Extracted from:

Seven Databases in Seven Weeks

A Guide to Modern Databases and the NoSQL Movement

The Pragmatic Bookshelf

Dallas, Texas • Raleigh, North Carolina

The Pragmatic Programmers

Seven Databases in Seven Weeks

A Guide to Modern Databases and the NoSQL Movement

Eric Redmond
and Jim R. Wilson

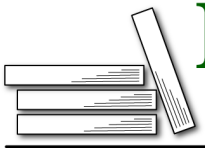
Edited by Jacquelyn Carter

Seven Databases in Seven Weeks

A Guide to Modern Databases and the NoSQL Movement

Eric Redmond
Jim R. Wilson

The Pragmatic Bookshelf
Dallas, Texas • Raleigh, North Carolina



Our Pragmatic courses, workshops, and other products can help you and your team create better software and have more fun. For more information, as well as the latest Pragmatic titles, please visit us at *http://pragprog.com*.

The team that produced this book includes:

Jackie Carter (editor)
Potomac Indexing, LLC (indexer)
Kim Wimpsett (copyeditor)
David J Kelly (typesetter)
Janet Furlow (producer)
Juliet Benda (rights)
Ellie Callahan (support)

Printed in the United States of America.
ISBN-13: 978-1-93435-692-0
Encoded using the finest acid-free high-entropy binary digits.
Book version: P2.0—August 2012

PUT the Value in the Bucket

Riak is a key-value store, so it expects you to pass in a key to retrieve a value. Riak breaks up classes of keys into *buckets* to avoid key collisions—for example, a key for java the *language* will not collide with java the *drink*.

We're going to create a system to keep track of animals in a dog hotel. We'll start by creating a bucket of animals that contain each furry guest's details. The URL follows this pattern:

```
http://SERVER:PORT/riak/BUCKET/KEY
```

A straightforward way of populating a Riak bucket is to know your key in advance. We'll first add *Ace, The Wonder Dog* and give him the key ace with the value {"nickname" : "The Wonder Dog", "breed" : "German Shepherd"}. You don't need to explicitly create a bucket—putting a first value into a bucket name will create that bucket.

```
$ curl -v -X PUT http://localhost:8091/riak/animals/ace \
  -H "Content-Type: application/json" \
  -d '{"nickname" : "The Wonder Dog", "breed" : "German Shepherd"}'
```

Putting a new value returns a 204 code. The -v (verbose) attribute in the curl command outputs this header line.

```
< HTTP/1.1 204 No Content
```

We can view our list of buckets that have been created.

```
$ curl -X GET http://localhost:8091/riak?buckets=true
{"buckets":["favs","animals"]}
```

Optionally, you can return the set results with the ?returnbody=true parameter, which we'll test by adding another animal, Polly:

```
$ curl -v -X PUT http://localhost:8091/riak/animals/polly?returnbody=true \
  -H "Content-Type: application/json" \
  -d '{"nickname" : "Sweet Polly Purebred", "breed" : "Purebred"}'
```

This time you'll see a 200 code.

```
< HTTP/1.1 200 OK
```

If we aren't picky about our key name, Riak will generate one when using POST.

```
$ curl -i -X POST http://localhost:8091/riak/animals \
  -H "Content-Type: application/json" \
  -d '{"nickname" : "Sergeant Stubby", "breed" : "Terrier"}'
```

The generated key will be in the header under Location—also note the 201 success code in the header.

```
HTTP/1.1 201 Created
Vary: Accept-Encoding
Server: MochiWeb/1.1 WebMachine/1.7.3 (participate in the frantic)
Location: /riak/animals/6VZc2o7zKxq2B34kJrm1S0ma3PO
Date: Tue, 05 Apr 2011 07:45:33 GMT
Content-Type: application/json
Content-Length: 0
```

A GET request (cURL's default if left unspecified) to that location will retrieve the value.

```
$ curl http://localhost:8091/riak/animals/6VZc2o7zKxq2B34kJrm1S0ma3PO
```

