Beyond Relational: Riak's Journey into Distributed Data Management

Avril Mauro & Katelyn Donn

DS 4300

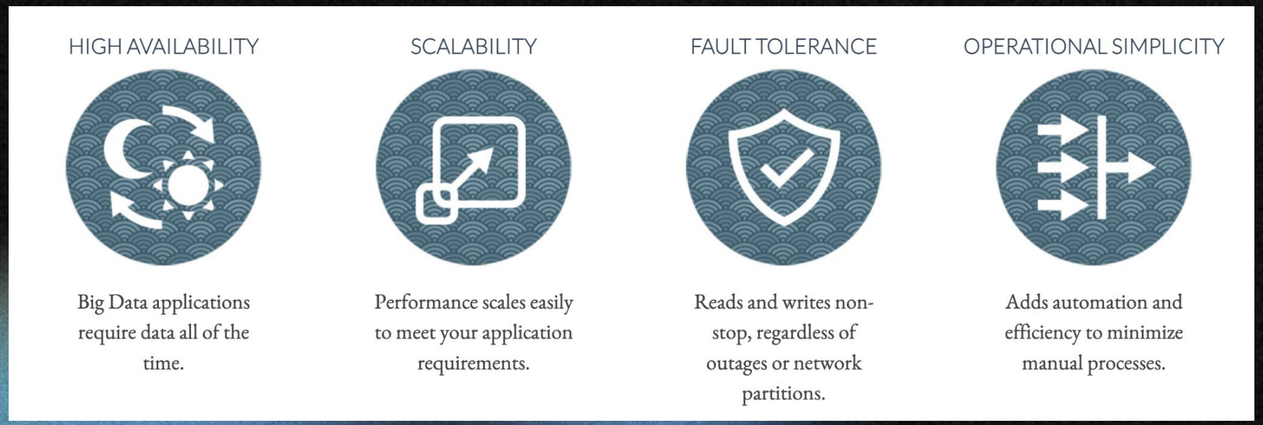
riak

TABLE OF CONTENTS

What is Riak?

Riak is a distributed NoSQL database renowned for its high availability, fault tolerance, operational simplicity, and scalability.

Riak has 3 main products: Riak KV, Riak TS, and Riak CS. Each product is tailored to address specific data storage and management needs.



Main Products

Riak KV

- Optimized for key-value data storage and retrieval.
- Features high availability, fault tolerance, and scalability.
- Ideal for real-time data processing and low-latency access to large volumes of data.
- Supports multi-datacenter replication and flexible consistency options.

Riak TS

- Designed for time-series data handling, such as sensor readings or financial metrics.
- Offers efficient storage, retrieval, and analysis capabilities.
- Excels at managing time-stamped data with predictable performance and scalability.
- Features automatic data expiration, time-based aggregation queries, and distributed processing.

Riak CS

- Scalable and durable object storage system built on Riak KV, tailored for cloud environments.
- Provides RESTful APIs compatible with Amazon S3 for easy integration with cloud applications.
- Supports multi-tenancy, data replication, and configurable durability settings.
- Facilitates storage and retrieval of diverse unstructured data types in cloud-based applications.

Use Cases

Highly Available Web Applications

- Distributed architecture → can build highly available web applications.
- Automatically replicate data across multiple nodes → minimizes downtime and ensuring uninterrupted service for end-users.
- Easily can add or remove nodes from cluster to accommodate varying traffic loads, optimizing resource utilization and cost efficiency.

IoT Data Storage and Processing

- Scalability and fault tolerance → can handle the volumes and diverse types of data generated by IoT devices.
- Integration with stream processing frameworks or analytics platforms → in-depth analysis on the incoming data streams, identifying patterns, anomalies, and trends in real-time.

Brief Riak Tutorial

Mac Installation

With **Homebrew**, you can install Riak with this command in the terminal: `**pip install riak**`. Build the client with this command: `**python setup.py install**`. Then, you can begin the client with this command: `**riak start**` and verify that it is running with `**riak ping**`.

Mac OS Terminal Database Creation

Create a Bucket: `riak-admin bucket-type create <bucket_type> '{"props":{"<property_name>":"<value>"}}'`
Verify Bucket Creation: `riak-admin bucket-type list`
Store Data in the Bucket: `riak-admin bucket-type store <bucket_type> <key> '{"field1":"value1","field2":"value2", ...}'`
Retrieve Data: `riak-admin bucket-type get <bucket_type> <key>`

Python

Riak Object:

- Data unit in Riak KV.
- Consists of a key-value pair.
- Holds various data formats.
- Distributed across nodes for fault tolerance.
- Can include metadata for context or control.

Riak Vector Clock (vclock):

- Tracks causality between data updates.
- Comprises node-counter pairs.
- Enables conflict resolution.
- Ensures eventual consistency.

Riak Siblings:

- Conflicting versions of an object.
- Arise from concurrent updates without clear causality.
- Detected using vector clocks.
- Resolved through mechanisms or manual intervention.
- Maintains availability and eventual consistency.

```python
from riak import RiakClient

client = RiakClient(protocol='http', host='127.0.0.1', http_port=8098)
mybucket = client.bucket('mybucket')

mybucket.set_property('key', 'value')
mybucket.get_property('key') # returns value
```

```python
from riak.content import RiakContent


class RiakObject(object):
    """
    Holds meta information about a Riak object and its data
    """
    def __init__(self, client, bucket, key=None):
        """
        Construct a new RiakObject.
        """
        self._resolver = None
        self.client = client
        self.bucket = bucket
        self.key = key
        self.vclock = None
        self.siblings = [RiakContent(self)]

    #: The list of sibling values contained in this object
    siblings = []

    def __hash__(self):
        return hash((self.key, self.bucket, self.vclock))
```

riak

Strengths

- Fault tolerance, scalability, and eventual consistency.
- Suitable for high availability and handling large data volumes.
- Enables seamless expansion through horizontal scaling.
- Provides flexibility with eventual consistency for availability-focused applications.

Weaknesses

- Complex configuration and management requirements.
- Requires intricate tuning for optimal performance.
- Limited querying capabilities compared to relational databases.
- May not be suitable for complex ad-hoc queries or sophisticated data analysis tasks.


riak
THANK YOU!