

Microservices and the art of taming the Dependency Hell Monster

Michael Bryzek
Cofounder & ex-CTO Gilt
@mbryzek
mbryzek@alum.mit.edu

Dependency Hell

- What is it and how does it happen?
- How do we mitigate?
 - API design must be First Class
 - Backward and Forward Compatibility
 - Accurate Documentation
 - Generated client libraries

Dependency hell is a colloquial term for the frustration of some software users who have installed software packages which have dependencies on specific versions of other software packages.

http://en.wikipedia.org/wiki/Dependency_hell

Example

service a depends on lib-foo version 1.7

service b depends on lib-foo version 1.6

Build pulls in version 1.7.

At runtime, turns out there was a breaking change in lib-foo that the compiler could not verify.

Long chains of dependencies make this hard:

service a depends on

lib-foo depends on

lib-bar depends on

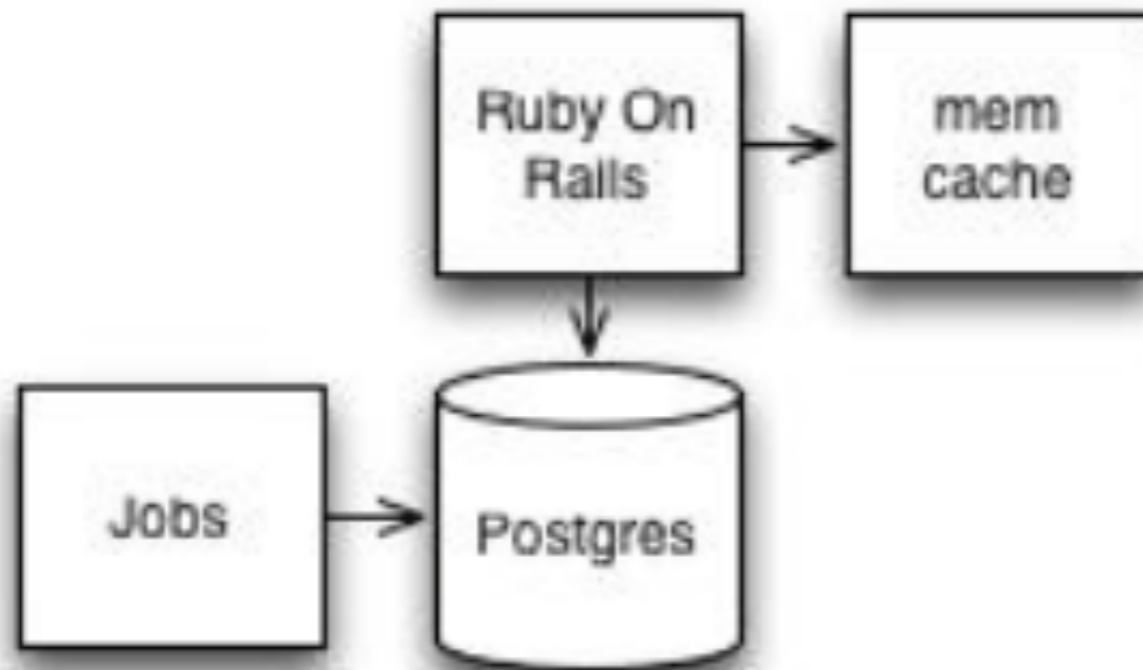
lib-baz

Real World Dependency Injection

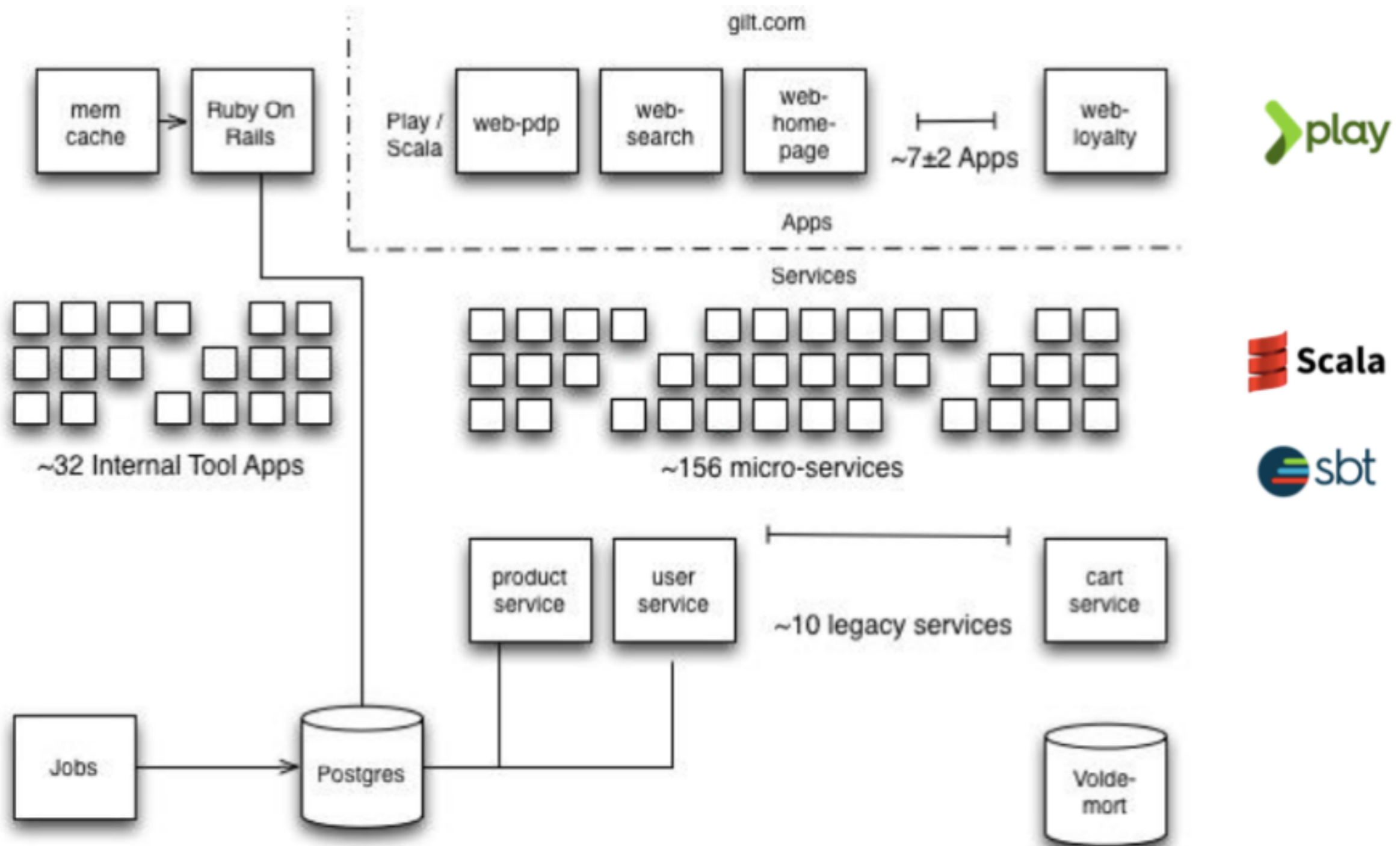


Dependency hell!

From Simple Architecture



To Fully Distributed



0 to 150+ People in Tech



How do you manage dependencies?

And specifically those dependencies in the libraries we use.

