Discovering RESTful Web Microservices

Mike Amundsen
@mamund
training.amundsen.com
Rate today’s session

Effective enterprise architecture

Who is this presentation for?
- Architects of all stripes, tech leads, senior developers, and managers

Prerequisite knowledge
- Experience as a senior developer or architect working on software projects

What you’ll learn
- Learn a holistic approach to architecture that explains how to bring business architecture, information architecture, data architecture, application (software) architecture together to have the best chance for your system’s success
- Explore a practical set of architecture practices to create winning technical architectural guidance
- Understand how architecture works effectively with development teams, management, and product management teams through the value chain
- Get usable templates you can start incorporating into your teams immediately

Session page on oreillysacon.com/ny

O’Reilly Events App
training.amundsen.com

Learning with Mike Amundsen

Technical  Transformation  Teams  Inspiration
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
Discovering
A few years ago, in a slide deck far away...

Mapping the API Landscape

Mike Amundsen
API Academy
@mamund
<table>
<thead>
<tr>
<th>Path</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/user</td>
<td></td>
</tr>
<tr>
<td>POST /user.json/cre...</td>
<td>Creates list of users with given input array</td>
</tr>
<tr>
<td>POST /user.json</td>
<td>Create user</td>
</tr>
<tr>
<td>POST /user.json/cre...</td>
<td>Creates list of users with given list input</td>
</tr>
<tr>
<td>PUT /user.json/{...</td>
<td>Updated user</td>
</tr>
<tr>
<td>DELETE /user.json/...</td>
<td>Delete user</td>
</tr>
<tr>
<td>GET /user.json/{...</td>
<td>Get user by user name</td>
</tr>
<tr>
<td>GET /user.json/login</td>
<td>Logs user into the system</td>
</tr>
<tr>
<td>GET /user.json/logout</td>
<td>Logs out current logged in user session</td>
</tr>
<tr>
<td>/pet</td>
<td></td>
</tr>
<tr>
<td>GET /pet.json/{petI...</td>
<td>Find pet by ID</td>
</tr>
<tr>
<td>POST /pet.json</td>
<td>Add a new pet to the store</td>
</tr>
<tr>
<td>PUT /pet.json</td>
<td>Update an existing pet</td>
</tr>
<tr>
<td>GET /pet.json/findB...</td>
<td>Finds Pets by status</td>
</tr>
<tr>
<td>Feature</td>
<td>Town</td>
</tr>
<tr>
<td>-----------------</td>
<td>---------------</td>
</tr>
<tr>
<td>desert/wasteland</td>
<td>town/village</td>
</tr>
<tr>
<td>grassland</td>
<td>cities</td>
</tr>
<tr>
<td>swamp</td>
<td>tower/fortress</td>
</tr>
<tr>
<td>hills</td>
<td>castle</td>
</tr>
<tr>
<td>mountains</td>
<td>country capital</td>
</tr>
<tr>
<td>cave</td>
<td>windmill</td>
</tr>
<tr>
<td>political border</td>
<td>forest</td>
</tr>
<tr>
<td>road</td>
<td>lake</td>
</tr>
<tr>
<td>river</td>
<td>reef</td>
</tr>
<tr>
<td></td>
<td>canyon</td>
</tr>
</tbody>
</table>
RESTful Web Microservices?
The words hypertext, hyperlink, and hypermedia were coined by Ted Nelson around 1965.
HTTP and HTML

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

-- Tim Berners-Lee, 1989
"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
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

Loosely-coupled components running in an engineered system.
Traveling
Traveling
Traveling the Network
Programming the Network
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.

Programming the Network

"There is no simultaneity at a distance."

-- Pat Helland (2005)
“Bugs will happen. They cannot be eliminated, so they must be survived instead.”

-- Michael T. Nygard
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...
Let's talk about code for a bit...
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

```javascript
// 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

1. Fail-Fast

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

```javascript
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;
}
```
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…)

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);
    }, request.timeBudget);
    return response;
}
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.
BUT WAIT, there’s more!
Let *not* talk about code for a bit…
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, Sign, and Symbol

- **Signal:** Protocol
- **Sign:** Format
- **Symbol:** Vocabulary
Signal, Sign, and Symbol

- **Signal**: Protocol
  *HTTP, CoAP, etc.*

- **Sign**: Format
  *HTML, HAL, etc.*

- **Symbol**: Vocabulary
  *ALPS, DCAP, etc.*
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...

```plaintext
*** 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...

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

*** RESPONSE ***
200 OK
...
{
  "machinesActive" : "42"
}
```

```plaintext
*** 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"
}
What is the best practice for versioning a REST API?
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
SEA OF DISCOVERY
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
ADVANCED RESEARCH PROJECTS AGENCY
Washington, 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 2 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 16th, 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 independent. 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
DNS for discovering machines
But we need to discover services...
Discovering Interoperative Services for Continuous Operation (DISCO)

Mike Amundsen
<mca@amund.com>

Table of Contents
Status
Summary
DISCO Resources
DISCO Design Goals
Service-Id Implementation for NodeJS
The DISCO Registry Specification
DISCO Registry Service Resources 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/rambook/registry-docs

Last Updated

2018-03-10

Summary

This document details an open 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/implmenters may find things “missing” or “undescribed.” 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

ServiceA

registry

ServiceA is now ’findable’

find(services)

serviceCollection

ServiceA now has pointers to dependent services

bind(sourceRegID, targetRegID)

bbToken

ServiceA ”binds” to another Service

health()

healthURL()

ServiceA returns health stats

renew(regID)

ServiceA can auto-renew its own registration

unregister(regID)

ServiceA is dropped from the Registry

Registry

open-disco.org
Discovering Interoperative 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 is is still up and running
- `unregister`: remove a service from the registry
// 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,

```javascript
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