# Configuration Files

## Contents



---

Riak has two configuration files located in `/etc` if you are using a source install or in `/etc/riak` if you used a binary install. Those files are `app.config` and `vm.args`.

The `app.config` file is used to set various attributes for the Riak node, such as the storage backend that the node will use to store data, while the `vm.args` file is used to pass parameters to the Erlang node, such as the name or cookie of the node.

## Configuring Your `app.config`

Riak and the Erlang applications it depends on are configured by settings in the `app.config` file, which looks something like this:

Erlang

```
[
    {riak_core, [
        {ring_state_dir, "data/ring"}
        %% More riak_core settings...
    ]},
    {riak_kv, [
        {storage_backend, riak_kv_bitcask_backend},
        %% More riak_kv settings...
    ]},
    %% Other application configurations...
].
```

Below is a series of tables listing the configurable parameters in `app.config`.

## `riak_api` Settings

| Parameter | Description | Default |
|---|---|---|
| | | |

| | | |
|---|---|---|
| `pb_backlog` | The maximum length to which the queue of pending*simultaneous* Protocol Buffers connections may grow. If set, it must be an integer >= 0. If you anticipate a larger number of connections than the default being simultaneously initialized, set this number to a higher value accordingly. You should adjust this value to meet your anticipated simultaneous connection demand or if experiencing connection resets. | `5` |
| `pb` | A list of IP addresses and ports on which Riak's Protocol Buffers interface should listen. | `{"127.0.0.1",8087}` |
| `pb_ip` | The IP address to which the Protocol Buffers interface will bind. If not set, the PBC interface will not be started. The IP address may be specified as a string or tuple of address components as integers (4 for IPv4, 8 for IPv6).<br><br>Examples:<br>- Bind to a specific IPv4 interface: `{pb_ip, {10,1,1,56}}`<br>- Bind to all IPv6 interfaces: `{pb_ip, "::0"}`<br>- Bind to a specific IPv6 interface: `{pb_ip, {65152,0,0,0,64030,57343,65250,15801}}` | `127.0.0.1` |
| `pb_port` | The port to which the Protocol Buffers interface will bind. | `8087` |
| `disable_pb_nagle` | Turns off Nagle's algorithm (aka TCP slow start) for Protocol Buffer connections. This is equivalent to setting the `TCP_NODELAY` option on the socket. | `false` |

## `riak_core` Settings

| Parameter | Description | Default |
|---|---|---|
| `choose_claim_fun` | Designates a module/function pair—using a `{Module, Function}` syntax—to claim vnodes from the passed-in ring and then return the resulting ring.<br><br>**Note**: Please contact Basho support for more information. | `{riak_core_claim, default_choose_claim}`, which defaults to `{riak_core_claim, choose_claim_v2}` for most cases. |
| `cluster_name` | The name of the cluster (as a | `riak` |

| | | |
|---|---|---|
| | string). This is used internal to the ring, and can be used for identifying multiple clusters within a larger infrastructure. Any custom `cluster_name` values should be established before your cluster is started and should not be changed thereafter. | |
| `default_bucket_props` | See detailed discussion below in the **Default Bucket Properties** section below | |
| `delayed_start` | Sleep a specified number of milliseconds before starting `riak_core`. | `unset` |
| `disable_http_nagle` | When set to `true`, this option will disable the Nagle buffering algorithm for HTTP traffic. This is equivalent to setting the `TCP_NODELAY` option on the HTTP socket. If you experience consistent minimum latencies in multiples of 20 milliseconds, setting this option to `true` may reduce latency. | `false` |
| `gossip_interval` | How often nodes in the cluster will share information about their ring state, in milliseconds. | `60000` |
| `handoff_concurrency` | Number of vnodes per physical node that are allowed to perform handoff at once. | `2` |
| `handoff_port` | TCP port number for the handoff listener. | `8099` |
| `handoff_ip` | The IP address to which the handoff listener will bind. The IP address may be specified as a string or tuple of address components as integers (4 for IPv4, 8 for IPv6). See `pb_ip` above for examples. | `0.0.0.0` |
| `http` | A list of IP addresses and ports | `{"127.0.0.1", 8091}` |

