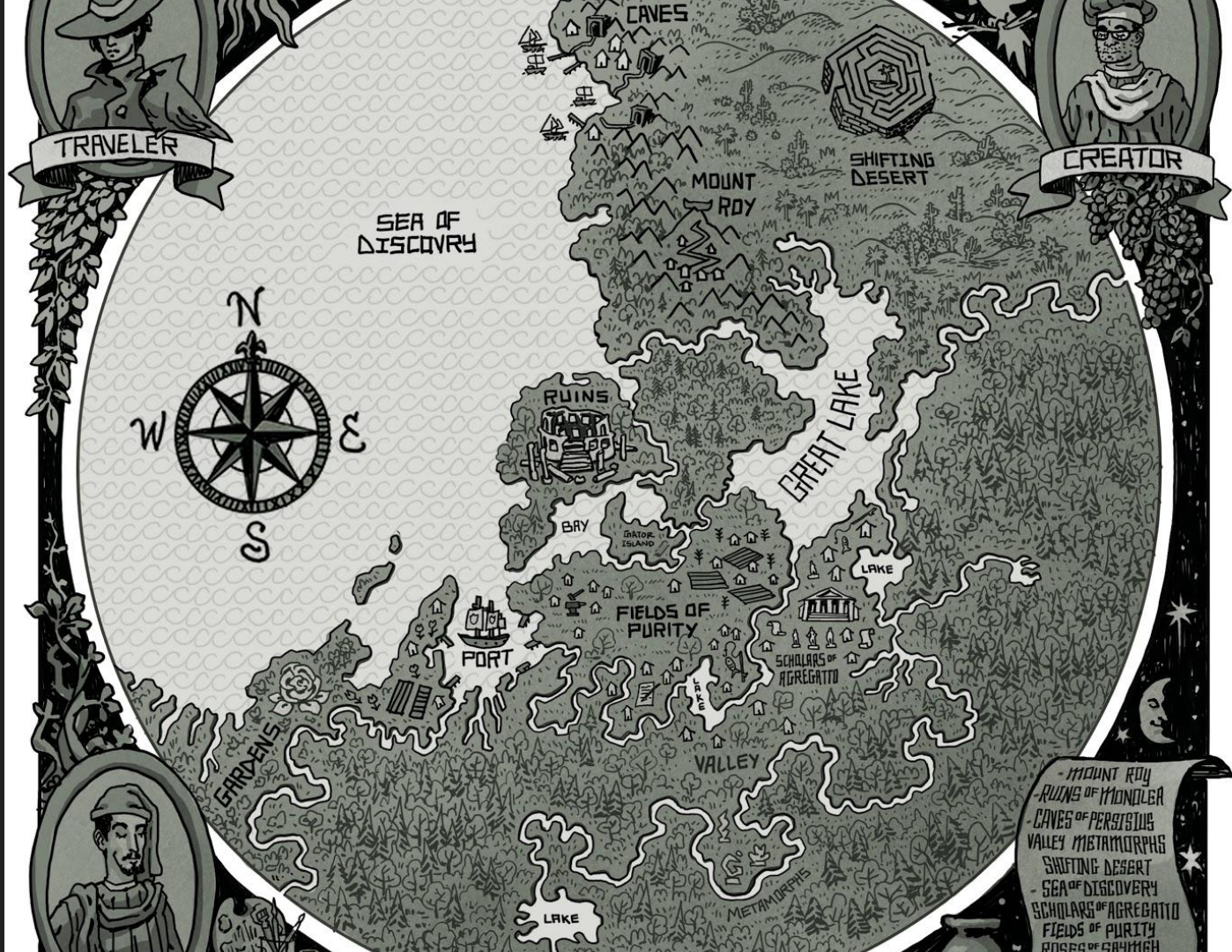
lake



reef



canyon

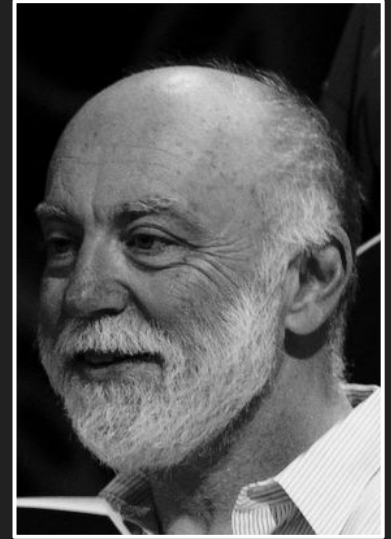


MOUNT ROY
- RUINS OF MONDLER
- CAVES OF PERSEUS
- VALLEY METAMORPHS
- SHIFTING DESERT
- SEA OF DISCOVERY
- SCHOLARS OF AGREGATTO
- FIELDS OF PURITY
- ROADS OF BATHIGH



Fallacies of Distributed Computing (1994)

1. The network is reliable.
2. Latency is zero.
3. Bandwidth is infinite.
4. The network is secure.
5. Topology doesn't change.
6. There is one administrator.
7. Transport cost is zero.
8. The network is homogeneous.



L Peter Deutsch

Programming the Network

"There is no simultaneity at a distance."

-- Pat Helland (2005)



Pat Helland

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

-- Michael T. Nygard



The
Pragmatic
Programmers

Release It!

Second Edition

Design and Deploy
Production-Ready Software



Michael T. Nygard
Edited by Katharine Dvorak

Nygard Stability Patterns

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



*"The journey of a thousand miles begins
with one step." -- Lao Tzu*

Let's talk about code for a bit...

OMNI TERRA



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

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

Stateless Microservices

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

But, what about the network?

Programming the network

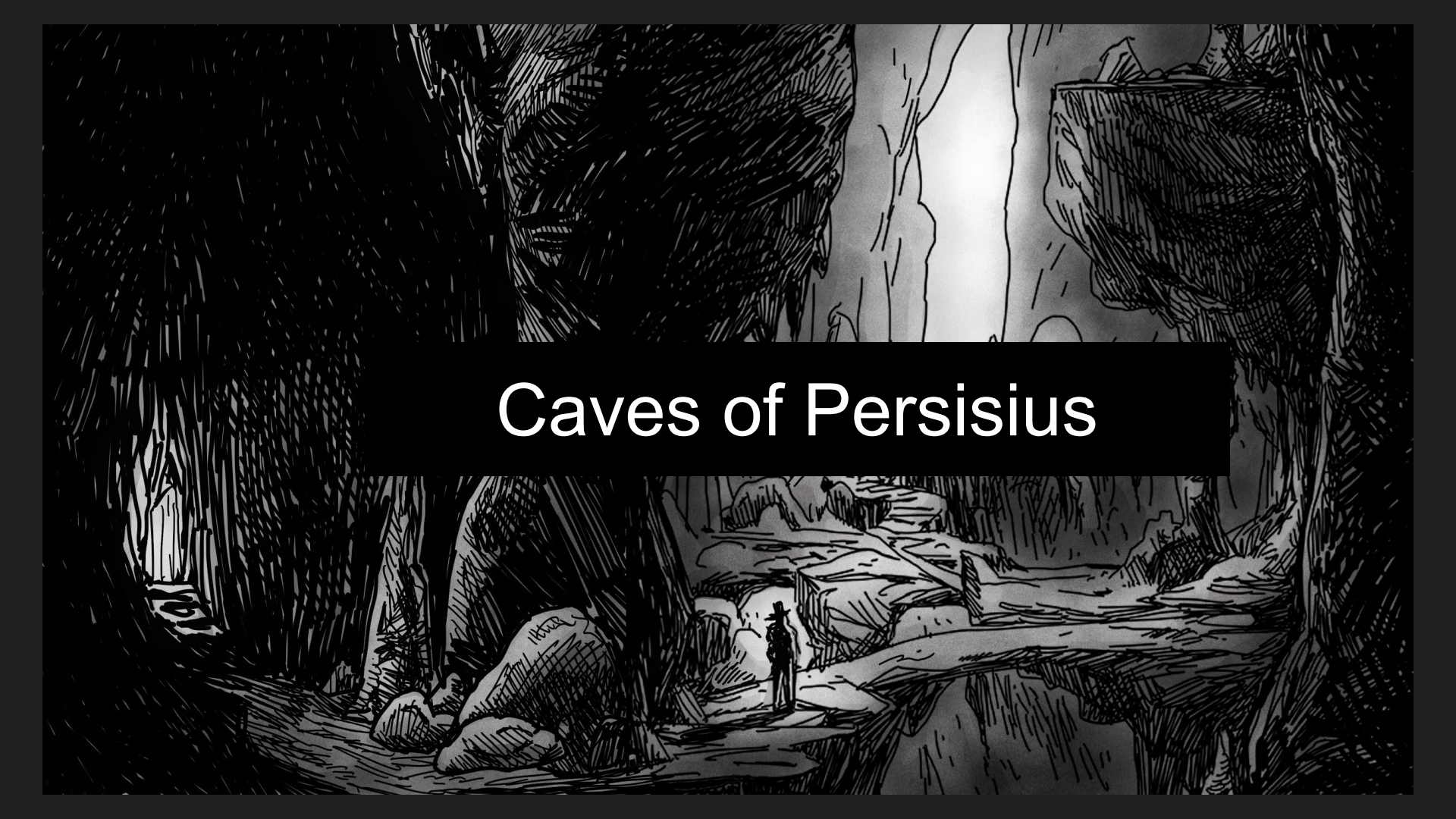
- *What if the work takes too long?*

Stateless Microservices

```
// http server handling data conversions  
function conversionServer(request, response) {  
  if(request.timeBudget > my.averageResponse) {  
    response = FailFastError(request);  
  }  
  else {  
    response = convertValue(request);  
  }  
  return response;  
}
```

1. Fail-Fast





Caves of Persisius

Persistence Microservices

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

Commonly needed MSC, not the easiest to implement.

Persistence Microservices

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

Persistence Microservices

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

But, what about the network?

Programming the network

- *What if the work takes too long?*
- *What if the dependent service doesn't respond in time?*
- *What if the dependent service is down?*
- *What if the storage overflows (data, logs, etc.)?*

Persistence Microservices

```
function updateOrders(request, response) {  
  if(request.timeBudget < localStorage.latency) {  
    response = FailFastError(request);  
  }  
  else {  
    response = setTimeout(circuitBreaker(  
      localStorage.write(request),  
      {timeout:10,maxFail:3,reset:30}  
    ), timeBudget);  
  }  
  return response;  
}
```

1. Fail-Fast
2. Timeout
3. Circuit Breaker
4. Steady State





Scholars of Aggregato

Aggregator Microservices

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

The most often-needed; most challenging, too.

Aggregator Microservices

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

Aggregator Microservices

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

But, what about the network?

