Let’s Write A Test!

@joedevivo
basho technologies

http://joedevivo.github.io/euc2013

I love Sweden!

The Vasa

?


THE ORIGINAL
Swedish Fish®
SOFT & CHEWY CANDY
NET WT. 2 OZ (57 g)
Swedish


Delta
Frontier
Jet Blue
Sun Country
DOCTOR WHO
MY OTHER CAR IS A TARDIS

Engineering
since like forever

Build Thing

Test Thing

Use Thing

Software Engineering

now we have options!

Test Thing

Build Thing

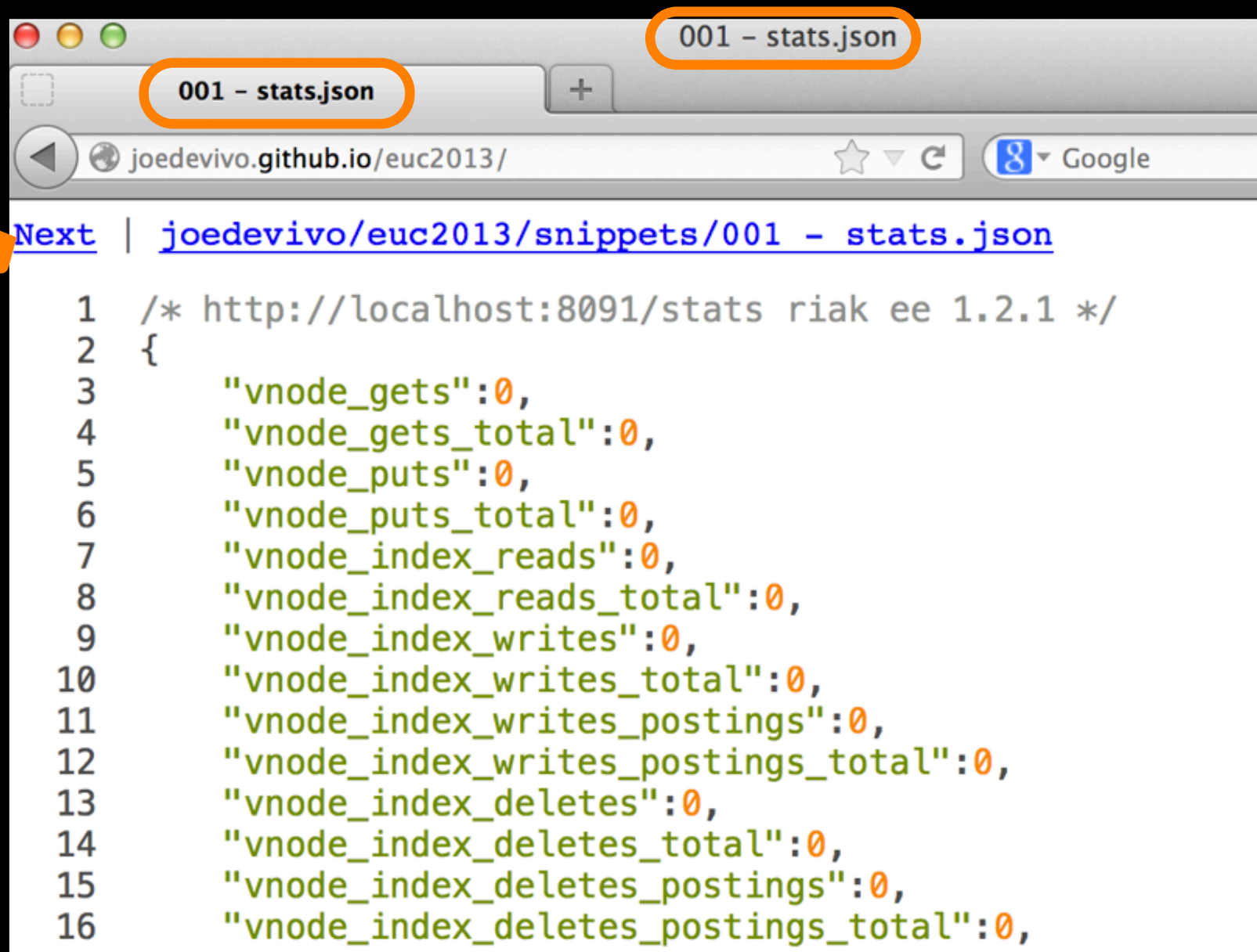
Use Thing

these ➜ code: 001

How to watch this talk

go here ➜ http://joedevivo.github.io/euc2013

that! ➜



Why are we here?

Add a feature to Riak

with riak_test

using TDD

Add a feature to Riak

with riak_test

using TDD

Riak is Basho's open-source, available, fault-tolerant, operationally simple, scalable, so you can rest databass

Add a feature to Riak

with riak_test

using TDD

riak_test is Basho's functional testing framework for testing distributed systems

riak_test

- does our release validation
- its roots are as a TDD framework

Add a feature to Riak with riak_test using TDD

Feature?

JMX Monitoring for Riak

Why JMX?

- It's pretty isolated
- We don't need to talk about complexities of riak
- It's closed source!
- I worked on it

Umm… hasn’t Riak JMX been around for a while?

What is Riak JMX?

/stats exposed!
(in theory)

what's /stats?

- http://riak-node:8098/stats
- riak-admin status
- general information and statistics about the node

code: 001


```
{
    "vnode_gets":0,
    "vnode_gets_total":0,
    "vnode_puts":0,
    "vnode_puts_total":0,
    "vnode_index_reads":0,
    "vnode_index_reads_total":0,
    "vnode_index_writes":0,
    "vnode_index_writes_total":0,
    "vnode_index_writes_postings":0,
    "vnode_index_writes_postings_total":0,
    "vnode_index_deletes":0,
    "vnode_index_deletes_total":0,
    "vnode_index_deletes_postings":0,
    "vnode_index_deletes_postings_total":0,
    "node_gets":0,
    "node_gets_total":0,
    "node_get_fsm_siblings_mean":0,
    "node_get_fsm_siblings_median":0,
    "node_get_fsm_siblings_95":0,
    "node_get_fsm_siblings_99":0,
    "node_get_fsm_siblings_100":0,
    "node_get_fsm_objsize_mean":0,
    "node_get_fsm_objsize_median":0,
    "node_get_fsm_objsize_95":0,
    "node_get_fsm_objsize_99":0,
    "node_get_fsm_objsize_100":0,
```

```
[// →  java -cp  riak_jmx.jar com.basho.riak.jmx.Dump localhost 41110
    {"CPUNProcs":145},
    {"MemAllocated":5888280064},
    {"NodeGets":0},
    {"NodeGetsTotal":0},
    {"NodePuts":0},
    {"NodePutsTotal":0},
    {"MemTotal":7924740096},
    {"VnodeGets":0},
    {"VnodeGetsTotal":0},
    {"VnodePuts":0},
    {"VnodePutsTotal":0},
    {"PbcActive":0},
    {"PbcConnects":0},
    {"PbcConnectsTotal":0},
    {"NodeName":"dev1@127.0.0.1"},
    {"RingCreationSize":64},
    {"CpuAvg1":320},
    {"CpuAvg5":279},
    {"CpuAvg15":274},
    {"NodeGetFsmTimeMax":0},
    {"NodeGetFsmTimeMean":0},
    {"NodeGetFsmTimeMedian":0},
    {"NodeGetFsmTime95":0},
    {"NodeGetFsmTime99":0},
    {"NodePutFsmTimeMax":0},
    {"NodePutFsmTimeMean":0},
    {"NodePutFsmTimeMedian":0},
    {"NodePutFsmTime95":0},
    {"NodePutFsmTime99":0},
    {"ReadRepairs":0},
    {"ReadRepairsTotal":0}
```

