**MSCD664**

**Lab 4 – Using and learning the Riak environment**

**Name: Rohan Purohit**

**Date: Sept 24 2016**

-----------------------------------------------------------------------------------

The purpose of this lab is to become familiar with the Riak Database Environment and how to access the system. You will perform the following tasks in the lab:

* Starting and stopping your Virtual Machine
* Starting and stopping both a single instance and a 5 node Riak Database system
* Perform some of the same statements in the Seven Weeks text to demonstrate how storing and retrieving data works in Riak.

*Read chapter 8 in the NoSQL Distilled text and chapter 3 in the Seven Weeks text. Also, ensure you have downloaded and successfully installed the Riak-Ubuntu virtual machine for VirtualBox. You may use your own Riak installation, if you desire, but this lab is specifically developed using the VM from the course. For those of you not familiar with Linux environments please review the Linux videos and other resources within the course.*

To get an “A” grade on this lab:

* Answer all questions.
* Provide output of steps that require you to perform a task. The output can be a cut and paste from the screen to a word document or screen shots. I need to see something to verify that you ran the labs. I’m expecting you to turn in a word document that contains each question and also the corresponding answers to questions and any output from statements. Tasks that require an output will be indicated clearly.
* Text you should enter will look like this:

Enter this text

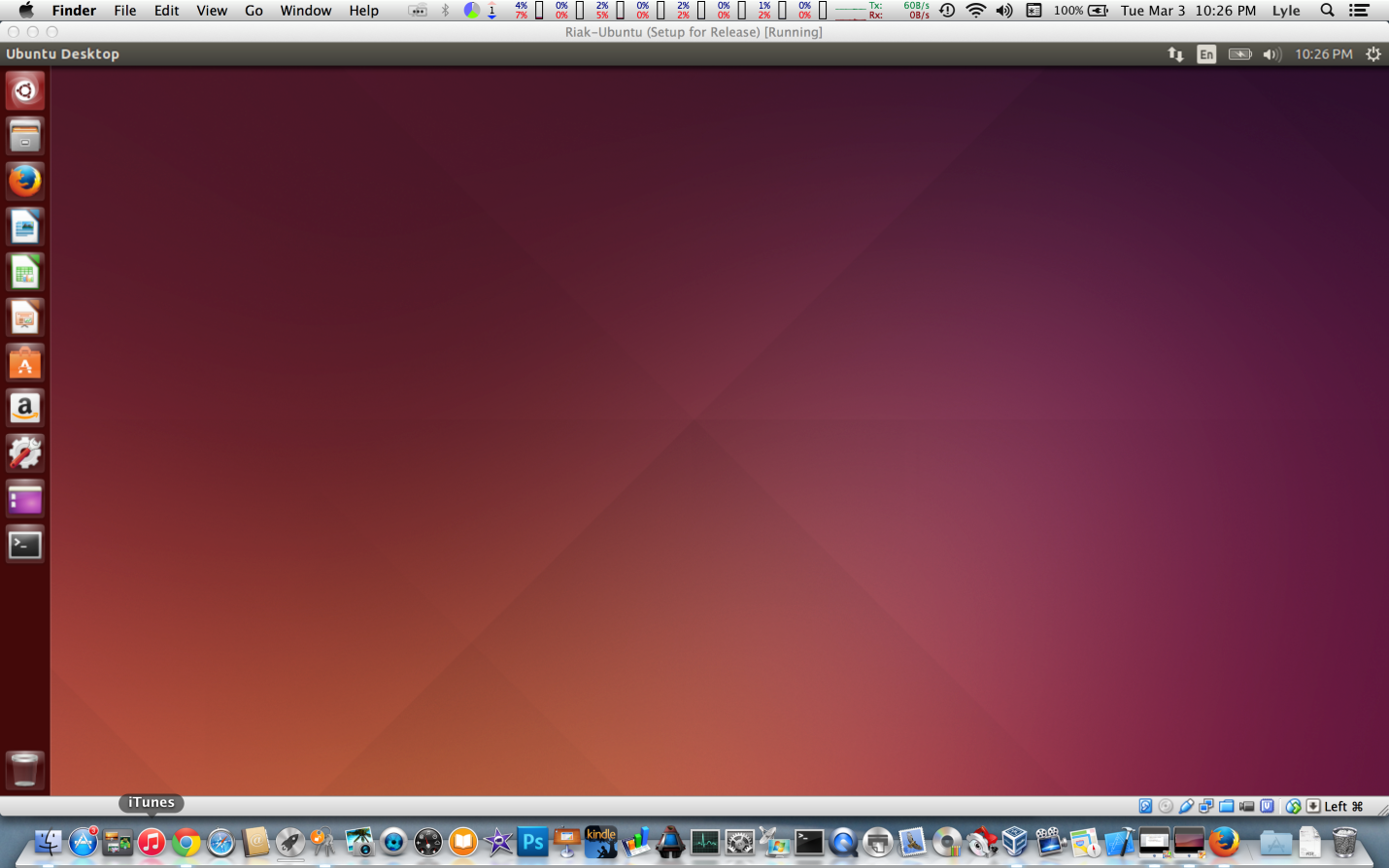
<return> means to hit the return or enter key on your keyboard.

