Administration & Operations

Storage

AEROSPIKE

Objectives

In this section we will be covering:

- How Aerospike Reads/Writes/Updates Data.
- Defragmentation.
- Data Hygiene.
- Configuring Storage.

How Aerospike Reads/Writes/Updates Data

Data Access Patterns

How does Aerospike handle the following operations

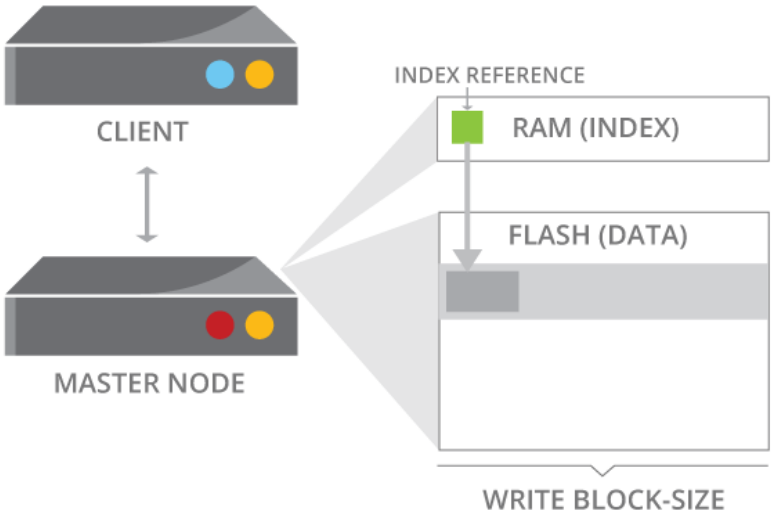
- Read
- Write*
- Update*
- Replace*
- Delete*

* The following slides do not illustrate how replica are written.

Reading data in Flash

The entire record is read from storage into the server RAM, **only the requested Bins** are returned to the client.

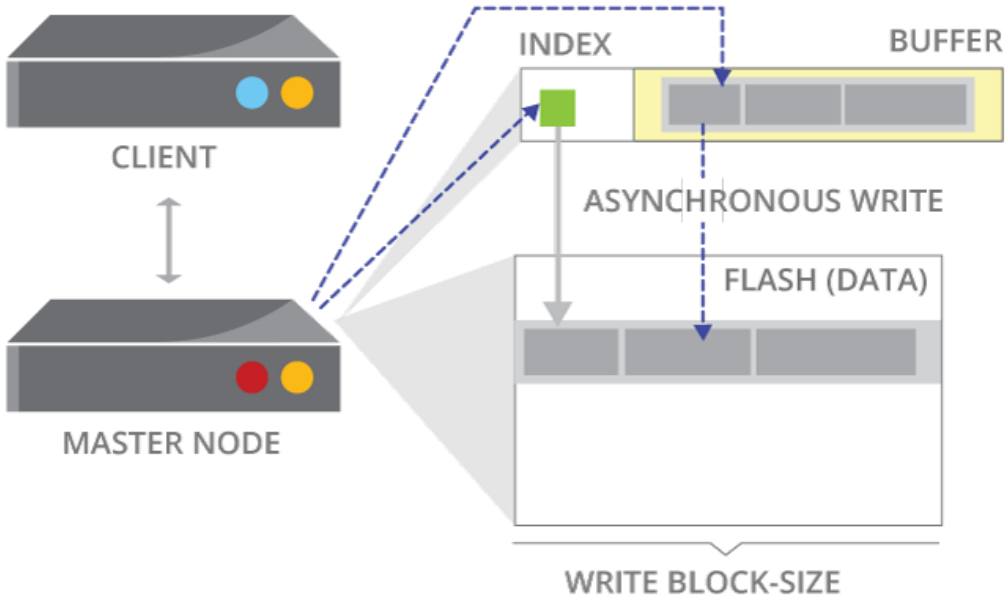
1. Client finds Master Node from partition map.
2. Client makes read request to Master Node.
3. Master node finds data location from index in RAM.
4. Master node reads entire object from flash. This is true even if only reading bin.
5. Master node returns value.



Writing data in Persistent store

Entries are added into the primary and any secondary indexes. The record is placed in a write buffer to be written to the next available block.

1. Client finds Master Node from partition map.
2. Client makes write request to Master Node.
3. Master Node make an entry into index (in RAM) and queues write in temporary write buffer.
4. Master Node coordinates write with replica nodes (not shown).
5. Master Node returns success to client.
6. Master Node asynchronously writes data in blocks.
7. Index in RAM points to location on flash.