code: 002

```
{
    "vnode_gets":0,
    "vnode_gets_total":0,
    "vnode_puts":0,
    "vnode_puts_total":0,
    "vnode_index_reads":0,
    "vnode_index_reads_total":0,
    "vnode_index_writes":0,
    "vnode_index_writes_total":0,
    "vnode_index_writes_postings":0,
    "vnode_index_writes_postings_total":0,
    "vnode_index_deletes":0,
    "vnode_index_deletes_total":0,
    "vnode_index_deletes_postings":0,
    "vnode_index_deletes_postings_total":0,
    "node_gets":0,
    "node_gets_total":0,
    "node_get_fsm_siblings_mean":0,
    "node_get_fsm_siblings_median":0,
    "node_get_fsm_siblings_95":0,
    "node_get_fsm_siblings_99":0,
    "node_get_fsm_siblings_100":0,
    "node_get_fsm_objsize_mean":0,
    "node_get_fsm_objsize_median":0,
    "node_get_fsm_objsize_95":0,
    "node_get_fsm_objsize_99":0,
    "node_get_fsm_objsize_100":0,
    "node_get_fsm_time_mean":0,
    "node_get_fsm_time_median":0,
    "node_get_fsm_time_95":0,
    "node_get_fsm_time_99":0,
    "node_get_fsm_time_100":0,
    "node_puts":0,
    "node_puts_total":0,
    "node_put_fsm_time_mean":0,
    "node_put_fsm_time_median":0,
    "node_put_fsm_time_95":0,
    "node_put_fsm_time_99":0,
    "node_put_fsm_time_100":0,
    "read_repairs":0,
    "read_repairs_total":0,
    "coord_redirs_total":0,
    "executing_mappers":0,
    "precommit_fail":0,
    "postcommit_fail":0,
    "pbc_active":0,
    "pbc_connects":0,
    "pbc_connects_total":0,
    "cpu_nprocs":155,
    "cpu_avg1":233,
    "cpu_avg5":228,
    "cpu_avg15":241,
    "mem_total":8164060000,
    "mem_allocated":7943324000,
    "nodename":"dev1@127.0.0.1",
    "connected_nodes":[],
    "sys_driver_version":"2.0",
    "sys_global_heaps_size":0,
    "sys_heap_type":"private",
    "sys_logical_processors":4,
    "sys_otp_release":"R15B01",
    "sys_process_count":1440,
    "sys_smp_support":true,
    "sys_system_version":"Erlang R15B01 (erts-5.9.1) [source] [64-bit] [smp:4:4] [async-threads:64] [kernel-poll:true]",
    "sys_system_architecture":"i386-apple-darwin12.2.1",
    "sys_threads_enabled":true,
    "sys_thread_pool_size":64,
    "sys_wordsize":8,
    "ring_members":["dev1@127.0.0.1"],
    "ring_num_partitions":64,
    "ring_ownership":"[{'dev1@127.0.0.1',64}]",
    "ring_creation_size":64,
    "storage_backend":"riak_kv_bitcask_backend",
    "erlydtl_version":"0.7.0",
    "riak_control_version":"1.2.0",
    "cluster_info_version":"1.2.2",
    "riak_jmx_version":"1.2.0",
    "riak_snmp_version":"1.2.0",
    "riak_repl_version":"1.2.1",
    "ranch_version":"0.2.1",
    "riak_api_version":"1.2.0",
    "riak_search_version":"1.2.1",
    "merge_index_version":"1.2.1",
    "riak_kv_version":"1.2.1",
    "riak_pipe_version":"1.2.1",
    "riak_core_version":"1.2.1",
    "lager_version":"1.2.0",
    "syntax_tools_version":"1.6.8",
    "compiler_version":"4.8.1",
    "bitcask_version":"1.5.2",
    "basho_stats_version":"1.0.2",
    "luke_version":"0.2.5",
    "webmachine_version":"1.9.2",
    "mochiweb_version":"1.5.1",
    "inets_version":"5.9",
    "erlang_js_version":"1.2.1",
    "mnesia_version":"4.7",
    "snmp_version":"4.22",
    "runtime_tools_version":"1.8.8",
    "os_mon_version":"2.2.9",
    "riak_sysmon_version":"1.1.2",
    "ssl_version":"5.0.1",
    "public_key_version":"0.15",
    "crypto_version":"2.1",
    "sasl_version":"2.2.1",
    "stdlib_version":"1.18.1",
    "kernel_version":"2.15.1",
    "memory_total":30986792,
    "memory_processes":9083110,
    "memory_processes_used":9083050,
    "memory_system":21903682,
    "memory_atom":561761,
    "memory_atom_used":546330,
    "memory_binary":71424,
    "memory_code":13914313,
    "memory_ets":1977760,
    "ignored_gossip_total":0,
    "rings_reconciled_total":0,
    "rings_reconciled":0,
    "gossip_received":0,
    "rejected_handoffs":0,
    "handoff_timeouts":0,
    "converge_delay_min":0,
    "converge_delay_max":0,
    "converge_delay_mean":0,
    "converge_delay_last":0,
    "rebalance_delay_min":0,
```

```
[
    {"CPUNProcs":145},
    {"MemAllocated":5888280064},
    {"NodeGets":0},
    {"NodeGetsTotal":0},
    {"NodePuts":0},
    {"NodePutsTotal":0},
    {"MemTotal":7924740096},
    {"VnodeGets":0},
    {"VnodeGetsTotal":0},
    {"VnodePuts":0},
    {"VnodePutsTotal":0},
    {"PbcActive":0},
    {"PbcConnects":0},
    {"PbcConnectsTotal":0},
    {"NodeName":"dev1@127.0.0.1"},
    {"RingCreationSize":64},
    {"CpuAvg1":320},
    {"CpuAvg5":279},
    {"CpuAvg15":274},
    {"NodeGetFsmTimeMax":0},
    {"NodeGetFsmTimeMean":0},
    {"NodeGetFsmTimeMedian":0},
    {"NodeGetFsmTime95":0},
    {"NodeGetFsmTime99":0},
    {"NodePutFsmTimeMax":0},
    {"NodePutFsmTimeMean":0},
    {"NodePutFsmTimeMedian":0},
    {"NodePutFsmTime95":0},
    {"NodePutFsmTime99":0},
    {"ReadRepairs":0},
    {"ReadRepairsTotal":0}
]
```

history lesson:

Riak JMX predates riak_test

Why does that matter?