Programming the network

- *What if the work takes too long?*
- *What if a dependent services doesn't respond in time?*
- *What if a dependent service is down?*
- *What if storage overflows (data, logs, etc.)?*
- *What if a dependent service is unhealthy?*
- *What if traffic for a service spikes?*

Aggregator Microservices

```
function writeOrders(request, response) {  
  var resourceList = ["customerDB", "orderDB", "salesDB"]  
  
  setTimeout(function(request, response, resourceList) {  
    var serviceList = gatherResources(resourceList);  
    if(serviceList.estimatedCost > request.timeBudget) {  
      response = FailFast(request);  
    }  
    else {  
      if(serviceList.healthy === true) {  
        circuitBreaker(serviceList, request,  
          {timeout:10,maxFail:3,reset:30});  
      }  
    }  
  },request.timeBudget);  
  
  return response;  
}
```

1. Fail-Fast
2. Timeout
3. Circuit Breaker
4. Steady State
5. Handshaking
6. Bulkhead



Nygard's Admonition...

W/ Joe asks:

Is All This Clutter Really Necessary?

You may think, as I did when porting the sockets library, that handling all the possible timeouts creates undue complexity in your code. It certainly adds complexity. You may find that half your code is devoted to error handling instead of providing features. I argue, however, that the essence of aiming for production—instead of aiming for QA—is handling the slings and arrows of outrageous fortune. That error-handling code, if done well, adds resilience. Your users may not thank you for it, because nobody notices when a system *doesn't* go down, but you will sleep better at night.

*Let **not** talk about code for a bit...*

OMNI TERRA



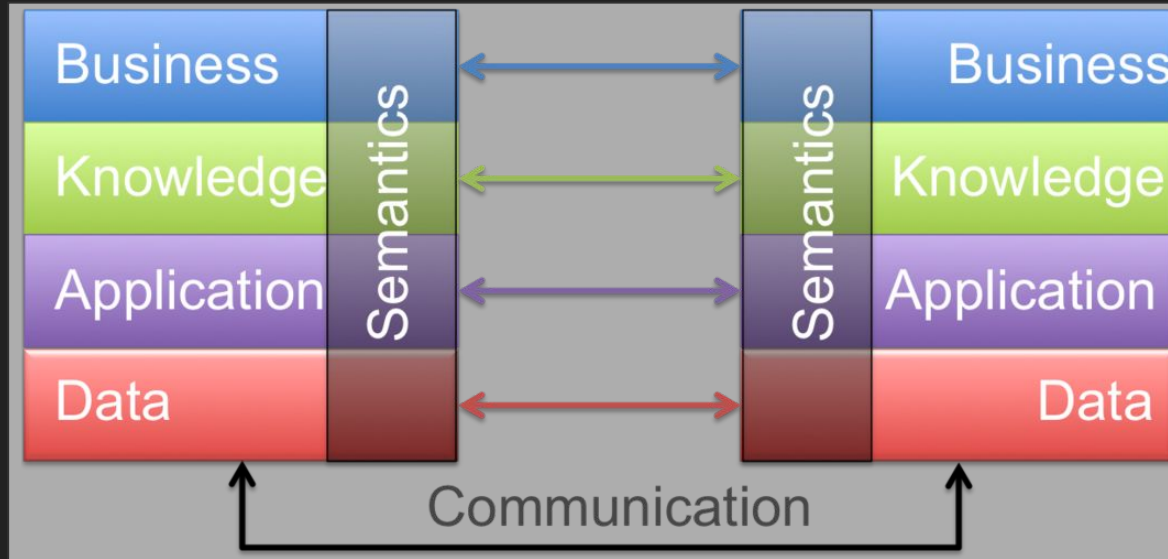
Aim for Interop, not Integration...

"Interoperation is peer to peer. Integration is where a system is subsumed within another."

-- Michael Platt, Microsoft



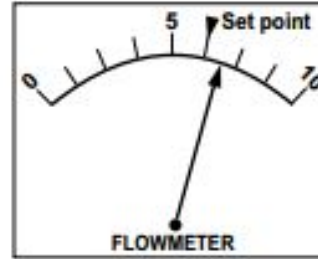
Aim for Interop, not Integration...



Signal, Sign, and Symbol

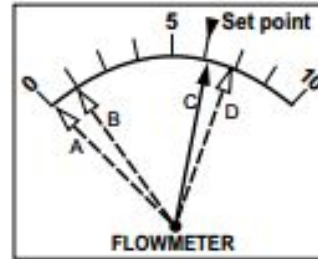


Jens Rasmussen



SIGNAL

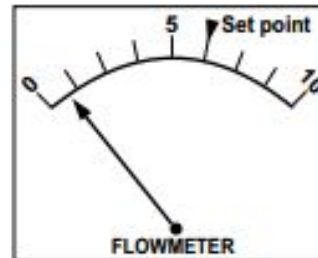
- Keep at set point
- Use deviation as error signal
- Track continuously



SIGN

Stereotype acts

If	If C, ok
Valve	If D, adjust flow
Open	
If	If A, ok
Valve	If B, recalibrate
Closed	meter



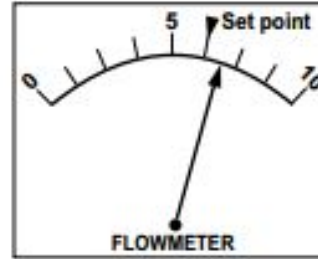
SYMBOL

If, after calibration, is still B, begin to read meter and speculate functionally (could be a leak)



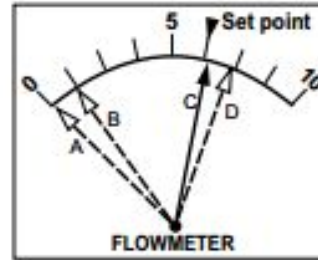
Signal, Sign, and Symbol

- Signal: Protocol
- Sign: Format
- Symbol: Vocabulary



SIGNAL

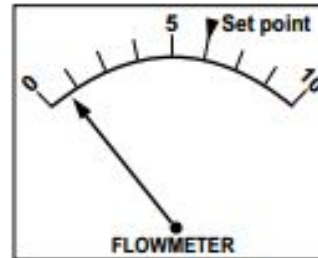
- Keep at set point
- Use deviation as error signal
- Track continuously



SIGN

Stereotype acts

If	If C, ok
Valve	If D, adjust flow
Open	
If	If A, ok
Valve	If B, recalibrate
Closed	meter



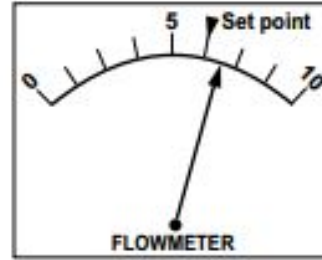
SYMBOL

If, after calibration, is still B, begin to read meter and speculate functionally (could be a leak)



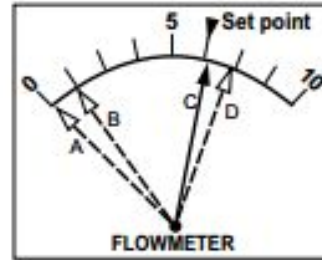
Signal, Sign, and Symbol

- Signal: Protocol
HTTP, CoAP, etc.
- Sign: Format
HTML, HAL, etc.
- Symbol: Vocabulary
ALPS, DCAP, etc.



SIGNAL

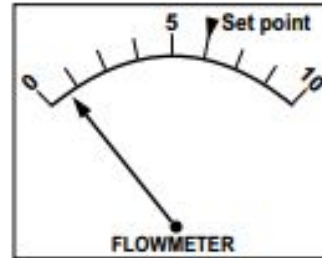
- Keep at set point
- Use deviation as error signal
- Track continuously



SIGN

Stereotype acts

If	If C, ok
Valve	If D, adjust flow
Open	
If	If A, ok
Valve	If B, recalibrate
Closed	meter

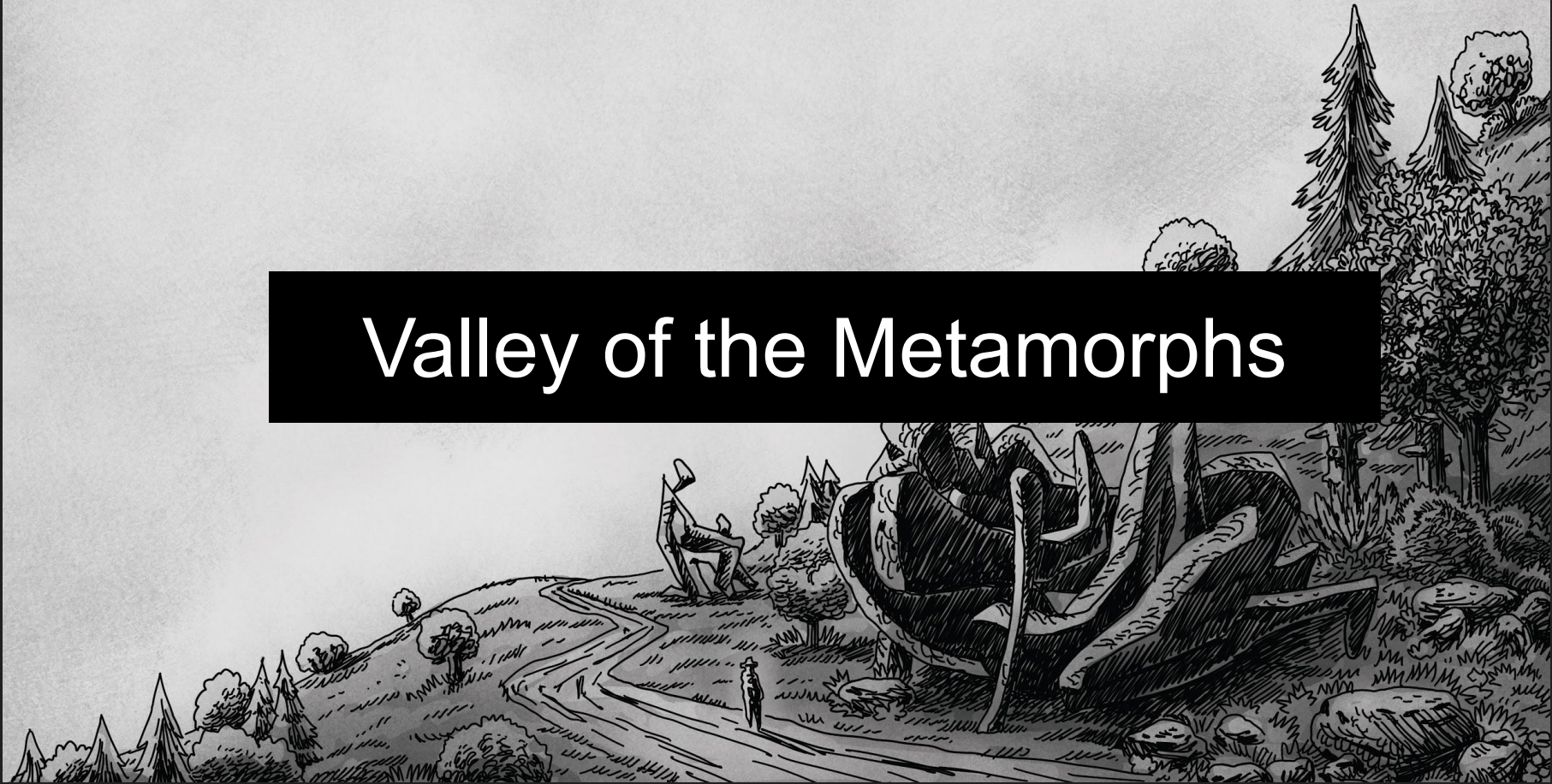


SYMBOL

If, after calibration, is still B, begin to read meter and speculate functionally (could be a leak)



Valley of the Metamorphs



Three Rules for Not Breaking Things...

