GitHub This repository Search Explore Features Enterprise Blog Sign up Sign in

basho / riak Watch 274 Star 2,102 Fork 351

branch: develop riak / README.org

cmeiklejohn on 25 Feb Add note for required VSN for Riak.

9 contributors

349 lines (255 sloc) 11.452 kb Raw Blame History

Welcome to Riak.

Overview

Riak is a distributed, decentralized data storage system.

Below, you will find the “quick start” directions for setting up and using Riak. For more information, browse the following files:

- README: this file
- LICENSE: the license under which Riak is released
- doc/
  - admin.org: Riak Administration Guide
  - architecture.txt: details about the underlying design of Riak
  - basic-client.txt: slightly more detail on using Riak
  - basic-setup.txt: slightly more detail on setting up Riak
  - man/riak.1.gz: manual page for the riak(1) command
  - man/riak-admin.1.gz manual page for the riak-admin(1) command
  - raw-http-howto.txt: using the Riak HTTP interface

Where to find more

Below, you’ll find a basic introduction to starting and using Riak as a key/value store. For more information about Riak’s extended feature set, including MapReduce, Search, Secondary Indexes, various storage strategies, and more, please visit our docs at http://docs.basho.com/.

Quick Start

This section assumes that you have copy of the Riak source tree. To get started, you need to:

1. Build Riak
2. Start the Riak server
3. Connect a client and store/fetch data

Building Riak

Note: the `develop` branch currently only supports Erlang R16B02.

Assuming you have a working Erlang (R14B02 or later) installation, building Riak should be as simple as:

```
$ cd $RIAK
$ make rel
```

RootFile

Starting Riak

Once you have successfully built Riak, you can start the server with the following commands:

```
$ cd $RIAK/rel/riak
$ bin/riak start
```

RootFile

Now, verify that the server started up cleanly and is working:

```
$ bin/riak-admin test
```

RootFile

Note that the $RIAK/rel/riak directory is a complete, self-contained instance of Riak and Erlang. It is strongly suggested that you move this directory outside the source tree if you plan to run a production instance.

Connecting a client to Riak

Now that you have a functional server, let's try storing some data in it. First, start up a erlang node using our embedded version of erlang:

```
$ erts-<vsn>/bin/erl -name riaktest@127.0.0.1 -setcookie riak

Eshell V5.7.4  (abort with ^G)
(riaktest@127.0.0.1)1>
```

RootFile inserting the shown text into the programs stdin

Now construct the node name of Riak server and make sure we can talk to it:

```
(riaktest@127.0.0.1)4> RiakNode = 'riak@127.0.0.1'.

(riaktest@127.0.0.1)2> net_adm:ping(RiakNode).
pong
(riaktest@127.0.0.1)2>
```

ContinueFile

We are now ready to start the Riak client:

```
(riaktest@127.0.0.1)2> {ok, C} = riak:client_connect(RiakNode).
{ok,{riak_client,'riak@127.0.0.1',<<4,136,81,151>>}}
```

same

Let's create a shopping list for bread at /groceries/mine:

```
(riaktest@127.0.0.1)6> O0 = riak_object:new(<<"groceries">>, <<"mine">>, ["bread"]).
O0 = riak_object:new(<<"groceries">>, <<"mine">>, ["bread"]).
{r_object,<<"groceries">>,<<"mine">>,
       [{r_content,{dict,0,16,16,8,80,48,
                         {[],[],[],[],[],[],[],[],[],[],[],[],[],[],...},
                         {{[],[],[],[],[],[],[],[],[],[],[],[],...}}},
                   ["bread"]}],
       [],
       {dict,1,16,16,8,80,48,
             {[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],...},
             {{[],[],[],[],[],[],[],[],[],[],[],[],[],...}}},
       undefined}

 (riaktest@127.0.0.1)3> C:put(O0, 1).
```

same

Now, read the list back from the Riak server and extract the value

```
(riaktest@127.0.0.1)4> {ok, O1} = C:get(<<"groceries">>, <<"mine">>, 1).
{ok,{r_object,<<"groceries">>,<<"mine">>,
          [{r_content,{dict,2,16,16,8,80,48,
                            {[],[],[],[],[],[],[],[],[],[],[],[],...},
                            {{[],[],[],[],[],[],
                              [["X-Riak-Last-Modified",87|...]],
                              [],[],[],...}}},
```

same

```
                       ["bread"]}],
             [{"20090722191020-riaktest@127.0.0.1-riakdemo@127.0.0.1-266664",
               {1,63415509105}}],
             {dict,0,16,16,8,80,48,
                   {[],[],[],[],[],[],[],[],[],[],[],[],[],...},
                   {{[],[],[],[],[],[],[],[],[],[],[],...}}},
             undefined}}

    (riaktest@127.0.0.1)5> %% extract the value
    (riaktest@127.0.0.1)5> V = riak_object:get_value(O1).
    ["bread"]
```