Let's Build an App

GILT

WOMEN MEN BABY & KIDS HOME CITY

Sign In | Register

\$1,116.21 credits available

0 Cart

Search



The Father's Day Event: Our best brands, one-of-a-kind finds, and can't-fail gift cards for every deserving dad [Buy Now](#)

WOMEN / VINTAGE FENDI HANDBAGS / Fendi Pink Leather 2Jours Medium



Just One



Fendi
Fendi Pink Leather 2Jours
Medium

\$1,990

Color: Pink



Qty: 1

You have \$1,116.21 credits available

Add to Cart

Add to Waitlist

This item is final sale and non-returnable.

Estimated Delivery: Fri 06/12/15 - Wed 06/17/15

Description: Pink Leather 2Jours Medium

- Condition: Carried - Excellent (minor signs of handling on leather/hardware; little to no signs of wear)
- Leather
- Gold-tone hardware
- Double top handles and adjustable shoulder strap

The Basics

- User registration and login
- Product catalog
- Inventory data
- Cart

user-service

- API to create a user; fetch user details
- High throughput: 10k RPS+
- Millions of users



user-service client lib

```
createUser(form: UserForm): Future[User]
getUser(guid: UUID): Future[Option[User]]
deactivateUser(guid: UUID): Future[Unit]
updateUser(guid: UUID, form: UserUpdateForm):
  Future[Unit]
authenticate(email: String, password: String):
  Future[Boolean]
```

...

catalog-service

- API to fetch rich product details
- Moderate throughput: 5k RPS+
- Millions of products



catalog-service client lib

```
getProduct(id: Long): Option[Product]
```

```
getProductsInSale(saleId: Long, limit: Int,  
offset: Int): List[Product]
```

```
getSkusForProduct(productId: Long): List[Sku]
```

```
...
```


inventory-service

- API to check stock of individual products
- High throughput: 10k RPS+
- Guarantee never oversold



