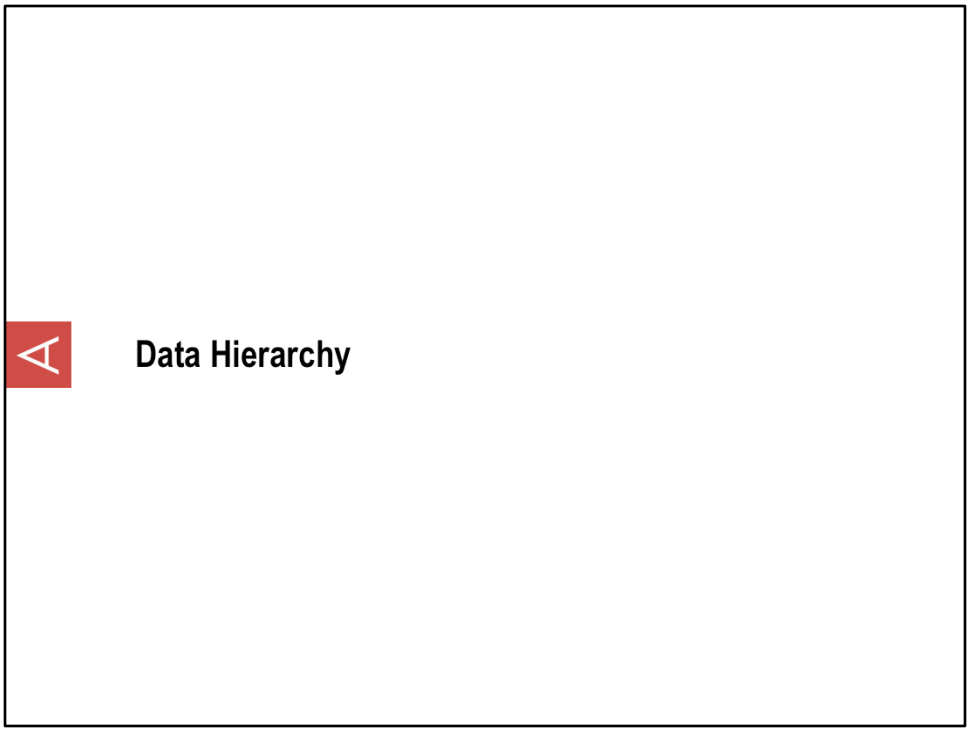
Architecture

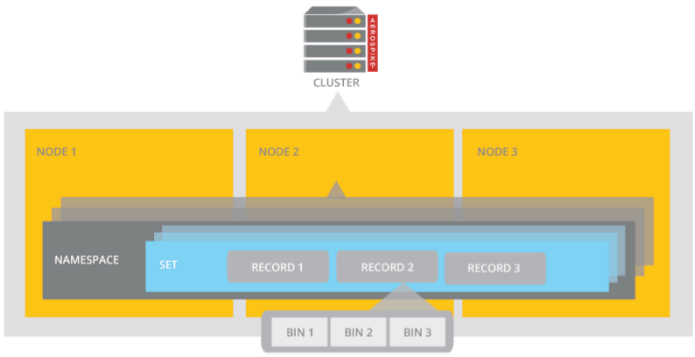
Objectives

This module covers the following:

- Data hierarchy.
- High level architecture.
- Data partitioning.


Data Hierarchy

Data Hierarchy



Database Hierarchy

Term	RDBMs	Definition	Notes
Cluster	-	An Aerospike cluster services a single database service.	While a company may deploy multiple clusters, applications will only connect to a single cluster.
Node	-	A single instance of an Aerospike database. A node will act as a part of the whole cluster.	For production deployments, a host should only have a single node. For development, you may place more than one node on a host.
Namespace	Database	An area of storage related to the media. Can be either RAM or flash (SSD based). Setting up new/removing namespaces requires a cluster-wide restart.	
Set	Table	An unstructured grouping of data that has some commonality.	Similar to “tables” in a relational database, but do not require a schema.
Record	Row	A key and all data related to that key.	Aerospike always stores all data for a record on the same node.
Bin	Column	One part of data related to a key.	Bins in Aerospike are typed, but the same bin in different records can have different types. Bins are not required. Single bin optimizations are allowed.

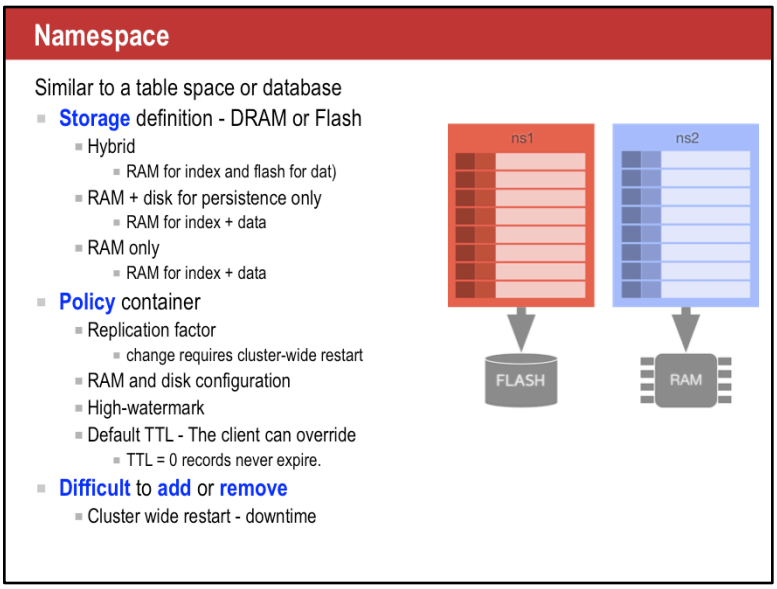
Nodes

- Each node should have **identical hardware**.
- Should have **identical configuration**.
- Data (and their associated traffic) will be **evenly balanced** across the nodes.
- Big **differences** between nodes implies a **problem**.
- Node **capacity** should take into account **node failure** patterns.

