The Riak Docs

Basho Technologies

1.4.9 2014-06-23

Body typeface is Crimson.

Contents

What is Riak?

Riak is a distributed database designed to deliver maximum data availability by distributing data across multiple servers. As long as your client can reach *one* Riak server, it should be able to write data. In most failure scenarios, the data you want to read should be available, although it may not be the most up-to-date version of that data.

This fundamental tradeoff---high availability in exchange for possibly outdated information---informs the key architectural decisions behind Riak. This idea of "eventual consistency" is a common one in distributed systems, with DNS and web caches as two notable examples.

Basho's goals for Riak

Goal	Description
Availability	Riak writes to and reads from multiple servers to offer data availability even when hardwa
Operational simplicity	Easily add new machines to your Riak cluster without incurring a larger operational burde
Scalability	Riak automatically distributes data around the cluster and yields a near-linear performanc
Masterless	Your requests are not held hostage to a specific server in the cluster that may or may not l

When Riak makes sense

If your data does not fit on a single server and demands a distributed database architecture, then you should absolutely take a close look at Riak as a potential solution to your data availability issues. Getting distributed databases right is **very** difficult, and Riak was built to address data availability issues with as few trade-offs and downsides as possible.

In essence, Riak's focus on availability makes it a good fit whenever downtime is unacceptable. No one can promise 100% uptime, but Riak is designed to survive network partitions and hardware failures that would significantly disrupt most databases.

A less-heralded feature of Riak is its predictable latency. Because its fundamental operations---read, write, and delete---do not involve complex data joins or locks, it services those requests promptly. Thanks to this capability Riak is often selected as a data storage backend for other data management software from a variety of paradigms.

When Riak is Less of a Good Fit

Basho recommends that you run no fewer than 5 data servers in a cluster. This means that Riak can be overkill for small databases. If you're not already sure that you will need a distributed database, there's a good chance that you won't need Riak.

Nonetheless, if explosive growth is a possibility, you are always highly advised to prepare for that in advance. Scaling at Internet speeds is sometimes compared to overhauling an airplane mid-flight. If you feel that such a transition might be necessary in the future, then you might want to consider Riak.

Riak's simple data model, consisting of keys and values as its atomic elements, means that your data must be denormalized if your system is to be reasonably performant. For most applications this is not a serious hurdle. But if your data simply cannot be effectively managed as keys and values Riak will most likely not be the best fit for you.

On a related note: while Riak offers ways to find values that match certain criteria, if your application demands a high query load by any means other than the keys---e.g. SQL-style `SELECT * FROM table` operations---Riak will not be as efficient as other databases. If you wish to compare Riak with other data technologies, Basho offers a tool called **basho_bench** to help measure its performance so that you can decide whether the availability and operational benefits of Riak outweigh its disadvantages.

How Does a Riak Cluster Work?

What is a Riak Node?

A Riak node is not quite the same as a server, but in a production environment the two should be equivalent. A developer may run multiple nodes on a single laptop, but this would never be advisable in a real production cluster.

Each node in a Riak cluster is equivalent, containing a complete, independent copy of the whole Riak package. There is no "master." No node has more responsibilities than others, and no node has special tasks not performed by other nodes. This uniformity provides the basis for Riak's fault tolerance and scalability.

Each node is responsible for multiple data partitions as discussed below.

Riak Automatically Re-Distributes Data When Capacity is Added

When you add (or remove) machines, data is rebalanced automatically with no downtime. New machines claim data until ownership is equally spread around the cluster, with the resulting cluster status updates shared to every node via a gossip protocol and used to route requests. This is what makes it possible for any node in the cluster to receive requests. The end result is that developers don't need to deal with the underlying complexity of where data lives.

Consistent Hashing

Data is distributed across nodes using consistent hashing. Consistent hashing ensures that data is evenly distributed around the cluster and makes possible the automatic redistribution of data

as the cluster scales.

How does consistent hashing work? Riak stores data using a simple key/value scheme. These keys are associated with a namespace called a **bucket**. When you perform key/value operations in Riak, the bucket and key combination is hashed. The resulting hash maps onto a 160-bit integer space. You can think of this integer space as a ring used to determine what data to put on which physical machines.

How is this determination made? Riak divides the integer space into equally-sized partitions. Each partition owns a range of values on the ring, and is responsible for all buckets and keys that, when hashed, fall into that range. Each partition is managed by a process called a virtual node (or **vnode***). Physical machines evenly divide responsibility for vnodes. Let's say that you have a 4-node cluster with 32 partitions managed by 32 vnode processes. Each of the four physical machines claim eight vnodes (as illustrated below). Each physical machine thus becomes responsible for all keys represented by its eight vnodes.

Intelligent Replication

Riak's replication scheme ensures that you can still read, write, and update data if nodes go down. Riak allows you to set a replication variable, *n*. An *n* value of 3 (the default) means that each object is replicated 3 times. When an object's key is mapped onto a given partition, Riak won't stop there: it will automatically replicate the data onto the next two partitions as well.

When Things Go Wrong

Riak retains fault tolerance, data integrity, and availability even in failure conditions such as hardware failure and network partitions. Riak has a number of means of addressing these scenarios and other bumps in the road, like version conflicts in data.

Hinted Handoff

Hinted handoff enables Riak to handle node failure. If a node goes down, a neighboring node will take over its storage operations. When the failed node returns, the updates received by the neighboring node are handed back to it. This ensures availability for writes and updates and happens automatically, minimizing the operational burden of failure conditions.

Version Conflicts

In any system that replicates data, conflicts can arise, for example when two clients update the same object at the exact same time or when not all updates have yet reached hardware that is experiencing lag. Furthermore, in Riak, replicas are "eventually consistent," meaning that while data is always available, not all replicas may have the most recent update at the exact same time,