inventory-service client lib

`numberAvailable(id: Long): Long`

`reserve(id: Long): Token`

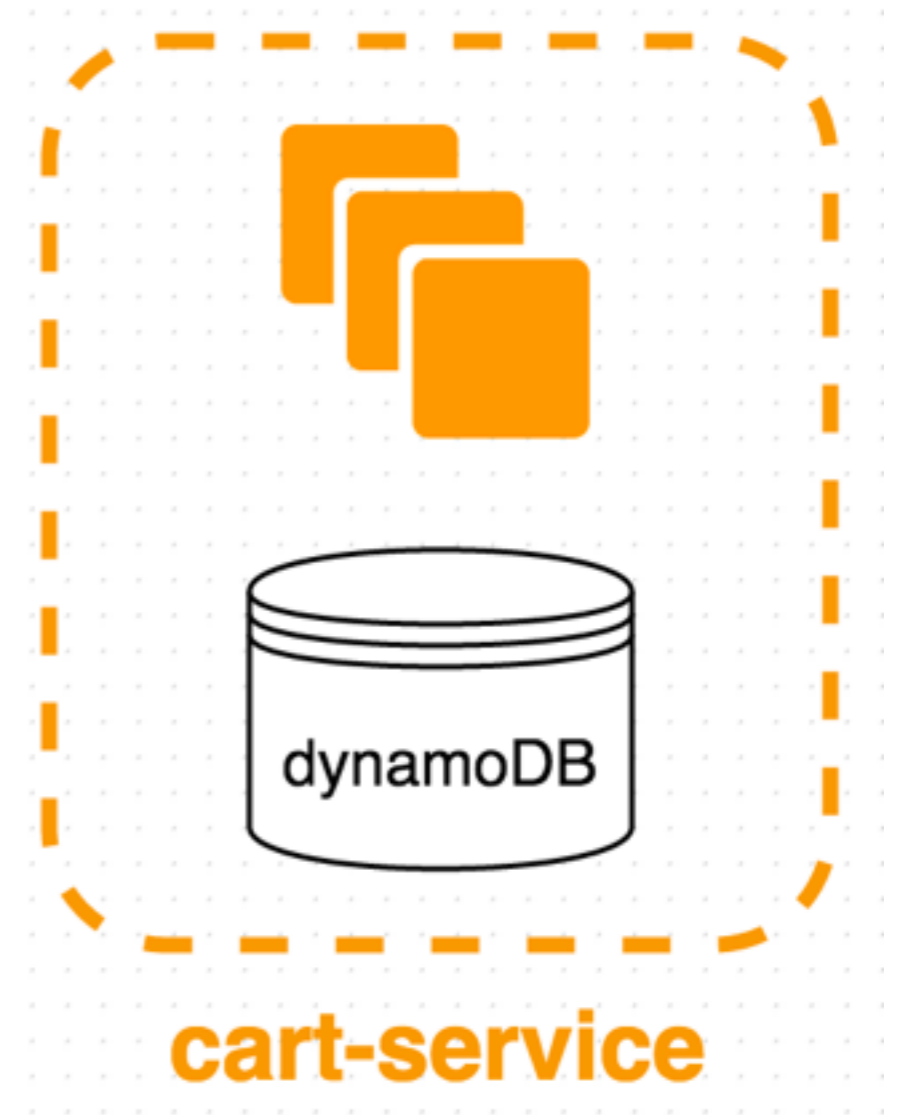
`clearReservation(token: Token)`

`lock(reservationToken: Token, externalId: UUID)`

...

cart-service

- API to add/remove to a shopping cart
- Integrates with checkout
- Low throughput



cart-service client lib

`addToCart(id: String, skuId: Long)`

`getCart(id: String): Cart`

`clearCart(id: String)`