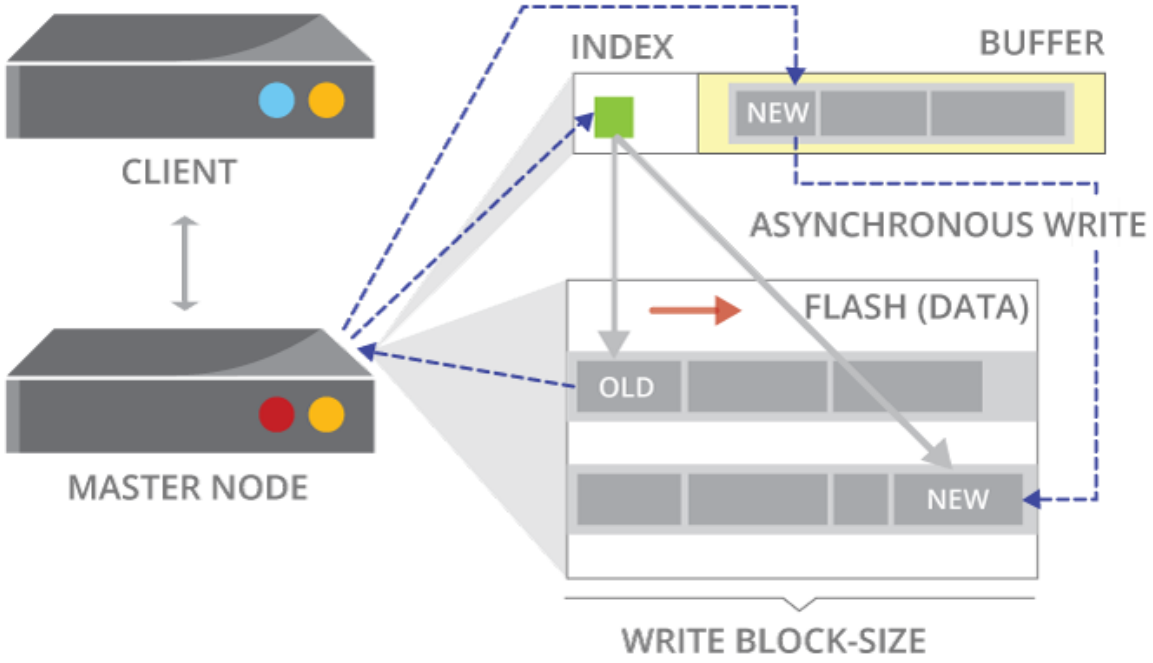


Updating data in Persistent store

The entire record is read in to server RAM, updated as needed, then written to a new block (copy on write).

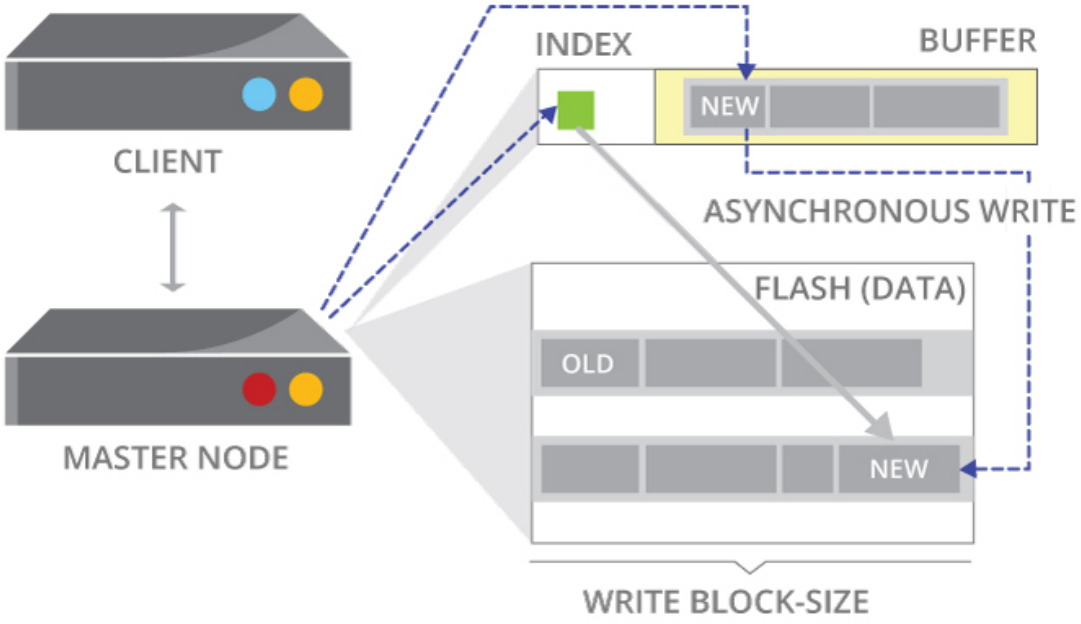
The index(es) point to the new location.

1. Client finds Master Node from partition map.
2. Client makes update request to Master Node.
3. Master Node reads the existing record
4. Master Node queues write of updated record in a temporary write buffer
5. Master Node coordinates write with replica nodes (not shown).
6. Master Node returns success to client.
7. Master Node asynchronously writes data in blocks.
8. Index in RAM points to new location on flash.



Replacing data in Persistent store

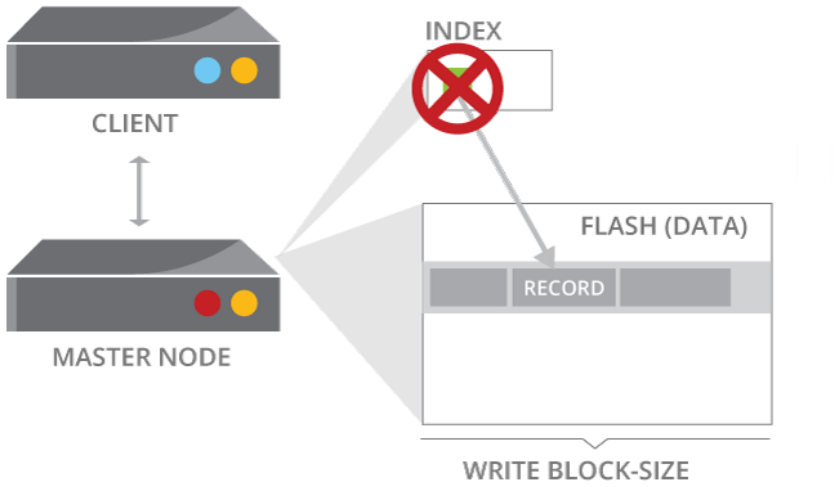
1) Client finds Master Node from partition map.
2) Client makes replace request to Master Node.
3) Master Node queues write of replaced record in a temporary write buffer
4) Master Node coordinates replace with replica nodes (not shown).
5) Master Node returns success to client.
6) Master Node asynchronously writes data in blocks.
7) Index in RAM points to new location on flash.



Deleting data

Deleting a record **removes** the entries from the indexes only. Very fast. The background defragmentation will physically delete the record at a **future** time.

1) Client finds Master Node from partition map.
2) Client makes delete request to Master Node.
3) Master Node coordinates deletion with replica nodes (not shown).
4) Master Node returns success to client.
5) Data is eventually deleted from flash by defragmentation and new data written in the block.



Note: Aerospike 3.10 has introduced “Durable Delete” which allows you to “tombstone” deleted records.