| | | |
|---|---|---|
| | on which Riak's HTTP interface should listen (along the lines of `[{host1, port1}, {host2, port2}]`).*Note: Riak's HTTP interface will not start if this setting is not defined*. | |
| `http_logdir` | Override the default location of the access logs. See the `webmachine_logger` settings to enable access logs. | **none** |
| `https` | A list of IP addresses and ports on which Riak's HTTPS interface should listen (along the lines of `[{addr1, port1}, {addr2, port2}`) | not enabled |
| `legacy_vnode_routing` | Boolean for compatibility with older versions. | |
| `platform_data_dir` | Base directory for backend data storage. | `./data` |
| `ring_state_dir` | The directory on disk in which to store the ring state.<br><br>Riak's ring state is stored on-disk by each node, such that each node may be restarted at any time (purposely, or via automatic failover) and know what its place in the cluster was before it terminated, without needing immediate access to the rest of the cluster. | `/data/ring` |
| `ring_creation_size` | The number of partitions into which the hash space is divided.<br><br>By default, each Riak node will own `ring_creation_size` / (number of nodes in the cluster) partitions. It is generally a good idea to specify a `ring_creation_size` that is several times greater than the number of nodes in your cluster (e.g. specify 64 to 256 | `64` |

| | | |
|---|---|---|
| | partitions for a 4-node cluster). This gives you room to expand the number of nodes in the cluster without worrying about underuse due to owning too few partitions. This number should be a power of 2 (64, 128, 256…). | |
| `ssl` | You can override the default SSL key and certificate settings. | `etc/cert.pem`,`etc/key.pem` |
| `target_n_val` | The highest `n_val` that you generally intend to use. This affects how partitions are distributed amongst the cluster and how preflists are calculated, helping to ensure that data is never stored to the same physical node more than once. You will need to change this setting only in rare circumstances.<br><br>Assuming that `ring_creation_size` is a power of 2, the ideal value for this setting is both greater than or equal to the largest `n_val` of any bucket, and an even divisor of the number of partitions in your ring (i.e. `ring_creation_size`).<br><br>The default value is 4, and for this to be effective at preventing hot spots, your cluster size (the number of physical nodes) must be equal to or larger than `target_n_val`. | `4` |
| `vnode_management_timer` | Frequency (in milliseconds) at which Riak checks for primary partitions that need to be transferred. | `10000` |
| `wants_claim_fun` | A module/function pairing, in `{Module, Function}` format, that returns a Boolean expressing whether | `{riak_core_claim, default_wants_claim}`, which defaults to `{riak_core_claim,` |

| | | |
|---|---|---|
| | or not this node wants to claim more vnodes.<br><br>**Note**: Please contact Basho support for more information. | `wants_claim_v2}` for most cases. |
| `enable_health_checks` | Boolean expressing whether or not all health checks should be enabled. Set to `true` to enable all health checks. | **none** |
| `stat_cache_ttl` | The interval, in seconds, between stat cache population runs. All Riak stats are served from the stat cache. This setting controls how frequently that cache is refreshed. | `1` |

## Default Bucket Properties (`default_bucket_props`)

These properties are for buckets that have not been explicitly defined. Imagine a scenario, for example, in which you create two buckets with explicitly defined properties but you know that you'll end up using other buckets beyond the initial two. If you'd like to define the properties of *those* additional buckets, you would set properties using the `default_bucket_props` parameter, which is constructed as a list of Erlang tuples along the lines of `[{prop1,value},{prop2,value}]`. Here's an example:

app.config

```
{default_bucket_props, [
    {n_val,3},
    {allow_mult,false},
    {last_write_wins,false},
    {precommit, []},
    {postcommit, []},
    {chash_keyfun, {riak_core_util, chash_std_keyfun}},
    {linkfun, {modfun, riak_kv_wm_link_walker, mapreduce_linkfun}}
]}
```