`addToUserCart(userGuid: UUID, skuId: Long)`

`getUserCart(userGuid: UUID): Cart`

`clearUserCart(userGuid: UUID)`

...

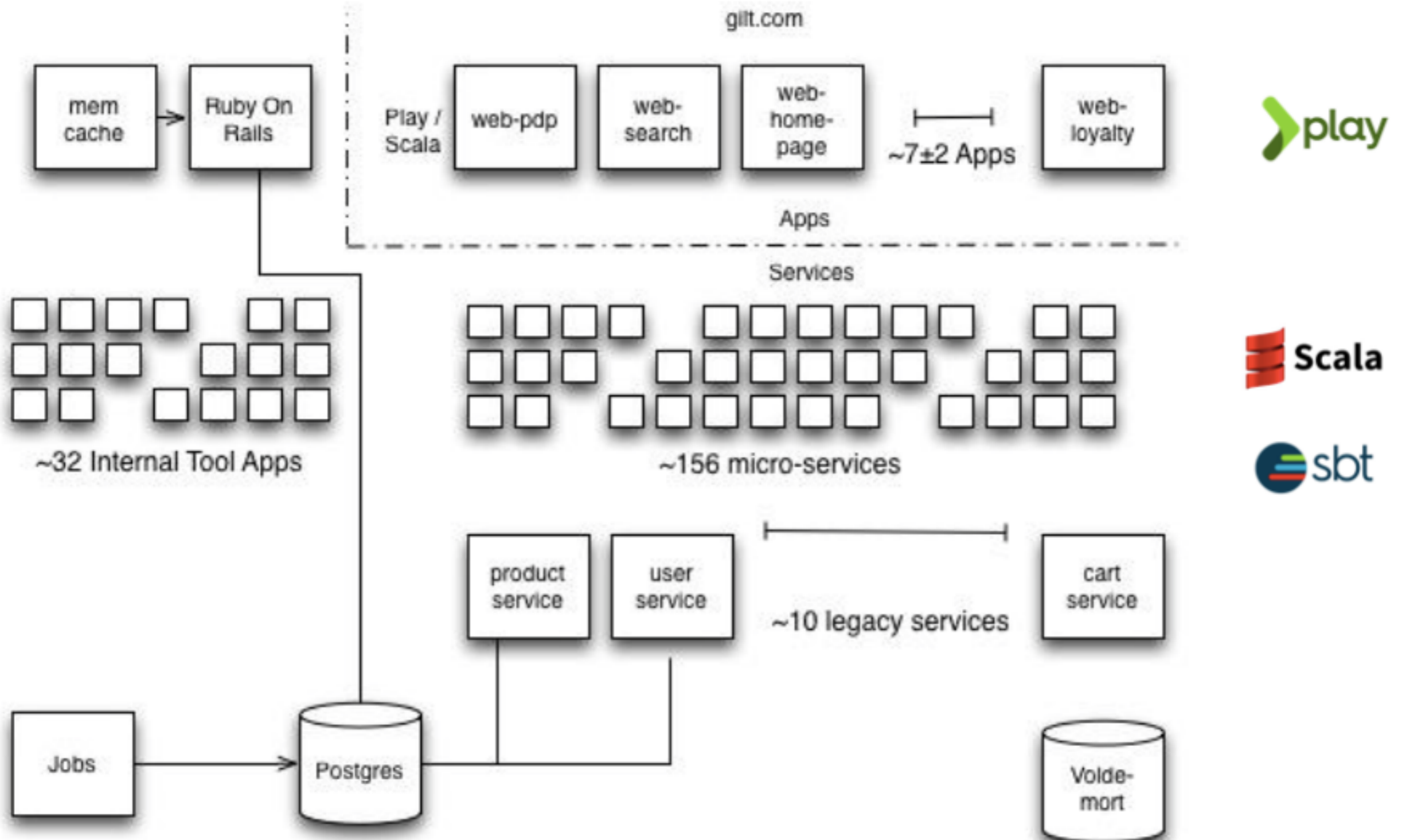
Service	Year of Latest Update	Client Dependencies	Futures?	Example Methods
user	2015	Scala 2.11, Ning 1.9	Yes	createUser, deactivate
catalog	2013	Scala 2.10, Ning 1.7	No	createProduct
inventory	2009	Java 6, Netty HTTP client.	No	reserve, lock
cart	2008	Java 6, Apache HTTP Client.	No	addToCart