1. You can't take things away
2. You can't change the meaning of things
3. All new things must be optional

You can't take things away...

```
*** REQUEST ***
GET /status HTTP/1.1
...

*** RESPONSE ***
200 OK
...
{
  "status" : "All OK"
}
```

```
*** REQUEST ***
GET /status HTTP/1.1
...

*** RESPONSE ***
400 Bad Request
```

```
*** REQUEST ***
GET /status HTTP/1.1
...

*** RESPONSE ***
HTTP/1.1. 301 Moved Permanently
Location: http://new-status

*** RESPONSE ***
HTTP/1.1 200 OK
...
{
  "status" : "All OK"
}
```

You can't change the meaning of things...

```
*** REQUEST ***  
GET /status HTTP/1.1  
...  
  
*** RESPONSE ***  
200 OK  
...  
{  
  "machinesActive" : "42"  
}
```

```
*** REQUEST ***  
GET /status HTTP/1.1  
...  
  
*** RESPONSE ***  
200 OK  
...  
{  
  "status" : "All OK",  
  "machinesActive" : "42"  
}
```

All new things MUST be optional...

```
*** REQUEST ***  
GET /status?machines HTTP/1.1  
...  
  
*** RESPONSE ***  
200 OK  
...  
{  
  "machinesActive" : "42"  
}
```

```
*** REQUEST ***  
GET /status HTTP/1.1  
...  
  
*** RESPONSE ***  
400 Bad Request
```

```
*** REQUEST ***  
GET /status HTTP/1.1  
...  
  
*** RESPONSE ***  
200 OK  
...  
{  
  "status" : "All OK",  
  "machinesActive" : "42"  
}
```

SOFTWARE ARCHITECTURE

What is
the best practice for
versioning
a REST API?



Roy Fielding, 2013

REST

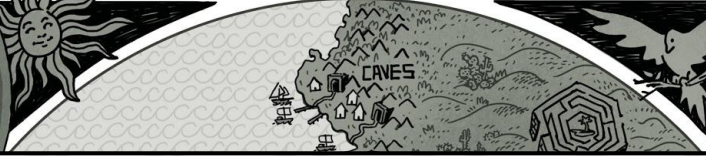
What is
the best practice for
versioning
a REST API?

DON'T

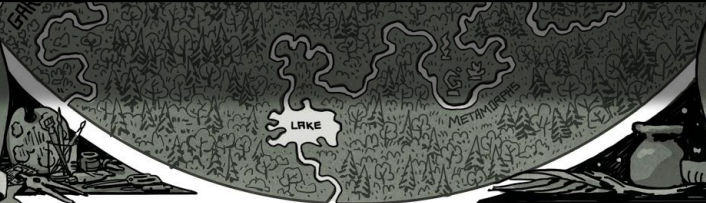
Versioning an interface
is just a “polite” way
to kill deployed applications

Roy Fielding, 2013

OMNI TERRA



SEA OF
DISCOVERY



RUINS OF MINDLEA
CAVES OF PERISTUS
VALLEY METAMORPHS
SHADING DESERT
SEA OF DISCOVERY
SCHOLARS OF AGGREGATO
FIELDS OF PURITY
ROSES OF SHYMEH

Service Discovery

*"How do you get
communication started among
totally uncorrelated 'sapient'
beings?"*