The table below provides more information about each of the other parameters listed in the code sample above:

| Parameter | Description |
|---|---|
| `n_val` | The number of replicas stored. See Replication Properties and for more information. |
| `allow_mult` | Whether or not siblings are allowed. See Vector Clocks for a discussion of sibling resolution. |
| `precommit` | Global [[pre-commit hook |

| | |
|---|---|
| `postcommit` | Global [[post-commit hook |

In addition to the above, there are also a variety of read, write and delete quorum values that can be configured within the `default_bucket_properties` list. Valid options include numeric values—e.g. `{r, 2}`—as well as the following symbolic values:

- `all` — All *N* replicas must respond
- `quorum` — A majority of the replicas must respond, equivalent to (`n_val` / 2) + 1. Thus, an `n_val` of 5 would require a `quorum` of 3, an `n_val` of 6 a `quorum` of 4, an `n_val` of 7 also a `quorum` of 4, and so on.

| Parameter | Description | Default |
|---|---|---|
| `r` | Read quorum value. The number of Riak nodes that must return results for a `GET` request before it is considered successful. | `quorum` |
| `pr` | Primary read quorum. The number of primary, non-fallback nodes that must return results for a successful `GET` request. | `0` |
| `w` | Write quorom value. The number of Riak nodes that must *accept* a `PUT` request. | `quorum` |
| `dw` | Durable write quorum. The number of Riak nodes that have received an acknowledgment of the write from the storage backend. | `quorum` |
| `pw` | Primary write quorum. The number of primary, non-fallback nodes that must accept a `PUT` request. | `0` |
| `rw` | Delete quorum. | `quorum` |

## `riak_kv` Settings

| Parameter | Description | Default |
|---|---|---|
| `anti_entropy` | Enable the AAE subsystem and optional debug messages.<br><br>AAE with no debugging: `{anti_entropy, {on, []}}`<br>For AAE with debugging: `{anti_entropy, {on, [debug]}}`<br>No AAE: | **none** |

| | | |
|---|---|---|
| | `{anti_entropy, {off, []}}` | |
| `anti_entropy_build_limit` | Restrict how quickly AAE can build hash trees. Building the tree for a given partition requires a full scan over that partition's data. Once built, trees stay built until they are expired. The format is `{number-of-builds, per-timespan-in-milliseconds}`. Example:<br><br>`{anti_entropy_build_limit, {1, 3600000}},` | **none** |
| `anti_entropy_concurrency` | Limit how many AAE exchanges/builds can happen concurrently, e.g. `{anti_entropy_concurrency, 2}`. | **none** |
| `anti_entropy_data_dir` | The directory in which AAE hash trees are stored. | `./data/anti_entropy` |
| `anti_entropy_expire` | Determine how often hash trees are expired after being built. Periodically expiring a hash tree ensures that the on-disk hash tree data stays consistent with the actual K/V backend data. It also helps Riak to identify silent disk failures and bit rot. However, expiration is not needed for normal AAE operation and should be infrequent for performance reasons. The time is specified in milliseconds. | `604800000` |
| `anti_entropy_leveldb_opts` | The LevelDB options used by AAE to generate the LevelDB-backed on-disk hash trees. Example:<br><br>`{anti_entropy_leveldb_opts, [{write_buffer_size, 4194304}, {max_open_files, 20}]},` | **none** |
| `anti_entropy_tick` | The tick determines how often the AAE manager looks for work to do, e.g. building/expiring trees or triggering exchanges. Lowering this value will speedup the rate that all replicas are synced across the cluster. Increasing the value is not recommended. Example: | `15000` |
| `add_paths` | A list of paths to add to the Erlang code path. This setting is especially useful for allowing Riak to use external modules during MapReduce queries. | **none** |
| `delete_mode` | Specifies behavior for the window of time | `{delay, 3000}` |