If you have an 8 node cluster distributed evenly on 4 racks, make sure that 6 nodes can handle the data and traffic volume in the event you lose a single rack.

Tracking what happens in the event of node loss is critical. You should understand what will happen under different failure modes, such a single node loss, single rack loss, etc.

If you have an 8 node cluster distributed evenly on 4 racks, make sure that 6 nodes can handle the data and traffic volume in the event you lose a single rack.



Namespaces

Namespaces are the top level containers for data. A namespace can actually be a part of a database or it can be a group of databases as you would think of them in a standard RDBMS – the reason you collect data into a namespace relates to how the data is going to be stored and managed.

A namespace contains records, indexes and policies. A policy dictates the behavior of the namespace, including:

- How data is stored: DRAM or disk
- How many replicas should exists for a record.
- When records should expire.
- Consistency

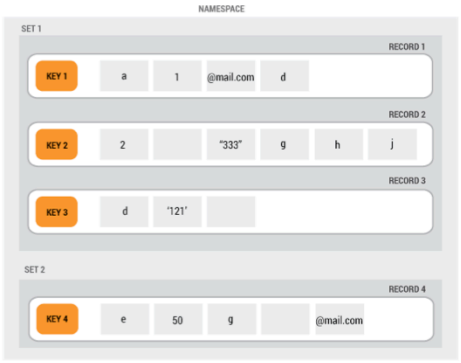
Namespaces can only be defined with a cluster wide restart. This makes them difficult to add or remove.

A database may specify multiple namespaces, each with differing policies, depending on the applications' needs. A namespace is considered a physical container, because it binds the data to a storage device, whether a segment of RAM, a disk or file.

Set

Sets are **similar** to tables

- But it has **no schema**
- Arbitrary grouping of records
- Inherits policy from namespace
- Prefix to **primary key**
- Set name <= 63 characters
- 1023 per namespace
- Cannot be deleted or renamed



Set

Within a namespace, records can belong to a logical container called set. A set provides applications the ability to group records in collections. A Set is similar to a table as it is a collection of records, but it has no schema. Sets use the policy defined by the namespace to which they belong, such as storage, replication and consistency. A Set may define additional policies specific to the set, such as a secondary index definition. Sets names are like a prefix to the primary key, and cannot be deleted or renamed.

The total number of sets that can be created in a namespace is 1023. This is 1023 “named” sets and 1 unnamed set (or null set).

Note: Moving a record from one Set to another may move it from one server to another. Sets can be iterated using the Scan operation.

Record

A record is a "**row**" of Key-value – (Object)

- **Value**: one or more bins
- Bin has a **name and type**
  - Bin types: String, integer, blob, list, map
- Bins can be added at any time
- **Generation** counter
  - Optimistic Concurrency
- **Time-to-live** = auto expiration
- Reads/Writes are **atomic**
- Any change = **complete rewrite**
- Data stored **contiguously** on the same node.

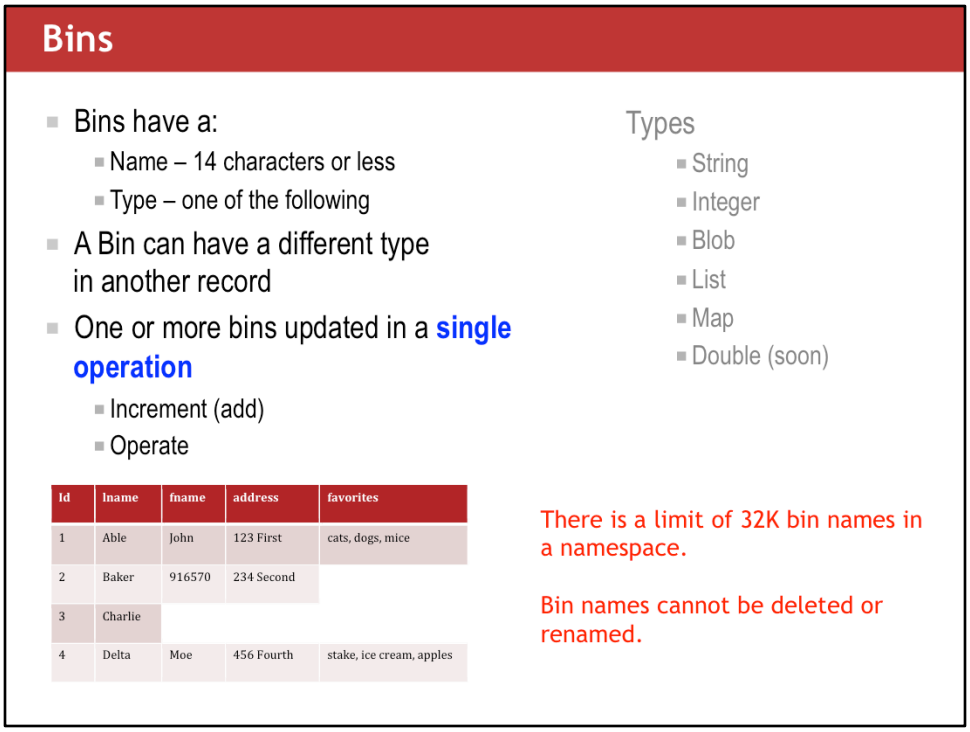
Re

Component	Description
Key	A 20-byte is addressable via a hash of its key called a Digest
Metadata	The metadata provides information about the version of the record (generation) and the time-to-live (TTL).
Bins	Bins are the equivalent of columns or fields in an conventional RDBMS.