We can’t go full big TDD on it

Let’s go back in time to the riak 1.3 release

Reuse!

verify_stats.erl code: 003

```
confirm() ->
    Nodes = [Node1] = rt:deploy_nodes(1),
    ?assertEqual(ok,
        rt:wait_until_nodes_ready(Nodes)),
```

```
Nodes = [Node1, Node2, Node3] = rt:deploy_nodes(3),
```

```
                Stats1 = get_stats(Node1),


get_stats(Node) ->
    timer:sleep(10000),
    StatString = os:cmd(
        io_lib:format("curl -s -S ~s/stats",
                      [rt:http_url(Node)])),
    {struct, Stats} = mochijson2:decode(StatString),
    Stats.
```

```
    %% make sure a set of stats have valid values
    verify_nz(Stats1,[<<"cpu_nprocs">>,
                      <<"mem_total">>,
                      <<"mem_allocated">>,
                      <<"sys_logical_processors">>,
                      <<"sys_process_count">>,
                      <<"sys_thread_pool_size">>,
                      <<"sys_wordsize">>,
                      <<"ring_num_partitions">>,
                      <<"ring_creation_size">>,
                      <<"memory_total">>,
                      <<"memory_processes">>,
                      <<"memory_processes_used">>,
                      <<"memory_system">>,
                      <<"memory_atom">>,
                      <<"memory_atom_used">>,
                      <<"memory_binary">>,
                      <<"memory_code">>,
                      <<"memory_ets">>]),
```

```
verify_nz(Props, Keys) ->
    [?assertNotEqual(
        proplists:get_value(Key,Props,0),
        0)
    || Key <- Keys].
```

```
    %% make sure a set of stats have valid values
    verify_nz(Stats1,[<<"cpu_nprocs">>,
                      <<"mem_total">>,
                      <<"mem_allocated">>,
                      <<"sys_logical_processors">>,
                      <<"sys_process_count">>,
                      <<"sys_thread_pool_size">>,
                      <<"sys_wordsize">>,
                      <<"ring_num_partitions">>,
                      <<"ring_creation_size">>,
                      <<"memory_total">>,
                      <<"memory_processes">>,
                      <<"memory_processes_used">>,
                      <<"memory_system">>,
                      <<"memory_atom">>,
                      <<"memory_atom_used">>,
                      <<"memory_binary">>,
                      <<"memory_code">>,
                      <<"memory_ets">>]),
```

```
    lager:info(
      "perform 5 x  PUT and a GET to increment the stats"),
    lager:info(
      "as the stat system only does calcs for > 5 readings"),

    C = rt:httpc(Node1),

    [rt:httpc_write(C,
                    <<"systest">>,
                    <<X>>,
                    <<"12345">>)
    || X <- lists:seq(1, 5)],

    [rt:httpc_read(C,
                   <<"systest">>,
                   <<X>>)
    || X <- lists:seq(1, 5)],
```

```
Stats1 = get_stats(Node1),
```

remember me?

I’m still bound!

```
Stats2 = get_stats(Node1),
```

```
    %% make sure the stats that were supposed to increment did
    verify_inc(Stats1, Stats2, [{<<"node_gets">>, 10},
                                {<<"node_puts">>, 5},
                                {<<"node_gets_total">>, 10},
                                {<<"node_puts_total">>, 5},
                                {<<"vnode_gets">>, 30},
                                {<<"vnode_puts">>, 15},
                                {<<"vnode_gets_total">>, 30},
                                {<<"vnode_puts_total">>, 15}]),
```

```
    verify_inc(Stats1, Stats2, [{<<"node_gets">>, 10},
                                ...]),



verify_inc(Prev, Props, Keys) ->
    [begin
         Old = proplists:get_value(Key, Prev, 0),
         New = proplists:get_value(Key, Props, 0),
         lager:info("~s: ~p -> ~p (expected ~p)",
           [Key, Old, New, Old + Inc]),
         ?assertEqual(New, (Old + Inc))
     end || {Key, Inc} <- Keys].
```

```
    %% make sure the stats that were supposed to increment did
    verify_inc(Stats1, Stats2, [{<<"node_gets">>, 10},
                                {<<"node_puts">>, 5},
                                {<<"node_gets_total">>, 10},
                                {<<"node_puts_total">>, 5},
                                {<<"vnode_gets">>, 30},
                                {<<"vnode_puts">>, 15},
                                {<<"vnode_gets_total">>, 30},
                                {<<"vnode_puts_total">>, 15}]),
```

```
    %% verify that fsm times were tallied
    verify_nz(Stats2, [<<"node_get_fsm_time_mean">>,
                       <<"node_get_fsm_time_median">>,
                       <<"node_get_fsm_time_95">>,
                       <<"node_get_fsm_time_99">>,
                       <<"node_get_fsm_time_100">>,
                       <<"node_put_fsm_time_mean">>,
                       <<"node_put_fsm_time_median">>,
                       <<"node_put_fsm_time_95">>,
                       <<"node_put_fsm_time_99">>,
                       <<"node_put_fsm_time_100">>]),
```

```
lager:info("Make PBC Connection"),
Pid = rt:pbc(Node1),

Stats3 = get_stats(Node1),
rt:systest_write(Node1, 1),
%% make sure the stats that were supposed to increment did
verify_inc(Stats2, Stats3, [{<<"pbc_connects_total">>, 1},
                            {<<"pbc_connects">>, 1},
                            {<<"pbc_active">>, 1}]),
```

```
    lager:info("Force Read Repair"),
    rt:pbc_write(Pid, <<"testbucket">>, <<"1">>, <<"blah!">>),
    rt:pbc_set_bucket_prop(Pid, <<"testbucket">>, [{n_val, 4}]),

    Stats4 = get_stats(Node1),
    verify_inc(Stats3, Stats4, [{<<"read_repairs_total">>, 0},
                                {<<"read_repairs">>, 0}]),

    _Value = rt:pbc_read(Pid, <<"testbucket">>, <<"1">>),

    Stats5 = get_stats(Node1),

    verify_inc(Stats3, Stats5, [{<<"read_repairs_total">>, 1},
                                {<<"read_repairs">>, 1}]),
```

pass.

large font because this is great!

What’s different about JMX?

Oh no!

Dump.java code: 004

```
for( ObjectInstance instance : beans )
{
    MBeanInfo info = mbs.getMBeanInfo( instance.getObjectName() );
    MBeanAttributeInfo[] attribs = info.getAttributes();
    for (MBeanAttributeInfo attr: attribs)
    {
        String name = attr.getName();
        Object value = mbs.getAttribute(riakBeanName, name);
        results.object().key(name);
        if(value instanceof Number)
        {
            results.value(((Number)value).longValue());
        } else {
            results.value(value);
        }
        results.endObject();
    }
}
System.out.println(results.endArray().toString());
```

