# RIAK KV

Chris Lambert, Sam Pastoriza, Joel Shapiro

## Table of Contents

## Riak KV

### Introduction

Riak is a distributed NoSQL database. This database is designed from the ground up to allow for maximum data availability by distributing data across multiple servers intelligently. The Riak KV service stores data as a combination of keys and values, but while the data model is extremely simple it is still powerful. Keys are simply binary values used to uniquely identify a value. Riak uses the JSON format to store key-value pairs in the database. Riak is tightly integrated with Apache Spark, Solr, and Redis. It has a singular api with many open sourced, well maintained, official client libraries for Java, Ruby, Python, C# and NodeJS.

### Use Cases

Cases where it might be a good idea to use Riak include:

- Session Data
- Contents and Documents
- Sensor Data
- Messaging and Chat
- Log Storage
- Mobile Platforms

## The Data Model

Riak stores data in buckets, and each bucket can store multiple key-value pairs. An example of this follows below.

```
usernames = {
        'mohan': {
                {'name': 'Sriram Mohan'},
                {'password': 'password'},
                {'salt': 'mohanSalt'}
        },
        'mellor': {
                {'name': 'JP Mellor'},
                {'password': 'password'},
                {'salt': 'mellorSalt'}
        }
}
```

In the above example, 'usernames' is the name of the bucket while 'mohan' and 'mellor' are keys within the bucket that map to the data about the particular user.

## Installation Instructions

### For Ubuntu

First run the following command to install the prerequisite packages:

```
sudo apt-get install libc6 libc6-dev libc6-dbg
```

Next, we need to get the signing key and add it to apt-get:

```
curl https://packagecloud.io/gpg.key | sudo apt-key add -
```

Then, install apt-transport-https to enable the ability to fetch packages of https:

```
sudo apt-get install -y apt-transport-https
```

Finally, download and install Riak KV:

```
curl -s https://packagecloud.io/install/repositories/basho/riak/script.deb.sh | sudo bash
sudo apt-get install riak=2.1.4-1
```

### For Mac

Use brew to install Riak

```
brew install Riak
```

## Configuration Instructions

### Initial Configuration

To make sure that Riak has been installed correctly, type in the terminal:

```
riak ping
```

If the console returns ‘pong’, riak is correctly configured. If there are issues, please follow the guide on this website: http://docs.basho.com/riak/kv/2.0.5/configuring/

### Configuring Python Library

First, we need to install the necessary ubuntu packages:

```
sudo apt-get install python-dev libffi-dev libssl-dev
```

Next, we need to install the riak python libraries (Use either pip or easy_install):

```
sudo pip install riak
```

or

```
easy_install riak
```

Riak is now ready to use with python.

## Riak Basic Commands

If you want to test Riak out using the console, feel free to follow the tutorial on this website. We will be using the python client to perform basic commands.
To start the console, use the command:

```
riak console
```

To start Riak as a service, use the command:

```
riak start
```

Then, start the python client:

```
python
```

Start by importing the riak library:

```
import riak
```

Then establish connection with the database

```
myClient = riak.RiakClient(pb_port=8087, protocol='pbc', host='hostname')
```

Riak uses bucket types to define configuration specifications, including indexing, for sets of key value pairs contained inside.

```
testBucket = myCleint.bucketType('firstBucket')
```

Retrieve a reference to the bucket called 'bucketName' from the database.

```
myBucket = testBucket.bucket('bucketName')
```

Create your first key in the bucket and store a value of '1'

```
key = myBucket.new('one', data='1')
```

Store the key in the database

```
key.store()
```

Retrieve the value of 'one' from the bucket in the database

```
fetchedData = myBucket.get('one')
```

Delete the key-value pair {'one', '1'} from the bucket

```
fetchedData.delete()
```

We are now going to create two more key value pairs and store them in the database. You can store JSON data as values too. Riak uses the json format to store all key-value pairs in the database

```
valTwo = '2'
jsonData = {newValue: 4}
key2 = myBucket.new('two', data = valTwo}
jsonKey = myBucket.new{'json', data = jsonData}
key2.store()
jsonKey.store()
```

Now we will get the json data and update it

```
getJson = myBucket.get('json')
getJson['newValue'] = 42
getJson.store();
```

Lets make sure the value has been updated

```
assert myBucket.get('json')['newValue'] == 42
```

Delete the rest of the key-value pairs

```
getJson.delete()
getKey2 = myBucket.get('two')
```

```
getKey2.delete()
```

Unfortunately, there is no automatic way to delete all key-value pairs from a bucket. It must be done manually.