Ae[illegible] rows in a standard RDBMS). A record is the basic unit of storage in the database and its Bins (columns) are stored contiguously. A mentioned earlier, records may belong to a namespace or a set within a namespace. A record is addressable via a key, which is used to uniquely identify the records in the namespace.
A record is composed of the following:

Key / Digest

In the application, each record will have a key associated with it. This key is what the application will use to read or write the record.
However, when the key is sent to the database, the key is hashed into a 160-bit digest. Within the database, the digest is used address the record for all operations. The key is used primarily in the application, while the digest is primarily used for addressing the record in the database.



Bins

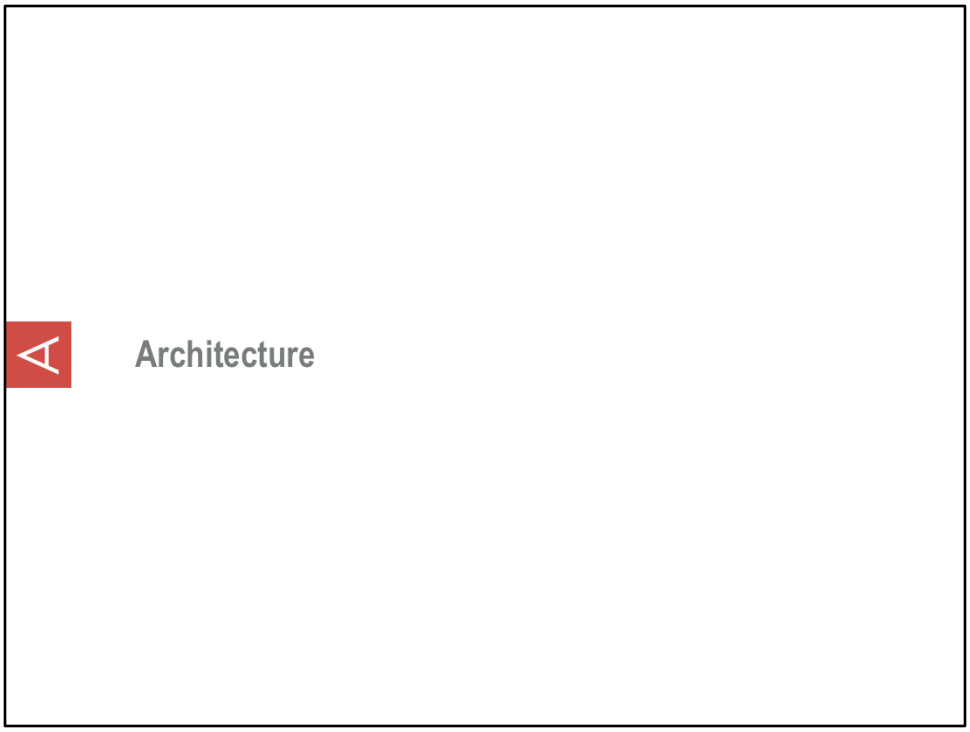
Within a record, data is stored in one or many bins. A bin consists of a name and a value. A bin does not specify the type, instead the type is defined by the value contained in the bin.

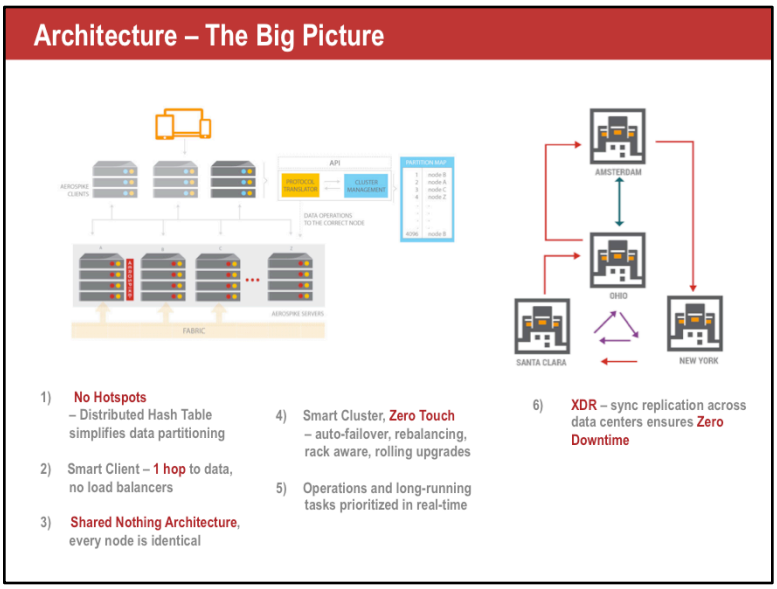
This dynamic typing provides much flexibility in the data model. For example, a record may contain a bin named "id" with a string value of "bob". The value of the bin can always change to a different string value, but to a value of a different type, such as the integer 72.