Max write cache

- When a storage device is not keeping up, Aerospike uses cache and will try to keep up until a certain point (until this cache is full) and will then throw a device overload error.
- This cache is configurable through max-write-cache, at the namespace level, and you can dynamically increase it from the default (64M) to a higher multiple of the write-block-size (usually 128KB for SSD devices).

- For example:
- `asinfo -v 'set-config:context=namespace;id=test;max-write-cache=128M'`

- In the logs, look for w-q for a specific device:
- device /dev/sdc: used 296160983424, contig-free 110637M (885103 wblocks), swb-free 16, w-q 0 w-tot 12659541 (43.3/s), defrag-q 0 defrag-tot 11936852 (39.1/s)

Post write queue

- After a write block has been flushed to disk, it is actually kept in memory for faster access on reads.
- 256 such write blocks kept in the “post-write-queue” by default.
- The “post-write-queue” is per device.
- Depending on the use case, this can give huge read latencies improvements.
- Big for XDR, as reads happen right after writes.
- Blocks in the post-write-queue are not eligible to be defragmented.

- Dynamically configurable:

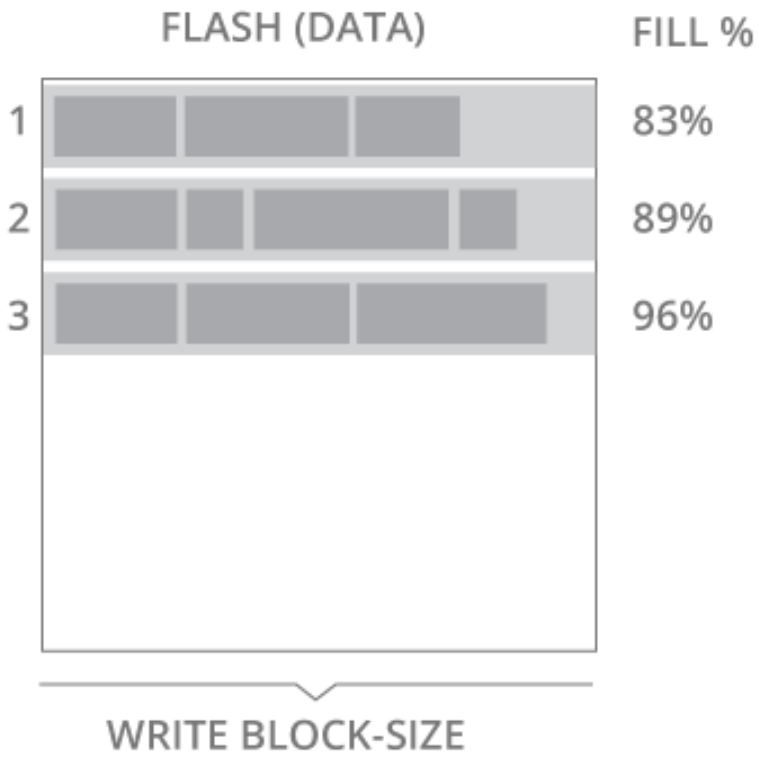
```
asinfo -v 'set-config:context=namespace;id=test;post-write-queue=512'
```

- In the logs, look for cache-read pct for a given namespace:

{test} device-usage: used-bytes 1458377696640 avail-pct 5 cache-read-pct 11.07

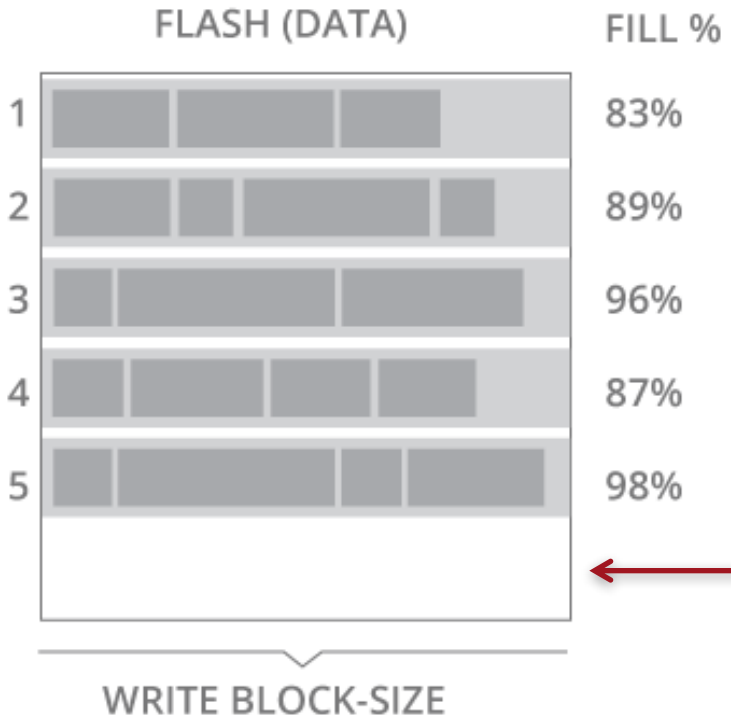
Defragmentation

Records in blocks



Aerospike writes the data in large data blocks.