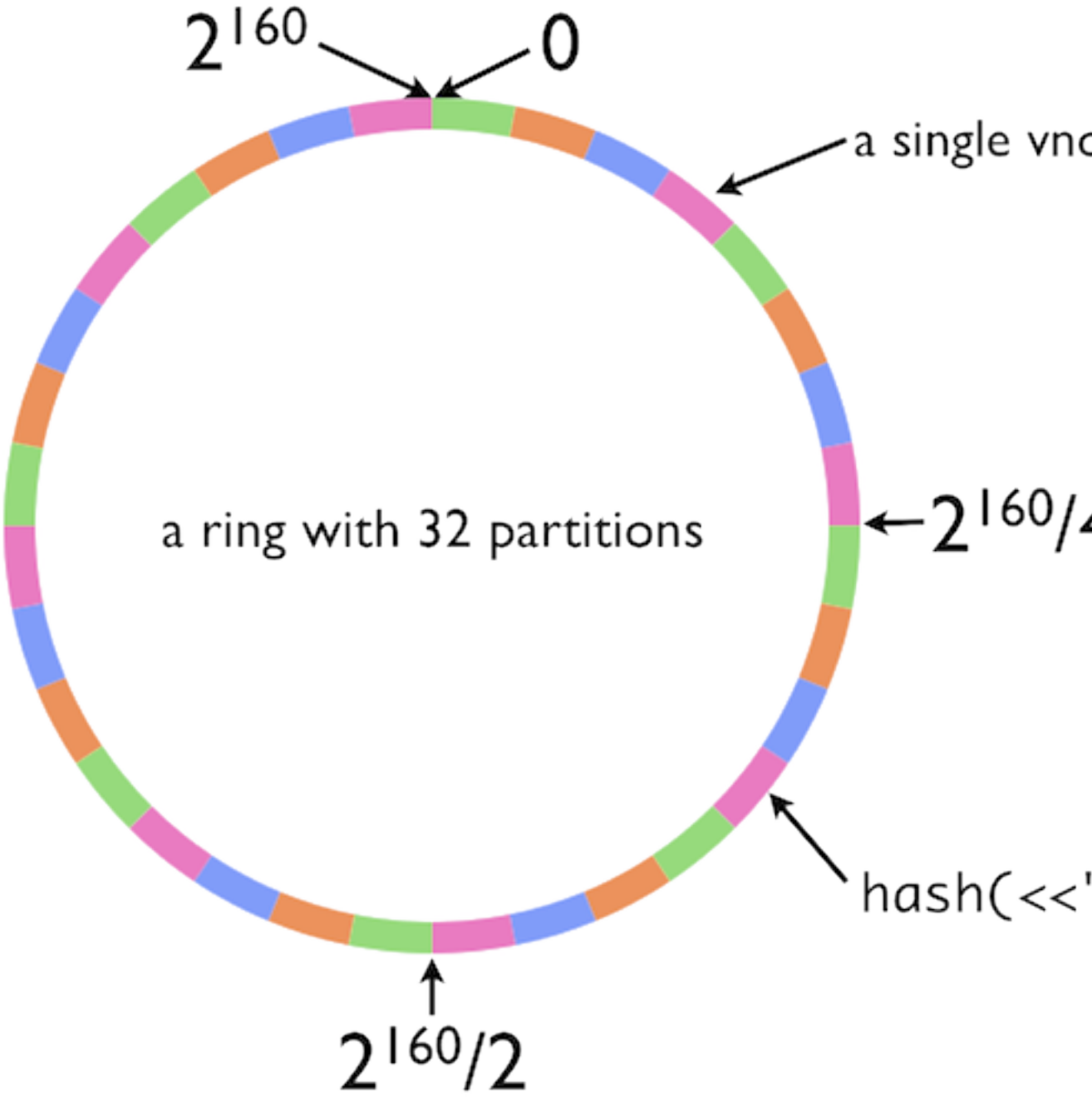

Figure 1: A Riak Ring

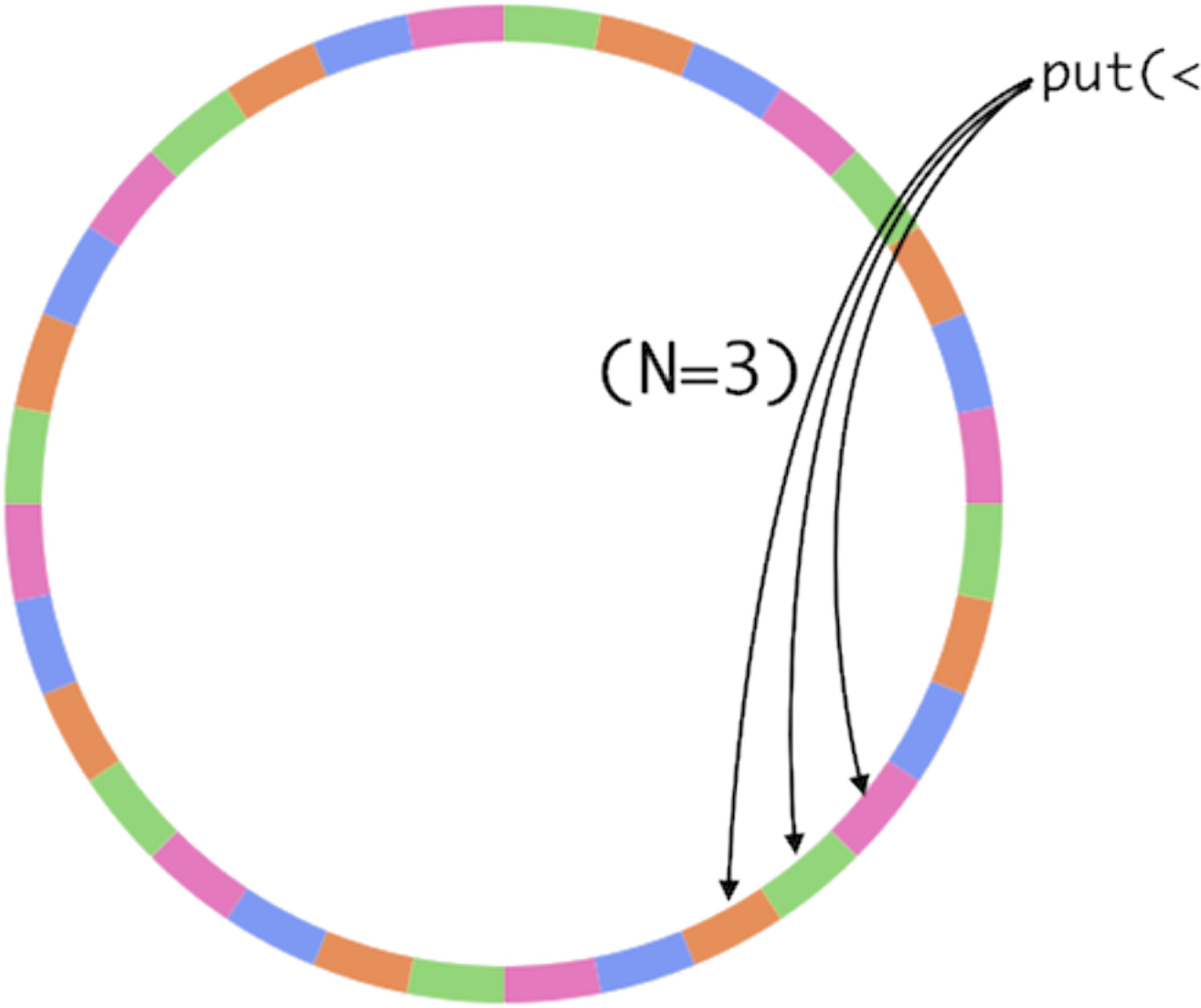

Figure 2: A Riak Ring

causing brief periods---generally on the order of milliseconds---of inconsistency while all state changes are synchronized.

How is this divergence addressed? When you make a read request, Riak looks up all replicas for that object. By default, Riak will return the most recently updated version, determined by looking at the object's vector clock. Vector clocks are metadata attached to each replica when it is created. They are extended each time a replica is updated to keep track of versions. You can also allow clients to resolve conflicts themselves if that is a better fit for your use case.