| | | |
|---|---|---|
| | between Riak identifying an object for deletion and actual deletion of the object. There are three modes of operation: `delay` (in milliseconds), `immediate`, and `keep`. Setting `delete_mode` to `immediate` removes the tombstone for the object when the delete request is received. Setting `delete_mode` to `keep` disables tombstone removal altogether. | |
| `mapred_name` | The base of the path in the URL exposing MapReduce via HTTP. | `mapred` |
| `mapred_queue_dir` | The directory used to store a transient queue for pending map tasks. Only valid when `mapred_system` is set to `legacy` (discussed immediately below). | `data/mrqueue` |
| `mapred_system` | Indicates which version of the MapReduce system should be used. `pipe` means that `riak_pipe` will power MapReduce queries, while `legacy` means that `luke` will be used. | `pipe` |
| `map_js_vm_count` | The number of Javascript VMs started to handle map phases. | `8` |
| `reduce_js_vm_count` | The number of Javascript VMs started to handle reduce phases. | `6` |
| `hook_js_vm_count` | The number of Javascript VMs started to handle pre-commit hooks. | `2` |
| `mapper_batch_size` | Number of items the mapper will fetch in one request. Larger values can impact read/write performance for non-MapReduce requests. Only valid when `mapred_system` is set to `legacy`. | `5` |
| `js_max_vm_mem` | The maximum amount of memory (in megabytes) allocated to each Javascript virtual machine. | `8` |
| `js_thread_stack` | The maximum amount of thread stack space (in megabytes) to allocate to Javascript virtual machines. | `16` |
| `map_cache_size` | Number of objects held in the MapReduce cache. These will be ejected when the cache runs out of room or the bucket/key pair for that entry. Only valid | `10000` |

| | | |
|---|---|---|
| | when `mapred_system` is set to `legacy`. | |
| `js_source_dir` | Where to load user-defined built-in Javascript functions | `unset` |
| `http_url_encoding` | Determines how Riak treats URL-encoded buckets, keys, and links over the REST API. When set to `on`, Riak always decodes encoded values sent as URLs and headers. Otherwise, Riak defaults to compatibility mode, in which links are decoded but buckets and keys are not. The compatibility mode will be removed in a future release. | `off` |
| `vnode_vclocks` | When set to `true`, Riak uses vnode-based vclocks rather than client ids. This significantly reduces the number of vclock entries. Only set to `true` if all nodes in the cluster are upgraded to 1.0. | `false` |
| `legacy_keylisting` | This option enables compatibility of bucket and key listing with 0.14 and earlier versions. Once a rolling upgrade to a version >= 1.0 is completed for a cluster, this should be set to `false` for improved performance for bucket and key listing operations. | `true` |
| `raw_name` | The base of the path in the URL exposing Riak's HTTP interface. The default (`riak`) will expose data at `/riak/Bucket/Key`. Thus, changing this setting to `bar` would expose the interface at `/bar/Bucket/Key`. | `riak` |
| `riak_kv_stat` | Enables the statistics-aggregator—`/stats` URL and `riak-admin status` command—if set to `true`. | `true` |
| `stats_urlpath` | The base of the path in the URL exposing the statistics-aggregator. | `stats` |
| `storage_backend` | The module name of the storage backend that Riak should use. For more on data backends, see the **Riak Backends** section below. | `riak_kv_bitcask_backend` |
| `riak_search` | Riak Search is now enabled via the `app.config`. To enable it in your app, simply set it to `true` in Riak Search configs section (more on this in **Riak Search Settings** below). | **none** |
| `vnode_mr_timeout` | How long (in milliseconds) a map function | `1000` |

| | | |
|---|---|---|
| | is permitted to execute on a vnode before it times out and is retried on another vnode. | |
| `vnode_mailbox_limit` | Configures the `riak_kv` health check that monitors message queue lengths of `riak_kv` vnodes, in `{EnableThreshold, DisableThreshold}` format. If a K/V vnode's message queue length reaches `DisableThreshold`, the `riak_kv` service is disabled on the node. The service will not be re-enabled until the message queue length drops below `EnableThreshold`. | **none** |
| `secondary_index_timeout` | The number of milliseconds before a secondary index query times out. A value of `0` indicates that no timeout will occur. | `0` |