jmx_verify.erl

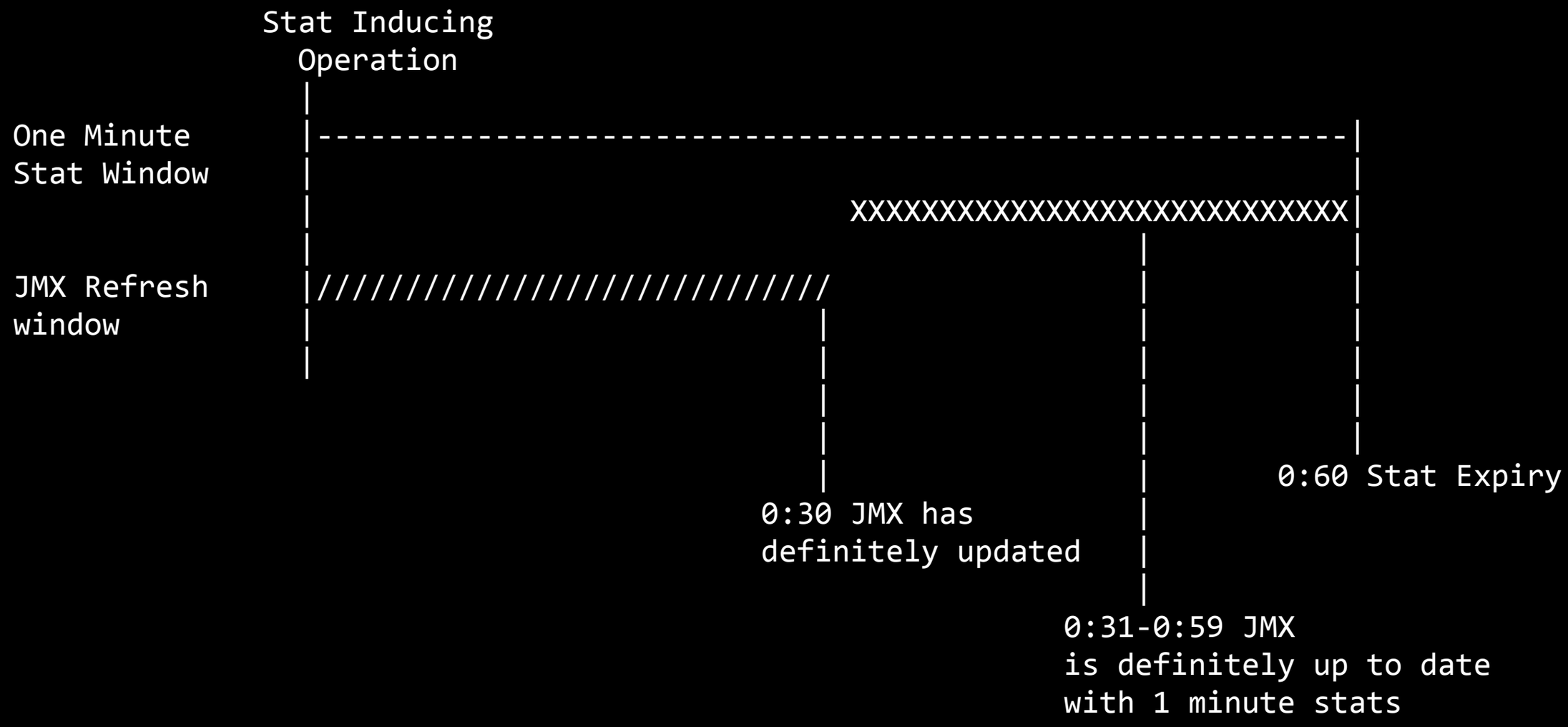
code: 005

```
    JMXDumpCmd = jmx_dump_cmd(IP, JMXPort),
    JMX1 = jmx_dump(JMXDumpCmd),
```

```
jmx_jar_path() ->
    %% Find riak_jmx.jar
    DepsPath = rt:get_deps(),
    Deps = string:tokens(os:cmd("ls " ++ DepsPath), "\n"),
    [RiakJMX] = lists:filter(fun(X) -
> string:str(X, "riak_jmx") == 1 end, Deps),
    filename:join([DepsPath, RiakJMX, "priv", "riak_jmx.jar"]).

jmx_dump_cmd(IP, Port) ->
    io_lib:format("java -cp ~s com.basho.riak.jmx.Dump ~s ~p",
        [jmx_jar_path(), IP, Port]).

jmx_dump(Cmd) ->
    timer:sleep(40000), %% JMX only updates every 30seconds
    Output = string:strip(os:cmd(Cmd), both, $\n),
    JSONOutput = mochijson2:decode(Output),
    [ {Key, Value} || {struct, [{Key, Value}]} <- JSONOutput].
```


Stat Inducing
Operation
One Minute
Stat Window
JMX Refresh
window
0:30 JMX has
definitely updated
0:31-0:59 JMX
is definitely up to date
with 1 minute stats
0:60 Stat Expiry

How do we run it?

```
➜  riak_ee git:(master) make devrel
...
➜  riak_ee git:(master) ./riak_test/bin/rteedev-current.sh
Making /Users/joe/dev/basho/riak_ee the current release:
 - Determining version: riak_ee-1.4wip-70-gf5768c5-master
 - Resetting existing /Users/joe/riak_test_ee
 - Removing and recreating /Users/joe/riak_test_ee/current
 - Copying devrel to /Users/joe/riak_test_ee/current
 - Writing /Users/joe/riak_test_ee/current/VERSION
 - Reinitializing git state
➜  riak_ee git:(master) cd ../riak_test
➜  riak_test git:(master) make clean && make
➜  riak_test git:(master)./riak_test -c ee -t jmx_verify
```

The test should work!

It doesn’t

Parity

We’ll rage on this later

code: 006

```
============== jmx_verify failure stack trace ==================
{{assertNotEqual_failed,[{module,jmx_verify},
                         {line,113},
                         {expression,"{ Key , 0 }"},
                         {value,{<<"cpu_nprocs">>,0}}]},
 [{jmx_verify,'-verify_nz/2-fun-0-',2,
              [{file,"tests/jmx_verify.erl"},{line,113}]},
  {jmx_verify,'-verify_nz/2-lc$^0/1-0-',2,
              [{file,"tests/jmx_verify.erl"},{line,113}]},
  {jmx_verify,confirm,0,
              [{file,"tests/jmx_verify.erl"},{line,42}]}]}
================================================================
```


THERE ARE NO cpu_nprocs!

code: 007

cpu_nprocs	CPUNProcs
mem_allocated	MemAllocated
mem_total	MemTotal
node_gets	NodeGets
node_gets_total	NodeGetsTotal
node_puts	NodePuts
node_puts_total	NodePutsTotal
vnode_gets	VnodeGets
vnode_gets_total	VnodeGetsTotal
vnode_puts	VnodePuts
vnode_puts_total	VnodePutsTotal
pbc_active	PbcActive
pbc_connects	PbcConnects
pbc_connects_total	PbcConnectsTotal
nodename	NodeName
ring_creation_size	RingCreationSize
cpu_avg1	CpuAvg1
cpu_avg5	CpuAvg5
cpu_avg15	CpuAvg15
node_get_fsm_time_95	NodeGetFsmTime95
node_get_fsm_time_99	NodeGetFsmTime99
node_get_fsm_time_100	NodeGetFsmTimeMax
node_get_fsm_time_mean	NodeGetFsmTimeMean
node_get_fsm_time_median	NodeGetFsmTimeMedian
node_put_fsm_time_95	NodePutFsmTime95
node_put_fsm_time_99	NodePutFsmTime99
node_put_fsm_time_100	NodePutFsmTimeMax
node_put_fsm_time_mean	NodePutFsmTimeMean
node_put_fsm_time_median	NodePutFsmTimeMedian
read_repairs	ReadRepairs
read_repairs_total	ReadRepairsTotal