Also, records within a namespace or set maybe composed of very different collection of bins. There is no schema for the records, so it is possible for each record to have completely different set of bins. Also, bins can be added or removed at any point in the lifetime of a record.

There is a limit on the number of bin names currently in use within a namespace, due to an optimized string-table implementation. That limit is 32K unique bin names per namespace. If more bin names are required, consider using a map. With a map, you can store an arbitrary set of key-value pairs, and access those values in a UDF for efficiency.

A bin name is limited to 14 characters.


Architecture



The Big Picture

Aerospike is a distributed, scalable NoSQL database. It is architected with three key objectives:

- To create a flexible, scalable platform that would meet the needs of today's web-scale applications
- To provide the robustness and reliability expected from traditional databases.
- To provide operational efficiency (minimal manual involvement)

First published in the Proceedings of VLDB (Very Large Databases) in 2011, the Aerospike architecture consists of 3 layers:

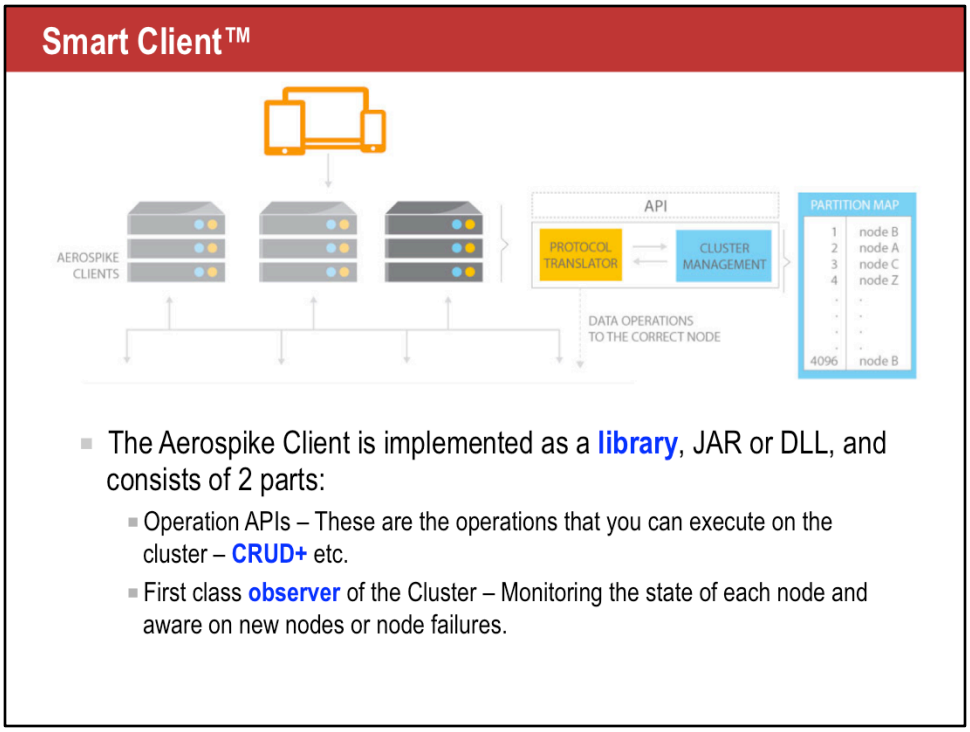
1. The cluster-aware Client Layer includes open source client libraries that implement Aerospike APIs, track nodes and know where data reside in the cluster.
2. The self-managing Clustering and Data Distribution Layer oversees cluster communications and automates fail-over, replication, cross data center synchronization and intelligent re-balancing and data migration.
3. The flash-optimized Data Storage Layer reliably stores data in RAM and Flash.

Aerospike scales up and out on commodity servers.

1) Each multi-threaded, multi-core, multi-cpu server is fast
2) Each server scales up to manage up to 16TB of data, with parallel access to up to 20 SSDs
3) Aerospike scales out linearly with each identical server added, to parallel process 100TB+ across the cluster

(AppNexus was in production with a 50 node cluster that has now scaled up by increasing SSD capacity per node and reducing node count to 12, in preparation for scaling back up on node counts and SSD capacity)
- see http://www.aerospike.com/wp-content/uploads/2013/12/Aerospike-AppNexus-SSD-Case-Study_Final.pdf

SYNC replication within cluster for immediate consistency, #replicas from 1 to #nodes in cluster, typical deployment has 2 or 3 copies of data; Can support ASYNC within cluster. ASYNC replication across clusters using Cross Data Center Replication (XDR). Writes

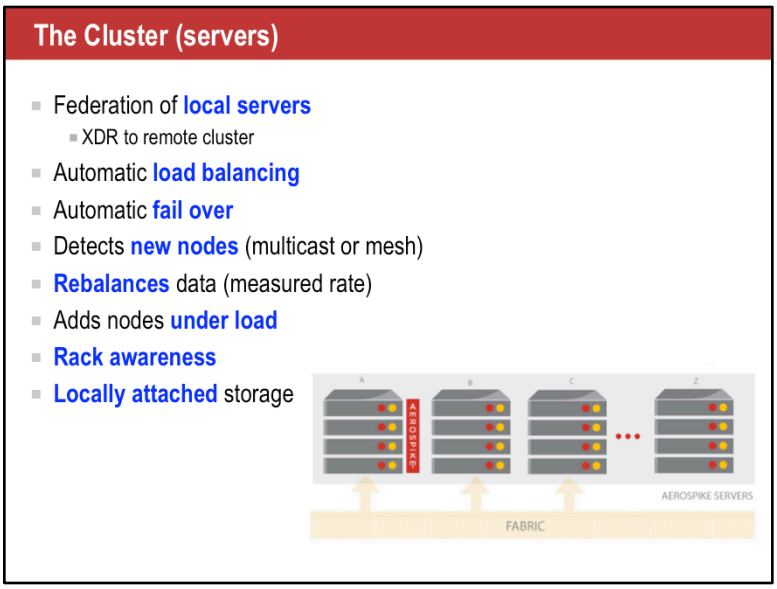


Client Layer

The Aerospike "smart client" is designed for speed. It is implemented as an open source linkable library available in C, C#, Java, PHP, Go, node.js and Python, and developers are free to contribute new clients or modify them as needed. The Client Layer has the following functions:

- Implements the Aerospike API, the client-server protocol and talks directly to the cluster.
- Tracks nodes and knows where data is stored, instantly learning of changes to cluster configuration or when nodes go up or down.
- Implements its own TCP/IP connection pool for efficiency. Also detects transaction failures that have not risen to the level of node failures in the cluster and re-routes those transactions to nodes with copies of the data.
- Transparently sends requests directly to the node with the data and re-tries or re-routes requests as needed. One example is during cluster re-configurations.

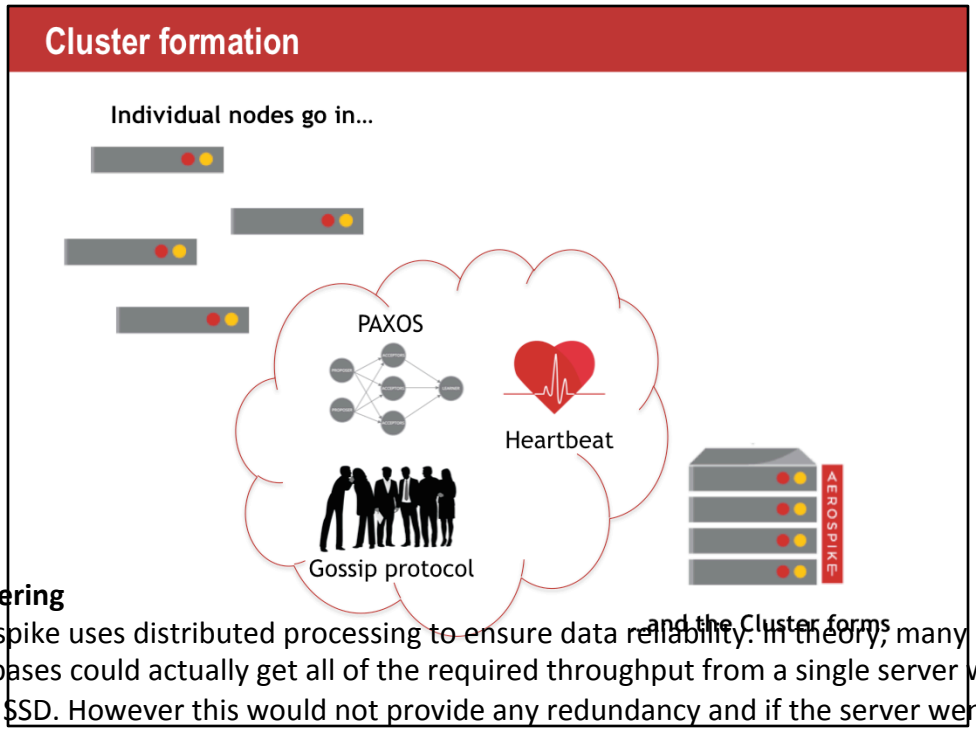
This architecture reduces transaction latency, offloads work from the cluster and eliminates work for the developer. It also ensures that applications do not have to be restarted when nodes are brought up or down. Finally, it eliminates the need to setup and manage additional cluster management servers or proxies.



Cluster Layer

The Aerospike "shared nothing" architecture is designed to reliably stores terabytes of data with automatic fail-over, replication and cross data-center synchronization. This layer scales linearly and implements many of the ACID guarantees. This layer is also designed to eliminate manual operations with the systematic automation of all cluster management functions. It includes 3 modules:

- The Cluster Management Module tracks nodes in the cluster. The key algorithm is a Paxos-like consensus voting process which determines which nodes are considered part of the cluster. Aerospike implement special heartbeat (active and passive) to monitor inter-node connectivity.
- When a node is added or removed and cluster membership is ascertained, each node uses distributed hash algorithm to divide the primary index space into data 'slices' and assign owners. Data Migration Module then intelligently balances the distribution of data across nodes in the cluster, and ensures that each piece of data is duplicated across nodes and across data centers, as specified by the system's configured replication factor.
- Division is purely algorithmic, the system scales without a master and eliminates the need for additional configuration that is required in a sharded environment.
- The Transaction Processing Module reads and writes data as requested and provides many of the consistency and isolation guarantees. This module is responsible for
    1. Sync/Async Replication : For writes with immediate consistency, it propagates changes to all replicas before committing the data and returning the result to the client.
    2. Proxy : In rare cases during cluster re-configurations when the Client Layer may be briefly out of date, it transparently proxies the request to another node.
    3. Duplicate Resolution : when a cluster is recovering from being partitioned, it resolves any conflicts that may have occurred between different copies of data. Resolution can be configured to be
        - Automatic, in which case the data with the latest timestamp is canonical



Clustering

Aerospike uses distributed processing to ensure data reliability. In theory, many databases could actually get all of the required throughput from a single server with a huge SSD. However this would not provide any redundancy and if the server went down, all database access would stop. So a more typical configuration is several nodes, with each node having several SSD devices. Aerospike typically runs on fewer servers than other databases.