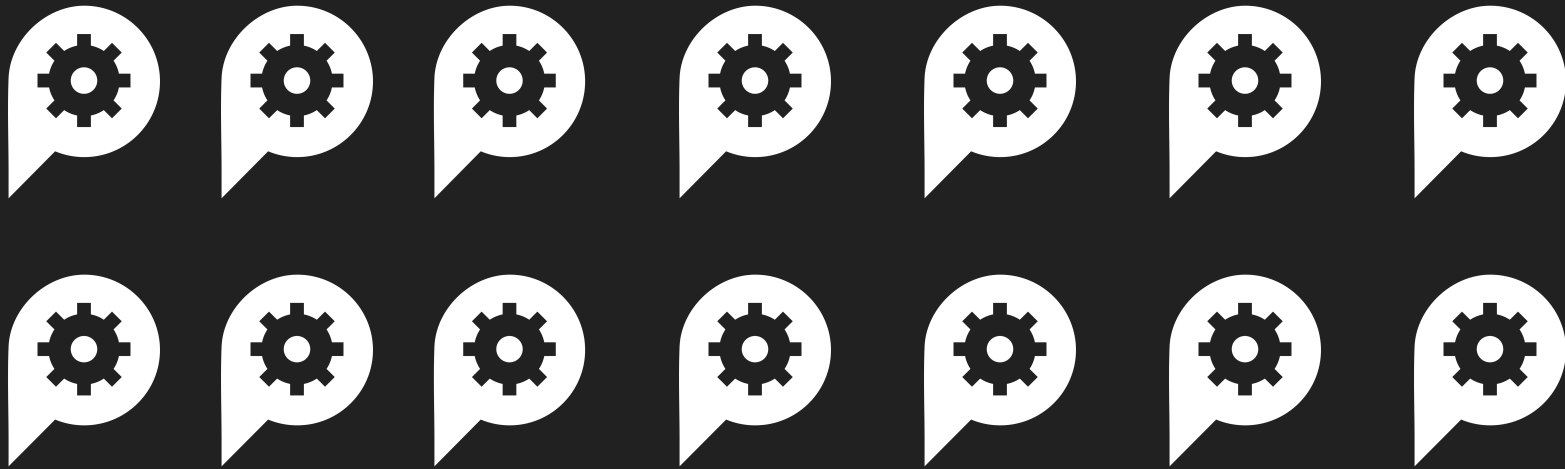
-- J. C. R. Licklider, 1963



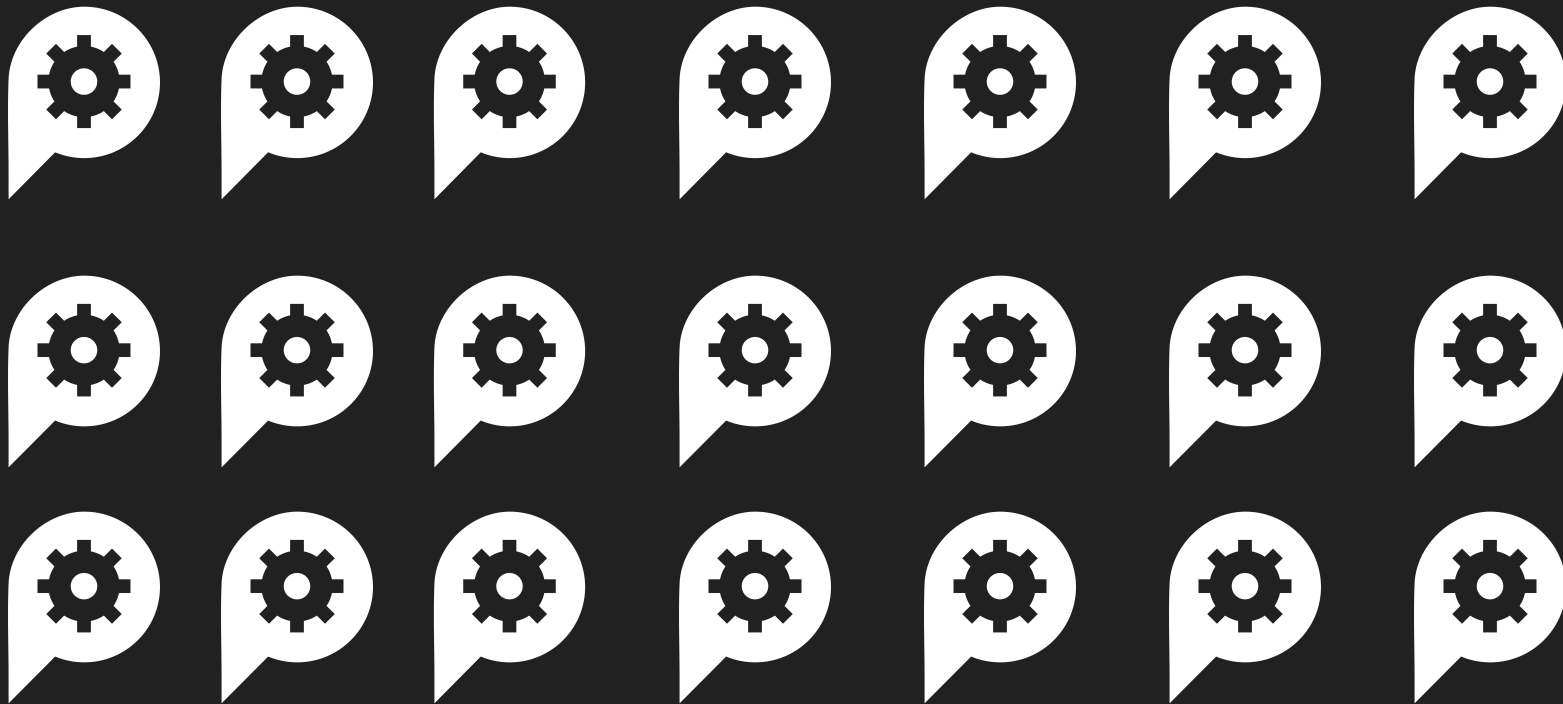
Service Discovery



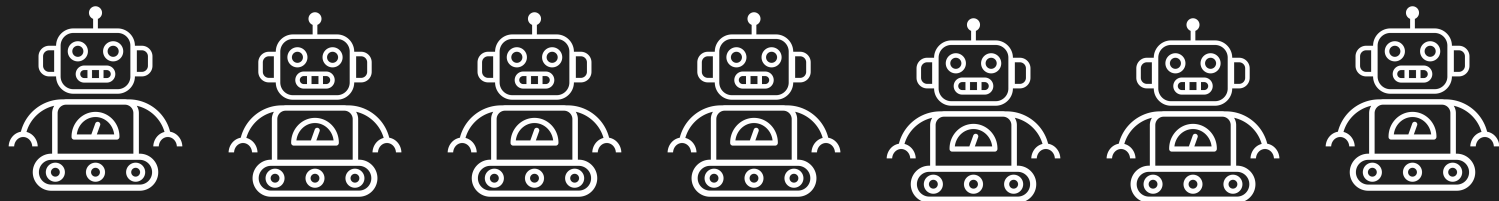
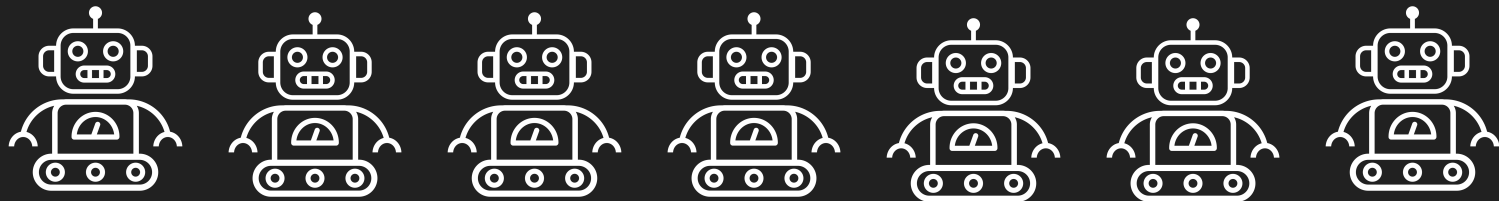
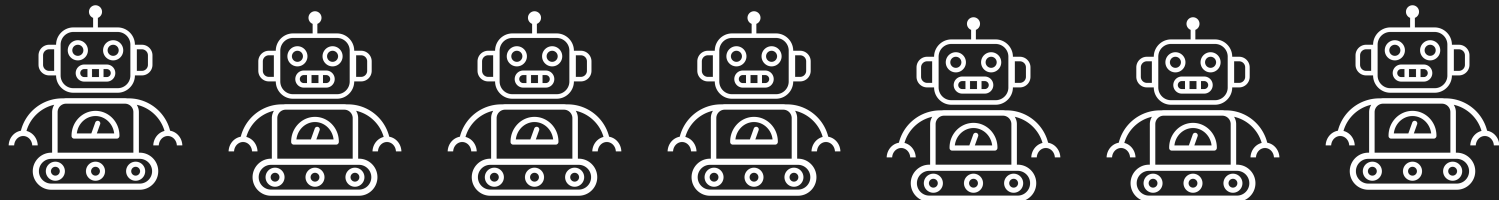
Service Discovery



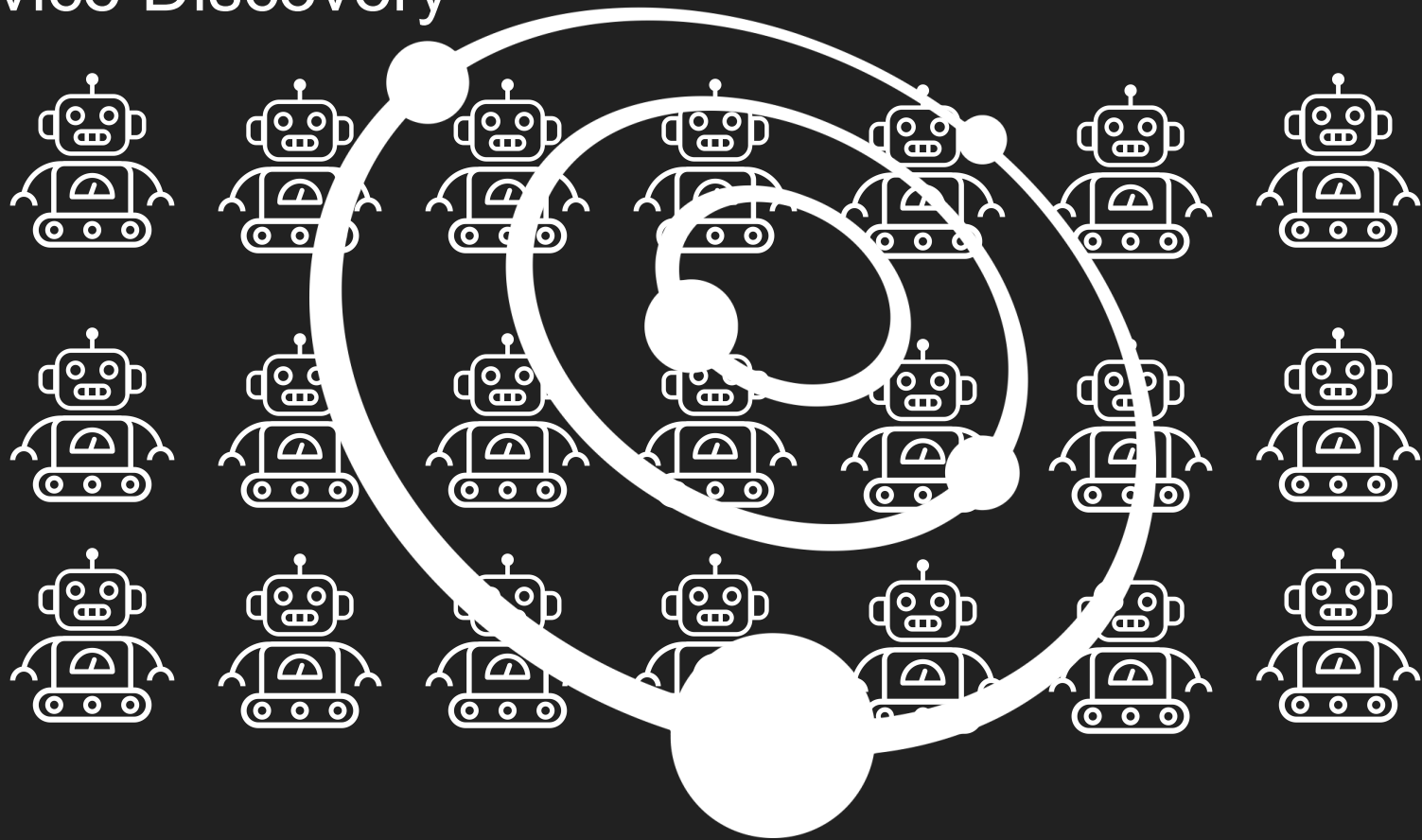
Service Discovery



Service Discovery



Service Discovery



Licklider Memo (1963)

ADVANCED RESEARCH PROJECTS AGENCY

Washington 25, D.C. April 23, 1963

MEMORANDUM FOR: Members and Affiliates of the Intergalactic
Computer Network

FROM: J. C. R. Licklider

SUBJECT: Topics for Discussion at the Forthcoming Meeting

First, I apologize humbly for having to postpone the meeting scheduled for 3 May 1963 in Palo Alto. The ARPA Command & Control Research office has just been assigned a new task that must be activated immediately, and I must devote the whole of the coming week to it. The priority is externally enforced. I am extremely sorry to inconvenience those of you who have made plans for May 3rd. Inasmuch as I shall be in Cambridge the rest of this week, I am asking my colleagues here to re-schedule the meeting, with May 10th, Palo Alto, as target time and place.

The need for the meeting and the purpose of the meeting are things that I feel intuitively, not things that I perceive in clear structure. I am afraid that that fact will be too evident in the following paragraphs. Nevertheless, I shall try to set forth some background material and some thoughts about possible interactions among the various activities in the overall enterprise for which, as you may have detected in the above subject, I am at a loss for a name.

In the first place, it is evident that we have among us a collection of individual (personal and/or organizational) aspirations, efforts, activities, and projects. These have in common, I think, the characteristics that they are in some way connected with advancement of the art or technology of information processing, the advancement of intellectual capability (man, man-machine, or machine), and the approach to a theory of science. The individual parts are, at least to some extent, mutually interdependent. To make progress, each of the active research needs a software base and a hardware facility more complex and more extensive than he, himself, can create in reasonable time.

In pursuing the individual objectives, various members of the group will be preparing executive the monitoring

Licklider Memo (1963)

ADVANCED RESEARCH PROJECTS AGENCY

Washington 25, D.C. April 23, 1963

MEMORANDUM FOR: Members and Affiliates of the Intergalactic Computer Network

FROM: J. C. R. Licklider

SUBJECT: Topics for Discussion at the Forthcoming Meeting

First, I apologize humbly for having to postpone the meeting scheduled for 3 May 1963 in Palo Alto. The ARPA Command & Control Research office has just been assigned a new task that must be activated immediately, and I must devote the whole of the coming week to it. The priority is externally enforced. I am extremely sorry to inconvenience those of you who have made plans for May 3rd. Inasmuch as I shall be in Cambridge the rest of this week, I am asking my colleagues here to re-schedule the meeting, with May 10th, Palo Alto, as target time and place.

The need for the meeting and the purpose of the meeting are things that I feel intuitively, not things that I perceive in clear structure. I am afraid that that fact will be too evident in the following paragraphs. Nevertheless, I shall try to set forth some background material and some thoughts about possible interactions among the various activities in the overall enterprise for which, as you may have detected in the above subject, I am at a loss for a name.

In the first place, it is evident that we have among us a collection of individual (personal and/or organizational) aspirations, efforts, activities, and projects. These have in common, I think, the characteristics that they are in some way connected with advancement of the art or technology of information processing, the advancement of intellectual capability (man, man-machine, or machine), and the approach to a theory of science. The individual parts are, at least to some extent, mutually interdependent. To make progress, each of the active research needs a software base and a hardware facility more complex and more extensive than he, himself, can create in reasonable time.

In pursuing the individual objectives, various members of the group will be preparing executive the monitoring

Licklider Protocol (2008)

Networking Working Group
Request for Comments: 5326
Category: Experimental

M. Ramadas
ISTRAC, ISRO
S. Burleigh
NASA/Jet Propulsion Laboratory
S. Farrell
Trinity College Dublin
September 2008

Licklider Transmission Protocol - Specification

Status of This Memo

This memo defines an Experimental Protocol for the Internet community. It does not specify an Internet standard of any kind. Discussion and suggestions for improvement are requested. Distribution of this memo is unlimited.