Then We Add Features

- Loyalty
- Recommendation
- Account Credits
- Nav bar with context, related sales
- Tracking
- and dozens more...

And with micro service architectures,
significant new features often lead to
new services and new libraries.

Mature Microservice Arch



What happens next?

- Builds get larger and slower
- Create new client libraries that are each just a little bit different
- Produce custom APIs that reduce interoperability
- Increase amount of boilerplate code
- Reduce code quality; slow down development
- And Eventually you will see a production error

Caused by:

`java.lang.NoSuchMethodError`

Minimizing the Pain

- API design must be First Class
- Backward and Forward Compatibility
- Accurate Documentation
- Generated client libraries

Guiding Principle: The Open Source Way

- How do applications integrate with each other?
- How do you use a library?
- How much and what kind of documentation?
- How do I get support / contribute / report bugs?
- Public or Private is a detail

Tooling Matters

- www.apidoc.me codifies these practices
- very simple to get use
- zero dependencies on existing software process nor runtime
- Open source and free SAAS: <https://github.com/mbryzek/apidoc>
- First commit April 6, 2014.
- Over 100 services already built at Gilt w/ apidoc

API Design Must be First Class

- Protobufs, thrift, avro, swagger 2.0, and apidoc
- The design of your API and the data structures themselves are the hardest things to change
- Design them up front - and integrate these artifacts into your design process.