The distributed, shared-nothing architecture means that nodes can self-manage and coordinate to ensure that there are no outages to the user, while at the same time being easy to expand your cluster as traffic increases.

Heartbeat

The nodes in the cluster keep track of each other through a heartbeat so that they can coordinate among themselves. The nodes are peers – there is no one node that is the master, all of the nodes track the other nodes in the cluster. When nodes are added or removed, it is detected by the nodes in the cluster using heartbeat mechanism. Aerospike have two way of defining cluster

- Multicast in this case multicast IP:PORT is used to broadcast heartbeat message
- Mesh in this case address of one Aerospike server is used to join cluster

High speed distributed consensus

Aerospike's 24/7 reliability guarantee starts with its ability to detect cluster failure and quickly recovering from that situation to reform the cluster.

Once the cluster change is discovered, all surviving nodes use a Paxos-like consensus voting process to determine which nodes form the cluster and Aerospike Smart Partitions™ algorithm to automatically re-allocate partitions and re-balance. The hashing algorithm is deterministic – that is, it always maps a given record to the same partition. Data records stay in the same partition for their entire life although partitions may move from one server to another.

All of the nodes in the Aerospike system participate in a Paxos distributed consensus algorithm, which is used to ensure agreement on a minimal amount of critical shared state. The most critical part of this shared state is the list of nodes that are participating in the cluster. Consequently, every time a node arrives or departs, the consensus

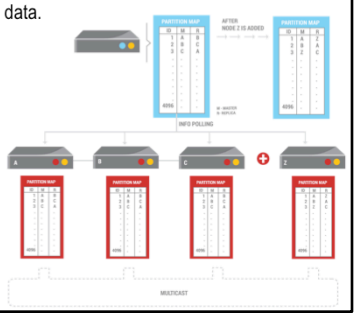
Distributing Data: The Partition Map

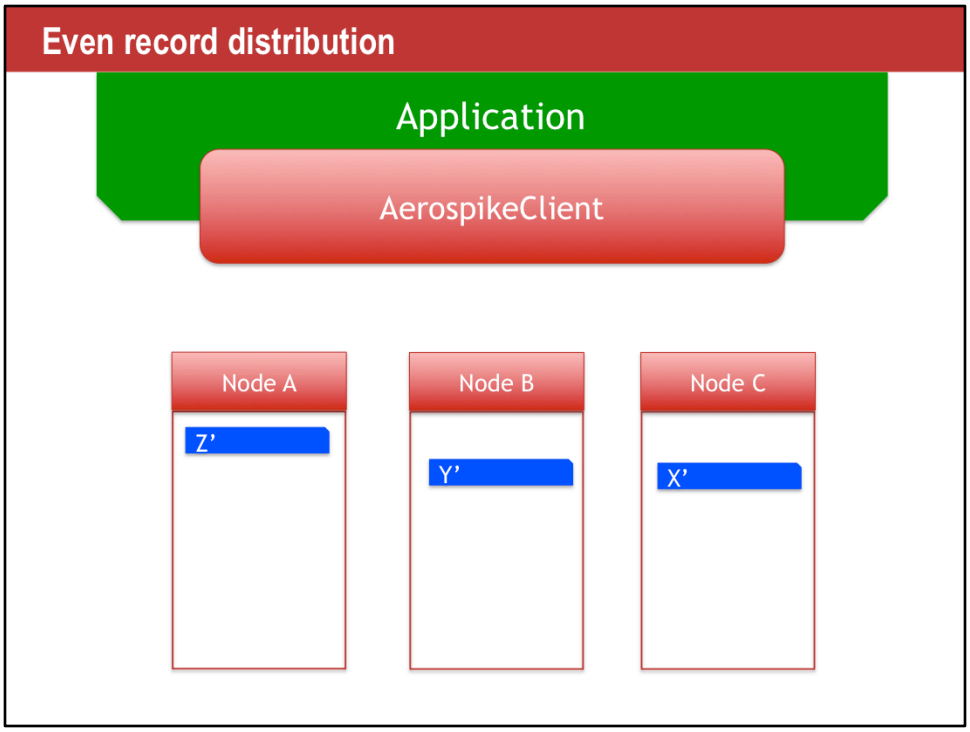
Distributing data can be done in many ways. Aerospike has chosen a method that:

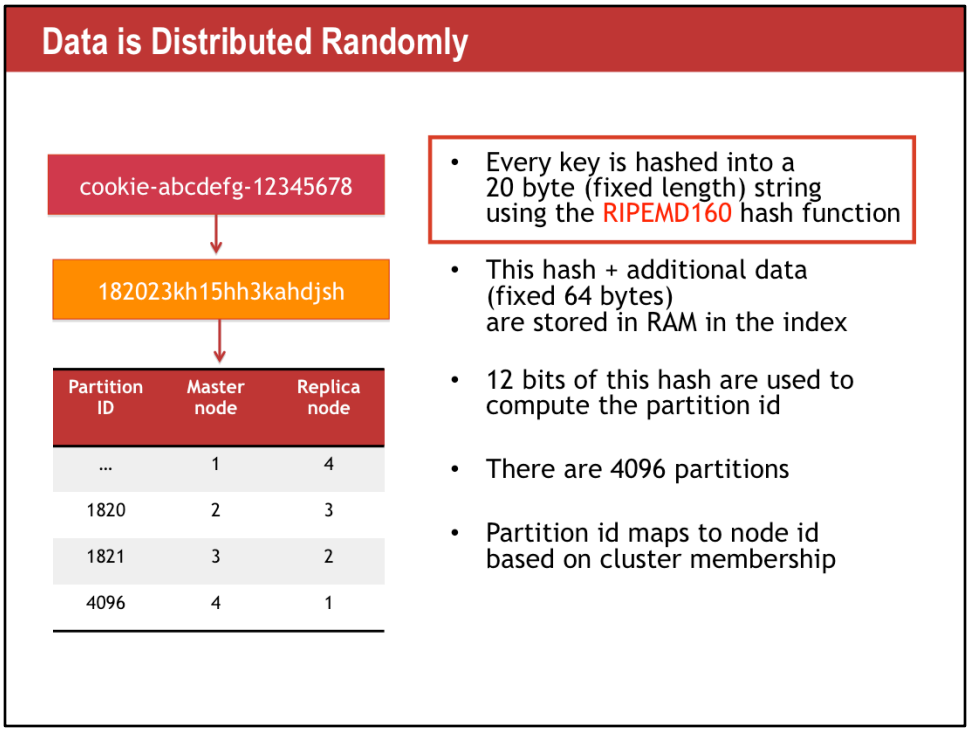
1. Automatically **balances** data across **nodes**.
2. Makes it easy to **migrate (rebalance)** should a node crash or be added.
3. Does not require the developer to understand how the data is **distributed**.
4. Takes into account **replica** copies of the data.

1 Hop to data

- Smart Client simply calculates Partition ID to determine Node ID
- No Load Balancers required




Even record distribution
Application
AerospikeClient
Node A
Z'
Node B
Y'
Node C
X'



When a cluster forms, it will generate a partition map.

The partition map is distributed to all the nodes in the cluster as well as all the clients. It is essential to the central operation of the database and allows for it to run without any true “master”.

For the curious: https://online.tugraz.at/tug_online/voe_main2.getvolltext?pCurrPk=17675. And: https://discuss.aerospike.com/t/what-will-aerospike-do-if-there-is-a-hash-collision-two-records-have-the-same-key/779.

Losing a Node

Let's take a 3 node cluster with 12 partitions and a replication factor of 2.
When everything is stable, every thing will be evenly distributed.

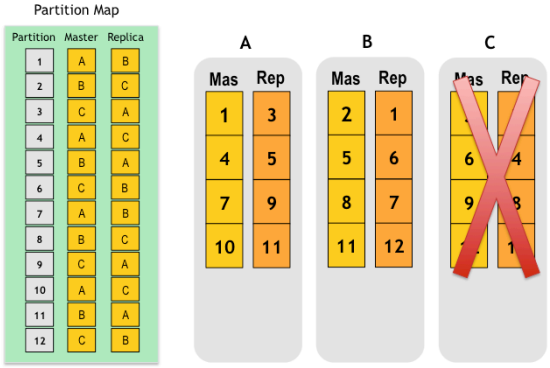
Partition Map

Partition	Master	Replica
1	A	B
2	B	C
3	C	A
4	A	C
5	B	A
6	C	B
7	A	B
8	B	C
9	C	A
10	A	C
11	B	A
12	C	B

A		B		C	
Mas	Rep	Mas	Rep	Mas	Rep
1	3	2	1	3	2
4	5	5	6	6	4
7	9	8	7	9	8
10	11	11	12	12	10

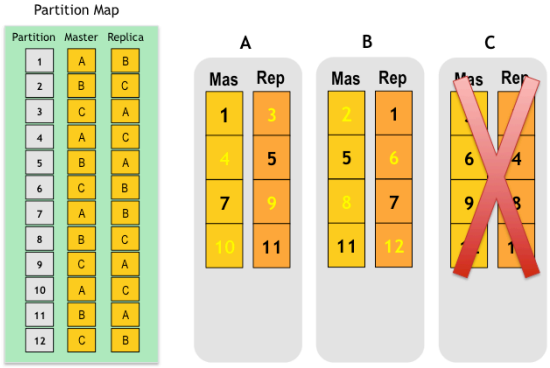
Losing a Node

So what happens if a node dies?



Losing a Node

Some of the partitions will only have a single copy.

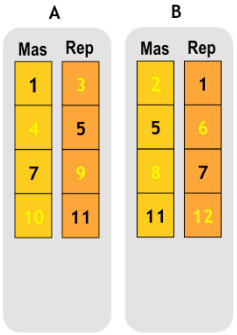


Losing a Node

So the cluster will exclude the missing node and create a new partition map.

New Partition Map

Partition	Master	Replica
1	A	B
2	B	A
3	B	A
4	A	B
5	B	A
6	A	B
7	A	B
8	B	A
9	B	A
10	A	B
11	B	A
12	A	B