Log structured writes

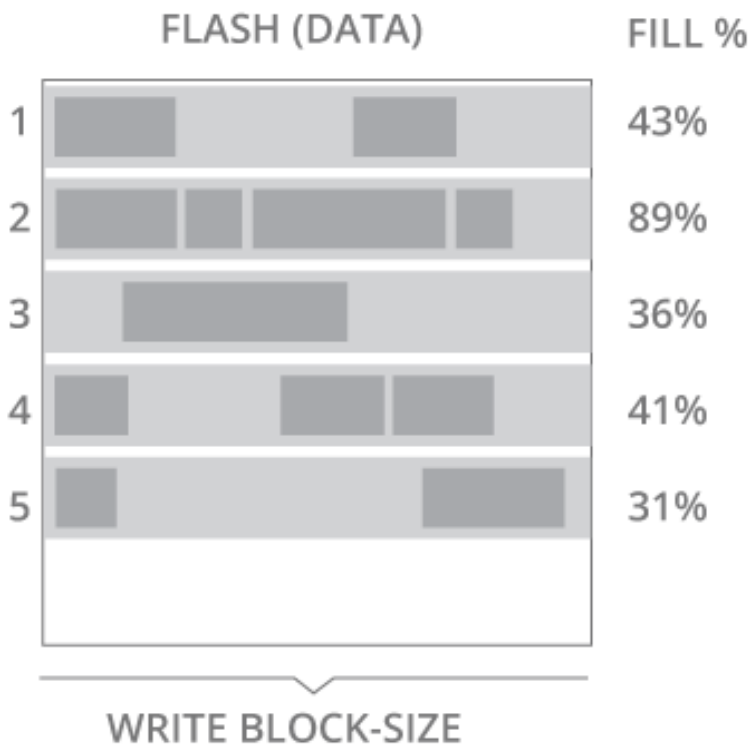


"A log-structured filesystem is a file system in which data and metadata are written sequentially to a circular buffer, called a log."

See(Log structured writes)

As new data is added to the disk, new blocks will be written to the flash device in a circular pattern.

Percent used vs Percent available

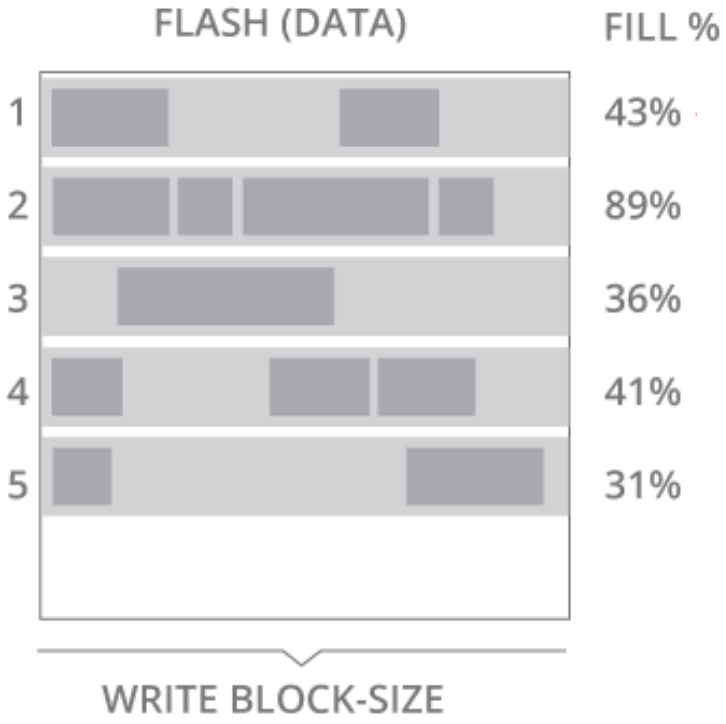


Over time, some records will be deleted or updated, resulting in fragmented usage on the flash/SSD disk.

- **Disk used percent** - The amount of space actually used .
- **Percent available on disk** - The percentage of free blocks (those at 0% filled).

This means it is possible for a flash device to be 50% utilized, but only have 20% available. The rest being taken by unused space in each block.

Defragmentation queue

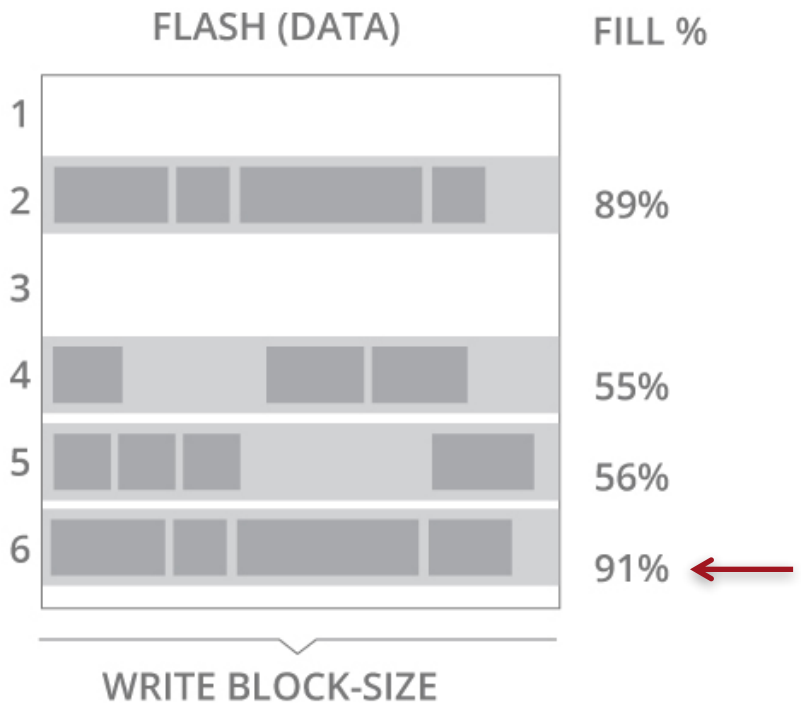


defrag-lwm-pct - Aerospike continuously checks for blocks below this level of use.

In this example, the defrag low watermark is 50%.

Blocks 1, 3, 4 and 5 are below 50% used and will be put on the defrag queue.

Result



The defragmenter runs **constantly**

- Every time period
- For a number of blocks

1. Records read from blocks
2. Put into the write queue.

Best when flash device **< 50%** occupied.

Each device in the namespace has it's own defrag thread.

Data Hygiene

Data Hygiene

Aerospike has a safety net to prevent reaching **maximum capacity**.