**-----------------------------------------------------------------------------------**

**Part 1: Examining the prepackaged Riak installed system**

**-----------------------------------------------------------------------------------**

1. In both Windows and Mac, start your VirtualBox software and startup your Riak-Ubuntu virtual machine. The VM should start and you should be presented with a desktop with the riak user already logged in. Along the left side of the desktop is the launch bar. This contains a number of application icons that you can use to start the respective application quickly. At this time let’s start by examining the installed single instance Riak database. This installation was done from the 2.0.5 debian package. Click on the Terminal icon on the launch bar. It should be the last icon on the left above the trashcan.

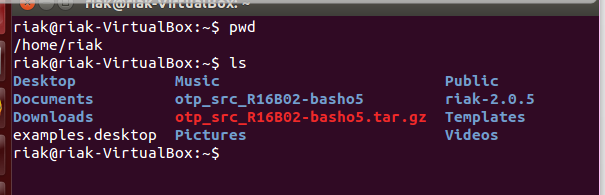


**-----------------------------------------------------------------------------------**

1. This will open a console terminal set to the riak user’s home directory.
   1. Type pwd<return> and you should see the current directory path displayed.
   2. Enter ls <return> and you should see a listing of the current directory.

This is your home directory. You can see several files and several directories. The red colored file is an archive source file of the Erlang language. This has already been installed into this VM, so you can leave it alone. If you want you can peruse the otp… directory and see the source code. The riak-2.0.5 directory is the source file distribution for riak and also where we have a 5 node setup. For now, leave these alone and don’t change anything with these directories.

Place a screenshot of your terminal screen below:



**-----------------------------------------------------------------------------------**

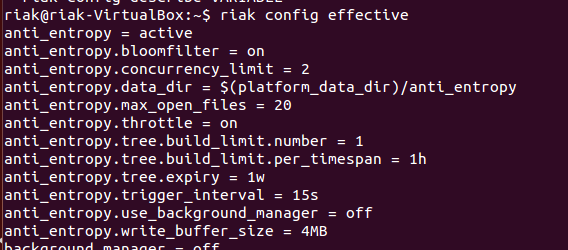
1. Now let’s start interacting with Riak.
   1. Enter riak ping<return> in the terminal.
   2. What was the response that was displayed? **Pong**



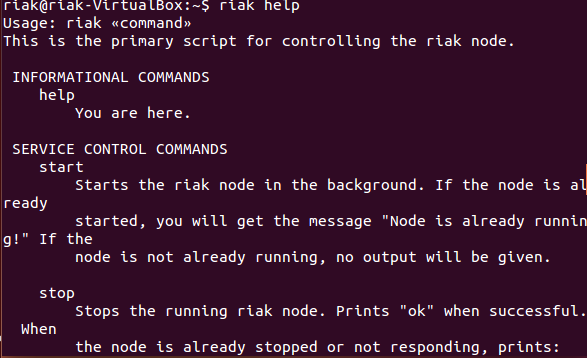
* 1. Try riak version <return> **2.0.5**



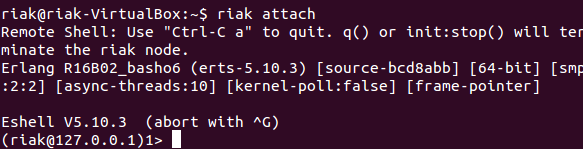
* 1. And riak config <return> **riak config effective**



* 1. Finally, let’s see what help will do for us. Enter riak help<return>



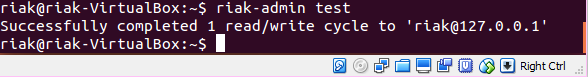
What does the riak attach command do? **Connects to Erlang remote shell.**



**-----------------------------------------------------------------------------------**

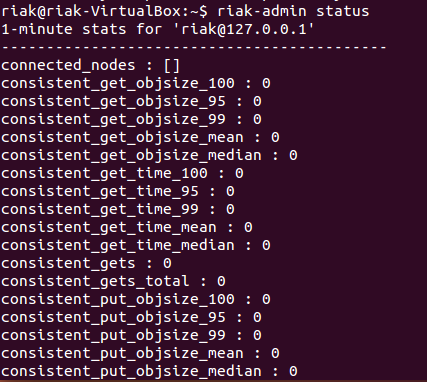
1. Enter riak-admin help<return> into your terminal. This is a command-line admin console where you can do some low level actions to Riak.
   1. Let’s make sure this riak node is properly working. Enter riak-admin test<return> into your terminal.