IESG Note

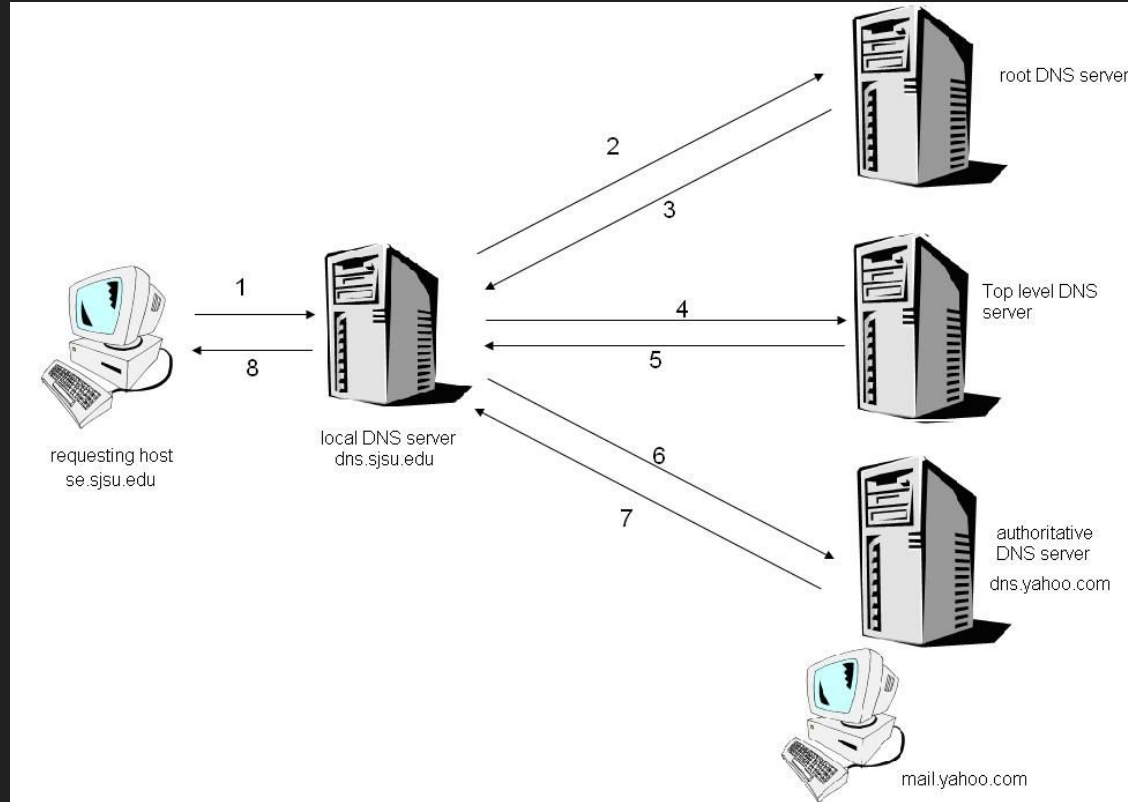
This RFC is not a candidate for any level of Internet Standard. It represents the consensus of the Delay Tolerant Networking (DTN) Research Group of the Internet Research Task Force (IRTF). It may be considered for standardization by the IETF in the future, but the IETF disclaims any knowledge of the fitness of this RFC for any purpose and in particular notes that the decision to publish is not based on IETF review for such things as security, congestion control, or inappropriate interaction with deployed protocols. See [RFC 3932](#) for more information.

Abstract

This document describes the Licklider Transmission Protocol (LTP), designed to provide retransmission-based reliability over links characterized by extremely long message round-trip times (RTTs) and/or frequent interruptions in connectivity. Since communication across interplanetary space is the most prominent example of this sort of environment, LTP is principally aimed at supporting "long-haul" reliable transmission in interplanetary space, but it has applications in other environments as well.

This document is a product of the Delay Tolerant Networking Research Group and has been reviewed by that group. No objections to its publication as an RFC were raised.

DNS for discovering machines



*But we need to discover **services...***

Discovering Interoperative Services... (DISCO)

5/15/2018

Discovering Interoperative Services for Continuous Operation (DISCO)

Discovering Interoperative Services for Continuous Operation (DISCO)

Mike Amundsen
<mca@mamund.com>

Table of Contents

[Status](#)

[Summary](#)

[DISCO Resources](#)

[DISCO Design Goals](#)

[Service-Side Implementation for NodeJS](#)

[The DISCO Registry Specification](#)

[DISCO Registry Service Sequence Diagram](#)

[DISCO Registry Service Basics](#)

[DISCO Registry Service Actions](#)

[Registry Bind Tokens](#)

[The Simple Bind Token](#)

Status

Status

Working Draft — Only experimental and 'proof-of-concept' apps should be built on this unstable draft.

Repository

<https://github.com/rwmbook/registry-docs>

Last Updated

2018-03-10

Summary

This document details an *ad hoc* specification called **DISCO** (Discovering Interoperative Services for Continuous Operation). DISCO is a simple language for managing the adding/removing of services as well as the ability to search ("find") and make connections with ("bind") registered services.

DISCO was designed to be easy, open, lightweight, and extensible. For this reason, readers/implementers may find things "missing" or "underspecified." This is intentional. Getting started is meant to be easy. And local customization is supported as needed. This allows the DISCO spec to safely grow and improve over time without breaking existing implementations.

The DISCO "language" supports the following features:

Open Discovery



Discovering Inter-operative Services... (DISCO)

The DISCO "language" supports the following features:

- `register` : add a service to the shared registry
- `find` : query the registry for services (dependents) to consume
- `bind` : notify the registry the intention to connect with and use another service
- `renew` : renew a service's registry *lease* to prove it is still up and running
- `unregister` : remove a service from the registry

Discovering Inter-operative Services... (DISCO)

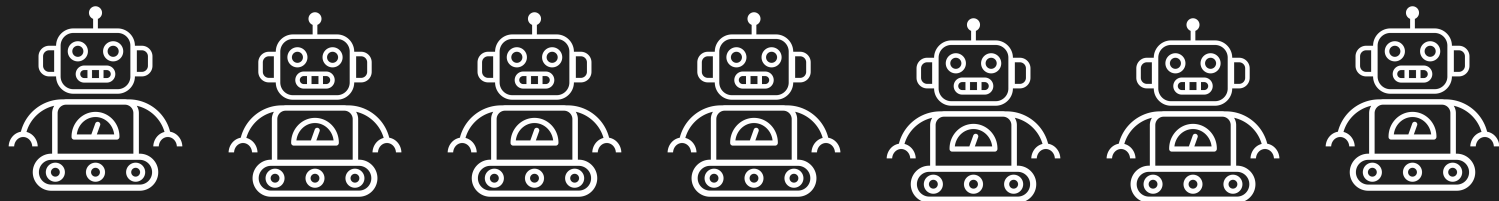
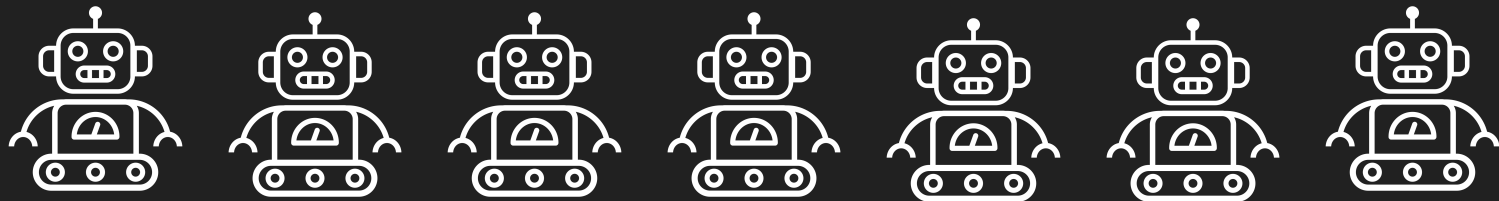
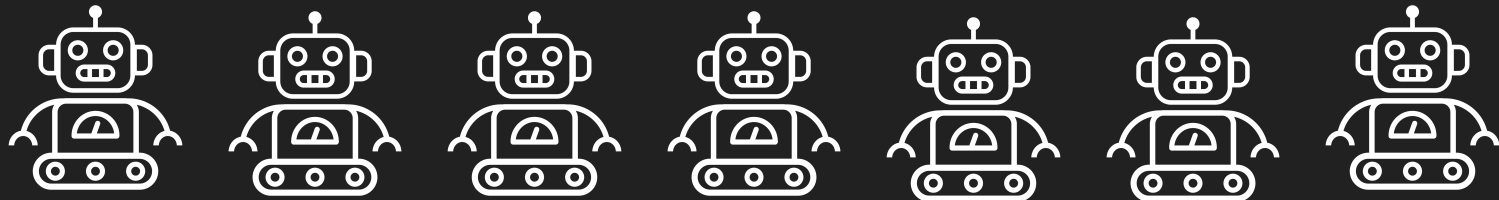
```
// register this service w/ defaults
discovery.register(null, function(response) {

    // sample service discovery action
    discovery.find(null, function(data, response) {

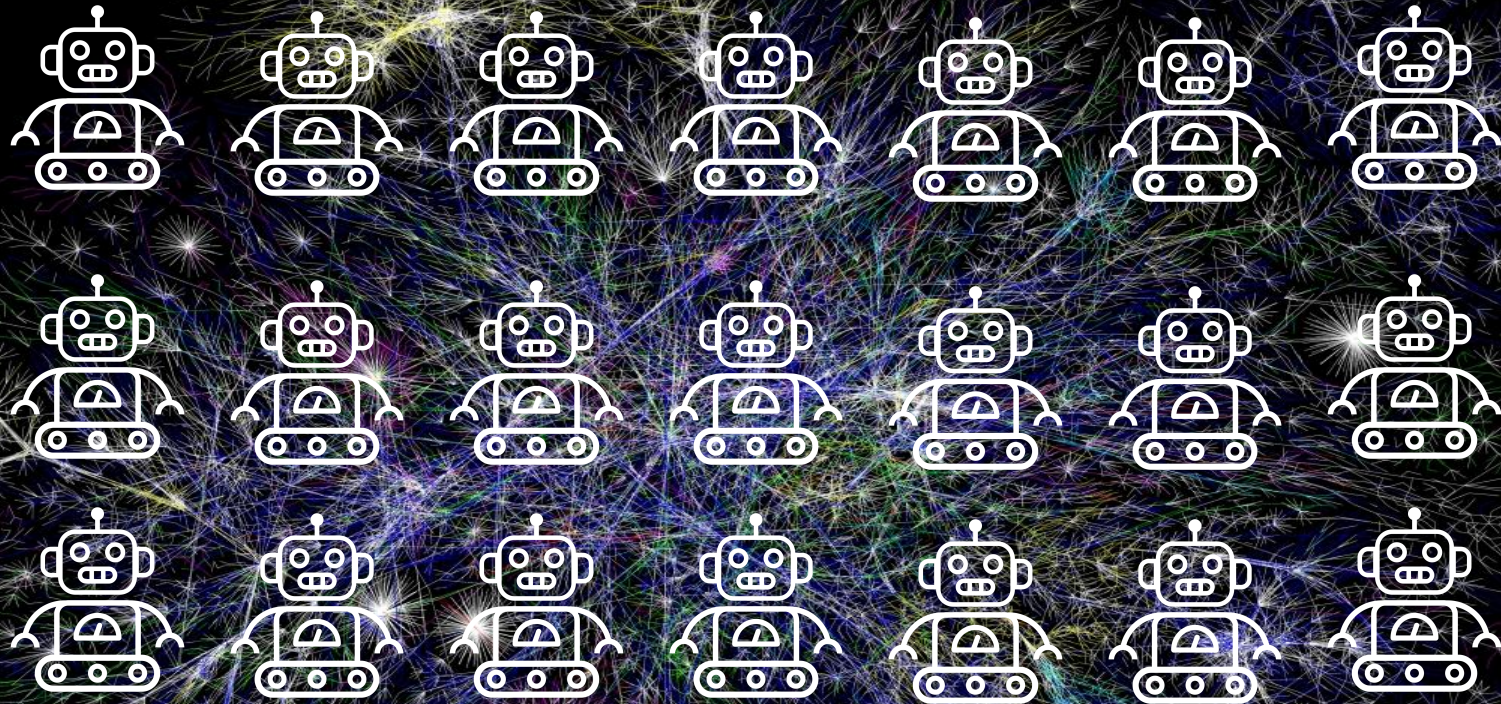
        // select endpoints from query
        if(data.success===true) {
            // launch http server
            http.createServer(zipServer).listen(8080);
            console.info('zip-server running on port 8080.');
```

```
        }
        else {
            console.error('unable to bind to dependent services');
            process.kill(process.pid, "SIGTERM");
        }
    });
});
```