Read Repair

When an outdated replica is returned as part of a read request, Riak will automatically update the out-of-sync replica to make it consistent. Read repair, a self-healing property of the database, will even update a replica that returns a `not_found` in the event that a node loses the data due to physical failure.

Reading and Writing Data in Failure Conditions

In Riak, you can set an *r* value for reads and a *w* value for writes. These values give you control over how many replicas must respond to a request for it to succeed. Let's say that you have an *n* value of 3, but one of the physical nodes responsible for a replica is down. With an r=2 setting, only 2 replicas must return results for read to be deemed successful. This allows Riak to provide read availability even when nodes are down or laggy. The same applies for the *w* in writes. If this value is not specified, Riak defaults to **quorum**, according to which the majority of nodes must respond. There is more on [[Replication Properties]] elsewhere in the documentation.

This is intended to be a brief, objective and technical comparison of Riak and Cassandra. The Cassandra version described is 1.2.x. The Riak version described is Riak 1.2.x. If you feel this comparison is unfaithful at all for whatever reason, please fix it or send an email to **docs@basho.com**.

At A Very High Level

- Both Riak and Cassandra are Apache 2.0 licensed databases based on Amazon's Dynamo paper.
- Riak is a faithful implementation of Dynamo, with the addition of functionality like links, MapReduce, indexes, full-text Search. Cassandra departs from the Dynamo paper slightly by omitting vector clocks and moving from partition-based consistent hashing to key ranges, while adding functionality like order-preserving partitioners and range queries.
- Riak is written primarily in Erlang with some bits in C. Cassandra is written in Java.

Feature/Capability Comparison

The table below gives a high level comparison of Riak and Cassandra features/capabilities. To keep this page relevant in the face of rapid development on both sides, low level details are found in links to Riak and Cassandra online documentation.

Feature/Capability

Riak

Cassandra

Data Model

Riak stores key/value pairs in a higher level namespace called a bucket.

[[Buckets, Keys, and Values|Concepts#Buckets-Keys-and-Values]]

Cassandra's data model resembles column storage, and consists of Keyspaces, Column Families, and several other parameters.

[[Cassandra Data Model|http://www.datastax.com/docs/0.7/data_model/index]]

Storage Model

Riak has a modular, extensible local storage system which lets you plug-in a backend store of your choice to suit your use case. The default backend is Bitcask.

[[Riak Supported Storage Backends|Choosing a Backend]]

```
  You can also write you own storage backend for Riak using our [[backend API|Backend API]].
</td>
  <td> Cassandra's write path starts with a write to a commit log followed by a subsequent write to an
       <ul>
      <li><a href="http://wiki.apache.org/cassandra/ArchitectureCommitLog">Commit Log</a></li>
    <li><a href="http://wiki.apache.org/cassandra/MemtableSSTable">Memtable</a></li>
    <li><a href="http://wiki.apache.org/cassandra/ArchitectureSSTable">SSTable Overview</a></li>
    <li><a href="http://www.datastax.com/docs/1.1/dml/about_writes">About Writes</a></li>
    <li><a href="http://www.datastax.com/docs/1.1/dml/about_reads">About Reads</a></li>
       </ul>
   </td>
</tr>
<tr>
    <td>Data Access and APIs</td>
    <td>Riak offers two primary interfaces (in addition to raw Erlang access):
<ul>
  <li>[[HTTP|HTTP API]]</li>
<li>[[Protocol Buffers|PBC API]]</li>
```

```
</ul>
Riak Client libraries are wrappers around these APIs, and client support exists for dozens of languages
<ul>
  <li>[[Client Libraries]]</li>
  <li>[[Community Projects]] </li>
</ul>
</td>
   <td>Cassandra provides various access methods including a Thrift API, CQL (Cassandra Query Language
<ul>
 <li><a href="http://www.datastax.com/docs/1.1/dml/about_clients">Cassandra Client APIs</a></li>
</ul>
 </td>
</tr>
<tr>
    <td>Query Types and Query-ability</td>
    <td>There are currently four ways to query data in Riak
        <ul>
        <li>Primary key operations (GET, PUT, DELETE, UPDATE)</li>
        <li>[[MapReduce|Using MapReduce]]</li>
        <li>[[Using Secondary Indexes]]</li>
        <li>[[Using Search]]</li>
        </ul>

</td>
    <td>Cassandra offers various ways to query data:
            <ul>
        <li><a href="http://www.datastax.com/docs/0.7/data_model/keyspaces">Keyspaces</a></li>
        <li><a href="http://www.datastax.com/docs/0.7/data_model/cfs_as_indexes">Column Family Oper
        <li><a href="http://www.datastax.com/docs/1.0/dml/using_cql">CQL</a></li>
        <li><a href="http://www.datastax.com/docs/0.7/data_model/secondary_indexes">Secondary Index
        <li><a href="http://wiki.apache.org/cassandra/HadoopSupport#ClusterConfig">Hadoop Support</
        <ul>

</td>
</tr>
<tr>
    <td>Data Versioning and Consistency</td>
   <td> Riak uses a data structure called a vector clock to reason about causality and staleness of stor
        <ul>
          <li>[[Vector Clocks]]</li>
       <li>[[Why Vector Clocks Are Easy|http://basho.com/blog/technical/2010/01/29/why-vector-clocks
```

```
      <li>[[Why Vector Clocks Are Hard|http://basho.com/blog/technical/2010/04/05/why-vector-clocks
       </ul>
     </td>

  <td>Cassandra uses timestamps at the column family level to determine the most-recent value when doi
       <ul>
      <li>[[About Read Consistency|http://www.datastax.com/docs/1.1/dml/data_consistency#about-rea
       </ul>
 </td>
</tr>
    </tr>
       <td>Concurrency</td>
     <td>In Riak, any node in the cluster can coordinate a read/write operation for any other node. Ria
       </td>

     <td>All nodes in Cassandra are peers. A client read or write request can go to any node in the clus
           <ul>
          <li>[[About Client Requests|http://www.datastax.com/docs/1.0/cluster_architecture/about_
           </ul>
     </td>
    </tr>
<tr>
    <td>Replication</td>
  <td>Riak's replication system is heavily influenced by the Dynamo Paper and Dr. Eric Brewer's CAP Th
       <ul>
         <li>[[Replication]]</li>
         <li>[[Clustering|Concepts#Clustering]]</li>
       </ul>

     The Riak APIs expose tunable consistency and availability parameters that let you select which le
           <ul>
          <li>[[Reading, Writing, and Updating Data|Concepts#Reading-Writing-and-Updating-Data]]</
           </ul>

 </td>
  <td>Replication in Cassandra starts when a user chooses a partitioner. Partitioners include Random
       <ul>
      <li>[[Replication|http://www.datastax.com/docs/1.0/cluster_architecture/replication]]</li>
       </ul>

     Like in Riak, Cassandra lets developers configure the consistency and availability requirements
```

```
        <ul>
        <li><a href="http://www.datastax.com/docs/1.1/dml/data_consistency#tunable-consistency">Tu
        </ul>

 </td>
</tr>
<tr>
    <td>Scaling Out and In</td>
   <td>Riak allows you to elastically grow and shrink your cluster while evenly balancing the load on ea

<ul>
    <li>[[Adding and Removing Nodes]]</li>
    <li>[[Command Line Tools]]</li>
</ul>

   <td>Cassandra allows you to add new nodes dynamically with the exception of manually calculating a n
        <ul>
       <li>[[Adding Capacity to an Existing Cluster|http://www.datastax.com/docs/1.1/operations/clus
        </ul>
</td>
</tr>
<tr>
    <td>Multi-Datacenter Replication</td>

   <td>Riak features two distinct types of replication. Users can replicate to any number of nodes in or

    <ul>
      <li><a href="http://basho.com/products/riak-enterprise/">Riak Enterprise</a></li>
    <ul>

   <td>Cassandra has the ability to spread nodes over multiple datacenters via various configuration pa
        <ul>
       <li>[[Multiple Datacenters|http://www.datastax.com/docs/1.1/initialize/cluster_init_multi_d
        </ul>

</td>
</tr>
<tr>
    <td>Graphical Monitoring/Admin Console</td>
   <td>Riak ships with Riak Control, an open source graphical console for monitoring and managing Riak
        <ul>
```

```
            <li>[[Riak Control]]</li>
        <li>[[Introducing Riak Control|http://basho.com/blog/technical/2012/02/22/Riak-Control/]]
        </ul>
</td>
   <td>Datastax distributes the DataStax OpsCenter, a graphical user interface for monitoring and admi
        <ul>
        <li>[[DataStax OpsCenter|http://www.datastax.com/products/opscenter]]</li>
        </ul>
 </td>
</tr>
```