- TTL (time-to-live) and Expiration
- High watermark and Eviction
- Read Only (stop-write) Mode

Recommendation: Size your cluster **correctly** for:

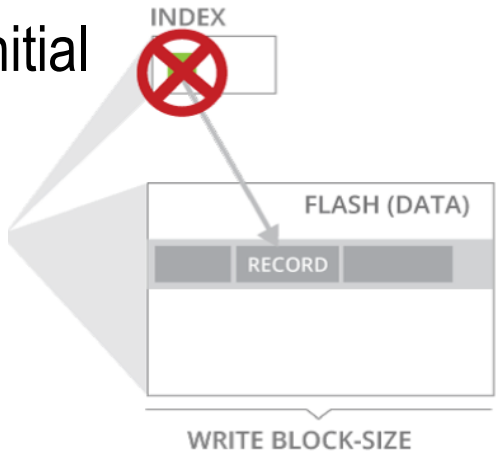
- Throughput
- Capacity

Time-to-live and Expiration

A record can optionally have an expiration time. When the expiration time is reached, the record is marked to be deleted. At some **future** time the record will be **deleted** by the namespace supervisor.

But as soon as the expiration time is reached, it cannot be read (it will be deleted and not returned to the application).

- The namespace contains the default TTL (time-to-live), used if no TTL is provided by the application.
- Every write recalculates a new expiry based on the initial TTL, unless if overwritten of course.



Namespace supervisor - nsup

- Runs sequentially across each namespace.
- Responsible for computing ttl histogram, expiring data, evicting data (when applicable), and going over set-delete (when issued).
- Runs every 2 minutes (120 s) by default.
- Can be dynamically changed:
- `asinfo -v 'set-config:context=service;nsup-period=30'`
- Check how long it takes in the logs:

```
{test} Records: 393851, 0 0-vt, 0(0) expired, 7890(14047) evicted, 0(0) set deletes, 0(0) set evicted. Evict ttls: 2240,2380,0.439. Waits: 0,0,0. Total time: 372 ms
```

- Can take time for larger datasets… Typical to see 30 minutes or longer for ~500 Million records datasets.

TTL configuration – Developers perspective

The default TTL is set in the namespace, it can be changed dynamically.
It can be overridden by the developer on a record by record basis.

Developer TTL Value	Result
0 or [not set]	Uses TTL set in the Config file
> 0	Update the TTL, but disregard the default value and use the given value. This is true even if the default is "0".
-1	Unset the TTL. The record will not expire. The record must be manually deleted.

NOTE: Zero default TTL in the config file means the “live for ever”

Eviction

A high watermark is a threshold for RAM or disk use.

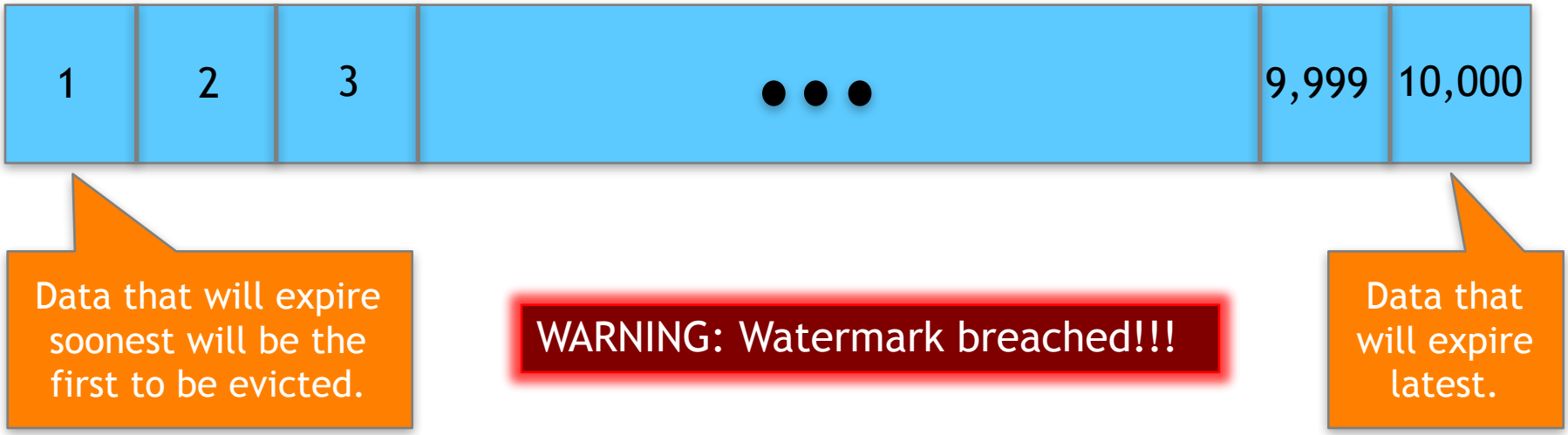
- When breached, the server will begin to evict data.
- Data closest to its TTL will be deleted first.
- This will only happen one of the watermarks (disk or memory) has been breached.

Data is divided into buckets

- Prior to version 3.8, only a fixed 100 buckets
- 3.8+, 10,000 buckets by default
  - Can change up to 10,000,000 with `evict-hist-buckets` parameter.

High Watermarks and Eviction

- Prior to 3.8, data in the lowest bucket that has data could be evicted
- Version 3.8+, all data in buckets under the threshold bucket ***will*** be evicted (whole buckets only).



Important note: The server does not discriminate between the data in the same bucket.

If you have data with wildly different TTLs, such as 100 years and 1 day, the data with the 1 day TTL will all be in the first bucket to expire. So be careful with your management of data.

This is especially true prior to version 3.8

TTL effects

- Records with no TTL are not affected by evictions.

- Here is how to check in the logs for how many records would not expire in a given namespace (look for the 0-vt which is the 0 void time number):

  May 30 2015 14:58:28 GMT: INFO (nsup): (thr_nsup.c::1237) **{test}**
  Records: 400008, **0 0-vt**, 0(0) expired, 2040(2040) evicted, 0(0) set deletes, 0(0) set evicted. Evict ttls: 2574,2717,0.020. Waits: 0,0,0.
  Total time: 386 ms

Read-Only Mode

Aerospike has some thresholds that will place the server in read-only mode.

stop-write-pct (for RAM)

When RAM usage goes above this level (90% by default), the server will be in a stop-write mode. Aerospike always uses RAM for indexes, even when using SSDs to store data, so this is always an active parameter.

min-avail-pct (for Flash/SSD)

When you are using SSDs to store data, the SSD must have at least 5% of the blocks available in order to properly operate. Do not lower this to less than 5% (the default).