code: 007

```
CPUNProcs -> cpu_nprocs
```

```
process_key(<<"CPUNProcs">>) -> <<"cpu_nprocs">>;
process_key(<<"MemAllocated">>) -> <<"mem_allocated">>;
process_key(<<"MemTotal">>) -> <<"mem_total">>;
process_key(<<"NodeGets">>) -> <<"node_gets">>;
process_key(<<"NodeGetsTotal">>) -> <<"node_gets_total">>;
process_key(<<"NodePuts">>) -> <<"node_puts">>;
process_key(<<"NodePutsTotal">>) -> <<"node_puts_total">>;
process_key(<<"VnodeGets">>) -> <<"vnode_gets">>;
process_key(<<"VnodeGetsTotal">>) -> <<"vnode_gets_total">>;
process_key(<<"VnodePuts">>) -> <<"vnode_puts">>;
process_key(<<"VnodePutsTotal">>) -> <<"vnode_puts_total">>;
process_key(<<"PbcActive">>) -> <<"pbc_active">>;
process_key(<<"PbcConnects">>) -> <<"pbc_connects">>;
process_key(<<"PbcConnectsTotal">>) -> <<"pbc_connects_total">>;
process_key(<<"NodeName">>) -> <<"nodename">>;
process_key(<<"RingCreationSize">>) -> <<"ring_creation_size">>;
process_key(<<"CpuAvg1">>) -> <<"cpu_avg1">>;
process_key(<<"CpuAvg5">>) -> <<"cpu_avg5">>;
process_key(<<"CpuAvg15">>) -> <<"cpu_avg15">>;
process_key(<<"NodeGetFsmTime95">>) -> <<"node_get_fsm_time_95">>;
process_key(<<"NodeGetFsmTime99">>) -> <<"node_get_fsm_time_99">>;
process_key(<<"NodeGetFsmTimeMax">>) -> <<"node_get_fsm_time_100">>;
process_key(<<"NodeGetFsmTimeMean">>) -> <<"node_get_fsm_time_mean">>;
process_key(<<"NodeGetFsmTimeMedian">>) -> <<"node_get_fsm_time_median">>;
process_key(<<"NodePutFsmTime95">>) -> <<"node_put_fsm_time_95">>;
process_key(<<"NodePutFsmTime99">>) -> <<"node_put_fsm_time_99">>;
process_key(<<"NodePutFsmTimeMax">>) -> <<"node_put_fsm_time_100">>;
process_key(<<"NodePutFsmTimeMean">>) -> <<"node_put_fsm_time_mean">>;
process_key(<<"NodePutFsmTimeMedian">>) -> <<"node_put_fsm_time_median">>;
process_key(<<"ReadRepairs">>) -> <<"read_repairs">>;
process_key(<<"ReadRepairsTotal">>) -> <<"read_repairs_total">>.
```

code: 008

```
09:32:46.305 [info] Wait until 'dev1@127.0.0.1' is pingable
09:32:46.306 [info] Check 'dev1@127.0.0.1' is a singleton
09:32:46.351 [info] Deployed nodes: ['dev1@127.0.0.1']
09:32:46.351 [info] Waiting for services [riak_kv] to start on
['dev1@127.0.0.1'].
09:32:47.354 [info] Wait until nodes are ready : ['dev1@127.0.0.1']
09:33:27.738 [info] perform 5 x  PUT and a GET to increment the stats
09:33:27.738 [info] as the stat system only does calcs for > 5 readings
09:34:08.247 [info] node_gets: 0 -> 10 (expected 10)
09:34:08.247 [info] node_puts: 0 -> 5 (expected 5)
09:34:08.247 [info] node_gets_total: 0 -> 10 (expected 10)
09:34:08.248 [info] node_puts_total: 0 -> 5 (expected 5)
09:34:08.248 [info] vnode_gets: 0 -> 30 (expected 30)
09:34:08.248 [info] vnode_puts: 0 -> 15 (expected 15)
09:34:08.248 [info] vnode_gets_total: 0 -> 30 (expected 30)
09:34:08.248 [info] vnode_puts_total: 0 -> 15 (expected 15)
09:34:08.249 [info] Make PBC Connection
09:34:48.673 [info] pbc_connects_total: 0 -> 1 (expected 1)
09:34:48.673 [info] pbc_connects: 0 -> 1 (expected 1)
09:34:48.673 [info] pbc_active: 0 -> 1 (expected 1)
09:34:48.673 [info] Force Read Repair
09:35:29.144 [info] read_repairs_total: 0 -> 0 (expected 0)
09:35:29.145 [info] read_repairs: 0 -> 0 (expected 0)
09:36:09.521 [info] read_repairs_total: 0 -> 1 (expected 1)
09:36:09.521 [info] read_repairs: 0 -> 1 (expected 1)
09:36:09.522 [notice] jmx_verify Test Run Complete

Test Results:
jmx_verify-bitcask: pass
```

code: 009

Time passes...

as time does

Somebody (me) wants to add features

add maven to the build

it should still work

```
10:47:31.088 [info] Wait until 'dev1@127.0.0.1' is pingable
10:47:31.089 [info] Check 'dev1@127.0.0.1' is a singleton
10:47:31.129 [info] Deployed nodes: ['dev1@127.0.0.1']
10:47:31.129 [info] Waiting for services [riak_kv] to start on
['dev1@127.0.0.1'].
10:47:31.131 [info] Wait until nodes are ready : ['dev1@127.0.0.1']
10:48:11.749 [info] perform 5 x  PUT and a GET to increment the stats
10:48:11.749 [info] as the stat system only does calcs for > 5 readings
10:48:52.398 [info] node_gets: 0 -> 10 (expected 10)
10:48:52.398 [info] node_puts: 0 -> 5 (expected 5)
10:48:52.399 [info] node_gets_total: 0 -> 10 (expected 10)
10:48:52.399 [info] node_puts_total: 0 -> 5 (expected 5)
10:48:52.399 [info] vnode_gets: 0 -> 30 (expected 30)
10:48:52.399 [info] vnode_puts: 0 -> 15 (expected 15)
10:48:52.399 [info] vnode_gets_total: 0 -> 30 (expected 30)
10:48:52.400 [info] vnode_puts_total: 0 -> 15 (expected 15)
10:48:52.400 [info] Make PBC Connection
10:49:32.908 [info] pbc_connects_total: 0 -> 1 (expected 1)
10:49:32.909 [info] pbc_connects: 0 -> 1 (expected 1)
10:49:32.909 [info] pbc_active: 0 -> 1 (expected 1)
10:49:32.909 [info] Force Read Repair
10:50:13.448 [info] read_repairs_total: 0 -> 0 (expected 0)
10:50:13.449 [info] read_repairs: 0 -> 0 (expected 0)
10:50:53.829 [info] read_repairs_total: 0 -> 1 (expected 1)
10:50:53.829 [info] read_repairs: 0 -> 1 (expected 1)
10:50:53.829 [notice] jmx_verify Test Run Complete

Test Results:
jmx_verify-bitcask: pass
```

code: 010

A nasty rumor

We’re going to need to kill some processes

So what do those processes look like?

Add this!

```
?assert(fail),
```

Now run it!

```
➜ make clean && make
➜ ./riak_test -c ee -t jmx_verify
...
So, we find ourselves in a tricky situation
here.
You've run a single test, and it has failed.
Would you like to leave Riak running in order
to debug?
[Y/n] Y
```

code: 011

What’s it look like?

```
→ ps -ax
96410 ??         0:00.57 /usr/bin/java -server -
Dcom.sun.management.jmxremote.authenticate=false -
Dcom.sun.management.jmxremote.ssl=false -
Dcom.sun.management.jmxremote.port=41111
→ kill 96410
→ ps -ax
97142 ??         0:00.43 /usr/bin/java -server -
Dcom.sun.management.jmxremote.authenticate=false -
Dcom.sun.management.jmxremote.ssl=false -
Dcom.sun.management.jmxremote.port=41111
```

Let’s kill them all!

code: 012

```
kill_jmx() ->
    {0, JMXProcs} = rt:cmd(
      "ps -ax | grep com.sun.management.jmxremote"),
    Pids = filter_procs(string:tokens(JMXProcs, "\n")),
    [ rt:cmd("kill -9 " ++ Pid) || Pid <- Pids].

filter_procs([]) -> [];
filter_procs([Proc|T]) ->
    case string:str(Proc, "java -server") of
        0 -> filter_procs(T);
        _ -> [hd(string:tokens(Proc, "\s"))
                | filter_procs(T)]
    end.
```

code: 013

```
test_supervision() ->
    JMXPort = 41111,
    Config = [{riak_jmx, [{enabled, true}, {port, JMXPort}]}],
    [Node] = rt:deploy_nodes(1, Config),

    [ kill_jmx() || _X <- lists:seq(0, 9)],

    case net_adm:ping(Node) of
        pang ->
            ?
assertEqual("riak_jmx crash able to crash riak node", true);
        _ ->
            yay
    end,
    ok.
```

It’s too slow!