## Riak Storage Backends

The storage backend that Riak should use is set using the `storage_backend` property (listed in the table above). Riak will refuse to start if no storage backend is specified. Here are the available backends:

| Backend | Description |
|---|---|
| `riak_kv_bitcask_backend` | Data is stored in Bitcask append-only storage. See the Bitcask configuration page for more information. |
| `riak_kv_eleveldb_backend` | Data is stored in LevelDB. See the LevelDB configuration page for more information. |
| `riak_kv_memory_backend` | A backend that behaves as an LRU-with-timed-expiry cache. Read the Memory backend configuration page for more information. |
| `riak_kv_multi_backend` | Enables storing data for different buckets in different backends. See the Multi backend configuration page for more details. |

## `webmachine_logger_module`

This needs to be set in order to enable access logs.

| Parameter | Description | Default |
| --- | --- | --- |
| `webmachine_logger_module` | This needs to be set in order to enable access logs.<br><br>**Note**: The additional disk I/O of an access log imposes a performance cost you may not wish to pay. Therefore, by default, Riak does not produce access logs. | **none** |

Here is an example:

app.config

```
{webmachine, [{webmachine_logger_module, webmachine_logger}]}
```

## `riak_search` Settings

| Parameter | Description | Default |
| --- | --- | --- |
| `enabled` | Enable search functionality. | `false` |
| `max_search_results` | Maximum number of results to accumulate before erroring, typically used to prevent or reduce memory exhaustion (which can sometimes reach levels that can bring down an entire VM). | `100000` |

Here is an example `riak_search` configuration:

app.config

```
%% Riak Search Config
{riak_search, [
    %% To enable Search functionality set this 'true'.
    {enabled, false}
    ]},
```

- **handoff_concurrency** Number of vnodes, per physical node, allowed to perform handoff at once. (default: `2`)

## lager

Lager is the logging engine introduced in Riak 1.0. It is designed to be more stable than Erlang's `error_logger` as well as to play nicely with standard logging tools.

| Parameter | Description | Default |
|---|---|---|
| `async_threshold` | The maximum number of log messages to be queued in asynchronous mode before switching to synchronous mode. | `20` |
| `async_threshold_window` | See detailed information in the**Async Threshold Window Settings** section below. | |
| `colored` | Enable color-coding messages logged to the console by level. Requires Erlang >= R16. | `false` |
| `colors` | Configure the colors to use for console messages, using ANSI escape sequences. The default colors are listed below in the**Default Colors** section. | **listed below** |
| `crash_log` | Whether or not to write a crash log and where. | **no crash logger** |
| `crash_log_count` | Number of rotated crash logs to keep. `0` means keep only the current one. | `0` |
| `crash_log_date` | What time to rotate the crash log. Formatting for crash logs is described in detail in the**Formatting Crash Logs** section below. | **no time-based rotation** |
| `crash_log_msg_size` | aximum size (in bytes) of events in the crash log. | `65536` |
| `crash_log_size` | Maximum size of the crash log (in bytes) before it is rotated. Set to `0` to disable rotation. | `0` |
| `error_logger_hwm` | Maximum number of messages per second allowed from `error_logger`. Permits weathering a flood of messages when many related processes crash. | **none** |
| `error_logger_redirect` | Whether to redirect `error_logger` messages | `true` |

| | | |
|---|---|---|
| | into lager. | |
| `handlers` | Allows the selection of log handlers with differing options.<br>- `lager_console_backend` logs to the the console, with the specified log level<br>- `lager_file_backend` logs to the given list of files, each with their own log level<br>Lager can rotate its own logs or have it done via an external process. To use internal rotation, use the `size`, `date`, and `count` values in the file backend's config.<br>See `crash_log_date` above for a description of the date field.<br>Below is an example:<br>`[{name, "error.log"}, {level, error}, {size, 10485760}, {date, "$D0"}, {count, 5}]` | |
| `error_logger_redirect` | Whether or not to redirect SASL `error_logger` messages into lager. | `true` |
| `traces` | Traces can be configured at startup by adding handlers to the lager configs, formatted as `{traces, [{handler1,filter1,level1}, {...}]}`. Refer to Lager Tracing for more information. | |