Example: AVRO idl

```
@namespace("mynamespace")
protocol User {
  record Employee {
    string email;
  }
}
```

Example: apidoc

```
{  
  "name": "user-service",  
  "models": {  
    "user": {  
      "fields": [  
        { "name": "id", "type": "uuid" },  
        { "name": "email", "type": "string" }  
      ]  
    }  
  }  
}
```


“Schema First Design”

Really the most important concept

Accurate Documentation

- What services exist? Think of how github helps us discover what libraries and applications exist.
- API as first class allows us to use these artifacts directly in our software - ensures accuracy
- Semantic Versioning (<http://semver.org/>)

Michael

www.apidoc.me/bryzek

apidoc Documentation github mbryzek Search


Bryzek

Subscriptions

Org Details

Bryzek

[+ Add application](#)

apidoc api	Host API documentation for applications providing REST APIs, facilitating the design of good...
apidoc Example Union Types	
apidoc generator	Documentation for an apidoc code generator API
apidoc internal	Internal models used in the implementation of apidoc
apidoc spec	
budget 	
Trainer	

apidoc api 0.9.21 x Michael

www.apidoc.me/bryzek/apidoc-api/0.9.21

apidoc Documentation github michael Search

Bryzek Subscriptions Org Details apidoc api Settings original (api.json) service.json Resources application change code com.bryzek.apidoc.generatc domain email_verification_confirmat generator_service healthcheck

apidoc api 0.9.21

0.9.21

[Upload new version](#) | [Delete this version](#) | [Watching](#)

Host API documentation for applications providing REST APIs, facilitating the design of good resource first APIs.

- Contact: [Michael Bryzek http://twitter.com/mbryzek](http://twitter.com/mbryzek)
- License: [MIT](#)

Resources

application

Operations

Method and Path	Description
GET /:orgKey	Search all applications. Results are always paginated.
POST /:orgKey	Create an application.
PUT /:orgKey/:applicationKey	Updates an application.
DELETE /:orgKey/:applicationKey	Deletes a specific application and its associated versions.
POST /:orgKey/:applicationKey/move	Moves application to a new organization.

change

Operations

Backward Compatibility

- Imagine storing all historical records
- General guidelines:
 - New fields are either optional or have defaults
 - Can't rename; Introduce new models where necessary and migration path

Forward Compatibility

- Imagine new messages arrive with new data
- Additional considerations:
 - Careful of enums; consider what happens when you add a value in the future
 - Careful with processing data (e.g. throwing an exception if an unknown field shows up)

Forward Compatible Enum

```
sealed trait OriginalType
```

```
object OriginalType {
```

```
  case object ApiJson extends OriginalType { override def toString = "api_json" }
```

```
  /**
```

```
   * UNDEFINED captures values that are sent either in error or  
   * that were added by the server after this library was  
   * generated. We want to make it easy and obvious for users of  
   * this library to handle this case gracefully.
```

```
   *
```

```
   * We use all CAPS for the variable name to avoid collisions  
   * with the camel cased values above.
```