Add milk to our list of groceries and write the new value to Riak:

```
(riaktest@127.0.0.1)6> %% add milk to the list
(riaktest@127.0.0.1)6> O2 = riak_object:update_value(O1, ["milk" | V]).
{r_object,<<"groceries">>,<<"mine">>,
     [{r_content,{dict,2,16,16,8,80,48,
                      {[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],...},
                      {{[],[],[],[],[],[],
                        [["X-Riak-Last-Modified",87,101,100|...]],
                        [],[],[],[],[],...}}},
                ["bread"]}],
     [{"20090722191020-riaktest@127.0.0.1-riakdemo@127.0.0.1-266664",
       {1,63415509105}}],
     {dict,0,16,16,8,80,48,
           {[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],...},
           {{[],[],[],[],[],[],[],[],[],[],[],[],[],[],...}}},
     ["milk","bread"]}

(riaktest@127.0.0.1)7> %% store the new list
(riaktest@127.0.0.1)7> C:put(O2, 1).
ok
```

same

Finally, see what other keys are available in groceries bucket:

same

```
(riaktest@127.0.0.1)8> C:list_keys(<<"groceries">>).
{ok,[<<"mine">>]}
```

Clients for Other Languages

Client libraries are available for many languages. Rather than bundle them with the Riak server source code, we have given them each their own source repository. Currently, official Riak client language libraries include:

- Javascript https://github.com/basho/riak-javascript-client
- Python https://github.com/basho/riak-python-client
- Ruby https://github.com/basho/riak-ruby-client
- Java https://github.com/basho/riak-java-client
- PHP https://github.com/basho/riak-php-client
- Erlang https://github.com/basho/riak-erlang-client (using protocol buffers instead of distributed Erlang)

Server Management

Configuration

Configuration for the Riak server is stored in $RIAK/rel/riak/etc directory. There are two files:

- vm.args This file contains the arguments that are passed to the Erlang VM in which Riak runs. The default settings in this file shouldn't need to be changed for most environments.
- app.config This file contains the configuration for the Erlang applications that run on the Riak server.

More information about this files is available in doc/basic-setup.txt.

Server Control

bin/riak

This script is the primary interface for starting and stopping the Riak server.

To start a daemonized (background) instance of Riak:

```
$ bin/riak start
```

Once a server is running in the background you can attach to the Erlang console via:

```
$ bin/riak attach
```

Alternatively, if you want to run a foreground instance of Riak, start it with:

```
$ bin/riak console
```

Stopping a foreground or background instance of Riak can be done from a shell prompt via:

```
$ bin/riak stop
```

Or if you are attached/on the Erlang console:

```
(riak@127.0.0.1)1> q().
```

You can determine if the server is running by:

```
$ bin/riak ping
```

bin/riak-admin

This script provides access to general administration of the Riak server. The below commands assume you are running a default configuration for parameters such as cookie.

To join a new Riak node to an existing cluster:

```
$ bin/riak start # If a local server is not already running
$ bin/riak-admin join <node in cluster>
```

(Note that you must have a local node already running for this to work)

To verify that the local Riak node is able to read/write data:

```
$ bin/riak-admin test
```

To backup a node or cluster run the following:

```
$ bin/riak-admin backup riak@X.X.X.X riak <directory/backup_file> node
$ bin/riak-admin backup riak@X.X.X.X riak <directory/backup_file> all
```

Restores can function in two ways, if the backup file was of a node the node will be restored and if the backup file contains the data for a cluster all nodes in the cluster will be restored.

To restore from a backup file:

```
$ riak-admin restore riak@X.X.X.X riak <directory/backup_file>
```

To view the status of a node:

```
$ bin/riak-admin status
```

If you change the IP or node name you will need to use the reip command:

```
$ bin/riak-admin reip <old_nodename> <new_nodename>
```

Contributing to Riak and Reporting Bugs

Basho encourages contributions to Riak from the community. Here's how to get started.

- Fork the appropriate sub-projects that are affected by your change. Fork this repository if your changes are for release generation or packaging.
- Make your changes and run the test suite. (see below)
- Commit your changes and push them to your fork.
- Open pull-requests for the appropriate projects.
- Basho engineers will review your pull-request, suggest changes, and merge it when it's ready and/or offer feedback.

To report a bug or issue, please open a new issue against this repository.

You can read the full guidelines for bug reporting and code contributions on the Riak Docs.

Testing

To make sure your patch works, be sure to run the test suite in each modified sub-project, and dialyzer from the top-level project to detect static code errors.

To run the QuickCheck properties included in Riak sub-projects, download QuickCheck Mini: http://quviq.com/index.html NOTE: Some properties that require features in the Full version will fail.

Running unit tests

The unit tests for each subproject can be run with `make` or `rebar` like so:

```
make eunit
```

```
./rebar skip_deps=true eunit
```

Running dialyzer

Dialyzer performs static analysis of the code to discover defects, edge-cases and discrepancies between type specifications and the actual implementation.

Dialyzer requires a pre-built code analysis table called a PLT. Building a PLT is expensive and can take up to 30 minutes on some machines. Once built, you generally want to avoid clearing or rebuilding the PLT unless you have had significant changes in your build (a new version of Erlang, for example).

Build the PLT

Run the command below to build the PLT.

```
make build_plt
```

Check the PLT

If you have built the PLT before, check it before you run Dialyzer again. This will take much less time than building the PLT from scratch.

```
make check_plt
```

Run Dialyzer

```
make dialyzer
```