Below are the default lager options:

app.config

```
{lager, [
    {handlers, [
        {lager_console_backend, info},
        {lager_file_backend, [
            {"/opt/riak/log/error.log", error},
            {"/opt/riak/log/console.log", info}
        ]},
     ]},
    {crash_log, "/crash.log"},
    {crash_log_msg_size, 65536},
    {crash_log_size, 0},
```

```
    {crash_log_count, 0},
    {error_logger_redirect, true},
    {error_logger_hwm, 50},
    {async_threshold, 20},
    {async_threshold_window, 5}
    {colored, false}
]},
```

## Async Threshold Window Settings

The `async_threshold_window` setting determinse how far below the `async_threshold` the log message queue must sink before re-enabling asynchronous mode. The default value is `5`.

Prior to lager 2.0, the `gen_event` at the core of lager operated solely in synchronous mode. Asynchronous mode is faster but has no protection against message queue overload. In lager 2.0, the `gen_event` takes a hybrid approach. It polls its own mailbox size and toggles the messaging between synchronous and asynchronous depending on mailbox size. Below is an example configuration:

app.config

```
{async_threshold, 20},
{async_threshold_window, 5}
```

This will use async messaging until the mailbox exceeds 20 messages, at which point synchronous messaging will be used, and then switch back to asynchronous when size is reduced to 20 - 5 = 15. If you wish to disable this behaviour, simply set `async_threshold` to `undefined`. It defaults to a low number to prevent the mailbox growing rapidly beyond the limit and causing problems. In general, lager should process messages as quickly as they come in, so getting 20 behind should be relatively exceptional anyway.

## Default Colors

Below are the default colors for console messages (using ANSI escape sequences):

app.config

```
{colors, [
    {debug,     "\e[0;38m" },
    {info,      "\e[1;37m" },
    {notice,    "\e[1;36m" },
    {warning,   "\e[1;33m" },
    {error,     "\e[1;31m" },
    {critical,  "\e[1;35m" },
    {alert,     "\e[1;44m" },
    {emergency, "\e[1;41m" }
]}
```

## Formatting Crash Logs

The `crash_log_date` setting determines at what time the crash log will be rotated. The default is to have no time-based rotation. The syntax for the value field

is taken from the `when` section of newsyslog.conf.

The lead-in character for the day, week, and month specification is a `$`. The format is as follows: `[Dhh]` for day format, `[Ww[Dhh]]` for week format, and `[Mdd[Dhh]]` for month format. Optional time fields default to midnight. The ranges for day and hour specifications are:

- `hh` — hours, range 0 … 23
- `w` — day of week, range 0..6, 0 = Sunday
- `d` — day of the month, range 1… 31, or `L` or `l` to specify the last day of the month.

Some examples:

- `$D0` — rotate every night at midnight
- `$D23` — rotate every day at 23:00 hr
- `$W0D23` — rotate every week on Sunday at 23:00 hr
- `$W5D16` — rotate every week on Friday at 16:00 hr
- `$M1D0` — rotate on the first day of every month at midnight (i.e. at the start of the day)
- `$M5D6` — rotate on every 5th day of the month at 6:00 hr

# Configuring Your `vm.args`

Parameters for the Erlang node on which Riak runs are set in the `vm.args` file in the `/etc` directory (or `/etc/riak` with a binary install) of the embedded Erlang node. Most of these settings can be left at their defaults until you are ready to tune performance.

Two settings in particular will be of immediate interest to most users: `-name` and `-setcookie`. These control, respectively, Erlang node names (possibly host specific) and Erlang inter-node communication access (cluster specific).

The format of the `vm.args` file is fairly loose. Lines that do not begin with `#` are concatenated and passed to the `erl` command as is.

More details about each of these settings can be found in the Erlang documentation for the `erl` Erlang virtual machine.