Configuring Storage

Aerospike Configuration File

The main Aerospike configuration file contains all the configuration variables for a node.

- **Located** on each node at:
  - `/etc/aerospike/aerospike.conf`
- **Not centrally** manage.
- Most variables can be **changed dynamically**.
- **Persistent** changes - edit the configuration file
- **Shorthand** for large numbers (K, M, G)
  - For example 4 gigabytes can be represented as 4G, which is mathematically 4*1024*1024*1024.

Configuration File

There are 7 major contexts in an Aerospike configuration file.

- service (required, covered in Configuring-Main)
- network (required, covered in Configuring-Main)
- namespace (at least 1 required)

Example:

```
namespace <NAMESPACE NAME> {
        ...
}
```

Check the configuration reference manual:

http://www.aerospike.com/docs/reference/configuration/

Special Note

Parameters that are most commonly problematic are denoted in RED. Pay special attention to these, since the ramifications of improperly setting these variables may take months to show up or be difficult to fix once set.

Namespace Configuration Parameters

Each namespace must be configured with 2 sets of variables:

- Common namespace variables.
- Storage variables
  - RAM (with or without HDD for persistence)
  - Flash/SSD

Namespace Parameters: Common Parameters

Some of the parameters for namespaces are common for all types.

Parameters covered:

- Replication factor
- RAM and flash high watermarks
- RAM allocation
- Time-to-live (TTL)

Replication Factor

Description	The total (master + replica) copies of data in the database cluster.
Context location	namespace
Config parameters (defaults)	replication-factor
Notes	Some databases refer to the replication factor as being only the number of copies, without the master. Aerospike refers to it as the total number of copies.
Change dynamically	No
Best practices	For almost all use cases, the best replication factor is “2”. “1” would not give any replication and means that the loss of a node means a loss of data. “3” or more would mean you would need additional storage to accommodate the additional copies.

Storage Watermarks

Description	Aerospike has built in a set of watermarks that trigger actions meant to act as safety valves.
Context location	namespace
Config parameters (defaults)	high-water-memory-pct high-water-disk-pct stop-writes-pct
Notes	Care should be taken to understand how these work. When the high-water-memory-pct or high-water-disk-pct is reached, the server will begin to evict records closest to their expiry. If either RAM or disk should reach the stop-writes-pct, the server will no longer accept write requests.
Change dynamically	Yes

Best practices	• Recommended settings are:
	  • **high-water-memory-pct 60**
	  • **high-water-disk-pct 50**
	  • **stop-writes-pct 90**
	• You should always account for changes in node could within the cluster. So consider what happens if you were to lose a node.
	• These settings are dynamically changeable, if you do want to increase these values temporarily, makes sure to take the appropriate corrective action.
	• In order to properly **defragment** Flash/SSD namespaces, keep the **high-water-disk at 50**.

Stop Writes

Description	When RAM usage hits this more than this level (90% by default), the database will stop writing new data. The server will respond with an error to new write requests.
Context location	namespace
Config parameters (defaults)	stop-writes-pct (90)
Notes	RAM is always used for indexes. If you are using a RAM based namespace, it will also include the data.
Change dynamically	Yes
Best practices	The default is 90. Use the default. The only time you may want to increase this is if you must do so temporarily. **Be very careful about increasing this to greater than 90.**

Minimum Available Percent (for Flash/SSD)

Description	The min-avail-pct sets the minimum percent of free blocks that must be available for new writes. Otherwise the database will be in a read-only mode.
Context location	namespace
Config parameters (defaults)	min-avail-pct (5)
Notes	The percentage is not technically based on the percentage of "used" Flash/SSD space. It is based on the number of free blocks. The database has background operations that automatically free blocks, but the process that handles this will lag a little. If write levels are very high, the values may be out of sync.
Change dynamically	Yes
Best practices	Use the default of 5%. **Do not set to below 5 permanently.**

RAM Allocation

Description	Maximum RAM available to the namespace. All namespaces require RAM for the indices, and optionally for Data.
Context location	namespace
Config parameters (defaults)	memory-size (4G)
Notes	Aerospike does **not allocate** all of this memory **immediately**. The minimum size for this is 128 MB. **Primary index** uses exactly **64 bytes** of memory for each record, multiplied by the replication factor. A RAM namespace, all data and indices will be stored in RAM. Aerospike **does not cache** data.
Change dynamically	Yes
Best practices	Node failure causes **RAM usage to increase** in each remaining node. The RAM configured should take into account the high water marks for memory.

Default TTL

Description	Default time-to-live (in seconds) for a record from the time of creation or last update. The record will expire in the system beyond this time.
Context location	namespace
Config parameters (defaults)	default-ttl (2592000 / 30 days)
Notes	0 means do not expire (lives forever). Can be overriden via API. 0 over API, means use namespace configured default on the server side. -1 via API means do not expire (use 0 on the server).
Change dynamically	Yes
Best practices	Try to keep ttl of records within a namespaces as homogeneous as possible to fully take advantage of the eviction mechanism. Following suffixes can be used: S (Seconds), M (Minutes), H (Hour), D (Day). **Be careful, M is for Minutes NOT Months**

Namespace Configuration: Common Parameters

The general configuration parameters for every namespace are:

```
namespace test_namespace {
    replication-factor 2
    high-water-memory-pct 60   # Not in default config file
    high-water-disk-pct 50     # Not in default config file
    stop-writes-pct 90         # Not in default config file
    min-avail-pct 5            # Not in default config file
    memory-size 4G
    default-ttl 30d            # Default 30 days expiration
    ...
}
```

Storage Engine

RAM (No Persistence)

Description	Sets how data will be stored.
Context location	namespace
Config parameters (defaults)	storage-engine memory
Notes	Aerospike does not cache. So all data (index + data) will be stored in RAM.
Change dynamically	No
Best practices	Be sure to take into consideration what will happen if you lose one or more nodes. See the Storage Watermark section above.