This is intended to be a brief, objective and technical comparison of Riak and Couchbase (i.e. Couchbase Server). The Couchbase version described is 2.0. The Riak version described is Riak 1.2.x. If you feel this comparison is unfaithful at all for whatever reason, please fix it or send an email to **docs@basho.com**.

At A Very High Level

- Riak is Apache 2.0 licensed; According to Couchbase, they have two free versions: Couchbase open source is Apache 2.0 licensed; Couchbase Server Community Edition (free version) is licensed under a community agreement
- Riak is written primarily in Erlang with some bits in C; Couchbase is written in Erlang and C/C++

Couchbase vs CouchDB

Keep in mind that Couchbase and CouchDB are two separate database projects. CouchDB is a document database providing replication, MapReduce and an HTTP API. Couchbase uses CouchDB as its backend, "wrapping" it with advanced features like caching, and is designed to be clustered.

Couchbase 2.0

As of the time of this writing, Couchbase 2.0 is still in developer preview, so some of these points may change between now and the final release. Caveat emptor

Feature/Capability Comparison

The table below gives a high level comparison of Riak and Couchbase features/capabilities. To keep this page relevant in the face of rapid development on both sides, low level details are found in links to Riak and Couchbase online documentation.

```
    <th WIDTH="15%">Feature/Capability</th>
    <th WIDTH="42%">Riak</th>
    <th WIDTH="43%">Couchbase</th>
</tr>
<tr>
    <td>Data Model</td>
    <td>Riak stores key/value pairs in a higher level namespace called a bucket.
        <ul>
        <li>[[Buckets, Keys, and Values|Concepts#Buckets-Keys-and-Values]] </li>
        </ul>
    </td>
   <td>Couchbase is a JSON-based document datastore. Like other document datastores, records have no i
        <ul>
        <li>[[How Should I Store an Object?|http://www.couchbase.com/docs/couchbase-manual-2.0/couc
        </ul>
    </td>
</tr>
<tr>
    <td>Storage Model</td>
   <td>Riak has a modular, extensible local storage system which lets you plug-in a backend store of yo
        <ul>
          <li>[[Riak Supported Storage Backends|Choosing a Backend]]</li>
        </ul>

   You can also write your own storage backend for Riak using our [[backend API|Backend API]].
 </td>
   <td>Couchbase 2.0 is largely memory-based, asynchronously persisting data using a CouchDB fork and
        <ul>
      <li>[[Persistence|http://www.couchbase.com/docs/couchbase-manual-2.0/couchbase-architecture-
      <li>[[Couchbase File Format|https://github.com/couchbaselabs/couchstore/wiki/Format]]</li>
        </ul>
    </td>
</tr>
<tr>
    <td>Data Access and APIs</td>
    <td>Riak offers two primary interfaces (in addition to raw Erlang access):
<ul>
<li>[[HTTP|HTTP API]]</li>
<li>[[Protocol Buffers|PBC API]]</li>
</ul>
```

```
Riak Client libraries are wrappers around these APIs, and client support exists for dozens of languages
<ul>
<li>[[Client Libraries]]</li>
<li>[[Community Projects]]</li>
</ul>
</td>
   <td>Couchbase provides drivers in several languages to access data through its binary memcached pro
        <ul>
        <li>[[Client Interface|http://www.couchbase.com/docs/couchbase-manual-2.0/couchbase-introd
            <li>[[Client-Libraries|http://www.couchbase.com/develop]]</li>
        <li>[[Management REST API|http://www.couchbase.com/docs/couchbase-manual-2.0/couchbase-adm
        </ul>
 </td>
</tr>
<tr>
    <td>Query Types and Query-ability</td>
    <td>There are currently four ways to query data in Riak
        <ul>
        <li>Primary key operations (GET, PUT, DELETE, UPDATE)</li>
        <li>[[Using MapReduce]]</li>
        <li>[[Using Secondary Indexes]]</li>
        <li>[[Using Search]]</li>
        </ul>
</td>
    <td>Couchbase also provides four query options
        <ul>
     <li>[[ID lookups|http://www.couchbase.com/docs/couchbase-manual-2.0/couchbase-developing-best
     <li>[[MapReduce Views|http://www.couchbase.com/docs/couchbase-manual-2.0/couchbase-views-basi
     <li>[[UnQL|http://www.couchbase.com/press-releases/unql-query-language]]</li>
        </ul>
     Hadoop support is also possible through a plugin that streams data to a Hadoop Distributed File Sy