DELETE will remove it.

```
$ curl -i -X DELETE http://localhost:8091/riak/animals/6VZc2o7zKxq2B34kJrm1S0ma3PO
HTTP/1.1 204 No Content
Vary: Accept-Encoding
Server: MochiWeb/1.1 WebMachine/1.7.3 (participate in the frantic)
Date: Mon, 11 Apr 2011 05:08:39 GMT
Content-Type: application/x-www-form-urlencoded
Content-Length: 0
```

DELETE won't return any body, but the HTTP code will be 204 if successful. Otherwise, as you'd expect, it returns a 404.

If we've forgotten any of our keys in a bucket, we can get them all with keys=true.

```
$ curl http://localhost:8091/riak/animals?keys=true
```

You can also get them as a stream with keys=stream, which can be a safer choice for huge datasets—it just keeps sending chunks of keys array objects and ends with an empty array.

Links

Links are metadata that associate one key to other keys. The basic structure is this:

```
Link: </riak/bucket/key>; riaktag=\"whatever\"
```

The key to where this value links is in pointy brackets (<…>), followed by a semicolon and then a tag describing how the link relates to this value (it can be whatever string we like).

Link Walking

Our little dog hotel has quite a few (large, comfortable, and humane) cages. To keep track of which animal is in what cage, we'll use a link. Cage 1 contains Polly by linking to her key (this also creates a new bucket named cages). The cage is installed in room 101, so we set that value as JSON data.

```
$ curl -X PUT http://localhost:8091/riak/cages/1 \
  -H "Content-Type: application/json" \
  -H "Link: </riak/animals/polly>; riaktag=\"contains\"" \
  -d '{"room" : 101}'
```

Note that this link relationship is one-directional. In effect, the cage we've just created knows that Polly is inside it, but no changes have been made to Polly. We can confirm this by pulling up Polly's data and checking that there have been no changes to the Link headers.

```
$ curl -i http://localhost:8091/riak/animals/polly
```

```
HTTP/1.1 200 OK
X-Riak-Vclock: a85hYGBgzGDKBVIcypz/fvrde/U5gymRMY+VwZw35gRfFgA=
Vary: Accept-Encoding
Server: MochiWeb/1.1 WebMachine/1.9.0 (participate in the frantic)
Link: </riak/animals>; rel="up"
Last-Modified: Tue, 13 Dec 2011 17:53:59 GMT
ETag: "VD0ZAfOTsIHsgG5PM3YZW"
Date: Tue, 13 Dec 2011 17:54:51 GMT
Content-Type: application/json
Content-Length: 59

{"nickname" : "Sweet Polly Purebred", "breed" : "Purebred"}
```

You can have as many metadata Links as necessary, separated by commas. We'll put Ace in cage 2 and also point to cage 1 tagged with *next_to* so we know that it's nearby.

```
$ curl -X PUT http://localhost:8091/riak/cages/2 \
-H "Content-Type: application/json" \
-H "Link:</riak/animals/ace>;riaktag=\"contains\",
  </riak/cages/1>;riaktag=\"next_to\"" \
-d '{"room" : 101}'
```

What makes Links special in Riak is *link walking* (and a more powerful variant, linked mapreduce queries, which we investigate tomorrow). Getting the linked data is achieved by appending a *link spec* to the URL that is structured like this: /_,_,_. The underscores (_) in the URL represent wildcards to each of the link criteria: bucket, tag, keep. We'll explain those terms shortly. First let's retrieve all links from cage 1.

```
$ curl http://localhost:8091/riak/cages/1/_,_,_
```

```
--4PYi9DW8iJK5aCvQQrrP7mh7jZs
Content-Type: multipart/mixed; boundary=Av1fawIA4WjypRlz5gHJtrRqklD

--Av1fawIA4WjypRlz5gHJtrRqklD
X-Riak-Vclock: a85hYGBgzGDKBVIcypz/fvrde/U5gymRMY+VwZw35gRfFgA=
Location: /riak/animals/polly
Content-Type: application/json
Link: </riak/animals>; rel="up"
Etag: VD0ZAfOTsIHsgG5PM3YZW
Last-Modified: Tue, 13 Dec 2011 17:53:59 GMT

{"nickname" : "Sweet Polly Purebred", "breed" : "Purebred"}
--Av1fawIA4WjypRlz5gHJtrRqklD--

--4PYi9DW8iJK5aCvQQrrP7mh7jZs--
```