Riak CS and Enterprise may make different choices for some of these configurations, so we advise relying on the `vm.args` file supplied with those packages.

| Parameter | Description | Default |
|---|---|---|
| `-name` | The name of the Erlang node. The default value ( `riak@127.0.0.2` ) will work for running Riak locally, but for distributed —i.e. multi-node—use, | `riak@127.0.0.1` |

| | | |
|---|---|---|
| | the portion of the name after the `@` should be changed to the IP address of the machine on which the node is running. If you have a properly configured DNS, the short form of this name can be used, e.g. `riak`. The name of the node will then be `riak@Host.Domain`. | |
| `-setcookie` | Cookie for the Erlang node. Erlang nodes grant or deny access based on the sharing of a previously shared cookie. You should use the same cookie for every node in your Riak cluster, but it should be a not-easily-guessed string unique to your deployment, for the sake of preventing non-authorized access. | `riak` |
| `-heart` | Enable `heart` node monitoring.<br><br>Heart will restart nodes automatically should they crash. However, `heart` is so good at restarting nodes that it can be difficult to prevent it from doing so.<br>Enable `heart` once you are sure that you wish to have the node restarted automatically on failure. | `disabled` |
| `+K` | Enable kernel polling. | `true` |
| `+A` | Number of threads in the async thread pool. | `64` |
| `-pa` | Adds the specified directories to the beginning of the code path, similar to `code:add_pathsa/1`. | |

| | | |
|---|---|---|
| | As an alternative to `-pa`, if several directories are to be prepended to the code and the directories have a common parent directory, that parent directory could be specified in the `ERL_LIBS` environment variable. | |
| `-env` | Set host environment variables for Erlang. | |
| `-smp` | Enables Erlang's SMP support. | `enable` |
| `+zdbbl` | Configures the buffer size for outbound messages between nodes. This is commented out by default because the ideal value varies significantly depending on available system memory, typical object size, and amount of traffic to the database.<br><br>Systems with lots of memory and under a heavy traffic load should consider increasing our default value; systems under lighter load but storing large objects may wish to lower it.Basho Bench is highly recommended to help determine the best values for this (and other tuning parameters) in your environment. | `1024` unless configured in `vm.args` — `32768` is the commented-out value |
| `+P` | Defines the Erlang process limit. Under the versions of Erlang supported by Riak through 1.4.x, the limit is very low, and thus using this to raise the limit is very important. | `256000` |

| | | |
|---|---|---|
| | **Note**: For anyone concerned about configuring such a high value, be aware that Erlang processes are *not* the same as system processes. All of these processes will exist solely inside a single system process, the Erlang BEAM. | |
| `+sfwi` | If using an appropriately patched Erlang VM (such as one downloaded directly from Basho), this will control the interval (in milliseconds) at which a supervisor thread wakes to check run queues for work to be executed. | `500` |
| `+W` | Determines whether or not warning messages sent to Erlang's `error_logger` are treated as errors, warnings, or informational. | `w` for warnings |
| `-env ERL_LIBS` | Alternative method for adding directories to the code path (see `-pa` above) | |
| `-env ERL_MAX_PORTS` | Maximum number of concurrent ports/sockets.<br><br>**Note**: As with processes, Erlang ports and system ports are similar but distinct. | `64000` |
| `-env ERL_FULLSWEEP_AFTER` | Run garbage collection more often. | `0` |
| `-env ERL_CRASH_DUMP` | Set the location of crash dumps. | `./log/erl_crash.dump` |
| `anti_entropy_expire` | Determine how often hash trees are expired after being built. Periodically expiring a hash tree ensures the on-disk hash | `604800000` |

| | | |
|---|---|---|
| | tree data stays consistent with the actual KV backend data. It also helps Riak identify silent disk failures and bit rot. However, expiration is not needed for normal AAE operation and should be infrequent for performance reasons. The time is specified in milliseconds. | |

---

If you are going to be rebuilding Riak often, you will want to edit the `vm.args` and `app.config` files in the `rel/files` directory. These files are used whenever a new release is generated using `make rel` or `rebar generate`. Each time a release is generated, any existing release must first be destroyed. Changes made to release files (`rel/riak/etc/vm.args`, `rel/riak/etc/app.config`, etc.) would be lost when the release is destroyed.


## These May Also Interest You

- Downloads
- Log Messages FAQs
- Operating Riak FAQs
- Load Balancing and Proxy Configuration
- Configuring Secondary Indexes
- Basic Cluster Setup