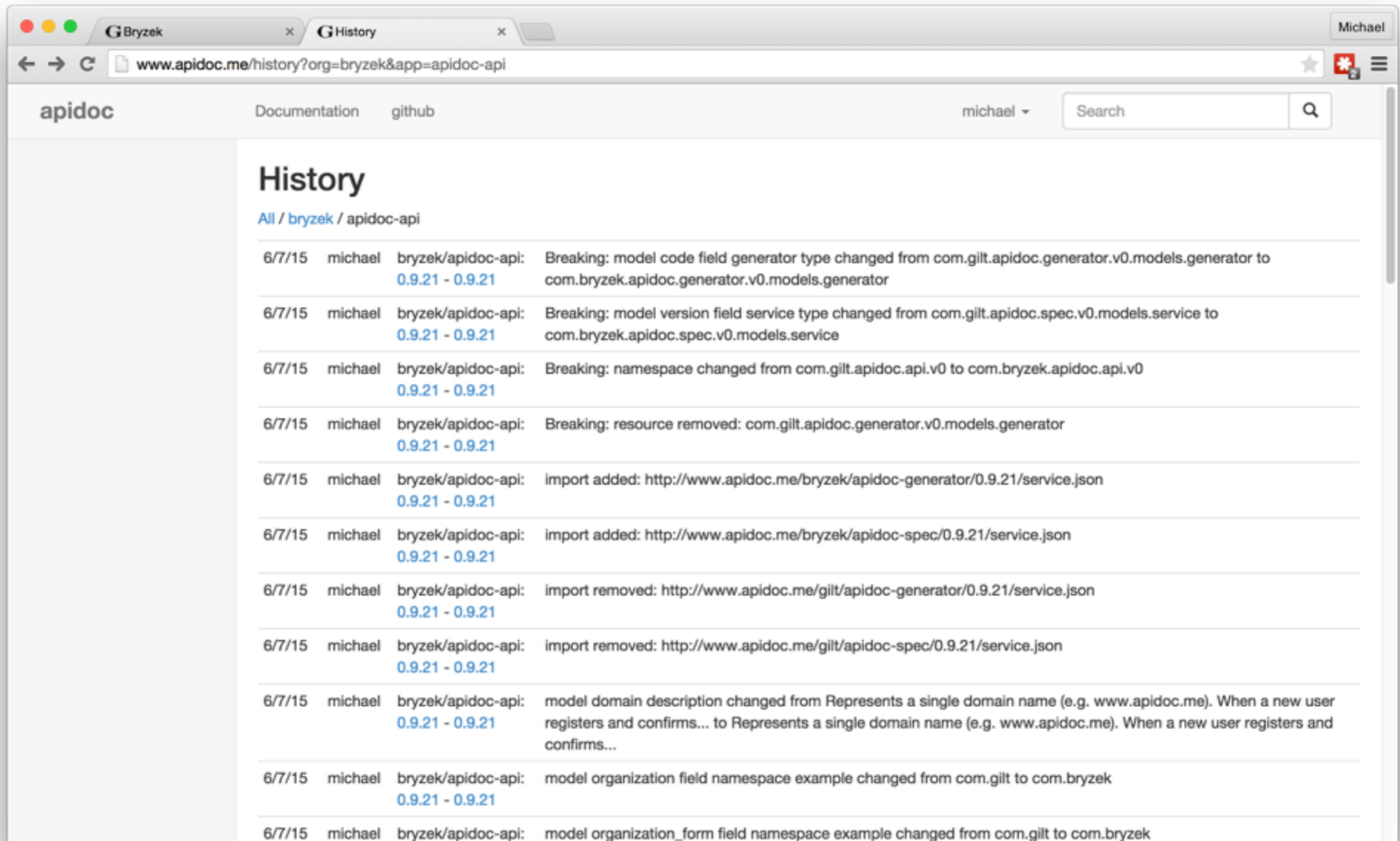
```
  */
```

```
  case class UNDEFINED(override val toString: String) extends OriginalType
```

```
  ...
```

```
}
```

Knowing When Things Change



The screenshot shows a web browser window with two tabs: 'Bryzek' and 'History'. The address bar shows the URL `www.apidoc.me/history?org=bryzek&app=apidoc-api`. The page header includes the 'apidoc' logo, navigation links for 'Documentation' and 'github', a user profile for 'michael', and a search bar. The main content area is titled 'History' and shows a list of changes for the 'bryzek/apidoc-api' repository. Each entry includes a date (6/7/15), the user 'michael', the repository name, a version range (0.9.21 - 0.9.21), and a description of the change.

