Creating a World-Class RESTful Web Services API

By David Keener

http://www.keenertech.com
dkeener@keenertech.com

But First, Who Am I?

David Keener

I'm a technical architect and writer with over 20 years of experience. Been doing web applications Since 1997, and Rails applications since version 1.1.

I'm a Technical Architect for **Grab Networks**, the company known for streaming the Beijing Olympics over the web and for distributing more news videos in the US than any other company except MSNBC.

- **Blog:** http://www.keenertech.com
  (New Rails-based version in late June)
- **Email:** dkeener@keenertech.com

Overview

I'm talking about the practical experiences gained from creating a real, RESTful Web Services API for use by external customers.

- One Minute RESTful Refresher
- Why would you want a RESTful API?
- Basic design steps for an API
- API Architecture Details
- Scalability
- Practical Tips

What's the Big Deal?

Rails has been RESTful since 1.2…what's so hard about doing an API?

```
def index
  @videos = Video.find(:all)

  respond_to do |wants|
    wants.xml { render :layout => false,
                       :xml => @videos.to_xml }
    wants.json { render :layout => false,
                       :json => @videos.to_json }
  end
end
```

What's the Big Deal?

Rails has been RESTful since 1.2…what's so hard about doing an API?

- No Authentication

```
def index
  @videos = Video.find(:all)

  respond_to do |wants|
    wants.xml { render :layout => false,
                       :xml => @videos.to_xml }
    wants.json { render :layout => false,
                       :json => @videos.to_json }
  end
end
```

What's the Big Deal?

Rails has been RESTful since 1.2…what's so hard about doing an API?

- No Authentication
- No Authorization

```
def index
  @videos = Video.find(:all)

  respond_to do |wants|
    wants.xml { render :layout => false,
                       :xml => @videos.to_xml }
    wants.json { render :layout => false,
                       :json => @videos.to_json }
  end
end
```

What's the Big Deal?

Rails has been RESTful since 1.2…what's so hard about doing an API?

- No Authentication
- No Authorization
- No Error Handling

```
def index
  @videos = Video.find(:all)

  respond_to do |wants|
    wants.xml { render :layout => false,
                       :xml => @videos.to_xml }
    wants.json { render :layout => false,
                       :json => @videos.to_json }
  end
end
```

What's the Big Deal?

Rails has been RESTful since 1.2…what's so hard about doing an API?

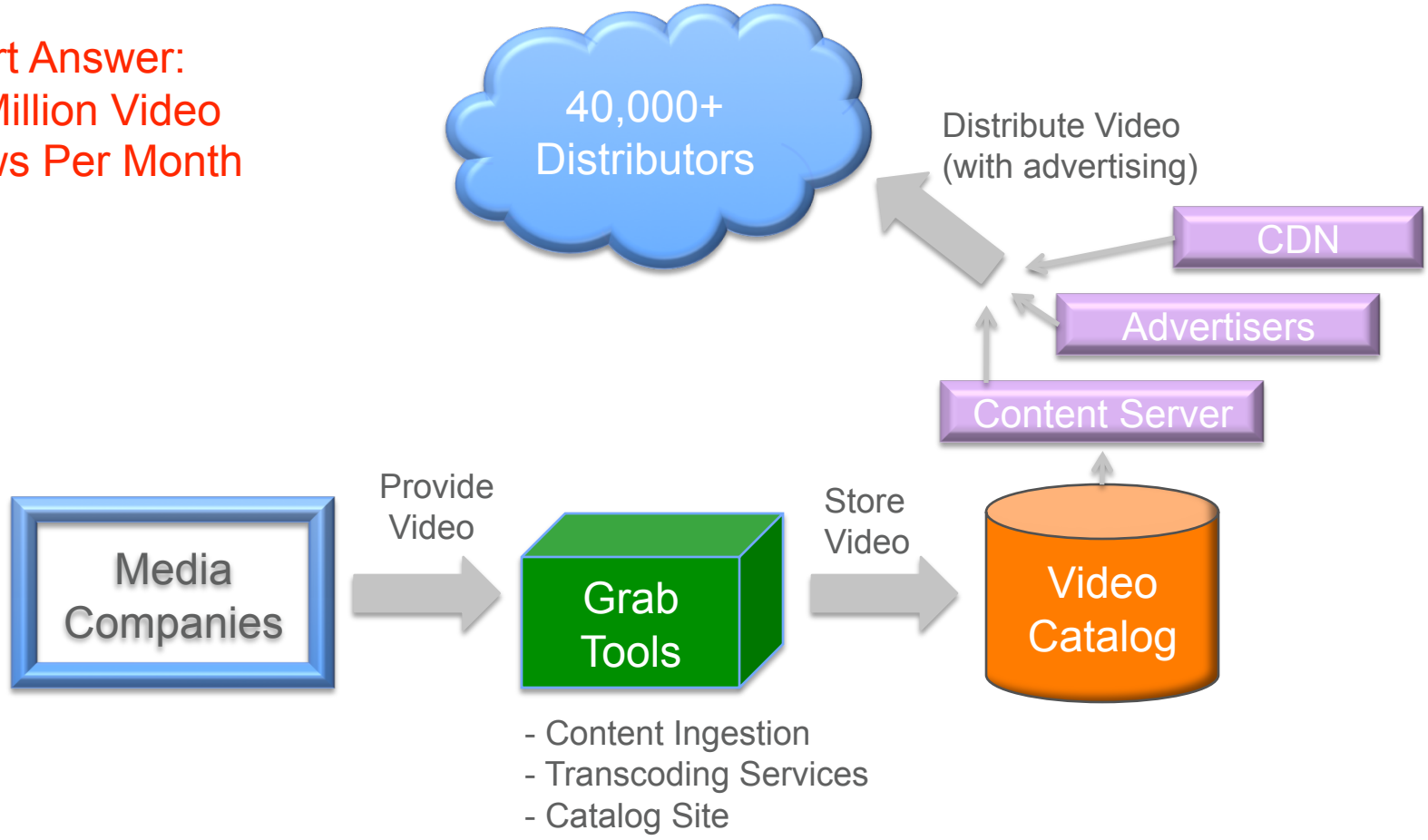
- No Authentication
- No Authorization
- No Error Handling
- No fine control over data elements

```
def index
  @videos = Video.find(:all)

  respond_to do |wants|
    wants.xml { render :layout => false,
                       :xml => @videos.to_xml }
    wants.json { render :layout => false,
                       :json => @videos.to_json }
  end
end
```