Enter a screenshot of the terminal window below:



* 1. Enter riak-admin status<return>

Enter a screenshot of the results below:



**-----------------------------------------------------------------------------------**

1. Now click the Firefox icon on the launch bar. This will open the Firefox web browser. Notice you have two “home” tabs opening (assuming you are connected to the internet). Click the Riak tab. You are now at the Riak Documentation pages. Take some time and look around. Then answer the below question. Feel free to cut and paste the answer from the website. Include the URL when you do however.

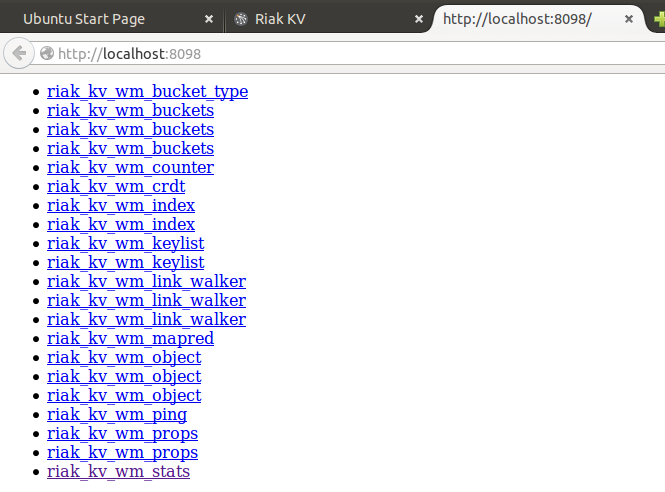
Is there a limit to the size of files that can be stored on Riak?

**There isn’t a limit on objects size, but it is recommended that we keep no more than 1-2 MB for performance reasons.**

**-----------------------------------------------------------------------------------**

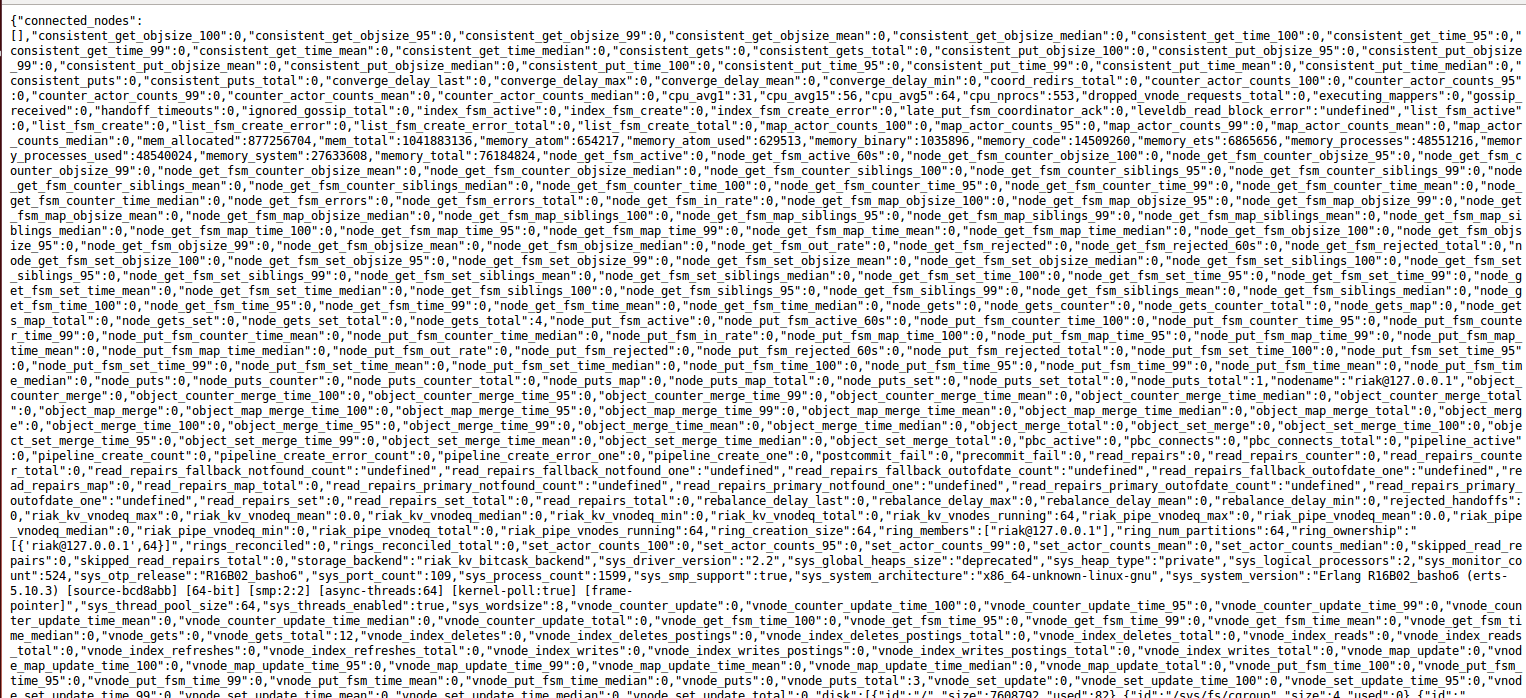
1. Open a new tab in the browser and enter the following in the address bar:
   1. [http://localhost:8098/<return](http://localhost:8098/%3creturn)>

What is displayed in the browser?