Let’s listen on port 80

code: 014

```
test_supervision() ->
    JMXPort = 80,
    Config = [{riak_jmx, [{enabled, true}, {port, JMXPort}]}],
    [Node] = rt:deploy_nodes(1, Config),
    timer:sleep(20000),
    case net_adm:ping(Node) of
        pang ->
            lager:error(
              "riak_jmx crash able to crash riak node"),
            ?assertEqual(
              "riak_jmx crash able to crash riak node", true);
        _ ->
            yay
    end,
ok.
```

code: 015

That triggers it!

```
============= jmx_verify failure stack trace ==================
{{assertEqual_failed,
     [{module,jmx_verify},
      {line,176},
      {expression,"true"},
      {expected,"riak_jmx crash able to crash riak node"},
      {value,true}]},
 [{jmx_verify,'-test_supervision/0-fun-0-',1,
              [{file,"tests/jmx_verify.erl"},{line,176}]},
  {jmx_verify,test_supervision,0,[{file,"tests/
jmx_verify.erl"},{line,176}]},
  {jmx_verify,confirm,0,[{file,"tests/jmx_verify.erl"},{line,
29}]}]}
================================================================
```

What should it do?

- Let the java process crash
- If it does, try 10 times like it used to
- If that fails, sleep for 10 minutes and try again

Let’s add to the test to check for that non-existent functionality

code: 016

```
%% Let's make sure the thing's restarting as planned
lager:info("calling riak_jmx:stop() to reset retry counters"),
rpc:call(Node, riak_jmx, stop, ["stopping for test purposes"]),

lager:info("loading lager backend on node"),
rt:load_modules_on_nodes([riak_test_lager_backend], [Node]),
ok = rpc:call(Node, gen_event, add_handler, [lager_event, riak_test_lager_b
ackend, [info, false]]),
ok = rpc:call(Node, lager, set_loglevel, [riak_test_lager_backend, info]),

lager:info("Now we're capturing logs on the node, let's start jmx"),
lager:info("calling riak_jmx:start() to get these retries started"),
rpc:call(Node, riak_jmx, start, []),

timer:sleep(40000), %% wait 2000 millis per restart + fudge factor
Logs = rpc:call(Node, riak_test_lager_backend, get_logs, []),

lager:info("It can fail, it can fail 10 times"),
RetryCount = lists:foldl(
    fun(Log, Sum) -
> case re:run(element(7,element(2,Log)), "JMX server monitor .* exited with
code .*\. Retrying", []) of
            {match, _} -> 1 + Sum;
            _ -> Sum
        end
    end,
    0, Logs),
?assertEqual({retry_count, 10}, {retry_count, RetryCount}),
rt:stop(Node),
ok_ok.
```

code: 017

```
============== jmx_verify failure stack trace ====================
{function_clause,
    [{lists,foldl,
         [#Fun<jmx_verify.3.111466063>,0,{badrpc,nodedown}],
         [{file,"lists.erl"},{line,1196}]},
     {jmx_verify,test_supervision,0,
         [{file,"tests/jmx_verify.erl"},{line,198}]},
     {jmx_verify,confirm,0,
         [{file,"tests/jmx_verify.erl"},{line,29}]}]}
=====================================================================
```

code: 018

```
rpc:call(Node, riak_jmx, stop, ["stopping for test purposes"]),
```

```
stop(Reason) ->
    error_logger:info_msg(io_lib:format("~p~n",[Reason])),
    init:stop().
```

code: 018

```
    rpc:call(Node, riak_jmx, stop, ["stopping for test purposes"]),
    stop(Reason) ->
        error_logger:info_msg(io_lib:format("~p~n",[Reason])),
        application:stop(riak_jmx).
```

code: 019

```
============ jmx_verify failure stack trace ====================
{{assertEqual_failed,
     [{module,jmx_verify},
      {line,205},
      {expression,"{ retry_count , RetryCount }"},
      {expected,{retry_count,10}},
      {value,{retry_count,15}}]},
 [{jmx_verify,'-test_supervision/0-fun-2-',2,
              [{file,"tests/jmx_verify.erl"},{line,205}]},
  {jmx_verify,test_supervision,0,[{file,"tests/jmx_verify.erl"},
{line,205}]},
  {jmx_verify,confirm,0,[{file,"tests/jmx_verify.erl"},{line,
29}]}]}
================================================================
```

code: 020,021

log the retry count

```
lager:info(
   "JMX server monitor ~s exited with code ~p. Retrying.",
                        [State#state.pid, Rc]),
```

code: 020,021

log the retry count

```
lager:info(
   "JMX server monitor ~s exited with code ~p. Retry #~p.",
                       [State#state.pid, Rc, Retry]),

          {expression,"{ retry_count , RetryCount }"},
                      {expected,{retry_count,10}},
                      {value,{retry_count,15}}]},
```

```
13:23:26.395 [info] JMX ... 40493 exited with code 1. Retry #0.
13:23:29.081 [info] JMX ... 40519 exited with code 1. Retry #0.
13:23:31.756 [info] JMX ... 40556 exited with code 1. Retry #0.
13:23:34.451 [info] JMX ... 40582 exited with code 1. Retry #0.
13:23:37.124 [info] JMX ... 40619 exited with code 1. Retry #0.
13:23:39.867 [info] JMX ... 40646 exited with code 1. Retry #0.
13:23:42.529 [info] JMX ... 40684 exited with code 1. Retry #0.
13:23:45.174 [info] JMX ... 40710 exited with code 1. Retry #0.
13:23:47.824 [info] JMX ... 40747 exited with code 1. Retry #0.
13:23:50.460 [info] JMX ... 40774 exited with code 1. Retry #0.
13:23:53.153 [info] JMX ... 40800 exited with code 1. Retry #0.
```

OOPS! Forgot to increment the retry counter

code: 022

log the retry count

```
lager:info(
   "JMX server monitor ~s exited with code ~p. Retry #~p.",
                        [State#state.pid, Rc, Retry]),
```

```
14:22:52.301 [info] "stopping for test purposes"
14:22:53.004 [info] JMX ... 47923 exited with code 1. Retry #0.
14:22:55.636 [info] JMX ... 47960 exited with code 1. Retry #1.
14:22:58.305 [info] JMX ... 47985 exited with code 1. Retry #2.
14:23:00.979 [info] JMX ... 48012 exited with code 1. Retry #3.
14:23:03.640 [info] JMX ... 48049 exited with code 1. Retry #4.
14:23:06.428 [info] JMX ... 48074 exited with code 1. Retry #5.
14:23:09.076 [info] JMX ... 48111 exited with code 1. Retry #6.
14:23:11.795 [info] JMX ... 48138 exited with code 1. Retry #7.
14:23:14.484 [info] JMX ... 48175 exited with code 1. Retry #8.
14:23:17.214 [info] JMX ... 48200 exited with code 1. Retry #9.
```

Cool!