Date	User	Repository	Version Range	Description
6/7/15	michael	bryzek/apidoc-api	0.9.21 - 0.9.21	Breaking: model code field generator type changed from <code>com.gilt.apidoc.generator.v0.models.generator</code> to <code>com.bryzek.apidoc.generator.v0.models.generator</code>
6/7/15	michael	bryzek/apidoc-api	0.9.21 - 0.9.21	Breaking: model version field service type changed from <code>com.gilt.apidoc.spec.v0.models.service</code> to <code>com.bryzek.apidoc.spec.v0.models.service</code>
6/7/15	michael	bryzek/apidoc-api	0.9.21 - 0.9.21	Breaking: namespace changed from <code>com.gilt.apidoc.api.v0</code> to <code>com.bryzek.apidoc.api.v0</code>
6/7/15	michael	bryzek/apidoc-api	0.9.21 - 0.9.21	Breaking: resource removed: <code>com.gilt.apidoc.generator.v0.models.generator</code>
6/7/15	michael	bryzek/apidoc-api	0.9.21 - 0.9.21	import added: http://www.apidoc.me/bryzek/apidoc-generator/0.9.21/service.json
6/7/15	michael	bryzek/apidoc-api	0.9.21 - 0.9.21	import added: http://www.apidoc.me/bryzek/apidoc-spec/0.9.21/service.json
6/7/15	michael	bryzek/apidoc-api	0.9.21 - 0.9.21	import removed: http://www.apidoc.me/gilt/apidoc-generator/0.9.21/service.json
6/7/15	michael	bryzek/apidoc-api	0.9.21 - 0.9.21	import removed: http://www.apidoc.me/gilt/apidoc-spec/0.9.21/service.json
6/7/15	michael	bryzek/apidoc-api	0.9.21 - 0.9.21	model domain description changed from Represents a single domain name (e.g. www.apidoc.me). When a new user registers and confirms... to Represents a single domain name (e.g. www.apidoc.me). When a new user registers and confirms...
6/7/15	michael	bryzek/apidoc-api	0.9.21 - 0.9.21	model organization field namespace example changed from <code>com.gilt</code> to <code>com.bryzek</code>
6/7/15	michael	bryzek/apidoc-api	0.9.21 - 0.9.21	model organization_form field namespace example changed from <code>com.gilt</code> to <code>com.bryzek</code>

Generating Client Libraries

- Potentially controversial; I was skeptical at first, but works
- Enables consistent naming
- Minimal external dependencies
- Challenge: Can you generate a client that developers love?

Clients

[Ning Async Http Client 1.8](#)

[Ning Async Http Client 1.9](#)

[Play 2.2 client](#)

[Play 2.3 client](#)

[Play 2.x json](#)

[Play 2.x routes](#)

[Ruby client](#)

[Scala models](#)

apidoc api 0.9.21

0.9.21 ▾

[Upload new version](#) | [Delete this version](#) | [Watch](#)

Host API documentation for applications providing REST APIs, facilitating the design of good resource first APIs.

- Contact: [Michael Bryzek](#) <http://twitter.com/mbryzek>
- License: [MIT](#)

Resources

application

Operations

Method and Path	Description
GET /:orgKey	Search all applications. Results are always

apidoc Ruby Client

```
client = MyService::Client.new("http://localhost:8000")

organizations = client.organizations.get(:limit => 10, :offset => 0)
organizations.each do |org|
  puts "Org %s is named %s" % [org.id, org.name]
end

neworg = client.organizations.post(:name => "My org")
puts "Created new org named %s" % neworg.name
```

apidoc Scala Client

```
val client = new com.bryzek.apidoc.api.v0.Client("http://localhost")
```

```
val organizations = client.organizations.get(limit = 10, offset = 0)
```

```
organizations.foreach { org =>
```

```
    println(s"Org ${org.name} is named ${org.id}")
```

```
}
```

```
val neworg = client.organizations.post(name = "My org")
```

```
println(s"Created new org named ${neworg.name}")
```

Consistency Really Matters

Original Consistent Naming based on REST

createUser POST /users/
Users.post

createProduct POST /products/
Products.post

reserve POST /reservations/
Reservations.post

addToCart POST /carts/:id/products/:productId
Products.postByIdAndProductId(id, productId, ...)

Summary: Mitigate Dependency Hell

- API design must be First Class
- Backward and Forward Compatibility
- Accurate Documentation
- Generated client libraries

Thank You

www.apidoc.me/doc/start

Michael Bryzek
@mbryzek
mbryzek@alum.mit.edu