* 1. Click the stats link on the bottom. Does this output look familiar?

Where else did you see this same information?



**The same information was in the riak-admin status output.**

**------------------------------------------------------------------------------**

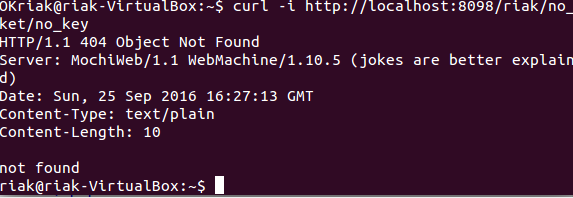
**PART II: Entering data on Riak**

**-----------------------------------------------------------------------------------**

1. Now let’s start entering data into the Riak database and see how that works. We will follow the examples in the Seven Weeks text, to avoid having to learn another language at this time. Since we are using only a single instance still, we will not be starting the multiple nodes at this time. Using the Terminal again, let’s talk to riak a new way.
   1. Enter curl http://localhost:8098/ping <return> Looking closely at the left corner of the screen, what was the output of the command? **Ok**



* 1. Enter curl -i <http://localhost:8098/riak/no_bucket/no_key>



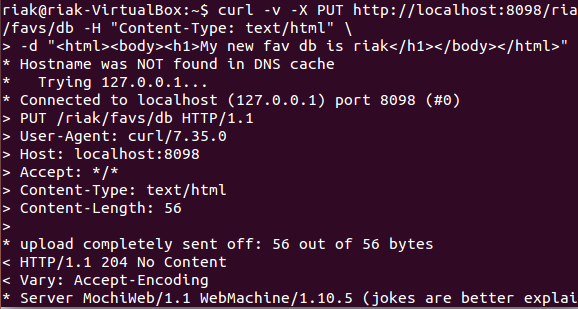
* 1. Let see how we can use riak to store html data. From page 55 of the Seven Weeks book, enter the curl statement listed at the top half of the page (changing the port number to 8098). Then goto Firefox and enter the corresponding http address.

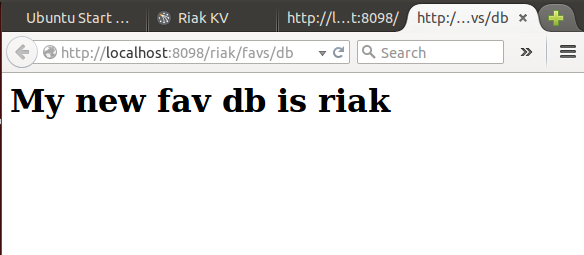
curl –v –X PUT [http://localhost:8098/riak/favs/db \](http://localhost:8098/riak/favs/db%20\)

-H “Content-Type: text/html” \

-d “<html><body><h1>My new favorite DB is RIAK</h1></body></html>” <return>

Paste a screenshot of the results below:





**-----------------------------------------------------------------------------------**

1. Remember Riak is a key-value database. It uses buckets to separate the namespaces of keys so they don’t collide.
   1. In the terminal enter: curl –v –X PUT [http://localhost:8098/riak/animals/ace \](http://localhost:8098/riak/animals/ace%20\)

-H “Content-Type: application/json” \

-d ‘{“nickname” : “The Wonder Dog”,”breed “ : “German Shepard”}’<return>

* 1. Now let’s see if it was stored. List the current buckets in the database:

curl –X GET [http://localhost:8098/riak?buckets=true<return](http://localhost:8098/riak?buckets=true%3creturn)>

Enter a screenshot of the results below:

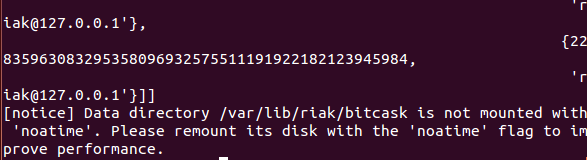


**-----------------------------------------------------------------------------------**

1. So we entered some data into riak but where is it storing the data?
   1. Enter riak-admin diag<return> into the terminal.

Look at the line that starts [notice]. It indicates the data directory. The bitcask directory is one of the directories that stores data. The actual data directory is one level up.

What is the full path of the riak data directory?