Now, let’s bring parity

cpu_nprocs	CPUNProcs
mem_allocated	MemAllocated
mem_total	MemTotal
node_gets	NodeGets
node_gets_total	NodeGetsTotal
node_puts	NodePuts
node_puts_total	NodePutsTotal
vnode_gets	VnodeGets
vnode_gets_total	VnodeGetsTotal
vnode_puts	VnodePuts
vnode_puts_total	VnodePutsTotal
pbc_active	PbcActive
pbc_connects	PbcConnects
pbc_connects_total	PbcConnectsTotal
nodename	NodeName
ring_creation_size	RingCreationSize
cpu_avg1	CpuAvg1
cpu_avg5	CpuAvg5
cpu_avg15	CpuAvg15
node_get_fsm_time_95	NodeGetFsmTime95
node_get_fsm_time_99	NodeGetFsmTime99
node_get_fsm_time_100	NodeGetFsmTimeMax
node_get_fsm_time_mean	NodeGetFsmTimeMean
node_get_fsm_time_median	NodeGetFsmTimeMedian
node_put_fsm_time_95	NodePutFsmTime95
node_put_fsm_time_99	NodePutFsmTime99
node_put_fsm_time_100	NodePutFsmTimeMax
node_put_fsm_time_mean	NodePutFsmTimeMean
node_put_fsm_time_median	NodePutFsmTimeMedian
read_repairs	ReadRepairs
read_repairs_total	ReadRepairsTotal

```
CPUNProcs -> cpu_nprocs
```

```
process_key(<<"CPUNProcs">>) -> <<"cpu_nprocs">>;
process_key(<<"MemAllocated">>) -> <<"mem_allocated">>;
process_key(<<"MemTotal">>) -> <<"mem_total">>;
process_key(<<"NodeGets">>) -> <<"node_gets">>;
process_key(<<"NodeGetsTotal">>) -> <<"node_gets_total">>;
process_key(<<"NodePuts">>) -> <<"node_puts">>;
process_key(<<"NodePutsTotal">>) -> <<"node_puts_total">>;
process_key(<<"VnodeGets">>) -> <<"vnode_gets">>;
process_key(<<"VnodeGetsTotal">>) -> <<"vnode_gets_total">>;
process_key(<<"VnodePuts">>) -> <<"vnode_puts">>;
process_key(<<"VnodePutsTotal">>) -> <<"vnode_puts_total">>;
process_key(<<"PbcActive">>) -> <<"pbc_active">>;
process_key(<<"PbcConnects">>) -> <<"pbc_connects">>;
process_key(<<"PbcConnectsTotal">>) -> <<"pbc_connects_total">>;
process_key(<<"NodeName">>) -> <<"nodename">>;
process_key(<<"RingCreationSize">>) -> <<"ring_creation_size">>;
process_key(<<"CpuAvg1">>) -> <<"cpu_avg1">>;
process_key(<<"CpuAvg5">>) -> <<"cpu_avg5">>;
process_key(<<"CpuAvg15">>) -> <<"cpu_avg15">>;
process_key(<<"NodeGetFsmTime95">>) -> <<"node_get_fsm_time_95">>;
process_key(<<"NodeGetFsmTime99">>) -> <<"node_get_fsm_time_99">>;
process_key(<<"NodeGetFsmTimeMax">>) -> <<"node_get_fsm_time_100">>;
process_key(<<"NodeGetFsmTimeMean">>) -> <<"node_get_fsm_time_mean">>;
process_key(<<"NodeGetFsmTimeMedian">>) -> <<"node_get_fsm_time_median">>;
process_key(<<"NodePutFsmTime95">>) -> <<"node_put_fsm_time_95">>;
process_key(<<"NodePutFsmTime99">>) -> <<"node_put_fsm_time_99">>;
process_key(<<"NodePutFsmTimeMax">>) -> <<"node_put_fsm_time_100">>;
process_key(<<"NodePutFsmTimeMean">>) -> <<"node_put_fsm_time_mean">>;
process_key(<<"NodePutFsmTimeMedian">>) -> <<"node_put_fsm_time_median">>;
process_key(<<"ReadRepairs">>) -> <<"read_repairs">>;
process_key(<<"ReadRepairsTotal">>) -> <<"read_repairs_total">>.
```

code: 023

code: 024

```
============== jmx_verify failure stack trace ====================
{{assertNotEqual_failed,[{module,jmx_verify},
                         {line,115},
                         {expression,"{ Key , 0 }"},
                         {value,{<<"cpu_nprocs">>,0}}]},
 [{jmx_verify,'-verify_nz/2-fun-0-',2,
              [{file,"tests/jmx_verify.erl"},{line,115}]},
  {jmx_verify,'-verify_nz/2-lc$^0/1-0-',2,
              [{file,"tests/jmx_verify.erl"},{line,115}]},
  {jmx_verify,confirm,0,[{file,"tests/jmx_verify.erl"},{line,
44}]}]}
====================================================================
```

Handwavey Java

- class & interface
- javassist
- curl vs jinterface
- crashing
- permgen space

code: 025

```
11:02:52.775 [info] Wait until 'dev1@127.0.0.1' is pingable
11:02:52.775 [info] Wait until the ring manager is up on 'dev1@127.0.0.1'
11:02:52.776 [info] Check 'dev1@127.0.0.1' is a singleton
11:02:52.776 [info] Deployed nodes: ['dev1@127.0.0.1']
11:02:52.776 [info] Waiting for services [riak_kv] to start on
['dev1@127.0.0.1'].
11:03:01.805 [info] Wait until nodes are ready : ['dev1@127.0.0.1']
11:03:42.309 [info] perform 5 x  PUT and a GET to increment the stats
11:03:42.310 [info] as the stat system only does calcs for > 5 readings
11:04:22.968 [info] node_gets: 0 -> 10 (expected 10)
11:04:22.969 [info] node_puts: 0 -> 5 (expected 5)
11:04:22.969 [info] node_gets_total: 0 -> 10 (expected 10)
11:04:22.969 [info] node_puts_total: 0 -> 5 (expected 5)
11:04:22.969 [info] vnode_gets: 0 -> 30 (expected 30)
11:04:22.969 [info] vnode_puts: 0 -> 15 (expected 15)
11:04:22.970 [info] vnode_gets_total: 0 -> 30 (expected 30)
11:04:22.970 [info] vnode_puts_total: 0 -> 15 (expected 15)
11:04:22.970 [info] Make PBC Connection
11:05:03.469 [info] pbc_connects_total: 0 -> 1 (expected 1)
11:05:03.469 [info] pbc_connects: 0 -> 1 (expected 1)
11:05:03.470 [info] pbc_active: 0 -> 1 (expected 1)
11:05:03.470 [info] Force Read Repair
11:05:44.104 [info] read_repairs_total: 0 -> 0 (expected 0)
11:05:44.104 [info] read_repairs: 0 -> 0 (expected 0)
11:06:24.543 [info] read_repairs_total: 0 -> 1 (expected 1)
11:06:24.543 [info] read_repairs: 0 -> 1 (expected 1)
11:06:24.544 [notice] jmx_verify Test Run Complete

Test Results:
jmx_verify-bitcask: pass
```