### Querying using secondary indexes

Secondary indexes can be extremely useful when you want to tag objects at runtime and categorically group them together so when one searching for a particular set of key value pairs with a given tag, those can easily be returned.

```
bucket = client.bucket_type('default').bucket('users')
```

In the Python client (and all clients), if you do not specify a bucket type, the client will use the default type. And so the following store command would be equivalent to the one above:

```
bucket = client.bucket('users')
```

Riak objects are another way of creating key value pairs. It is syntactically equivalent.

```
obj = RiakObject(client, bucket, 'john_smith')
obj.content_type = 'text/plain'
obj.data = '...user data…'
```

Here we add a secondary index

```
obj.add_index('twitter_bin', 'jsmith123')
obj.add_index('email_bin', 'jsmith@basho.com')
obj.store()
```

To query for this secondary index of ‘twitter_bin’

```
bucket.get_index('twitter_bin', 'jsmith123').results
```

This returns an array of one item, ‘john_smith’

## Mini Project: World #1 Golfers

| Weeks at the top | Country | Total weeks at the top | Golfer | |
|---|---|---|---|---|
| August 15, 1999- September 4, 2004 | USA | 683 | Name | # of Majors |
| | | | Tiger Woods | 14 |
| June 18, 1995- April 19, 1997 | Australia | 331 | Name | # of Majors |
| | | | Greg Norman | 2 |
| July 19, 1992- February 5, 1994 | England | 97 | Name | # of Majors |
| | | | Nick Faldo | 6 |
| August 3, 2014- August 15, 2015 | Northern Ireland | 95 | Name | # of Majors |

| | | | | |
|---|---|---|---|---|
| | | | Rory Mcilroy | 4 |
| April 2, 1989- August 19, 1989 | Spain | 61 | Name | # of Majors |
| | | | Seve Ballesteros | 5 |

Task 1: Store all of the information on Riak
Task 2: Sort all of the information based on the total weeks at the top in descending order
Task 3: Add the ability to have multiple time periods for weeks at the top.

## Golf Course Reviews + Tee Times

For this project, we are going to create a database that stores golf courses reviews and those users who make them.

Requirements:

1. Add a golf course, including the name, location, par for the course
    a. Par for the course means how many strokes it should take a golfer with a certain handicap to complete the course.
    See: https://en.wikipedia.org/wiki/Golf#Par
    b. Feel free to add a golf course id if you want to make sure that all golf courses are unique. This can help with searching.
2. Delete a golf course
3. Edit golf course information
4. Add a golfer, including their username, name and home golf course
    a. The home golf course must be in the system before the golfer is added
5. Remove a golfer from the database
6. Update a golfer's information
7. Add a golf course review
8. Remove a golf course review
9. Be able to search for golf courses and users, not specific reviews
10. Sort golf courses by their name and par for the course

**Note**: If a golfer is deleted from the database, all of their reviews become anonymous.

## Scalability

Scalability/sharding/replication instructions

1. Setting up a cluster
    1. Configure the first node
        1. run riak stop
        2. In the config file:
            1. modify listener.protobuf.internal to <public-ip>:8087
            2. modify listener.http.internal to <public-ip>:8098
            3. change nodename to some unique identifier
        3. run riak start

      4. run riak-admin cluster replace <prev-node-name> <new-node-name>
      5. run riak-admin cluster plan
      6. run riak-admin cluster commit
   2. Run the following commands on the node you would like to add to a cluster
      1. raik-admin cluster join <name-of-primary-node>
         1. adds current node to the cluster named <name-of-primary-node>
         2. Must modify listener.http.internal and listener.protobuf.internal to be a publicly accessible IP address
      2. riak-admin cluster plan
      3. riak-admin cluster commit
      4. riak admin cluster status
         1. Displays members of the cluster. Can be used to check if node was added
2. Setting up strong consistency
   1. For all the nodes in the cluster, in the config file set:
      1. strong_consistency = on
   2. It is disabled by default

## When using Riak, keep an eye out for

Setting up unique keys can be very difficult to set up.

## References

References to sources, documentation

-Riak Documentation
http://docs.basho.com/riak/kv/2.1.4/