VERSIONS
2.0.7

Getting Started:

CRUD Operations with NodeJS

Creating Objects In Riak KV

Pinging a Riak cluster sounds like a lot of fun, but eventually someone is going to want us to do productive work. Let’s create some data to save in Riak.

The Riak Node.js Client makes use of a `RiakObject` class to encapsulate Riak key/value objects. At the most basic, a `RiakObject` is responsible for identifying your object and for translating it into a format that can be easily saved to Riak.

- Javascript

```
var async = require('async');

var people = [
    {
        emailAddress: "bashoman@basho.com",
        firstName: "Basho",
        lastName: "Man"
    },
    {
        emailAddress: "johndoe@gmail.com",
        firstName: "John",
        lastName: "Doe"
    }
];

var storeFuncs = [];
people.forEach(function (person) {
    // Create functions to execute in parallel to store people
    storeFuncs.push(function (async_cb) {
        client.storeValue({
                bucket: 'contributors',
                key: person.emailAddress,
                value: person
            },
            function(err, rslt) {
                async_cb(err, rslt);
            }
        );
    });
});

async.parallel(storeFuncs, function (err, rslts) {
    if (err) {
        throw new Error(err);
    }
});
```

In this sample, we create a collection of `Person` objects and then save each `Person` to Riak. Once again, we check the response from Riak.

Reading from Riak

Let’s find a person!

- Javascript

```
var logger = require('winston');

client.fetchValue({ bucket: 'contributors', key: 'bashoman@basho.com', convertToJs: true },
    function (err, rslt) {
        if (err) {
            throw new Error(err);
        } else {
            var riakObj = rslt.values.shift();
            var bashoman = riakObj.value;
            logger.info("I found %s in 'contributors'", bashoman.emailAddress);
        }
    }
);
```

We use `client.fetchValue` to retrieve an object from Riak. This returns an array of `RiakObject` objects which helpfully encapsulates the communication with Riak.

After verifying that we’ve been able to communicate with Riak *and* that we have a successful result, we use the `value` property to get the object, which has already been converted to a javascript object due to the use of `convertToJs: true` in the options.

Modifying Existing Data

Let’s say that Basho Man has decided to be known as Riak Man:

- Javascript

```
bashoman.FirstName = "Riak";
riakObj.setValue(bashoman);

client.storeValue({ value: riakObj }, function (err, rslt) {
    if (err) {
        throw new Error(err);
    }
});
```

Updating an object involves modifying a `RiakObject` then using `client.storeValue` to save the existing object.

Deleting Data

- Javascript

```
client.deleteValue({ bucket: 'contributors', key: 'johndoe@gmail.com' }, function (err, rslt) {
    if (err) {
        throw new Error(err);
    }
});
```

Just like other operations, we check the results that have come back from Riak to make sure the object was successfully deleted. Of course, if you don’t care about that, you can just ignore the result.

The Riak Node.js Client has a lot of additional functionality that makes it easy to build rich, complex applications with Riak. Check out the documentation to learn more about working with the Riak Node.js Client and Riak.

Basho.com - Docs Home - Contact Basho