**-----------------------------------------------------------------------------------**

1. Let’s continue entering data into our animals bucket.
   1. curl –v –X PUT <http://localhost:8098/riak/animals/polly?returnbody=true> \

-H “Content-Type: application/json” \

-d ‘{“nickname” : “Sweet Polly Purebread”, “breed” : “Purebred”}’<return>

**And now the next one without a key to see how Riak automatically generates keys. Make sure to notice that this one is different. This has confused other students in the past who worked on this lab.**

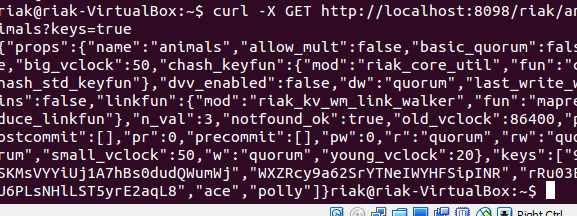
curl -i -X POST <http://localhost:8098/riak/animals> \  
-H "Content-Type: application/json" \

-d '{"nickname" : "Sergeant Stubby", "breed" : "Terrier"}' <return>

* 1. Now create and execute a curl request to return the information you just entered.

curl –x GET <http://localhost:8098/riak/animals>?keys=true

Paste a copy of your output below:



**-----------------------------------------------------------------------------------**

1. **Below is text from pages 56-60 in the Seven Weeks text. Please consider this section optional since many people had a hard time getting it to work. If you cannot get it to work, then skip it.**

**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

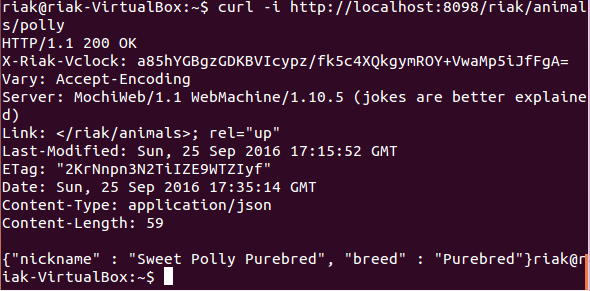
ETag: "VD0ZAfOTsIHsgG5PM3YZW"

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-Metaheader

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).

**I could not get this section to work completely. As its optional I am skipping it.**

Paste screen shot of your results below:

**I could not get this section to work completely. As its optional I am skipping it.**

When entering the color value for cage 1, enter a Size header as well and make it "Large"

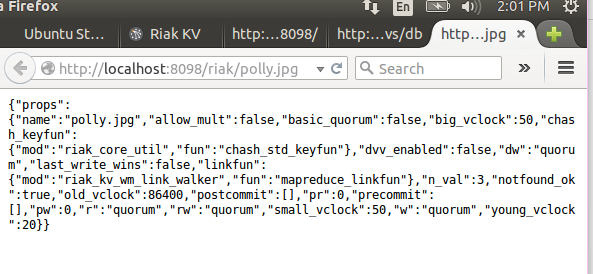
What is the command you entered to accomplish this task?

**I could not get this section to work completely. As its optional I am skipping it.**

Paste a screen shot of the query to return this header information below:

**I could not get this section to work completely. As its optional I am skipping it.**

Finally, paste a screenshot of your browser after you called up the photo you inserted into your database:



**-----------------------------------------------------------------------------------**

**Part III - Running a multi-node Riak system**

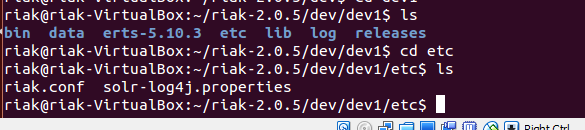
**-----------------------------------------------------------------------------------**

1. Now let's change gears a bit and move to our multi-node riak environment. but before we do that we need to make sure our single instance is shutdown.
   1. Enter riak stop<return> to stop the instance
   2. Change directory to the riak home directory: cd<return>
   3. Change directory to the riak-2.0.5 directory:

cd riak-2.0.5<return>

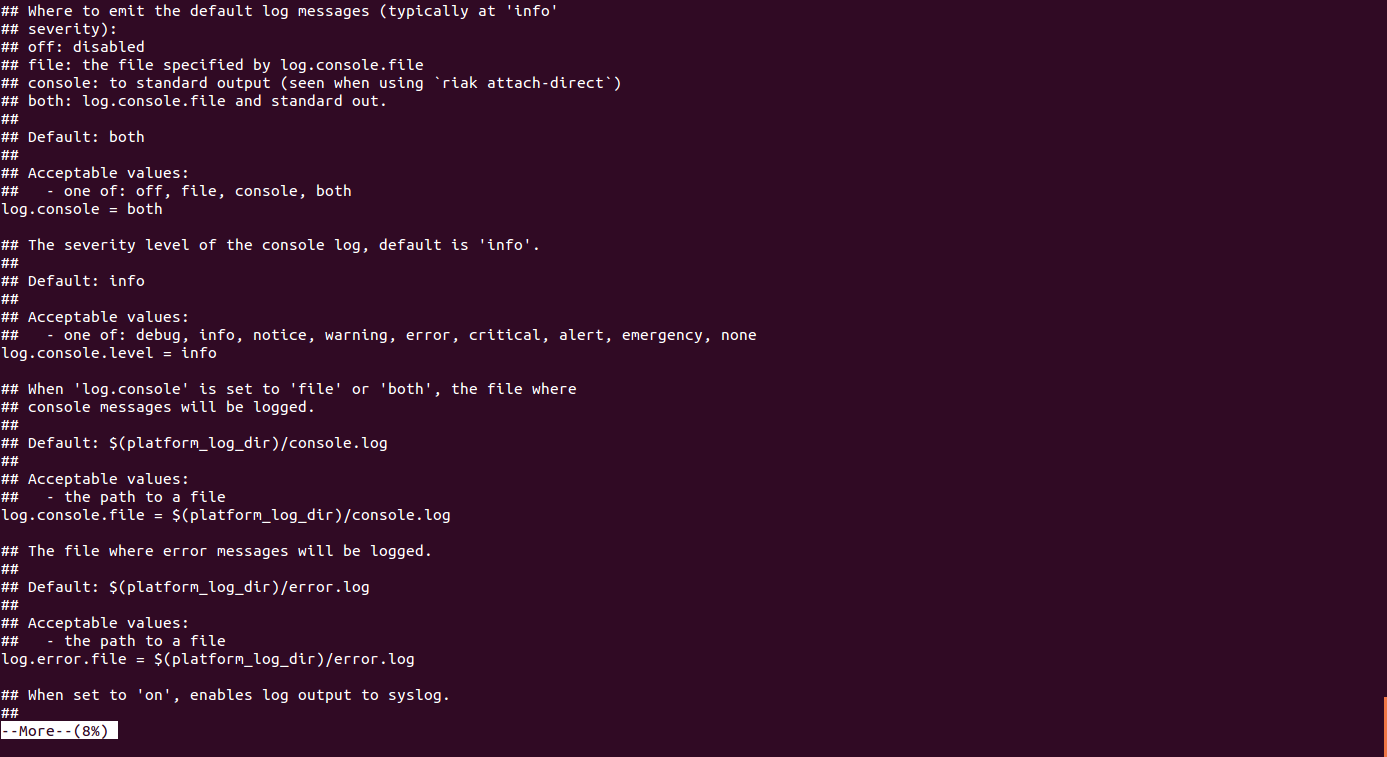
* 1. Look at a listing of this directory: ls<return>
  2. Change directory to the dev directory: cd dev<return>
  3. Notice that there are five directories labeled dev1 through dev5. Each of these directories includes a full instance of the riak system. They have already been configured as a 5 node system following the 5 Minute Install from the Basho Docs page.
  4. Enter the dev1 directory: cd dev1<return>
  5. Enter the etc directory: cd etc<return>
  6. List the files: ls<return>. Notice the riak.conf file. This is the configuration file for this particular instance of riak.

Paste a screenshot of the directory listing below:



**-----------------------------------------------------------------------------------**

1. Let's look at the configuration file for an important piece of information that we will need to interact with our riak database. Remember, the prepackaged single instance riak used port number 8098 to communicate with http. This is another separate riak system which uses a different port; we need this port number to be able to communicate with it.
   1. Enter: more riak.conf<return> You should see the file listed below.



* 1. Enter: /listener.http.internal<return>

You should now see the internal port number that we will need to communicate with riak. Note the format is: IP address:port number.

Verify the port number: 127.0.0.1:10017

Enter q<return> to exit the more application.

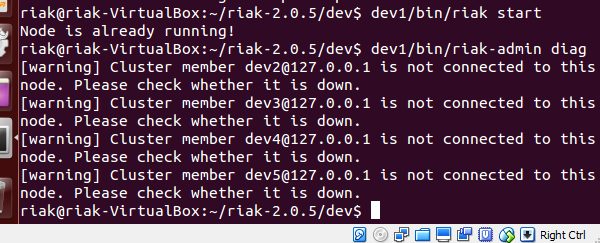
* 1. Return to the dev directory by entering: cd ../..<return>

**-----------------------------------------------------------------------------------**

1. From the dev directory, let's start our first node:
   1. Enter: dev1/bin/riak start<return>
   2. Let’s see what our diag information reports.

Enter: dev1/bin/riak-admin diag<return>

Past screenshot of results below:



**-----------------------------------------------------------------------------------**

1. Now start node 2. From the dev directory, let's start our second node:
   1. Enter: dev2/bin/riak start<return>
   2. This time let's see what the stats reports in the browser. Click on the firefox icon again and in a new tab enter the url for accessing our riak instance. When you get the initial page up, click the stats link and notice the first few lines.