We did wrong

DynamicMBean is the HashMap of JMX Beans

code: 026

```
09:42:05.367 [info] Wait until 'dev1@127.0.0.1' is pingable
09:42:05.368 [info] Wait until the ring manager is up on 'dev1@127.0.0.1'
09:42:05.369 [info] Check 'dev1@127.0.0.1' is a singleton
09:42:05.370 [info] Deployed nodes: ['dev1@127.0.0.1']
09:42:05.370 [info] Waiting for services [riak_kv] to start on
['dev1@127.0.0.1'].
09:42:14.393 [info] Wait until nodes are ready : ['dev1@127.0.0.1']
09:42:54.873 [info] perform 5 x  PUT and a GET to increment the stats
09:42:54.873 [info] as the stat system only does calcs for > 5 readings
09:43:35.560 [info] node_gets: 0 -> 10 (expected 10)
09:43:35.560 [info] node_puts: 0 -> 5 (expected 5)
09:43:35.560 [info] node_gets_total: 0 -> 10 (expected 10)
09:43:35.561 [info] node_puts_total: 0 -> 5 (expected 5)
09:43:35.561 [info] vnode_gets: 0 -> 30 (expected 30)
09:43:35.561 [info] vnode_puts: 0 -> 15 (expected 15)
09:43:35.562 [info] vnode_gets_total: 0 -> 30 (expected 30)
09:43:35.562 [info] vnode_puts_total: 0 -> 15 (expected 15)
09:43:35.562 [info] Make PBC Connection
09:44:16.102 [info] pbc_connects_total: 0 -> 1 (expected 1)
09:44:16.103 [info] pbc_connects: 0 -> 1 (expected 1)
09:44:16.103 [info] pbc_active: 0 -> 1 (expected 1)
09:44:16.103 [info] Force Read Repair
09:44:56.688 [info] read_repairs_total: 0 -> 0 (expected 0)
09:44:56.688 [info] read_repairs: 0 -> 0 (expected 0)
09:45:37.242 [info] read_repairs_total: 0 -> 1 (expected 1)
09:45:37.242 [info] read_repairs: 0 -> 1 (expected 1)
09:45:37.242 [notice] jmx_verify Test Run Complete

Test Results:
jmx_verify-bitcask: pass
```

Jump to Conclusions

???	JUMP AGAIN	STRIKE OUT
COULD BE	LOOSE ONE TURN	YES!
NO!	ACCEPT IT	GO WILD
ONE STEP BACK	THINK AGAIN	MOOT!

Lessons Learned

TDD is like going to the gym

Lessons Learned

riak_test is better at CI than TDD

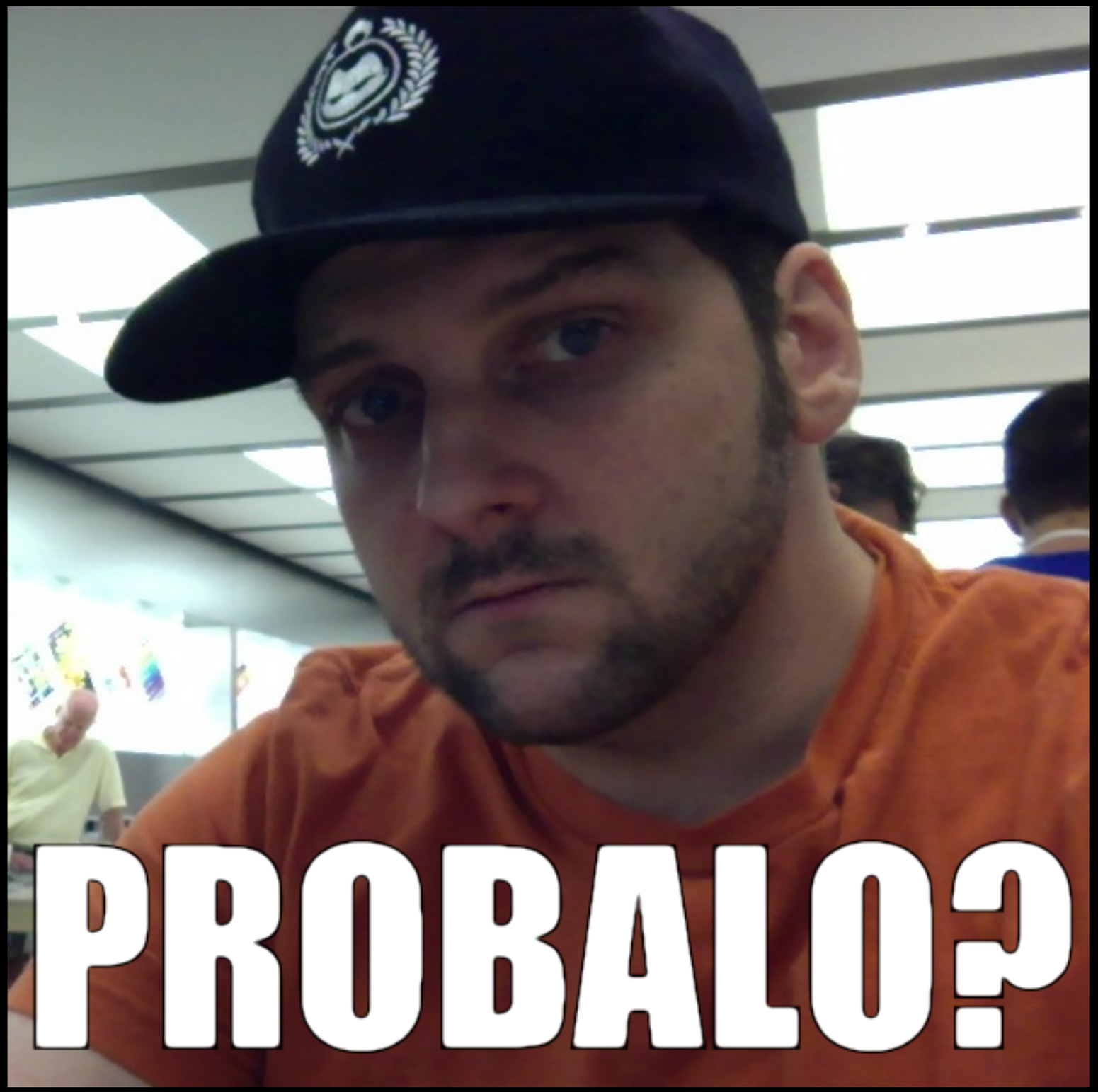
Lessons Learned

It would have been hard to test the restart stuff manually

find out more

http://joedevivo.github.io/euc2013

http://github.com/basho/riak_test


PROBALO?

riak_test's greatest hits!

basic_command_line

```
%% Stop node, to test console working
rt:console(Node, [
    {expect, "\(abort with ^G\)"},
    {send, "riak_core_ring_manager:get_my_ring()."},
    {expect, "dict,"},
    {send, "q()."},
    {expect, "ok"}]),
rt:wait_until_unpingable(Node),
ok.
```

counter converge

```
[N1, N2, N3, N4]=Nodes = rt:build_cluster(4),

PartInfo = rt:partition([N1, N2], [N3, N4]),

%% increment one side
increment_counter(HP1, Key, 5),

%% check value on one side is different from other
[?assertEqual(13, get_counter(HP, Key)) || HP <- [HP1, HP2]],
[?assertEqual(8, get_counter(HP, Key)) || HP <- [HP3, HP4]],

%% decrement other side
decrement_counter(HP3, Key, 2),

%% verify values differ
[?assertEqual(13, get_counter(HP, Key)) || HP <- [HP1, HP2]],
[?assertEqual(6, get_counter(HP, Key)) || HP <- [HP3, HP4]],

%% heal
lager:info("Heal and check merged values"),
ok = rt:heal(PartInfo),
ok = rt:wait_for_cluster_service(Nodes, riak_kv),

%% verify all nodes agree
[?assertEqual(ok, rt:wait_until(HP, fun(N) ->
                                        11 == get_counter(N, Key)
                                end)) ||  HP <- Hosts],

pass.
```

loaded upgrade

```
[begin
    exit(Sup, normal),
    lager:info("Upgrading ~p", [Node]),
    rt:upgrade(Node, current),
    {ok, NewSup} = rt_worker_sup:start_link([
        {concurrent, Concurrent},
        {node, Node},
        {backend, Backend},
        {version, current}
    ]),

    _NodeMon = init_node_monitor(Node, NewSup, self()),
    upgrade_recv_loop()

end || {{ok, Sup}, Node} <- Sups],

pass.
```

loaded upgrade