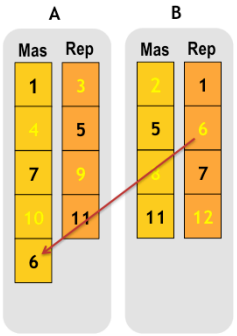


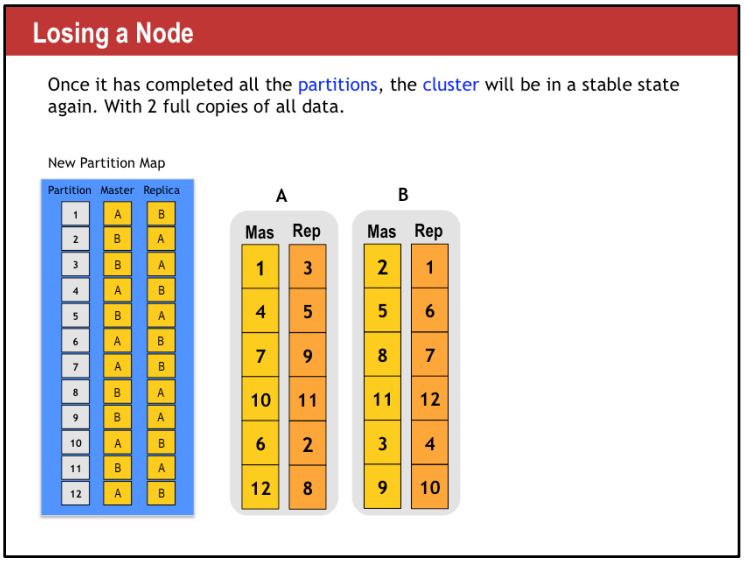
Losing a Node

It will then begin to make copies of all the data

New Partition Map

Partition	Master	Replica
1	A	B
2	B	A
3	B	A
4	A	B
5	B	A
6	A	B
7	A	B
8	B	A
9	B	A
10	A	B
11	B	A
12	A	B





Migrations can be tuned. They will impact performance in some situations, especially for writes, due to default write duplicate resolution.
Aerospike is working on improving and trying to minimize overhead of migrations.
If necessary, migrations can be sped up or slowed down. See: https://discuss.aerospike.com/t/speeding-up-migrations/683 for details.

Adding a Node

Now let's start with the same situation, but add a node this time. The same starting state: 12 partitions, 3 nodes, replication factor of 2.

Partition Map

Partition	Master	Replica
1	A	B
2	B	C
3	C	A
4	A	C
5	B	A
6	C	B
7	A	B
8	B	C
9	C	A
10	A	C
11	B	A
12	C	B

A

Mas	Rep
1	3
4	5
7	9
10	11

B

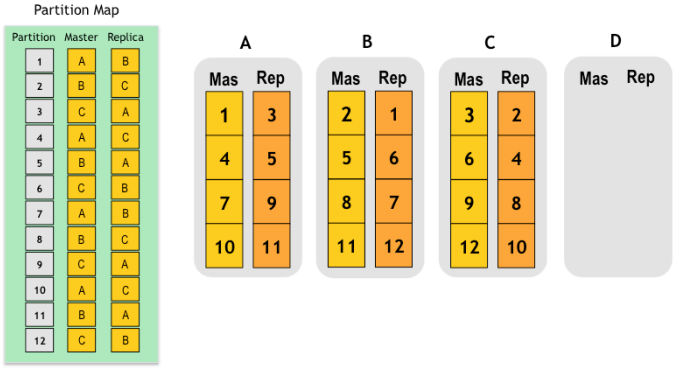
Mas	Rep
2	1
5	6
8	7
11	12

C

Mas	Rep
3	2
6	4
9	8
12	10

Adding a Node

When the new node is added, it starts empty.



Adding a Node

The cluster creates a new partition map, with the new node included.

Partition Map

Partition	Master	Replica
1	A	B
2	B	D
3	C	A
4	D	C
5	B	A
6	C	B
7	A	D
8	D	C
9	C	A
10	A	C
11	B	D
12	D	B

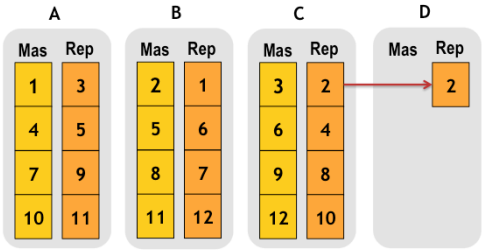
A		B		C		D	
Mas	Rep	Mas	Rep	Mas	Rep	Mas	Rep
1	3	2	1	3	2		
4	5	5	6	6	4		
7	9	8	7	9	8		
10	11	11	12	12	10		

Adding a Node

The cluster will then migrate (rebalance) the partitions, one at a time to the new node. During this time it is possible for the partition map to be out of sync with the actual data distribution. Aerospike nodes will proxy the request.

Partition Map

Partition	Master	Replica
1	A	B
2	B	D
3	C	A
4	D	C
5	B	A
6	C	B
7	A	D
8	D	C
9	C	A
10	A	C
11	B	D
12	D	B

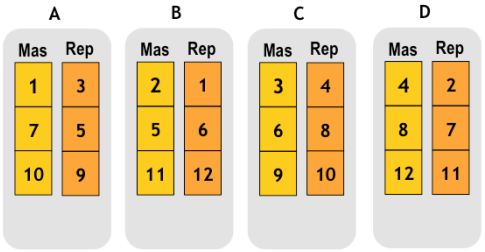


Adding a Node

Once all the partitions have migrated, the database will be in a new stable state, with replicated copies of all data again.

Partition Map

Partition	Master	Replica
1	A	B
2	B	D
3	C	A
4	D	C
5	B	A
6	C	B
7	A	D
8	D	C
9	C	A
10	A	C
11	B	D
12	D	B



Summary

In this module, we covered:

- Data hierarchy.
- High level architecture.
- Data partitioning.

These tasks are ones that you can use in your environments. Doing proper benchmarking will not only help to determine what your performance is, but also useful for finding basic connectivity issues.