        <ul>
     <li>[[Hadoop Connector|http://www.couchbase.com/develop/connectors/hadoop]]</li>
        <ul>
</td>
</tr>
<tr>
    <td>Data Versioning and Consistency</td>
   <td> Riak uses a data structure called a vector clock to reason about causality and staleness of stor
        <ul>
          <li>[[Vector Clocks]]</li>
```

```
       <li>[[Why Vector Clocks Are Easy|http://basho.com/blog/technical/2010/01/29/why-vector-clocks
       <li>[[Why Vector Clocks Are Hard|http://basho.com/blog/technical/2010/04/05/why-vector-clocks
        </ul>
     </td>

   <td>Couchbase is strongly consistent within a datacenter, replicating data between nodes in a cluste

      Via CouchDB, documents are internally revisioned (stored in a "_rev" value). However, prior revis

    <ul>
   <li>[[Couchbase Architecture|http://www.couchbase.com/docs/couchbase-manual-2.0/couchbase-archi
   <li>[[Internal Version Field|http://www.couchbase.com/docs/couchbase-manual-2.0/couchbase-views-
    </ul>
</tr>
    <td>Concurrency</td>
   <td> In Riak, any node in the cluster can coordinate a read/write operation for any other node. Riak
     </td>

   <td>Couchbase claims to be ACID-compliant on a per-item basis, but has no multi-operation transactio

    <ul>
   <li>[[Transaction and concurrency|http://www.couchbase.com/forums/thread/transaction-and-concure
   <li>[[Cluster Design|http://www.couchbase.com/docs/couchbase-manual-2.0/couchbase-architecture-c
   <li>[[Client-side Proxy|http://www.couchbase.com/docs/couchbase-manual-2.0/couchbase-deployment-
    </ul>

 </td>
</tr>
<tr>
    <td>Replication</td>
   <td>Riak's replication system is heavily influenced by the Dynamo Paper and Dr. Eric Brewer's CAP Th
        <ul>
          <li>[[Replication]]</li>
          <li>[[Clustering|Concepts#Clustering]]</li>
        </ul>

      The Riak APIs expose tunable consistency and availability parameters that let you select which le
            <ul>
           <li>[[Reading, Writing, and Updating Data|Concepts#Reading-Writing-and-Updating-Data]]</l
            </ul>
```

```
 </td>
 <td>Couchbase supports two types of replication.  For intra-datacenter clusters, Couchbase uses memba

        <ul>
      <li>[[CouchDB Replication|http://wiki.apache.org/couchdb/Replication]]</li>
        <li>[[Memcache Tap|http://code.google.com/p/memcached/wiki/Tap]]</li>
      <li>[[CouchDB, Couchbase, Membase|http://www.infoq.com/news/2012/05/couchdb-vs-couchbase-memb

        </ul>
 </td>
</tr>
<tr>
    <td>Scaling Out and In</td>
   <td>Riak allows you to elastically grow and shrink your cluster while evenly balancing the load on ea

<ul>
    <li>[[Adding and Removing Nodes]]</li>
    <li>[[Command Line Tools]]</li>
</ul>
    </td>
   <td>Couchbase scales elastically by auto-sharding. They can be rebalanced to grow or shrink through

    <ul>
   <li>[[Rebalancing|http://www.couchbase.com/docs/couchbase-manual-2.0/couchbase-admin-tasks-addr
   <li>[[Clone to Grow with Auto Sharding|http://www.couchbase.com/couchbase-server/features#clone_t
    </ul>
</td>
</tr>
<tr>
    <td>Multi-Datacenter Replication and Awareness</td>

   <td>Riak features two distinct types of replication. Users can replicate to any number of nodes in or

    <ul>
      <li><a href="http://basho.com/products/riak-enterprise/">Riak Enterprise</a></li>
    </ul>

    </td>
    <td>Couchbase 2.0 supports cross-datacenter replication (XDCR).

    <ul>
```

```
   <li>[[Stabilizing Couchbase Server 2.0|http://blog.couchbase.com/stabilizing-couchbase-server-2-
    </ul>
</td>
</tr>
<tr>
    <td>Graphical Monitoring/Admin Console</td>
   <td>Riak ships with Riak Control, an open source graphical console for monitoring and managing Riak
        <ul>
            <li>[[Riak Control]]</li>
        <li>[[Introducing Riak Control|http://basho.com/blog/technical/2012/02/22/Riak-Control/]]
        </ul>
</td>
    <td>Couchbase provides a web-based monitoring/admin console.
        <ul>
        <li>[[Admin Wed Console|http://www.couchbase.com/docs/couchbase-manual-2.0/couchbase-admin
        <li>[[Monitoring Couchbase|http://www.couchbase.com/docs/couchbase-manual-2.0/couchbase-mo
        </ul>

 </td>
</tr>
```