RAM (No Persistence) - Configuration

The general configuration parameters for every namespace are:

```
namespace test_namespace {
    replication-factor 2
    high-water-memory-pct 60   # Not in default config file
    high-water-disk-pct 50     # Not in default config file
    stop-writes-pct 90         # Not in default config file
    min-avail-pct 5            # Not in default config file
    memory-size 4G
    default-ttl 30d            # Default 30 days expiration
    storage-engine memory
}
```

RAM + HDD (Persistence device)

Description	Sets how data will be stored.
Context location	namespace
Config parameters (defaults)	storage-engine device
Notes	Aerospike does not cache. So all data (index + data) will be stored in RAM. Data is committed in RAM to all replica before returning to the client, but is written asynchronously to the drive. This data is not intended to be human readable. The drive will only be used for persistence and will only be used when starting up the node. The server will use this to rebuild the index in RAM.
Change dynamically	No
Best practices	Be sure to take into consideration what will happen if you lose one or more nodes. See the Storage Watermark section above.

RAM + HDD (Persistence File)

Description	These are the settings for the persistence file
Context location	namespace:storage-engine
Config parameters (defaults)	**File** filesize
Notes	When using a file, you specify the path of the file. When using the device, you specify the device (such as "/dev/sdb1") of the disk or partition. You must also specify the filesize. When using device, the server assumes it will use the entire disk, so please make sure there is no data on this partition.
Change dynamically	No
Best practices	• Aerospike recommends setting the size of the persistence file at 4x the amount of RAM allocated in the "memory-size" of the namespace. • You may use more than one file or device, but the server will balance between them.

Store Data In Memory

Description	Tells the server to store data in memory, rather than on storage.
Context location	namespace:storage-engine
Config parameters (defaults)	data-in-memory true
Notes	When set to “true”, the server will store data in RAM and use disk only for persistence. If set to “false”, the server will only store the index in RAM and use the disk for data storage.
Change dynamically	No
Note	Data is not stored in Shared Memory, therefore it will not FastRestart

RAM + HDD - Configuration

The configuration parameters for RAM + HDD are:

```
namespace RAM_persist_namespace {
    replication-factor 2
    high-water-memory-pct 60      # Not in default config file
    high-water-disk-pct 50        # Not in default config file
    stop-writes-pct 90            # Not in default config file
    min-avail-pct 5               # Not in default config file
    memory-size 4G
    default-ttl 30d               # Default 30 days expiration
    storage-engine device {
        file /opt/aerospike/data/test.data
        filesize 16G
        data-in-memory true
    }
}
```

RAM + Flash/SSD

Description	Sets how data will be stored.
Context location	namespace
Config parameters (defaults)	storage-engine device
Notes	RAM for indices only and Flash/SSD for data.
Change dynamically	No
Best practices	• Be sure to take into consideration what will happen if you lose one or more nodes. See the Storage Watermark section above. • Flash/SSDs should be no more than 50% full to allow for efficient defragmentation. Usage at above this rate is fine for short periods of time.

Flash/SSD Devices

Description	These are the settings for the Flash/SSD devices
Context location	namespace:storage-engine
Config parameters (defaults)	device
Notes	You must specify the device (such as /dev/sdb) of the Flash/SSD.
Change dynamically	No
Best practices	• You may use more than one device, the server will balance between them. Do not use heterogeneously sized devices, you will be locked at the smallest device's size. • Partition size currently limited to 2TiB • Partitioning may give better performance – dependent on device. • The device order in the config file is important. Devices should be added at the end of the list. If you replace a device, do not reorder the config entries.

Store Data In Memory

Description	Tells the server to store data in memory, rather than on storage.
Context location	namespace:storage-engine
Config parameters (defaults)	data-in-memory false
Change dynamically	No
Best practices	• In principle it is possible to use Flash/SSD as persistence only. However, most modern rotational hard drives are fast enough so that using Flash/SSD is not necessary. • The drives must be cleared or “dd”ed prior to use in the database. • Be sure to test the drives in the servers using the Aerospike ACT test tool (http://github.com/aerospike/act/).

Write Block Size

Description	Tells the server what block size to use when writing data.
Context location	namespace:storage-engine
Config parameters (defaults)	write-block-size
Notes	The value for this must be a multiple of 128 KB. Maximum supported value is 1 MB. No single record can be larger than this block size, so size this according to the maximum size in the database.
Change dynamically	No
Best practices	Size these according to the largest size of any object in your database. Test your workload against your device with different values to find the optimum one for performance.
Advanced Note	The write-block-size has an impact on how extra memory will be used by the post-write-queue. Default is 256 write blocks per device. Block size is a factor in performance, benchmark based on workload.

Flash/SSD Configuration

The configuration parameters for Flash/SSD

```
namespace ssd_namespace {
    replication-factor 2
    high-water-memory-pct 60       # Not in default config file
    high-water-disk-pct 50         # Not in default config file
    stop-writes-pct 90             # Not in default config file
    min-avail-pct 5                # Not in default config file
    memory-size 4G
    default-ttl 30d
    storage-engine device {
        device /dev/sdb
        device /dev/sdc
        data-in-memory false
        write-block-size 128K
    }
}
```

Flash vs RAM Configuration:

Comparing the 2 typical configurations

RAM + HDD

```
namespace RAM_persist_namespace {
     replication-factor 2
     high-water-memory-pct 60
     high-water-disk-pct 50
     stop-writes-pct 90
     min-avail-pct 5
     memory-size 4G
     default-ttl 30d
     storage-engine device {
          file /opt/aerospike/data/test.data
          filesize 16G
          data-in-memory true
     }
}
```

Flash/SSD

```
namespace ssd_namespace {
     replication-factor 2
     high-water-memory-pct 60
     high-water-disk-pct 50
     stop-writes-pct 90
     min-avail-pct 5
     memory-size 4G
     default-ttl 30d
     storage-engine device {
          device /dev/sdb
          device /dev/sdc
          data-in-memory false
          write-block-size 128K
     }
}
```

Special Note: Single Bin Optimization

If your data model is classic key-value, you can configure a “Single Bin” workspace.