Service Discovery

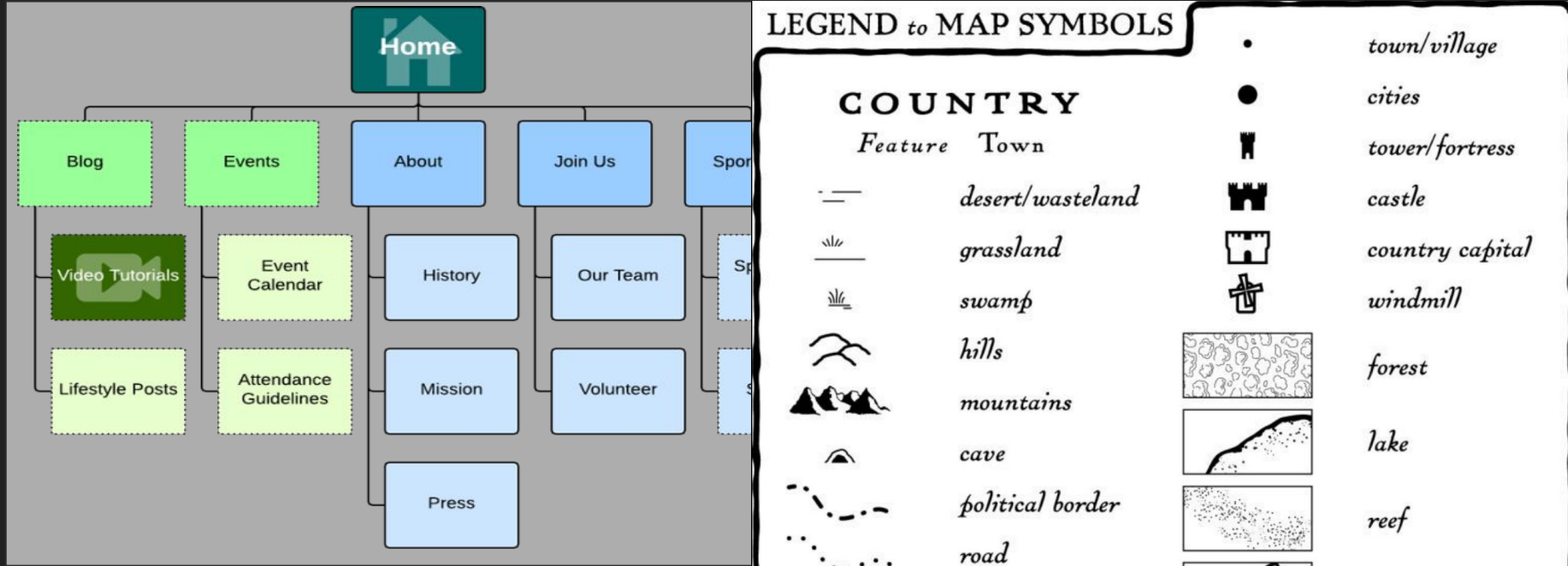


Service Discovery

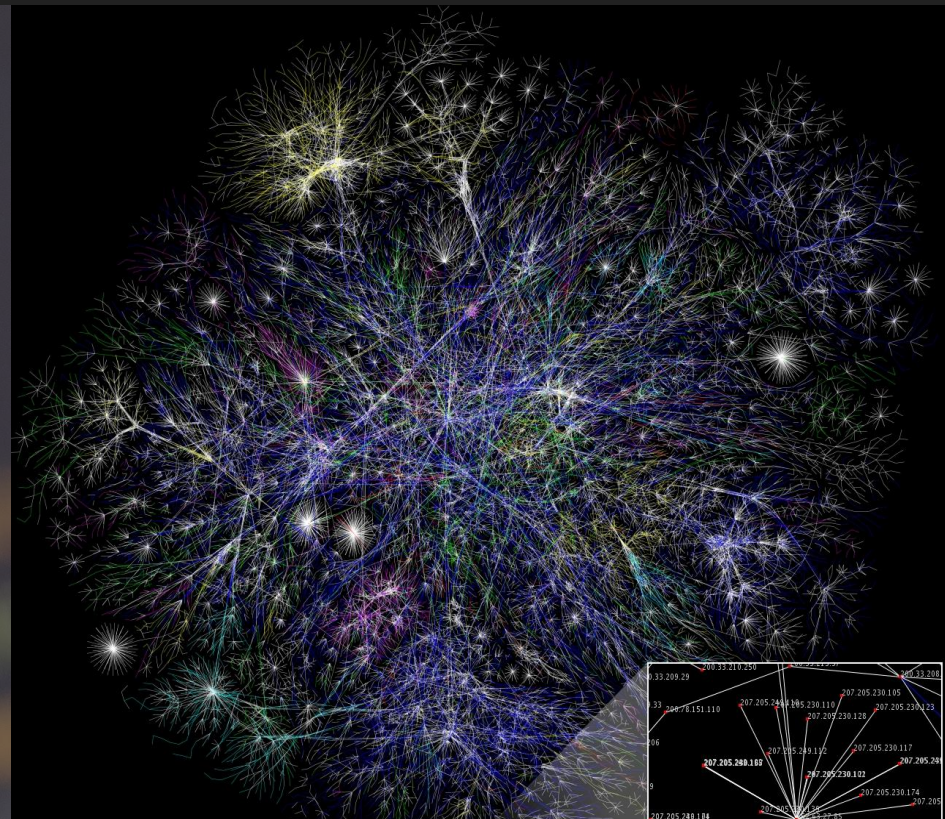
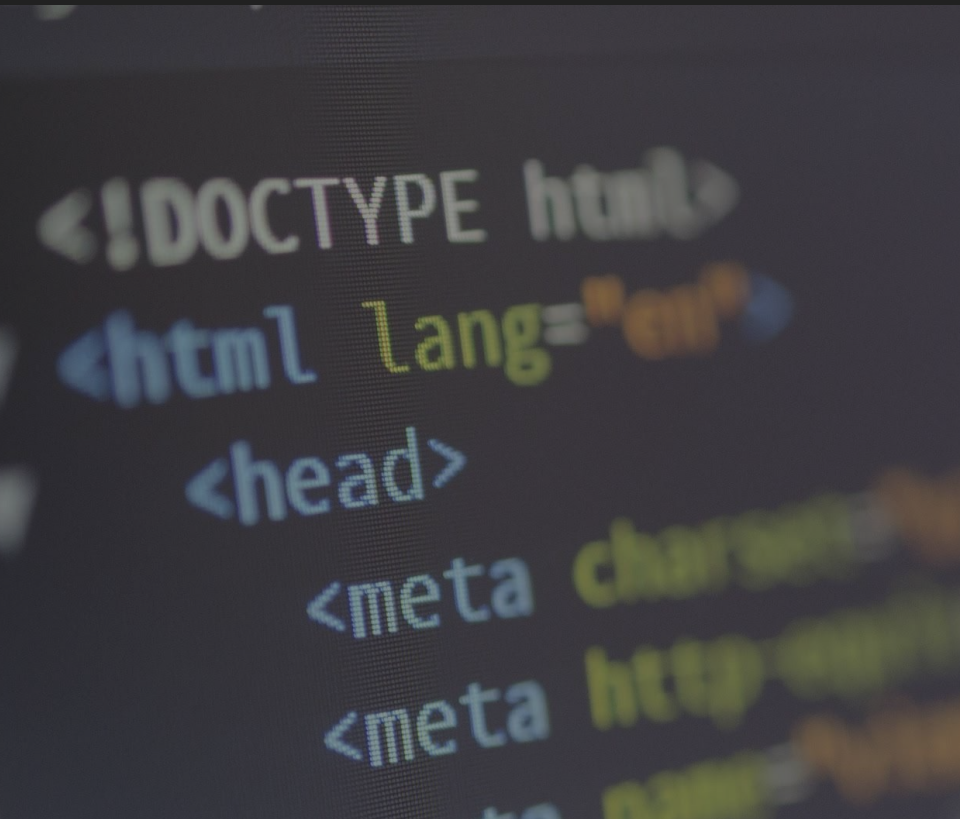


So...

We need better maps...



So that we can program the network...