* 1. Now start the remaining nodes following the same pattern as above, if possible.

Note: You may find that all the nodes are already running. This is okay.



**-----------------------------------------------------------------------------------**

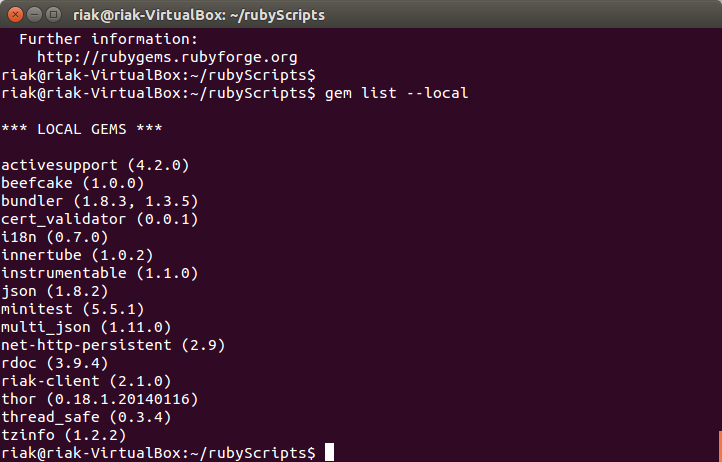
1. Now we need to do some more setup work to do the next few steps.
   1. Enter: sudo apt-get install ruby<return>
   2. Enter the riak password riak<return>
   3. Answer Y when needed
   4. When that is completed, enter:

sudo gem install riak-client json<return>

and follow the same steps as above if needed. This will install the Ruby language and some additional software for us to use.

* 1. When completed now enter: gem list --local<return>.

It should look similar to example, but it won’t match exactly:



**-----------------------------------------------------------------------------------**

1. Next we need to increase the size of our database. We will be using the ruby script found in your Seven Weeks text on page 63 with a few modifications for the updated riak software.
   1. Return to your terminal screen and goto your home directory: cd <return>
   2. Now create a new directory for scripts: mkdir myScripts<return>
   3. Click the top most icon on your launch bar. In the text field, type the word text<return> and you should see a list of icons show up. One of which should be a TextEditor. Click this icon to start the text editor program.
   4. Go to the new icon on the launch bar that represents this application and right click. Set it to lock to the dock so you can use it again later if you want a nice GUI text editor.
   5. Now copy and paste or re-type the following code into the editor and save the file as hotel.rb in the myScripts directory:

#generate loads and loads of rooms with random styles and capacities

require 'rubygems'

require 'riak'

STYLES = %w{single double queen king suite}

client = Riak::Client.new pb\_port:10017

#bucket = client.bucket 'rooms'

# a= bucket.get\_or\_new 'alice'

#a.data = 'Meeeeooowww'

#a.content\_type = 'text/plain'

#a.store

#client = Riak::Client.new(host: '127.0.0.1', pb\_port:10017)

bucket = client.bucket('rooms')

# create 100 floors to the building

for floor in 1...100

current\_rooms\_block = floor \* 100

puts "Making rooms #{current\_rooms\_block} - #{current\_rooms\_block + 100}"

# put 100 rooms on each floor

for room in 1...100

#create a unique room number as the key

# ro = Riak::RObject.new(bucket, (current\_rooms\_block + room))

ro = bucket.get\_or\_new "#{current\_rooms\_block + room}"

# Randomly gab a room style, and make up a capacity

style = STYLES[rand(STYLES.length)]

capacity = rand(8) + 1

#store the room informationas a JSON value

ro.content\_type = "application/json"

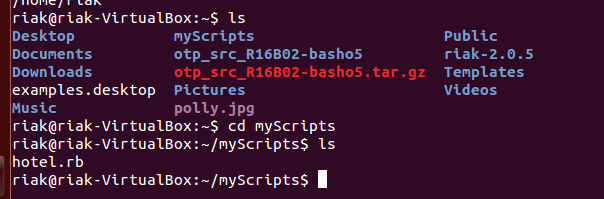
ro.data = {'style' => style,'capacity' => capacity}

ro.store

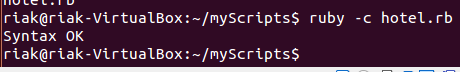
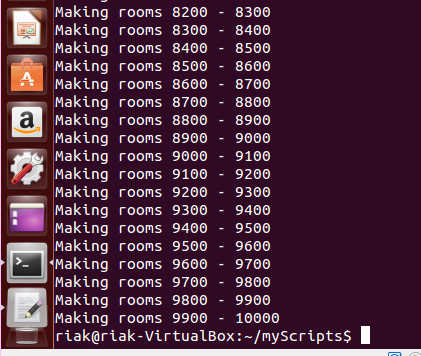
end

