Link Walking

Contents

What are Links?

One of the ways that we are able to extend the fairly-limited data model provided by a key/value store is with the notion “links” and a type of query known as “link walking.”

Links are metadata that establish one-way relationships between objects in Riak. Once attached, links then enable you to run queries that “walk” relations from one object to another. With links, you create lightweight pointers between your data, for example, from 'projects' to 'milestones' to 'tasks', and then select data along that hierarchy using simple client API commands. (In a pinch, this can substitute as a lightweight graph database, as long as the number of links attached to a given key are kept reasonably low.) Links are an incredibly powerful feature of Riak and can transform an application if used appropriately by developers.

Working with Links

Links live in the metadata of an object and are attached to it via the “Link” header. Here is what the link header looks like:

Shell

```
Link: </riak/people/dhh>; riaktag="friend"
```

So what's actually happening here? In the angle brackets, we have the URL the link points to. The prefix “riak” is followed by the bucket name, “people,” and the key, “dhh.” Next comes the “riaktag.” Contained in the riaktag field is an identifier that describes the relationship you are wishing to capture with your link. In this case the riaktag is “friend.”

Here is what a full PUT request through CURL with the link header would look like:

```
$ curl -v -XPUT http://127.0.0.1:8091/riak/people/timoreilly \
  -H 'Link: </riak/people/dhh>; riaktag="friend"' \
  -H "Content-Type: text/plain" \
  -d 'I am an excellent public speaker.'
```

In this request we are attaching 'Link: </riak/people/dhh>; riaktag=“friend”' to the key “timoreilly” located in the “people” bucket.

Try it. It's easy, right? You've just attached a link to an object in Riak!

> To remove a link from an object, it’s straightforward, too: read (GET) the object, remove the link information, and write it back into Riak.

To retrieve that object to see the attached link, simply to do the following:

HTTP

```
$ curl -v http://127.0.0.1:8091/riak/people/timoreilly
```

Look for the “Link” field in the response headers. This will show you your link information.

Alright. We've stored the “timoreilly” object with a “friend” tag pointing to the “dhh” object. Now we need to store the “dhh” object to which “timoreilly” is linked:

```
$ curl -v -XPUT http://127.0.0.1:8091/riak/people/dhh \
  -H "Content-Type: text/plain" \
  -d 'I drive a Zonda.'
```

Great. Now we have the “timoreilly” in the “people” bucket stored with a link that points to the “dhh” object that is also in the “people” bucket.

How do we connect the dots? Link Walking, that's how.

Link Walking

Once you have tagged objects in Riak with links, you can then traverse them with an operation called “Link Walking.” You can walk any number of links with one request, and you can choose to have all the objects matching a single step returned with the end result.

To continue with the example from above, the “timoreilly” object is now pointer to the “dhh” object located in the “people” bucket. We can use a link walking query to follow the link from “timoreilly” to “dhh.” Here is what that query would look like:

HTTP

```
$ curl -v http://127.0.0.1:8091/riak/people/timoreilly/peopl
```

You'll notice that at the end of that request we've tacked on “/people,friend,1” That is the link specification. It's composed of three parts:

- Bucket - a bucket name to limit the links to (in the above request it's 'people')
- Tag - the “riaktag” to limit the links ('friend' is the tag in the above request)
- Keep - 0 or 1, whether to return results from this step or phase

If all went well, the response body from the above request should include the record for the “dhh” object.

You can replace both the “bucket” and the “tag” fields in the link spec with an underscore. This will tell the query to match any bucket or tag name. For instance, the following request should return the same data as the fully-specified request above:

HTTP

```
$ curl -v http://127.0.0.1:8091/riak/people/timoreilly/_,fri
```

Each step you walk is referred to as a phase, because under the hood a link walking request uses the same mechanism as MapReduce, where every step specified in the URL is translated into a single MapReduce phase. If you want to walk multiple steps you can use the Keep parameter to specify which steps your particularly interested in.