This is intended to be a brief, objective and technical comparison of Riak and CouchDB. The CouchDB version described is 1.2.x. The Riak version described is Riak 1.2.x. If you feel this comparison is unfaithful at all for whatever reason, please fix it or send an email to **docs@basho.com**.

At A Very High Level

- Riak and CouchDB are both Apache 2.0 licensed
- Riak is written primarily in Erlang with some bits in C; CouchDB is written in Erlang

Feature/Capability Comparison

The table below gives a high level comparison of Riak and CouchDB features/capabilities. To keep this page relevant in the face of rapid development on both sides, low level details are found in links to Riak and CouchDB online documentation.

```
    <th WIDTH="15%">Feature/Capability</th>
    <th WIDTH="42%">Riak</th>
    <th WIDTH="43%">CouchDB</th>
</tr>
<tr>
    <td>Data Model</td>
    <td>Riak stores key/value pairs in a higher level namespace called a bucket.
        <ul>
        <li>[[Buckets, Keys, and Values|Concepts#Buckets-Keys-and-Values]] </li>
        </ul>
    </td>
   <td>CouchDB's data format is JSON stored as documents (self-contained records with no intrinsic rela
        <ul>
        <li>[[Document API|http://wiki.apache.org/couchdb/HTTP_Document_API]]</li>
        </ul>
    </td>
</tr>
<tr>
    <td>Storage Model</td>
   <td>Riak has a modular, extensible local storage system which lets you plug-in a backend store of you
        <ul>
          <li>[[Riak Supported Storage Backends|Choosing a Backend]]</li>
        </ul>

   You can also write your own storage backend for Riak using our [[backend API|Backend API]].
 </td>
   <td>CouchDB stores data to disk by "append-only" files. As the files continue to grow, they require c
        <ul>
        <li>[[Indexes and File|http://guide.couchdb.org/draft/btree.html]]</li>
        </ul>
    </td>
</tr>
<tr>
    <td>Data Access and APIs</td>
    <td>Riak offers two primary interfaces (in addition to raw Erlang access):
<ul>
  <li>[[HTTP|HTTP API]]</li>
<li>[[Protocol Buffers|PBC API]]</li>
</ul>
Riak Client libraries are wrappers around these APIs, and client support exists for dozens of languages
<ul>
```

```
<li>[[Client Libraries]]</li>
<li>[[Community Projects]]</li>
</ul>
</td>
    <td>CouchDB provides an HTTP API for both data access and administration.

            <ul>
        <li>[[Document API|http://wiki.apache.org/couchdb/HTTP_Document_API]]</li>
         <li>[[View API|http://wiki.apache.org/couchdb/HTTP_view_API]]</a></li>
        <li>[[DB API|http://wiki.apache.org/couchdb/HTTP_database_API]]</a></li>
            </ul>

        The CouchDB community supports many client libraries.
        <ul>
       <li>[[Client-Libraries|http://wiki.apache.org/couchdb/Related_Projects/#Libraries]]</li>
        </ul>
 </td>
</tr>
<tr>
    <td>Query Types and Query-ability</td>
    <td>There are currently four ways to query data in Riak
        <ul>
        <li>Primary key operations (GET, PUT, DELETE, UPDATE)</li>
        <li>[[MapReduce|Using MapReduce]]</li>
        <li>[[Using Secondary Indexes]]</li>
        <li>[[Using Search]]</li>
        </ul>

</td>
   <td>CouchDB is generally queried by direct ID lookups, or by creating MapReduce "views" that CouchDB

        <ul>
        <li>[[Views|http://wiki.apache.org/couchdb/HTTP_view_API]]</li>
     <li>[[Changes Notifications|http://guide.couchdb.org/draft/notifications.html]]</li>
        <li>[[Lucene Plugin|https://github.com/rnewson/couchdb-lucene/]]</li>
        <ul>
</td>
</tr>
<tr>
    <td>Data Versioning and Consistency</td>
   <td> Riak uses a data structure called a vector clock to reason about causality and staleness of stor
```

```
        <ul>
          <li>[[Vector Clocks]]</li>
       <li>[[Why Vector Clocks Are Easy|http://basho.com/blog/technical/2010/01/29/why-vector-clocks
       <li>[[Why Vector Clocks Are Hard|http://basho.com/blog/technical/2010/04/05/why-vector-clocks
        </ul>
     </td>

   <td>CouchDB replicates newer document versions between nodes, making it an eventually consistent sy
        <ul>
       <li>[[Eventual Consistency|http://guide.couchdb.org/draft/consistency.html]]</li>
        </ul>
 </td>
</tr>
    <td>Concurrency</td>
   <td> In Riak, any node in the cluster can coordinate a read/write operation for any other node. Riak
     </td>

   <td>Because of CouchDB's append-only value mutation, individual instances will not lock. When distr

        <ul>
        <li>[[No Locking|http://guide.couchdb.org/draft/consistency.html#locking]]</li>
        <li>[[Conflict Management|http://guide.couchdb.org/draft/conflicts.html]]</li>
        </ul>
 </td>
</tr>
<tr>
    <td>Replication</td>
   <td>Riak's replication system is heavily influenced by the Dynamo Paper and Dr. Eric Brewer's CAP Th
        <ul>
          <li>[[Replication]]</li>
          <li>[[Clustering|Concepts#Clustering]]</li>
        </ul>

      The Riak APIs expose tunable consistency and availability parameters that let you select which le
             <ul>
           <li>[[Reading, Writing, and Updating Data|Concepts#Reading-Writing-and-Updating-Data]]</
            </ul>

 </td>
   <td>CouchDB incrementally replicates document changes between nodes. It can be deployed with master
```

```
        <ul>
        <li>[[Replication|http://wiki.apache.org/couchdb/Replication]]</li>
        </ul>
 </td>
</tr>
<tr>
    <td>Scaling Out and In</td>
   <td>Riak allows you to elastically grow and shrink your cluster while evenly balancing the load on ea

<ul>
    <li>[[Adding and Removing Nodes]]</li>
    <li>[[Command Line Tools]]</li>
</ul>
    </td>
   <td>Out of the box, CouchDB is focused on a master-master replication of values (using MVCC to help w

        <ul>
            <li>[[BigCouch|http://bigcouch.cloudant.com/]]</li>
        <li>[[Sharding (on Wikipedia)|http://en.wikipedia.org/wiki/Sharding]]</li>
        </ul>
</td>
</tr>
<tr>
    <td>Multi-Datacenter Replication and Awareness</td>

   <td>Riak features two distinct types of replication. Users can replicate to any number of nodes in or

    <ul>
      <li><a href="http://basho.com/products/riak-enterprise/">Riak Enterprise</a></li>
    </ul>

    </td>
   <td>CouchDB can be configured to run in multiple datacenters. Robust awareness will generally requi

        <ul>
      <li>[[Filtered Replication|http://wiki.apache.org/couchdb/Replication#Filtered_Replication]]<
      <li>[[The Split Brain|http://guide.couchdb.org/draft/conflicts.html#brain]]</li>
        </ul>

</td>
</tr>
```

```
<tr>
    <td>Graphical Monitoring/Admin Console</td>
   <td>Riak ships with Riak Control, an open source graphical console for monitoring and managing Riak
        <ul>
            <li>[[Riak Control]]</li>
        <li>[[Introducing Riak Control|http://basho.com/blog/technical/2012/02/22/Riak-Control/]]
        </ul>
</td>
    <td>CouchDB ships with a graphical interface called Futon.

        <ul>
        <li>[[Welcome to Futon|http://guide.couchdb.org/draft/tour.html#welcome]]</li>
        </ul>
 </td>
</tr>
```

This is intended to be a brief, objective, and technical comparison of Riak and Amazon DynamoDB. The DynamoDB version described is API Version 2011-12-05. The Riak version described is Riak 1.3.x. If you feel this comparison is unfaithful at all for whatever reason, please fix it or send an email to **docs@basho.com**.

At A Very High Level

- Riak is an Apache 2.0 open source licensed project. DynamoDB is a fully managed NoSQL database service that is provided by Amazon as part of Amazon Web Services.
- Because DynamoDB is a database service, its implementation details (language, architecture, etc.) cannot be verified.

Feature/Capability Comparison

The table below gives a high level comparison of Riak and DynamoDB features/capabilities. To keep this page relevant in the face of rapid development on both sides, low level details are found in links to Riak and DynamoDB's online documentation.

Feature/Capability

Riak

DynamoDB

Data Model

Riak stores key/value pairs in a higher level namespsace called a bucket.

[[Buckets, Keys, and Values|Concepts#Buckets-Keys-and-Values]]

DynamoDB's data model contains tables, items, and attributes. A database is a collection of tables. A table is a collection of items and each item is a collection of attributes.

[[DynamoDB Data Model|http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/DataModel.html]]

Storage Model

Riak has a modular, extensible local storage system which lets you plug-in a backend store of your choice to suit your use case. The default backend is Bitcask.