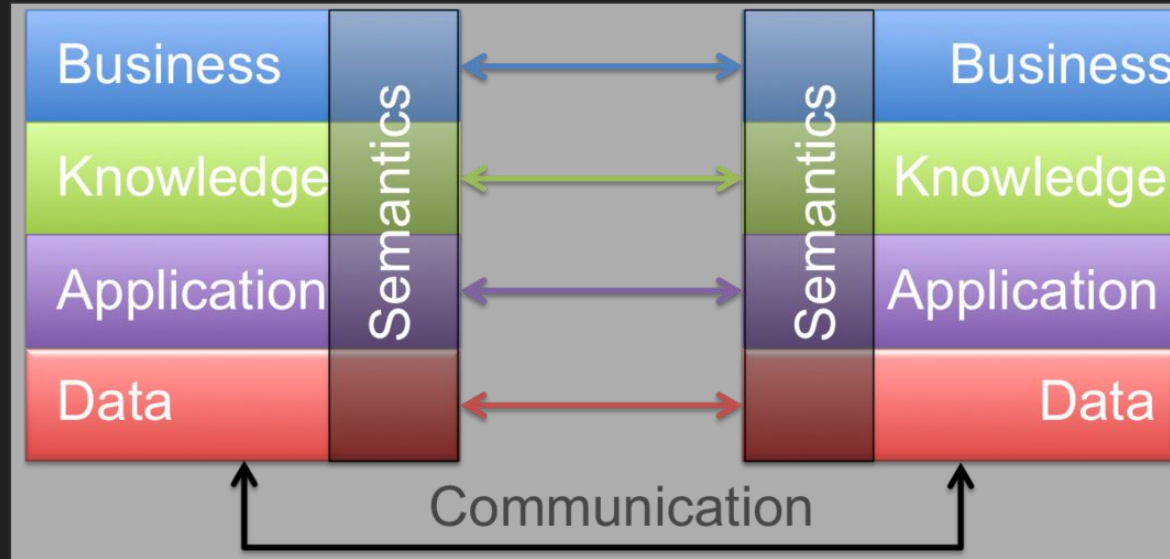
Which means applying patterns to our code..,

```
function writeOrders(request, response) {  
  var resourceList = ["customerDB", "orderDB", "salesDB"]  
  
  setTimeout(function(request, response, resourceList) {  
    var serviceList = gatherResources(resourceList);  
    if(serviceList.estimatedCost > request.timeBudget) {  
      response = FailFast(request);  
    }  
    else {  
      if(serviceList.healthy === true) {  
        circuitBreaker(serviceList, request,  
          {timeout:10,maxFail:3,reset:30});  
      }  
    }  
  },request.timeBudget);  
  
  return response;  
}
```

1. Fail-Fast
2. Timeout
3. Circuit Breaker
4. Steady State
5. Handshaking
6. Bulkhead



And that means understanding the role of semantics...

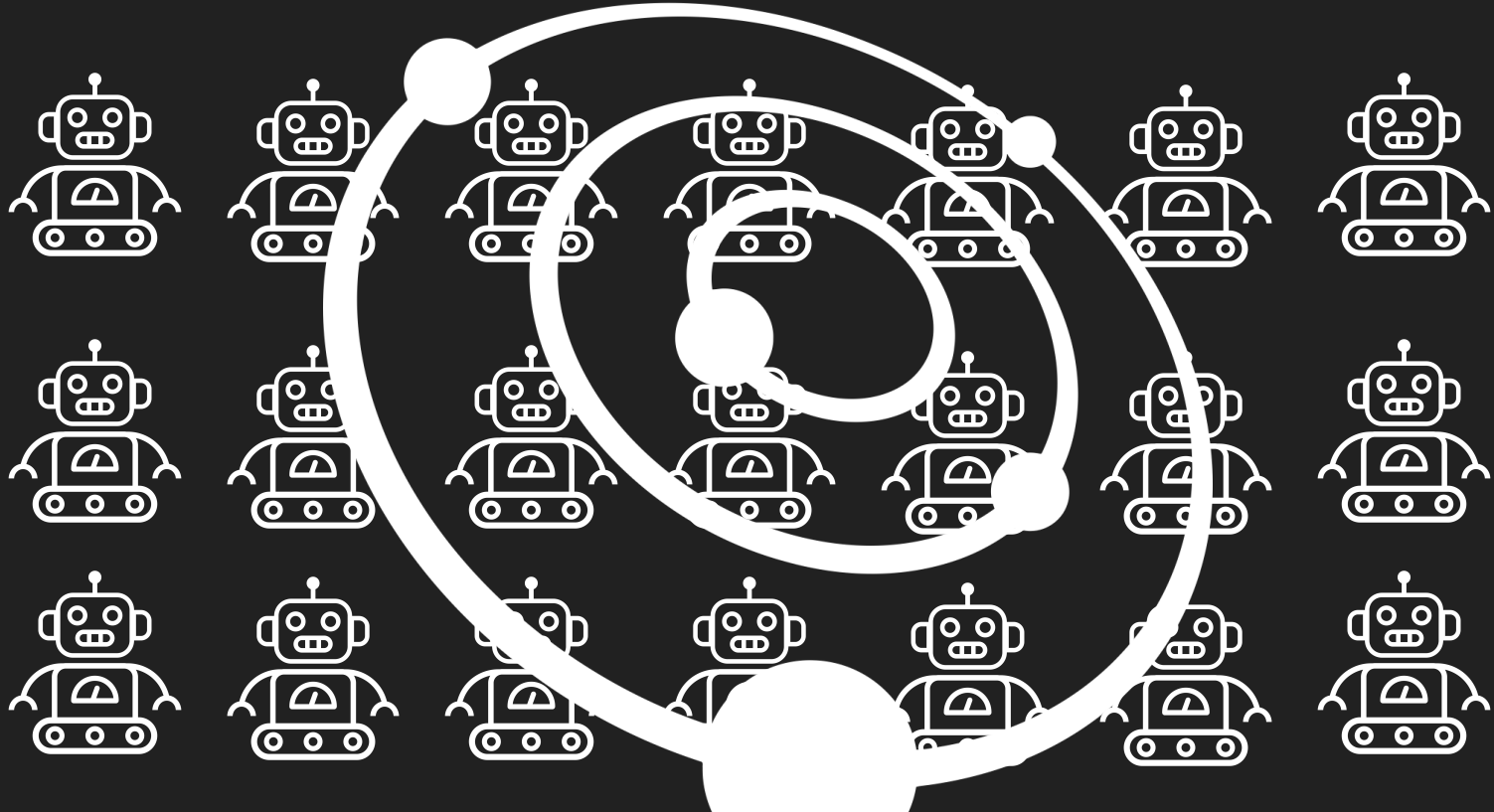


And the importance of change over time...

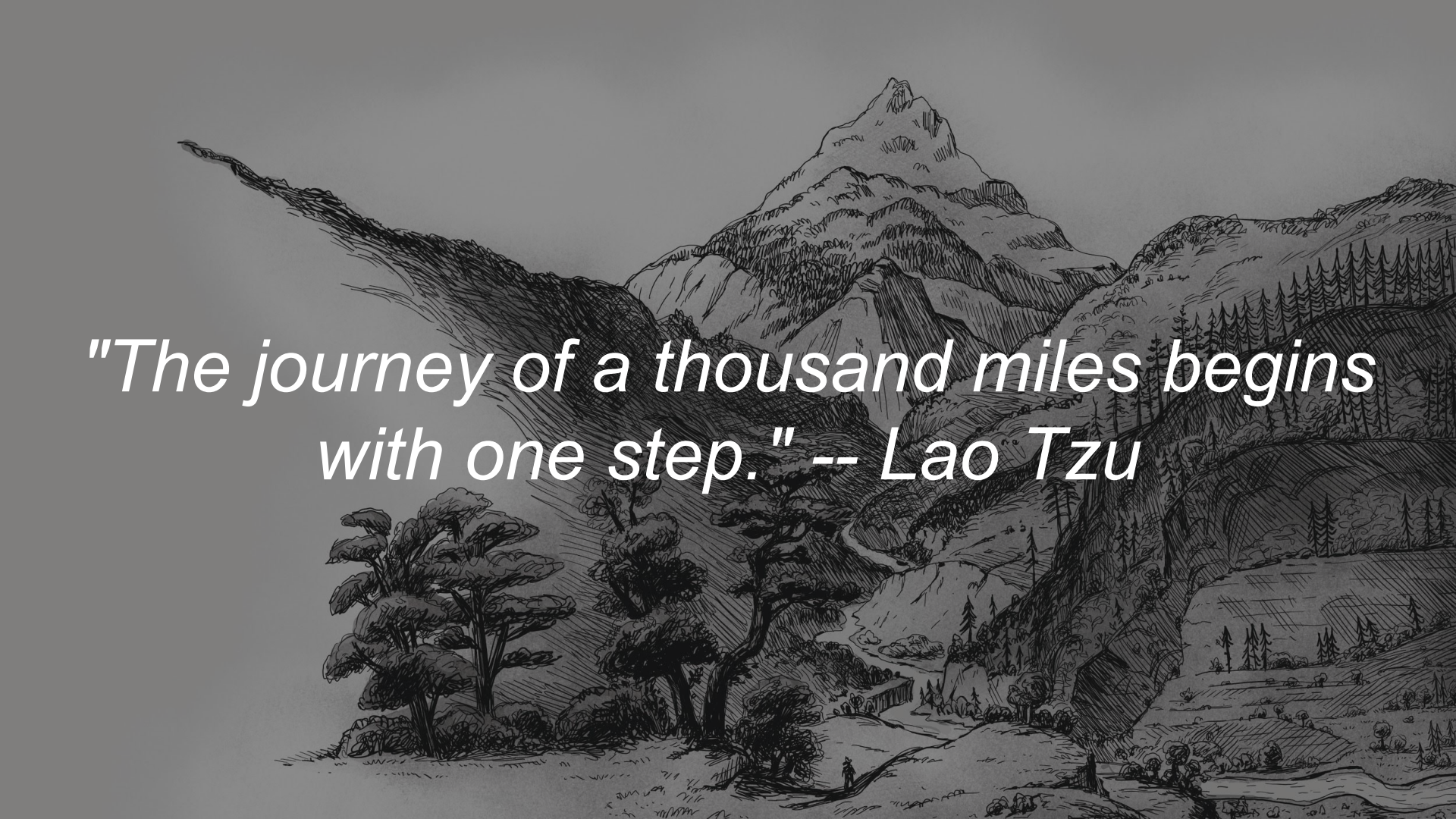
```
*** REQUEST ***  
GET /status HTTP/1.1  
...  
  
*** RESPONSE ***  
200 OK  
...  
{  
  "machinesActive" : "42"  
}
```

```
*** REQUEST ***  
GET /status HTTP/1.1  
...  
  
*** RESPONSE ***  
200 OK  
...  
{  
  "status" : "All OK",  
  "machinesActive" : "42"  
}
```

And the power of runtime service-level discovery...