This will store the data more compactly. To turn it on, simply add the parameter “single-bin” as below.

```
namespace sample_namespace {
    replication-factor 2
    memory-size 4G
    default-ttl 30d
    single-bin true
}
```

All writes are “replaces”

Special Note: Data In Index

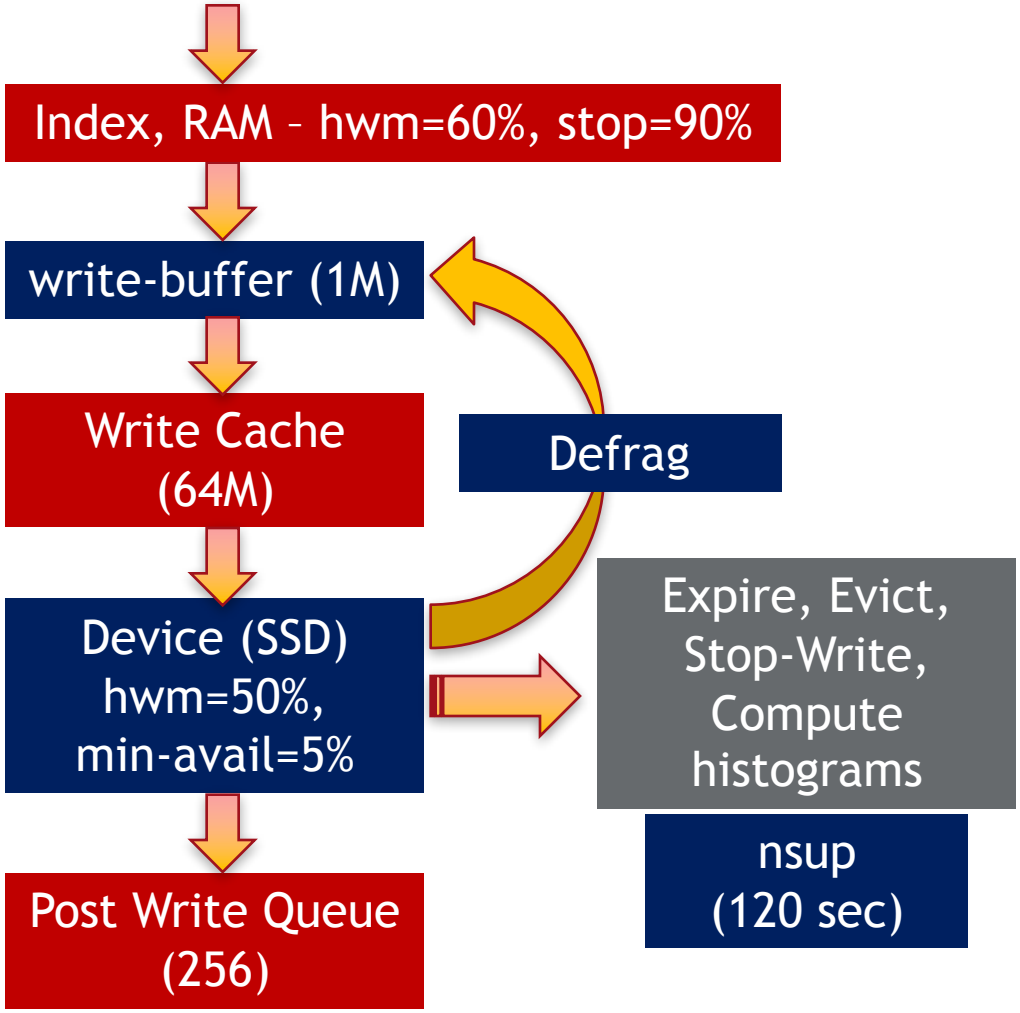
If your data is single-valued and an integer, you can store the data in the index only. Also use the single-bin option.

```
namespace sample_namespace {
    replication-factor 2
    memory-size 4G
    default-ttl 30d
    data-in-index true
    single-bin true
    storage-engine device {
        file /opt/aerospike/data/test.data
        filesize 16G
        data-in-memory true
    }

}
```

Key Config. Parameters for Storage Management

max-write-cache *(ns, 64M, dyn)*
write-block-size *(ns, 1M, static)*
Defrag:
defrag-lwm-pct *(ns, 50, dyn)*
defrag-sleep *(ns, 1000,dyn)*
post-write-queue *(ns, 256, dyn)*
Evict Records:
high-water-disk-pct *(ns, 50, dyn)*
high-water-memory-pct *(ns, 60, dyn)*
evict-hist-buckets *(ns, 10000, dyn)*
evict-tenths-pct *(ns, 5, dyn)*
Stop Writes:
stop-writes-pct *(ns, 90, dyn)*
min-avail-pct *(ns, 5, dyn)*
nsup-period *(service, 120, dyn)*



1-Cannot defrag records in Post Write Queue
2-Can only evict records till ttl bucket so # < evict-tenths-pct of total.
3-Defrag needs free space on Device to run.
Use TPS, Workload and Record size to tune. **Goal - Never get to eviction.**

nsup and defrag threads

- Each node has its own single nsup thread. nsup may launch other worker threads.
- Within a node, one nsup thread serves each namespace in sequence.
- Each device has its own defrag thread
- Defrag applies the same to both ssd and disk storage.
- When storing in pure memory, data is stored on the heap and free()'d right away on delete.
- OS heap manager will coalesce adjacent free memory.

Summary

What we have covered:

- How Aerospike Reads/Writes/Updates Data.
- Defragmentation.
- Data Hygiene.
- Configuring Storage.