[[Riak Supported Storage Backends|Choosing a Backend]]

```
   You can also write you own storage backend for Riak using our [[backend API|Backend API]].
 </td>
   <td>All data items are stored on Solid State Disks (SSDs) and replicated across multiple [[Availabil
    </td>
</tr>
<tr>
    <td>Data Access and APIs</td>
    <td>Riak offers two primary interfaces (in addition to raw Erlang access):
<ul>
  <li>[[HTTP|HTTP API]]</li>
<li>[[Protocol Buffers|PBC API]]</li>
</ul>
Riak Client libraries are wrappers around these APIs, and client support exists for dozens of languages
<ul>
  <li>[[Client Libraries]]</li>
  <li>[[Community Projects]] </li>
</ul>
</td>
   <td>DynamoDB is a web service that uses HTTP as a transport and JavaScript Object Notation (JSON) as
        <ul>
       <li>[[API Reference for DynamoDB|http://docs.aws.amazon.com/amazondynamodb/latest/developerg
   <li>[[Using the AWS SDKs with DynamoDB|http://docs.aws.amazon.com/amazondynamodb/latest/developer
        </ul>
 </td>
</tr>
<tr>
    <td>Query Types and Query-ability</td>
    <td>There are currently four ways to query data in Riak
        <ul>
```

```
        <li>Primary key operations (GET, PUT, DELETE, UPDATE)</li>
        <li>[[MapReduce|Using MapReduce]]</li>
        <li>[[Using Secondary Indexes]]</li>
        <li>[[Using Search]]</li>
        </ul>

</td>
    <td>DynamoDB offers three approaches to query data:
            <ul>
      <li>Primary key operations (GET, PUT, DELETE, UPDATE)</li>
    <li>[[Query|http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/queryingdynamodb.h
    <li>[[Scan|http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/scandynamodb.html]]
    <li>[[Local Secondary Indexes|http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/L
        <ul>
</td>
</tr>
<tr>
    <td>Data Versioning and Consistency</td>
   <td> Riak uses a data structure called a vector clock to reason about causality and staleness of stor
        <ul>
          <li>[[Vector Clocks]]</li>
       <li>[[Why Vector Clocks Are Easy|http://basho.com/blog/technical/2010/01/29/why-vector-clocks
       <li>[[Why Vector Clocks Are Hard|http://basho.com/blog/technical/2010/04/05/why-vector-clocks
        </ul>
     </td>

   <td>DynamoDB data is eventually consistent, meaning that your read request immediately after a write
        <ul>
       <li>[[Data Read and Consistency Considerations|http://docs.aws.amazon.com/amazondynamodb/late
        </ul>
 </td>
</tr>
    </tr>
        <td>Concurrency</td>
      <td>In Riak, any node in the cluster can coordinate a read/write operation for any other node. Ria
        </td>

      <td>Dedicated resources are allocated to your table (tunable via API) to meet performance require
            <ul>
           <li>[[Provisioned Throughput|http://docs.aws.amazon.com/amazondynamodb/latest/developerg
            </ul>
```

```
   Read and write capacity unit requirements are set at table creation time. When requests such as get,

    <ul>
    <li>[[Capacity Units Calculations|http://docs.aws.amazon.com/amazondynamodb/latest/developergu
    </ul>
     </td>
    </tr>
<tr>
    <td>Replication</td>
   <td>Riak's replication system is heavily influenced by the Dynamo Paper and Dr. Eric Brewer's CAP Th
        <ul>
          <li>[[Replication]]</li>
          <li>[[Clustering|Concepts#Clustering]]</li>
        </ul>

      The Riak APIs expose tunable consistency and availability parameters that let you select which le
            <ul>
           <li>[[Reading, Writing, and Updating Data|Concepts#Reading-Writing-and-Updating-Data]]</
            </ul>

 </td>
   <td>DynamoDB synchronously replicates your data across multiple [[Availability Zones|http://docs.a
 </td>
</tr>
<tr>
    <td>Scaling Out and In</td>
   <td>Riak allows you to elastically grow and shrink your cluster while evenly balancing the load on ea

     <ul>
      <li>[[Adding and Removing Nodes]]</li>
      <li>[[Command Line Tools]]</li>
    </ul>

   <td>DynamoDB requires that you specify your required read and write throughput values when you creat
throughput values can be increased and decreased later as access requirements change. This is used to r
      <ul>
     <li>[[Provisioned Throughput|http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
        </ul>
      </td>
</tr>
<tr>
```

```
    <td>Multi-Datacenter Replication</td>

   <td>Riak features two distinct types of replication. Users can replicate to any number of nodes in o

    <ul>
      <li><a href="http://basho.com/products/riak-enterprise/">Riak Enterprise</a></li>
    <ul>

   <td>DynamoDB has the ability to spread instances over multiple [[Availability Zones|http://docs.aw
    </td>
</tr>
<tr>
    <td>Graphical Monitoring/Admin Console</td>
   <td>Riak ships with Riak Control, an open source graphical console for monitoring and managing Riak
         <ul>
             <li>[[Riak Control]]</li>
         <li>[[Introducing Riak Control|http://basho.com/blog/technical/2012/02/22/Riak-Control/]]
         </ul>
</td>
   <td>DynamoDB and [[CloudWatch|http://aws.amazon.com/cloudwatch/]] are integrated, which allows yo
         <ul>
         <li>[[Monitoring Amazon DynamoDB|http://docs.aws.amazon.com/amazondynamodb/latest/develope
         </ul>
 </td>
</tr>
```

This is intended to be a brief, objective and technical comparison of Riak and HBase. The HBase version described is 0.94.x. The Riak version described is Riak 1.2.x. If you feel this comparison is unfaithful at all for whatever reason, please fix it or send an email to **docs@basho.com**.

At A Very High Level

- Riak and HBase are both Apache 2.0 licensed
- Riak is based on Amazon's Dynamo paper; HBase is based on Google's BigTable
- Riak is written primarily in Erlang with some C; HBase is written in Java

Feature/Capability Comparison

The table below gives a high level comparison of Riak and HBase features and capabilities. To keep this page relevant in the face of rapid development on both sides, low level details are found in links to Riak and HBase online documentation.

Feature/Capability

Riak

HBase

Data Model

Riak stores key/value pairs in a higher level namespace called a bucket.

[[Buckets, Keys, and Values|Concepts#Buckets, Keys, and Values]]

HBase stores data in a pre-defined column family format (each grouping of data has a key, and any number of column attributes which may be versioned individually). Data in HBase is sorted, sparse, and physically grouped by column family (rather than by row, as in a relational database). HBase calls their groupings "tables".

[[HBase Data Model|http://hbase.apache.org/book/datamodel.html]]

[[Supported Data Types|http://hbase.apache.org/book/supported.datatypes.html]]

Storage Model

Riak has a modular, extensible local storage system which features pluggable backend stores designed to fit a variety of use cases. The default Riak backend store is Bitcask.

[[Riak Supported Storage Backends|Choosing a Backend]]

You can also write your own storage backend for Riak using our [[backend API|Backend API]].

Hadoop Distributed File System (HDFS) is the storage system used by HBase. Data is stored in MemStores and StoreFiles, where data is streamed to disk (implemented via HFiles, a format based on BigTable's SSTable). Implementations generally use the native JVM-managed I/O file stream.

[[HDFS|http://en.wikipedia.org/wiki/Apache_Hadoop#Hadoop_Distributed_File_System]]

[[Hadoop Uses HDFS|http://hbase.apache.org/book/arch.hdfs.html]]

Data Access and Application Programming Interfaces (APIs)

In addition to raw Erlang access, Riak offers two primary APIs:

[[HTTP|HTTP API]]

[[Protocol Buffers|PBC API]]

Riak Client libraries are wrappers around these APIs, and client support exists for dozens of languages.

[[Client Libraries]]

[[Community Projects]]

HBase communicates primarily through code that runs on the JVM (Java, Jython, Groovy, etc.). Alternatively, HBase provides external protocols; either REST or Thrift (a cross-language data service format).