By default, Riak will only include the objects found by the last step. This could be interesting if you want e.g. to build a graph of how the original object (“timoreilly” in this case) relates to the ones found traversing the links. To see how this works out in practice, let's add another object to the mix, “davethomas”, who is friends with “timoreilly”.

```
$ curl -v -XPUT http://127.0.0.1:8091/riak/people/davethomas \
  -H 'Link: </riak/people/timoreilly>; riaktag="friend"' \
  -H "Content-Type: text/plain" \
  -d 'I publish books'
```

Now we can walk from “davethomas” to “dhh” in one go, and you'll see the last parameter in action:

HTTP

```
$ curl -v localhost:8091/riak/people/davethomas/_,friend,_/_
```

As a result you'll only get the “dhh” object. Leaving the last parameter for each step as “_” defaults to Riak not returning objects from intermittent steps (i.e. 0), and to 1 for the last step. So to get everything in between you set the last parameter to 1 for the steps you're interested in.

HTTP

```
$ curl -v localhost:8091/riak/people/davethomas/_,friend,1/_
```

When you try this out yourself you'll notice that the output has gotten slightly more confusing, because it now consists of two parts with a bunch of objects contained in each.

As a final sugar sprinkle on top, we can make “dhh” friends with “davethomas” directly, so we have a real graph and not just a single path.

```
$ curl -v -XPUT http://127.0.0.1:8091/riak/people/dhh \
  -H 'Link: </riak/people/davethomas>; riaktag="friend"' \
  -H "Content-Type: text/plain" \
  -d 'I drive a Zonda.'
```

You can add more link phases to the request, or you can walk from “dhh” to “timoreilly” through “davethomas”, or even from “davethomas” to “davethomas”, by adding another step to Link Walking specification.

HTTP

```
$ curl -v localhost:8091/riak/people/davethomas/_,friend,_/_
```

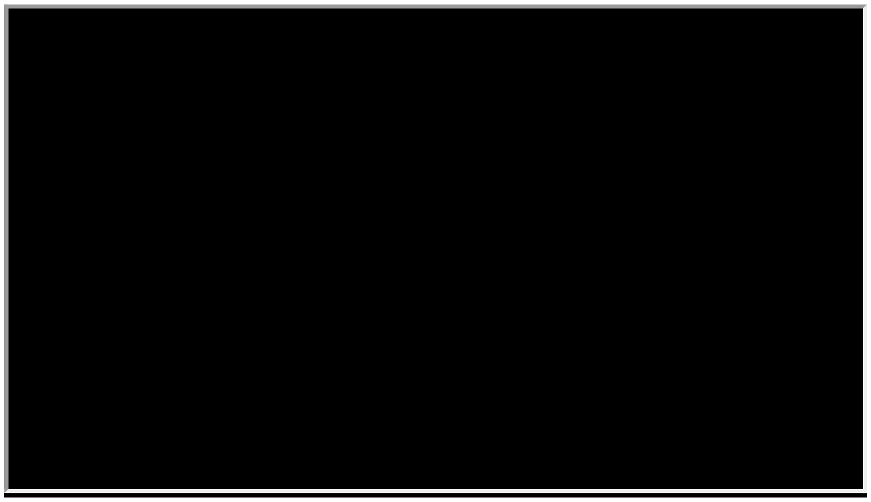
So, let's review what we just did:

1. Stored an object with a link attached to it.
2. Stored the object to which the link pointed.
3. Performed a link walking query to traverse the link from one object to another, and across a whole set of objects.

This is some pretty powerful stuff! And we've only just scratched the surface of what links can do and what they can be used for.

A Magnificent Link Walking Screen Cast

In this screencast, Basho Hacker Sean Cribbs will take you through link walking basics and then dive into more complex and advanced usage of links in Riak.