It returns a multipart/mixed dump of headers plus bodies of all linked keys/values. It's also a headache to look at. Tomorrow we'll find a more powerful way to get link-walked data that also happens to return nicer values—but today we'll dig a bit more into this syntax.

If you're not familiar with reading the multipart/mixed MIME type, the Content-Type definition describes a boundary string, which denotes the beginning and end of some HTTP header and body data.

```
--BcOdSWMLuhkisryp0GidDLqeA64
some HTTP header and body data
--BcOdSWMLuhkisryp0GidDLqeA64--
```

In our case, the data is what cage 1 links to: Polly Purebred. You may have noticed that the headers returned don't actually display the link information. This is OK; that data is still stored under the linked-to key.

When link walking, we can replace the underscores in the link spec to filter only values we want. Cage 2 has two links, so performing a link spec request will return both the animal Ace contained in the cage and the cage 1 next_to it. To specify only following the animals bucket, replace the first underscore with the bucket name.

```
$ curl http://localhost:8091/riak/cages/2/animals,_,_
```

Or follow the cages *next to* this one by populating the tag criteria.

```
$ curl http://localhost:8091/riak/cages/2/_,next_to,_
```

The final underscore—keep—accepts a 1 or 0. keep is useful when following second-order links, or links following other links, which you can do by just appending another link spec. Let's follow the keys next_to cage 2, which will return cage 1. Next, we walk to the animals linked to cage 1. Since we set

keep to 0, Riak will not return the intermediate step (the cage 1 data). It will return only Polly's information, who is next to Ace's cage.

```
$ curl http://localhost:8091/riak/cages/2/_,next_to,0/animals,_,_

--6mBdsboQ8kTT6MlUHg0rgvbLhzd
Content-Type: multipart/mixed; boundary=EZYdVz9Ox4xzR4jx1I2ugUFFiZh

--EZYdVz9Ox4xzR4jx1I2ugUFFiZh
X-Riak-Vclock: a85hYGBgzGDKBVIcypz/fvrde/U5gymRMY+VwZw35gRfFgA=
Location: /riak/animals/polly
Content-Type: application/json
Link: </riak/animals>; rel="up"
Etag: VD0ZAfOTsIHsgG5PM3YZW
Last-Modified: Tue, 13 Dec 2011 17:53:59 GMT

{"nickname" : "Sweet Polly Purebred", "breed" : "Purebred"}
--EZYdVz9Ox4xzR4jx1I2ugUFFiZh--

--6mBdsboQ8kTT6MlUHg0rgvbLhzd--
```