What Does Grab Networks Do?

★ Short Answer:
60 Million Video
Views Per Month



Where Does the API Fit In?

- Will allow distributors to integrate video content into their sites more effectively
- Will provide a better platform on which to build our own tools
- Will facilitate a level of innovation that could not exist before

Washington Times

Video content integrated in via RESTful API (Beta)

One-Minute RESTful Refresher

URL	HTTP Method	Operation
http://example.com/providers	GET	Get a list of content providers.
http://example.com/providers	POST	Create a new content provider.
http://example.com/providers/1	GET	Show data for provider 1.
http://example.com/providers/1	PUT	Update data for provider 1.
http://example.com/providers/1	DELETE	Delete provider 1.

- HTTP methods are the “verbs”
- Acting on Resources (“nouns”)
- Providing a simple, constrained, easy-to-understand interface

Reasons for Doing an API

Know WHY you're doing an API before you embark on creating one. Most reasons fall into two basic categories:

- Customer-Centric: To better serve customers
- Company-Centric: To better serve yourself

Customer-Centric Reasons

- To give customers direct access to content
- To facilitate innovation by giving customers the capability to leverage your resources for their own purposes
- To allow customers to explore your content more effectively
- To allow authorized customers to directly manipulate resources

Company-Centric Reasons

- To organize functionality in a more manageable, maintainable way
- To centralize key logic components, e.g. – authorization logic
- To facilitate the creation of your own tools
- To leverage the creativity and innovativeness of your customers
- To promote tighter coupling of customer applications with API, resulting in an enhanced exit barrier

The First Step…

All important projects must have a codename

Like….

- Tiger
- Leopard
- Longhorn

The First Step…

All important projects must have a codename

Like….

- Tiger
- Leopard
- Longhorn (um, maybe not)

The First Step…

All important projects must have a codename

Like….

- Tiger
- Leopard
- ~~Longhorn~~

The First Step…

All important projects must have a codename

Like….