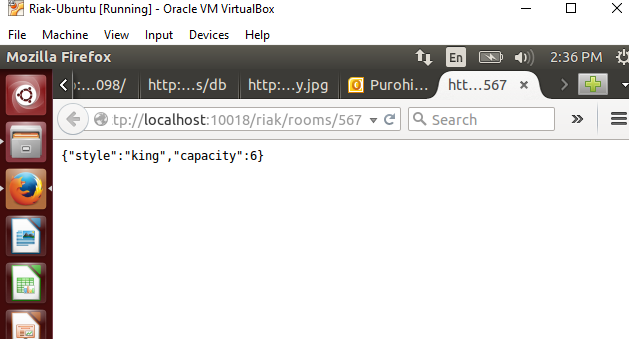
end

Paste a screenshot of your myScripts directory listing below:



**-----------------------------------------------------------------------------------**

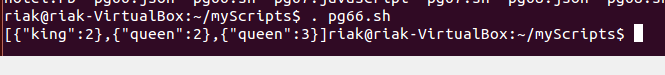
1. Return to the terminal screen and:
   1. Enter: ruby -c hotel.rb<return>
   2. What was the output of the command? 
   3. Now we will generate 10000 rooms for our riak hotel. Enter ruby hotel.rb<return>
   4. Go back to firefox and let's find out one of our room's information. Enter [http://localhost:10018/riak/rooms/567<return](http://localhost:10018/riak/rooms/567%3creturn)>
   5. Paste a screenshot of the result below:



**-----------------------------------------------------------------------------------**

1. At the bottom of page 66 of the Seven Weeks text, you see a curl statement with a mapreduce function. Enter this text as shown (except for the port number which should be 10018). (Note: you can use your editor to save the script for multiple executions.)

Paste a screenshot of your output below:



**-----------------------------------------------------------------------------------**

1. At the bottom on page 67 we store a function into our database. Complete this step and run curl statement on the following page.

Paste a screenshot of your output below:

I ran the pg67.sh and it did not return any output for me.

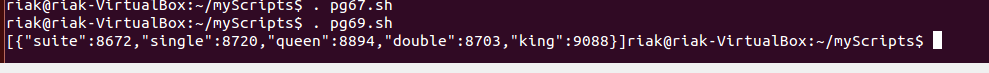
Does it match the previous screen?

Cant say, I didn’t get an output for this step.

**-----------------------------------------------------------------------------------**

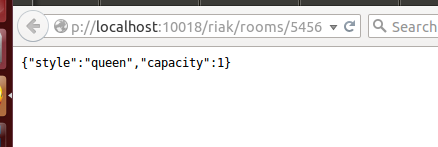
1. Now skip to page 69 and enter this curl statement.

Paste a screenshot of your results below:

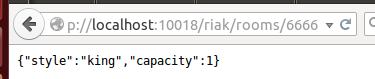
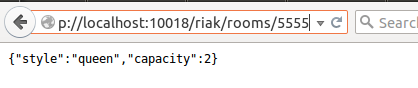


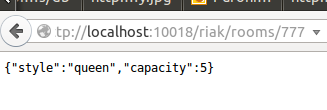
**-----------------------------------------------------------------------------------**

1. Now let's explore a little more about how riak returns some data. Go back to Firefox and enter the following in the address bar:
   1. [http://localhost:10018/riak/rooms/5456<return](http://localhost:10018/riak/rooms/5456%3creturn)>



* 1. Choose a few more room numbers from different floors and make sure you return data. Write down the room numbers here: 5555, 6666, 777,



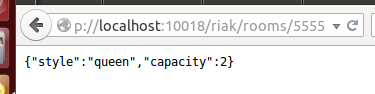


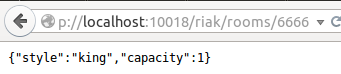
* 1. Now go to your terminal screen and cd into the dev directory: cd <return> cd riak-2.0.5/dev<return>
  2. Shutdown several nodes: dev2/bin/riak stop<return>
  3. dev3/bin/riak stop<return>
  4. dev4/bin/riak stop<return>
  5. Return to the firefox browser and reenter the same queries as before (The same room numbers)

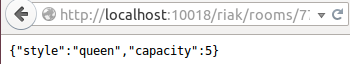
What happened? Paste a sample screenshot below:

**I am able to get the same data as before if I shutdown node 2, 3, 4 and keep 1 running.**

**I get only one result(room:5555) if I shutdown all the nodes(1, 2, 3, 4).**







We have just touched briefly on the Riak database and its capabilities. Take some time and play around with it. Continue in the Seven Weeks chapter and try some of the additional queries.

I hope you enjoyed your exploration of Riak.

Upload your completed lab assignment to the Dropbox titled for this assignment by the date specified by your facilitator.