If we want Polly's information and cage 1, set keep to 1.

```
$ curl http://localhost:8091/riak/cages/2/_,next_to,1/_,_,_

--PDVOEl7Rh1AP90jGln1mhz7x8r9
Content-Type: multipart/mixed; boundary=YliPQ9LPNEoAnDeAMiRkAjCbmed

--YliPQ9LPNEoAnDeAMiRkAjCbmed
X-Riak-Vclock: a85hYGBgzGDKBVIcypz/fvrde/U5gymRKY+VIYo35gRfFgA=
Location: /riak/cages/1
Content-Type: application/json
Link: </riak/animals/polly>; riaktag="contains", </riak/cages>; rel="up"
Etag: 6LYhRnMRrGIqsTmpE55PaU
Last-Modified: Tue, 13 Dec 2011 17:54:34 GMT

{"room" : 101}
--YliPQ9LPNEoAnDeAMiRkAjCbmed--

--PDVOEl7Rh1AP90jGln1mhz7x8r9
Content-Type: multipart/mixed; boundary=GS9J6KQLsI8zzMxJluDITfwiUKA

--GS9J6KQLsI8zzMxJluDITfwiUKA
X-Riak-Vclock: a85hYGBgzGDKBVIcypz/fvrde/U5gymRMY+VwZw35gRfFgA=
Location: /riak/animals/polly
Content-Type: application/json
Link: </riak/animals>; rel="up"
Etag: VD0ZAfOTsIHsgG5PM3YZW
Last-Modified: Tue, 13 Dec 2011 17:53:59 GMT

{"nickname" : "Sweet Polly Purebred", "breed" : "Purebred"}
```

```
--GS9J6KQLsI8zzMxJluDITfwiUKA--

--PDVOEl7Rh1AP90jGln1mhz7x8r9--
```

This returns the objects in the path to the final result. In other words, *keep* the step.

Beyond Links

Along with Links, you can store arbitrary metadata by using the X-Riak-Meta- header prefix. If we wanted to keep track of the color of a cage but it wasn't necessarily important in the day-to-day cage-managing tasks at hand, we could mark cage 1 as having the color pink. Getting the URL's header (the -I flag) will return your metadata name and value.

```
$ curl -X PUT http://localhost:8091/riak/cages/1 \
  -H "Content-Type: application/json" \
  -H "X-Riak-Meta-Color: Pink" \
  -H "Link: </riak/animals/polly>; riaktag=\"contains\"" \
  -d '{"room" : 101}'
```

MIME Types in Riak

Riak stores everything as a binary-encoded value, just like normal HTTP. The MIME type gives the binary data context—we've been dealing only with plain text up until now. MIME types are stored on the Riak server but are really just a flag to the client so that when it downloads the binary data, it knows how to render it.

We'd like our dog hotel to keep images of our guests. We need only use the data-binary flag on the curl command to upload an image to the server and specify the MIME type as image/jpeg. We'll add a link back to the /animals/polly key so we know who we are looking at.

First, create an image called polly_image.jpg and place it in the same directory you've been using to issue the curl commands.

```
$ curl -X PUT http://localhost:8091/riak/photos/polly.jpg \
  -H "Content-type: image/jpeg" \
  -H "Link: </riak/animals/polly>; riaktag=\"photo\"" \
  --data-binary @polly_image.jpg
```

Now visit the URL in a web browser, which will be delivered and rendered exactly as you'd expect any web client-server request to function.

```
http://localhost:8091/riak/photos/polly.jpg
```

Since we pointed the image to /animals/polly, we could link walk from the image key *to* Polly but not vice versa. Unlike a relational database, there is no "has

a" or "is a" rule concerning links. You link the direction you need to walk. If we believe our use case will require accessing image data from the animals bucket, a link should exist on that object instead (or in addition).

Day 1 Wrap-Up

We hope you're seeing a glimmer of Riak's potential as a flexible storage option. So far, we've covered only standard key-value practice with some links thrown in. When designing a Riak schema, think somewhere in between a caching system and PostgreSQL. You will break up your data into different logical classifications (buckets), and values can tacitly relate to each other. But you will not go so far as to normalize into fine components like you would in a relational database, since Riak performs no sense of relational joins to recompose values.

Day 1 Homework

Find

1. Bookmark the online Riak project documentation and discover the REST API documentation.
2. Find a good list of browser-supported MIME types.
3. Read the example Riak config dev/dev1/etc/app.config, and compare it to the other dev configurations.

Do

1. Using PUT, update animals/polly to have a Link pointing to photos/polly.jpg.
2. POST a file of a MIME type we haven't tried (such as application/pdf), find the generated key, and hit that URL from a web browser.
3. Create a new bucket type called *medicines*, PUT a JPEG image value (with the proper MIME type) keyed as *antibiotics*, and link to the animal Ace (poor, sick puppy).