Designing the API

Designing a RESTful API is an interesting challenge. Forget your existing database, and start at the logical level…

- Identify the basic objects in your problem domain
  - These are your candidate resources
- Identify the relationships between your resources
  - These help you define nesting
- Look for “actions” that need to become “nouns”
  Ex. – subscription (a standard example)
  Ex. – Publishing a video results in a “distribution”
- Stay in Beta a long time…you will make changes

Reality Sets In

This is where you compare your nice, clean, elegant resource design with your ugly, grown-over-time database

- Your database wasn't designed to have a RESTful API built on top of it
- Your database probably doesn't support the authorization needs of your API
- So you're going to need a massive re-factor (or a series of them)
- And management will still want you to develop new features during the massive re-factor(s)

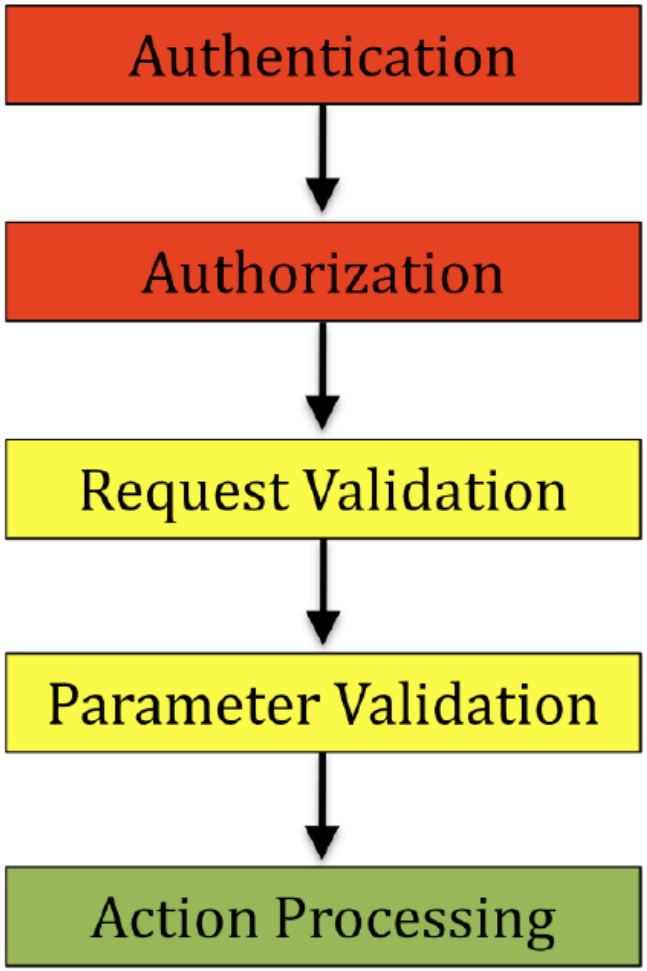
Need a (Painful) Example?

- Distributors were people with accounts
- Users were distributors who had filled out detailed profile information about themselves
- Two tables…
- Some objects were owned by distributors
- Some objects were owned by users

- Distributors => Users
- User => Profiles
- Re-factor to change ownership to users

Anatomy of an API Request

Next, we standardized how API requests should work…



Authentication

Authentication is the process of identifying who is accessing the API

- Since there's no money involved…
- Basic HTTP Authentication
- SSL for additional security
- Using a simple 20-character API key
- Easily supported in Rails
- Can add a more advanced scheme later if needed

Three Major Components

- **Acts As Authorized:** Handles privilege-checking to determine whether users can view, create, update or delete resources
- **Externalizable:** Domain-Specific Language (DSL) for exposing content. Also handles creates/updates in JSON and XML
- **Restful Behaviors:** Mini-framework for common controller logic related to manipulating resources

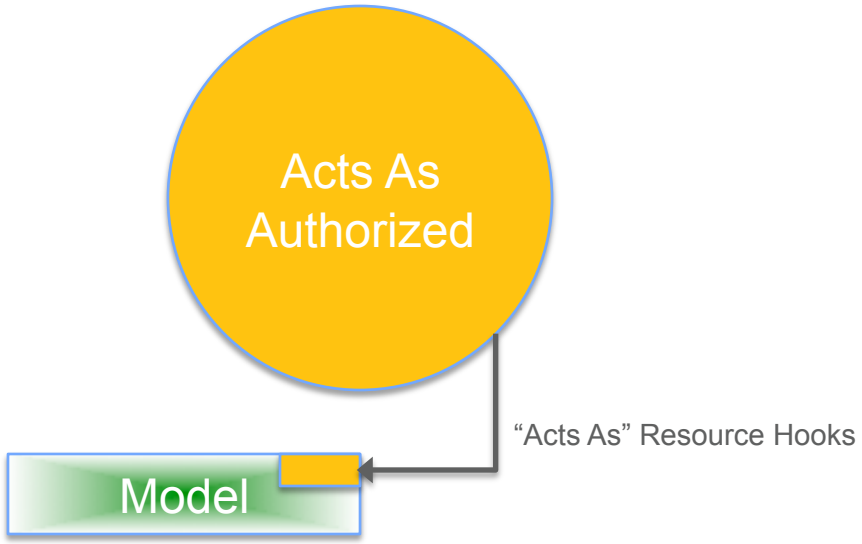
Acts As Authorized

- Handles most privilege checks for the API
- Relies on hooks in the model underlying each resource
  - auth_get_owner: Who “owns” the resource
  - auth_get_groups: Group sharing of resource
  - find_authorized: A method that honors privs