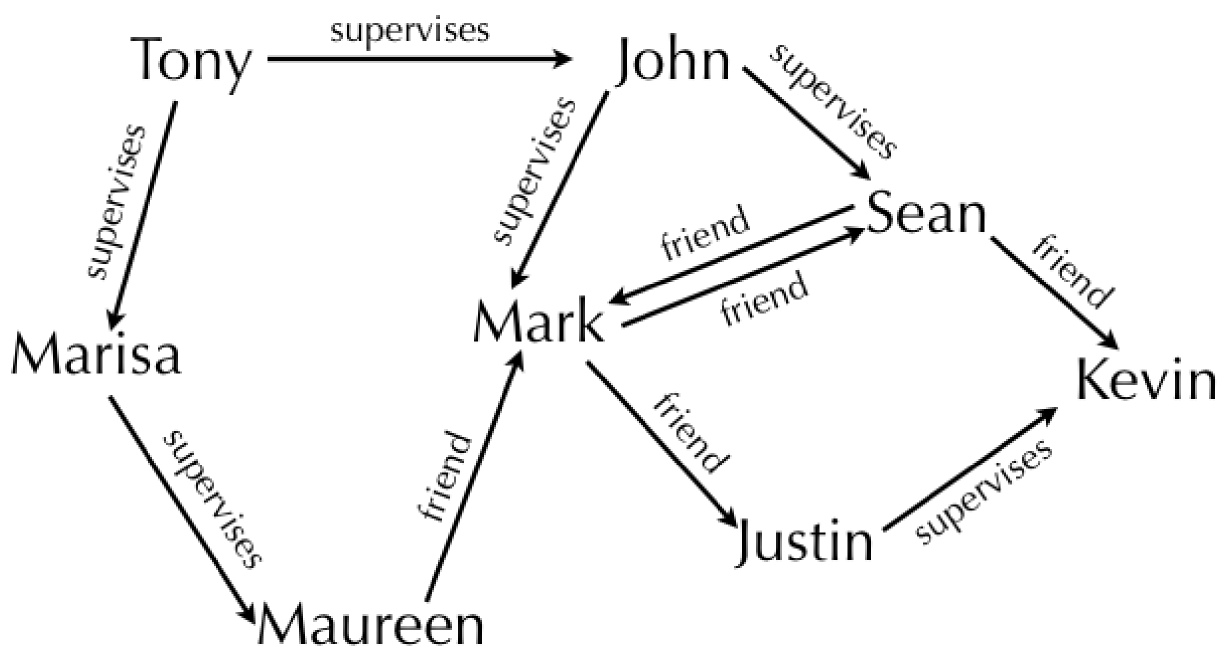
Links and Link Walking in Riak from Basho Technologies on Vimeo.

Link Walking Scripts

In the above screencast, Sean makes use of several scripts to demonstrate some deeper relationships expressed with links in Riak. Here are these scripts:

load_people.sh
people_queries.sh

If you watched the video, it's apparent how these scripts are used to demonstrate link walking. For those of you who didn't watch or who want to run and tweak the scripts themselves, check out this graphic:



`load_people.sh` will automatically load data into your running three node Riak Cluster that pertains to the the above graphic and has the requisite links attached.

`people_queries.sh` is a series of link walking queries that expresses the relationships that were preloaded with the `load_people.sh` script.

To use `load_people.sh` download it to your `dev` directory and run

Shell
```
$ chmod +x load_people.sh
```

followed by

Shell
```
$ ./load_people.sh
```

After the several lines of output finish, do the same for “people_queries.sh”:

Shell
```
$ chmod +x people_queries.sh
```

followed by

Shell
```
$ ./people_queries.sh
```

You should then see:

```
Press [[Enter]] after each query description to execute.
Q: Get Sean's friends (A:Mark, Kevin)
```

These May Also Interest You

- Five-Minute Install
- Advanced Secondary Indexes
- Replication Properties
- Advanced Commit Hooks
- Advanced MapReduce
- Advanced Search