[[Java Interface|http://hbase.apache.org/book/architecture.html]]

[[REST|http://wiki.apache.org/hadoop/Hbase/Stargate]]

[[Thrift|http://thrift.apache.org/]]

Query Types and Query-ability

There are currently four ways to query Riak.

Primary key operations (GET, PUT, DELETE, UPDATE)

[[MapReduce|Using MapReduce]]

[[Using Secondary Indexes]]

[[Using Search]]

```
</td>
   <td>HBase has two query options: looking up values by getting/scanning through ordered keys (optiona
         <ul>
          <li>[[Scanning|http://hbase.apache.org/book/client.filter.html]]</li>
             <li>[[MapReduce|http://hbase.apache.org/book/mapreduce.html]]</li>
         <li>[[Secondary Indexes|http://hbase.apache.org/book/secondary.indexes.html]]</li>
         </ul>
</td>
</tr>
<tr>
    <td>Data Versioning and Consistency</td>
   <td>Riak uses a data structure called a vector clock to reason about causality and staleness of store
         <ul>
           <li>[[Vector Clocks]]</li>
       <li>[[Why Vector Clocks Are Easy|http://basho.com/blog/technical/2010/01/29/why-vector-clocks
       <li>[[Why Vector Clocks Are Hard|http://basho.com/blog/technical/2010/04/05/why-vector-clocks
         </ul>
     </td>
   <td>HBase has strongly consistent reads/writes. Data may be autosharded across regions and redistri

   Column families may contain an unbounded number of versions, with optional TTL.
         <ul>
```

```
        <li>[[Consistent Architecture|http://hbase.apache.org/book/architecture.html#arch.overview
        </ul>
 </td>
</tr>
    <td>Concurrency</td>
   <td> In Riak, any node in the cluster can coordinate a read/write operation for any other node. Riak
     </td>

   <td>HBase guarantees write atomicity and locks per row. HBase has also recently added multi-action a
        <ul>
        <li>[[Consistency Guarantees|http://hbase.apache.org/acid-semantics.html]]</li>
        <li>[[http://hadoop-hbase.blogspot.com/2012/03/acid-in-hbase.html]]</li>
        </ul>
 </td>
</tr>
<tr>
    <td>Replication</td>
   <td>Riak's replication system is heavily influenced by the Dynamo Paper and Dr. Eric Brewer's CAP Th
        <ul>
          <li>[[Replication]]</li>
          <li>[[Clustering|Concepts#Clustering]]</li>
        </ul>

      The Riak APIs expose tunable consistency and availability parameters that let you select which le
            <ul>
           <li>[[Reading, Writing, and Updating Data|Concepts#Reading, Writing, and Updating Data]]</
            </ul>
 </td>
   <td>HBase supports in-cluster and between-cluster replication. In-cluster replication is handled by
    <ul>
    <li>[[Replication|http://hbase.apache.org/replication.html]]</li>
    </ul>
 </td>
</tr>
<tr>
    <td>Scaling Out and In</td>
   <td>Riak allows you to elastically grow and shrink your cluster while evenly balancing the load on ea
<ul>
    <li>[[Adding and Removing Nodes]]</li>
    <li>[[Command Line Tools]]</li>
</ul>
```

```
    </td>
   <td>HBase shards by way or regions, that automatically split and redistribute growing data. A crash
        <ul>
            <li>[[Regions|http://hbase.apache.org/book/regions.arch.html]]</li>
        <li>[[Node Management|http://hbase.apache.org/book/node.management.html]]</li>
        <li>[[HBase Architecture|http://hbase.apache.org/book/architecture.html]]</li>
        </ul>
</td>
</tr>
<tr>
    <td>Multi-Datacenter Replication and Awareness</td>

   <td>Riak features two distinct types of replication. Users can replicate to any number of nodes in o
    <ul>
     <li><a href="http://basho.com/products/riak-enterprise/">Riak Enterprise</a></li>
    </ul>

    </td>
   <td>HBase shards by way of regions, that themselves may be replicated across multiple datacenters.
        <ul>
         <li>[[Node Management|http://hbase.apache.org/replication.html]]</li>
        </ul>
</td>
</tr>
<tr>
    <td>Graphical Monitoring/Admin Console</td>
   <td>Riak ships with Riak Control, an open source graphical console for monitoring and managing Riak
        <ul>
            <li>[[Riak Control]]</li>
        <li>[[Introducing Riak Control|http://basho.com/blog/technical/2012/02/22/Riak-Control/]]
        </ul>
</td>
   <td>HBase has a few community supported graphical tools, and a command-line admin console.
    <ul>
   <li>[[Admin Console Tools|http://hbase.apache.org/book/ops_mgt.html#tools]]</li>
   <li>[[Eclipse Dev Plugin|http://wiki.apache.org/hadoop/Hbase/EclipseEnvironment]]</li>
    <li>[[HBase Manager|http://sourceforge.net/projects/hbasemanagergui/]]</li>
    <li>[[GUI Admin|https://github.com/zaharije/hbase-gui-admin]]</li>
    </ul>
 </td>
</tr>
```

The NoSQL space, as well as the database space in general, is growing ever-crowded. Because of this, we often find ourselves answering very high-level questions from developers, prospects, and customers along the lines of, "How does Riak compare to this database?" or "What is the main difference between your replication strategy and this NoSQL Database?" So, we thought it would be a worthwhile exercise to make available very brief and objective comparisons to as many databases as possible. (The list below will be growing as soon as we have the time to grow it.)

- [[Riak Compared to Cassandra]]
- [[Riak Compared to Couchbase]]
- [[Riak Compared to CouchDB]]
- [[Riak Compared to HBase]]
- [[Riak Compared to MongoDB]]
- [[Riak Compared to Neo4j]]
- [[Riak Compared to DynamoDB]]

Disclaimer: We tried to get this right, but software is complicated, and it changes rapidly. If you think we have made an error, please kindly correct us, and we will be happy to make the change.

Slide Decks

This is a sample of the slide decks used in presentations given by Riak Core Developers and Developer Advocates, and members of the Riak Community at conferences, meetups, and various other events worldwide. *(If you have a Slide Deck to add, please fork the Riak Docs Repo on GitHub and do so.)*

- [[Choosing The Right NoSQL Database - 4Developers|http://www.slideshare.net/juokaz/choosing-the-right-nosql-database-4developers]] - A whirlwind tour of a few NoSQL solutions, learning the very different ways they represent data and seeing their unique strengths and weaknesses in various kinds of applications. Along the way, we'll learn why new technologies must be introduced to address today's scaling challenges, and what compromises we'll have to make if we want to abandon the databases of our youth.
- [[Rolling With Riak|http://www.slideshare.net/johnthethird/rolling-with-riak]] - Overview of Riak's NoSQL distributed key/value data store by John Lynch from Rigel Group.
- [[How does Riak compare to Cassandra?|http://www.slideshare.net/ukdi1/how-does-riak-compare-to-cassandra-cassandra-london-user-group-july-2011]] - A presentation about Riak and quick comparison to Cassandra. Presented originally at the Cassandra London User Group in July 2011.