Acts As Authorized (2)

- Users can view any resource they created or that is shared with a group to which they belong
- Users can update a resource if they have the “update_<resource>” privilege for a group with which the resource is shared
- There are a few case-by-case restrictions

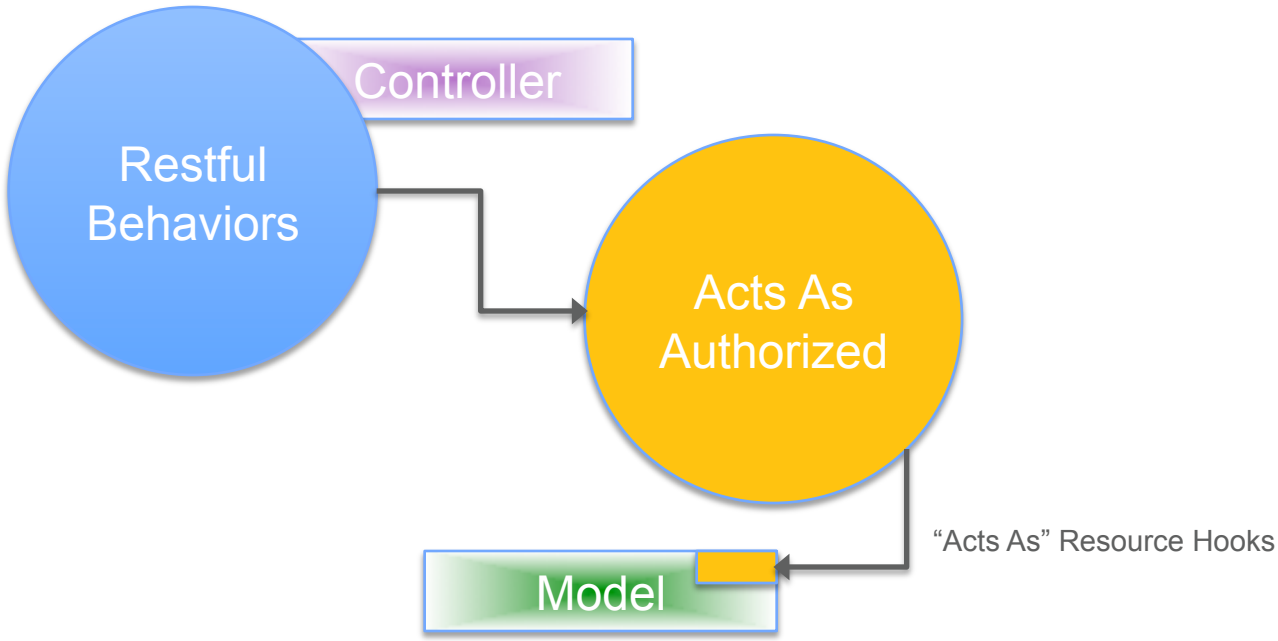
Acts As Authorized (3)



Restful Behaviors

- Mini-framework collecting common controller logic
- Functions as a mix-in for API controllers
- Developers just need to override key methods to tailor controllers for new resources
- Centralizes key features like externalizing content, error handling, single-asset privilege checking, etc.

Restful Behaviors (2)