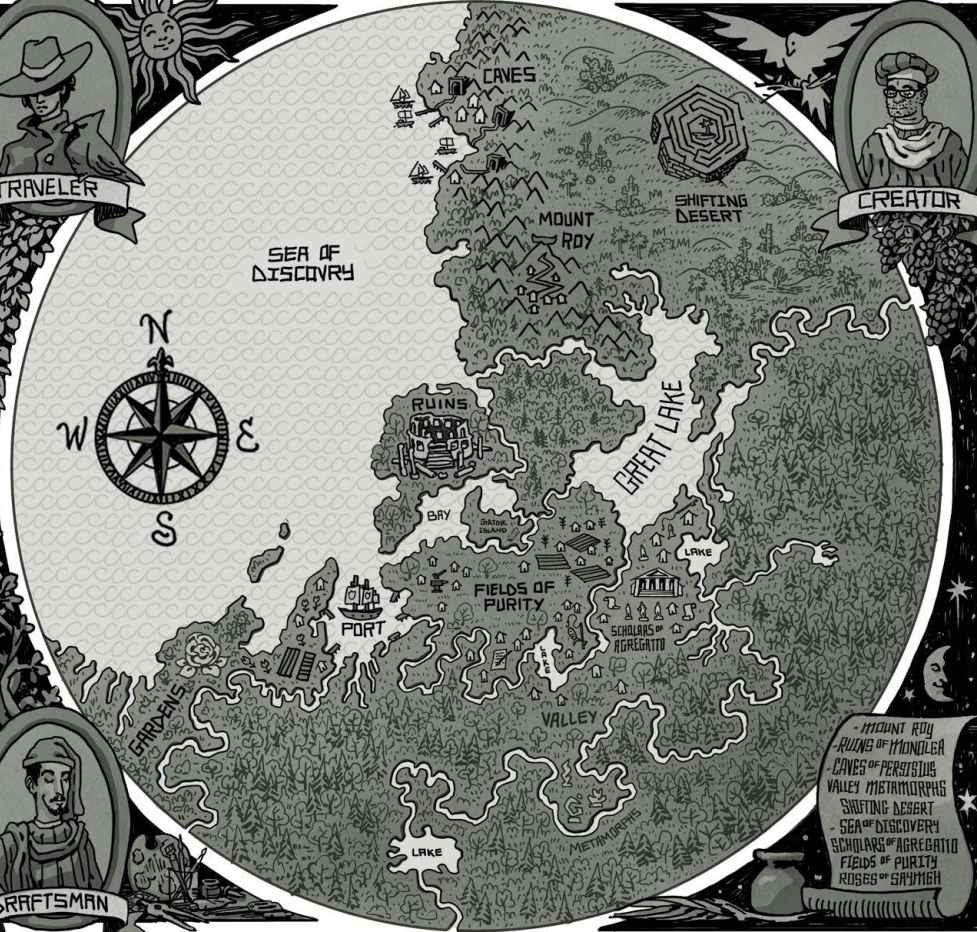
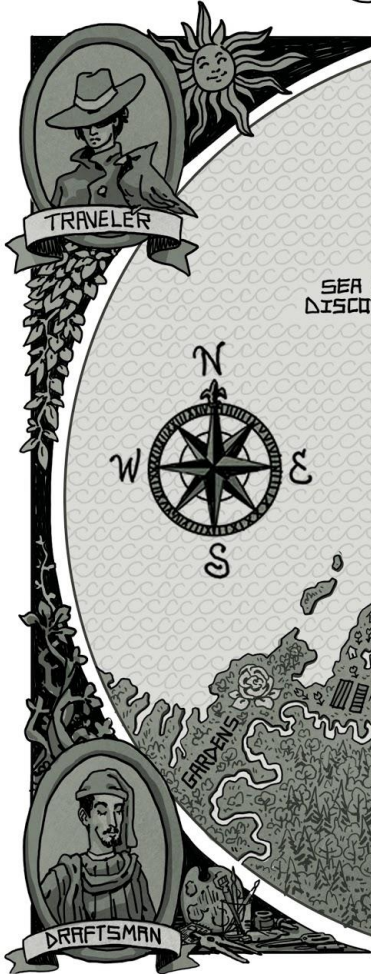


That's a lot!



*"The journey of a thousand miles begins
with one step." -- Lao Tzu*

OMNI TERRA



- MOUNT ROY
 - CAVES OF PERISTUS
 - VALLEY METAMORPHS
 - SHIFTING DESERT
 - SEA OF DISCOVERY
 - SCROLLS OF AGGREGATO
 - FIELDS OF PURITY
 - ROSES OF SHYMEH

OMNI TERRA

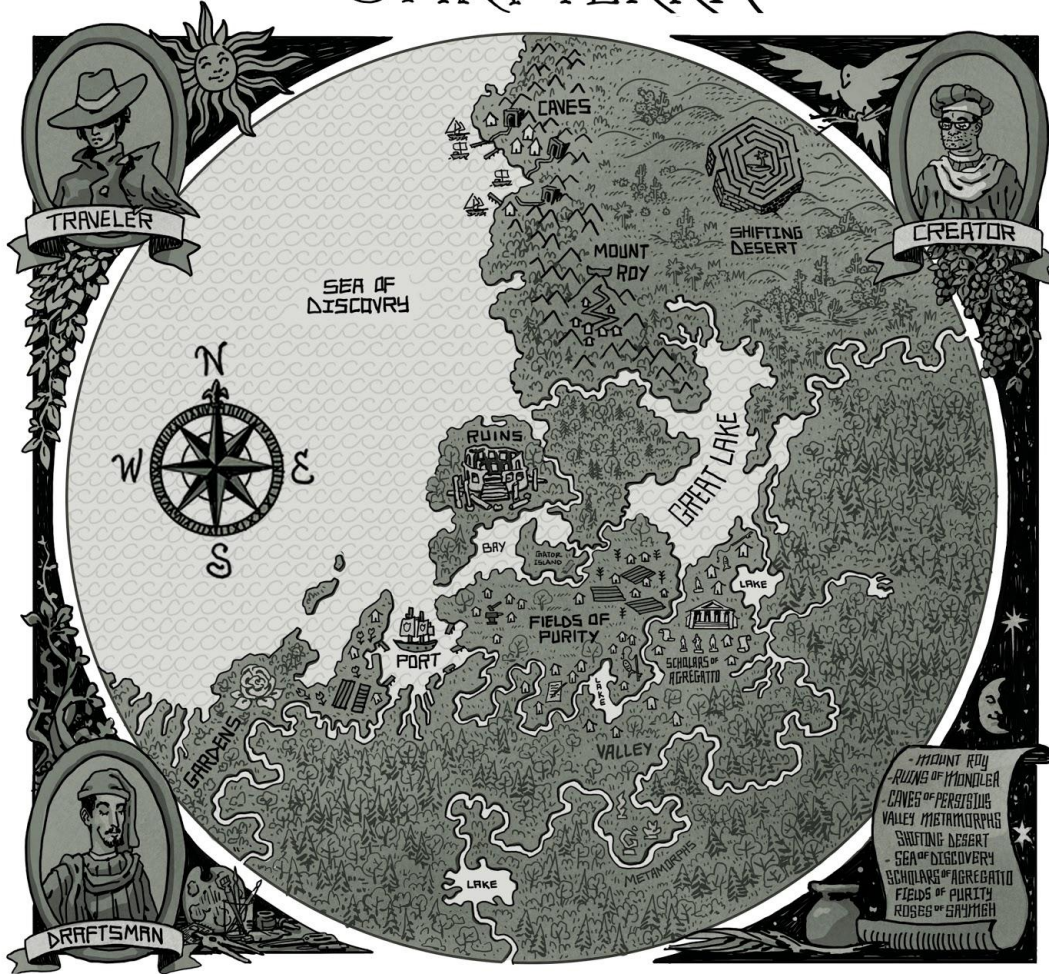
Discovering RESTful Web Microservices

<http://g.mamund.com/2019-04-goto-chicago>

Mike Amundsen
@mamund
training.amundsen.com



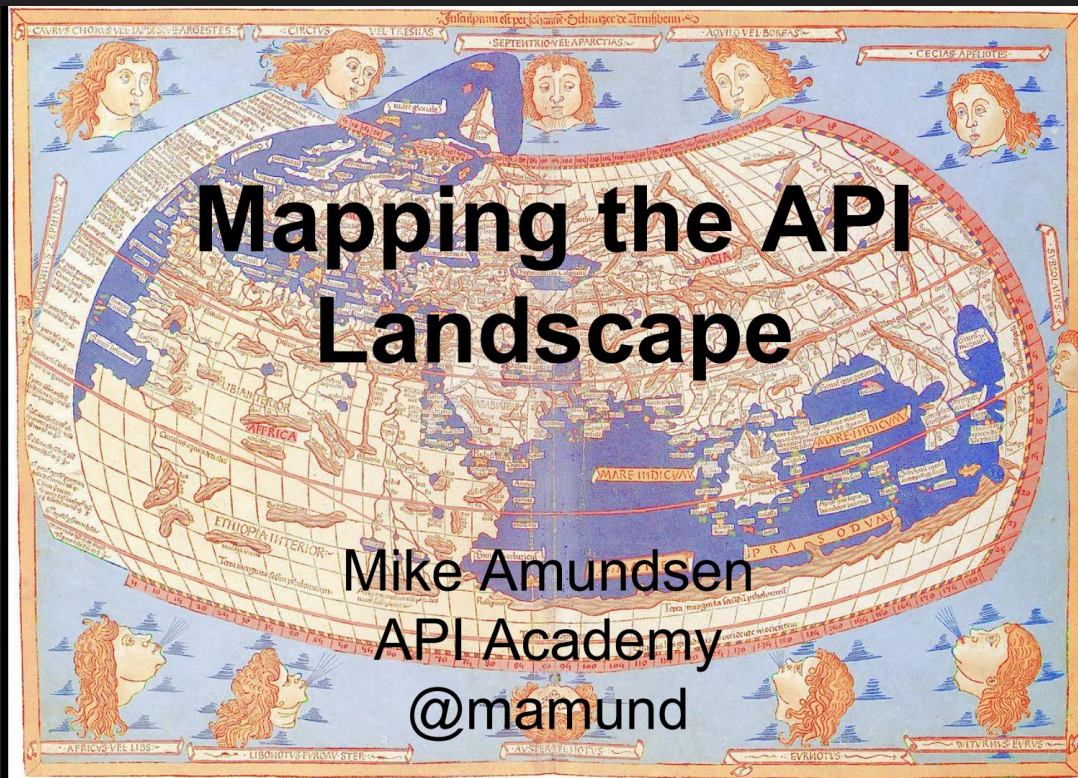
OMNI TERRA



Parking lot

Discovering

A few years ago, in a slide deck far away...



Microservice Characteristics

- Make each program do one thing well
- Expect the output of every program to be the input of another program
- Design and build software to be tried early
- Use tools to lighten the programming task

Unix Operating Principles (1978)

- Make each program do one thing well
- Expect the output of every program to be the input of another program
- Design and build software to be tried early
- Use tools to lighten the programming task

Traveling



Traveling

RedBoot(tm) bootstrap and debug environment [RAM]
(Panasonic Avionics Corporation) release, version ("560328-212" v "1.07" b "0126"
- built 15:35:59, May 22 2013

Platform: SM-02 (I386)
Copyright (C) 2000, 2001, 2002, Red Hat, Inc.

RAM: 0x00000000-0x000a0000, 0x00100000-0x01000000 available
Current Boot Count is 0

Verifying MBR... Fix MBR:
Partition 0: already exists
Partition 1: already exists
Partition 2: already exists
Partition 3: already exists

Verifying image... OK.
== Executing kernel in 5 seconds - enter ^C to abort
Load Address 0x00000000
Image length 0x00e2f5d5
Loading kernel binary...
Read image signature... 1f 8b
Decompressing image...



But Wait...
**There's
MORE!**