Externalizable

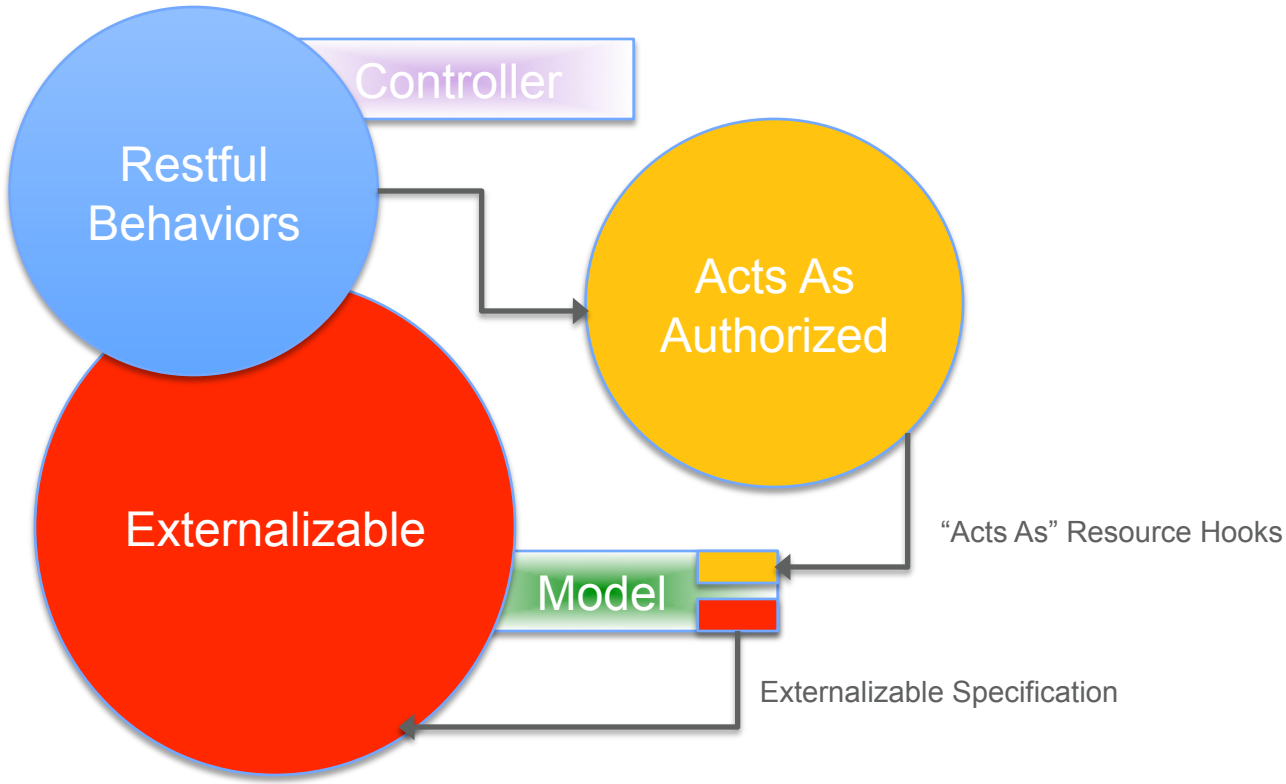
- Provides a Domain-Specific Language (DSL) for externalizing resources
- Included in models as a mix-in
- Centralizes functionality for producing output
- Centralizes processing of incoming XML/ JSON
- Centralizes create/update logic and ensures that only exposed fields can be set
- Can externalize database columns under different names

Externalizable (2)

include Externalizable

externalize_model_as “video”
externalize_attribute **:asset_id,** **:label => "id",**
:methods => [:index, :show]
externalize_attribute **:headline,** **:label => "title”**
externalize_attribute **:abstract,** **:label => "summary”**
externalize_attribute **:keywords**
externalize_attribute **:created_at,** **:methods => [:index, :show]**
externalize_attribute **:updated_at,** **:methods => [:index, :show]**

Overall Architecture



Error Handling

The diversity of technologies used to interact with the API makes it challenging to provide meaningful feedback to callers when errors occur.

- HTTP Status Codes
- HTTP “Warn” Header

  Ex. - 199 WAS-002: An unauthorized network was specified

  (the “199” => “Miscellaneous Message”)
- Error Messages in JSON/XML response
- Last Resort: The “always_ok” option

  - Always return 200… (Flash) caller has to parse response to determine success or failure

Searches

- Set up searches as Index and Create, so it accepts both GET and POST actions
- Searches are networks-specific
  - Can search Grab Networks public content or FOX private content (if authorized)
- Using Sphinx open source search engine
- Returns videos, but with extra “confidence”

Scalability

Scalability is a process, not a binary condition.

- Load Balancer in front of multiple Web Servers for the API – Can add servers as needed
- Separate Web Server in Amazon Cloud for contracted partners…handles file uploads and video transcoding tasks – Can load balance and add servers as needed
- Searches run against replicate production database – Can add replicates as needed
- Apache magic as needed

Practical Tips

- Start small, with a few resources
  - Work out the kinks, then expand the scope
- Start with an extended Beta
  - So you can change the API as needed without annoying users
- Recognize that a re-factor will be required
  - Just deal with it
- Eat your own dog food
  - It will never be solid unless you use your own API
- Challenge assumptions
  - Don’t be afraid to re-evaluate and adjust the API as you go
- Documentation, documentation, documentation

Resources

- RESTful Web Services by Leonard Richardson and Sam Ruby, Published by O'Reilly
- http://wasabi.grabnetworks.com
  - API is not publically available yet, but the documentation is