

AEROSPIKE

---

## Configuration: Storage

---

## Objectives

In this section we will be covering:

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

A

## How Aerospike Reads/Writes/Updates Data

## Data Access Patterns

How does Aerospike handle the following operations

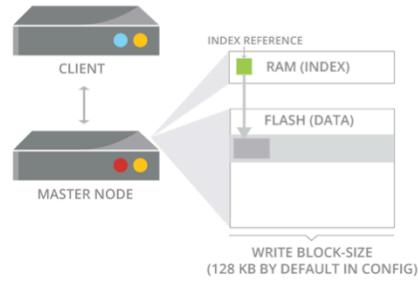
- Read
- Write\*
- Update\*
- Replace\*
- Delete\*

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

## Accessing An Object In Aerospike

### Reading A Standard Data Type With Flash

- 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.

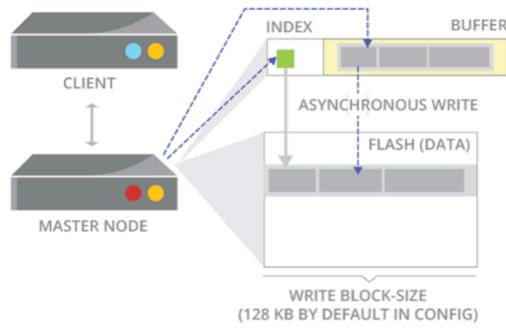


In this case, if the master node is down, the client may connect to a replica.

## Accessing An Object In Aerospike

### Writing A New Standard Data Type Record With Flash

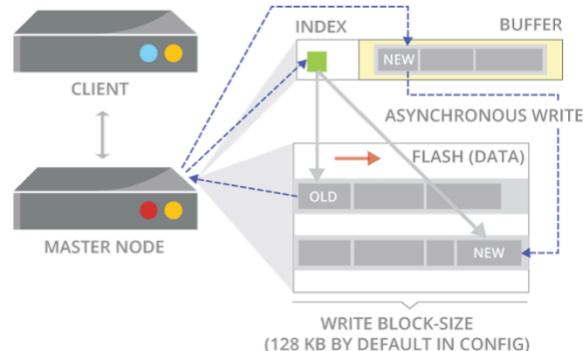
- 1) Client finds Master Node from partition map.
- 2) Client makes write request to Master Node.
- 3) Master Node make an entry into index (in RAM) and queues write in temporary write buffer.
- 4) Master Node coordinates write with replica nodes (not shown).
- 5) Master Node returns success to client.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



## Accessing An Object In Aerospike

Updating A Standard Data Type Record With Flash

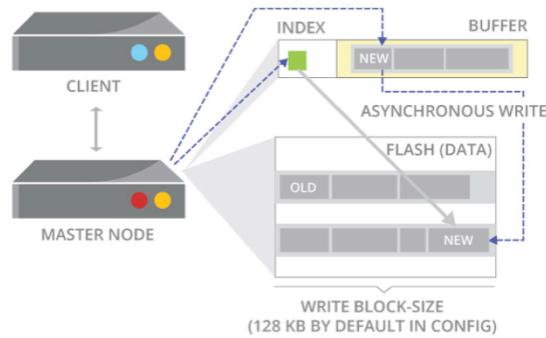
- 1) Client finds Master Node from partition map.
- 2) Client makes update request to Master Node.
- 3) Master Node reads the existing record
- 4) Master Node queues write of updated record in a temporary write buffer
- 5) Master Node coordinates write with replica nodes (not shown).
- 6) Master Node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



## Accessing An Object In Aerospike

Replacing A Standard Data Type Record With Flash

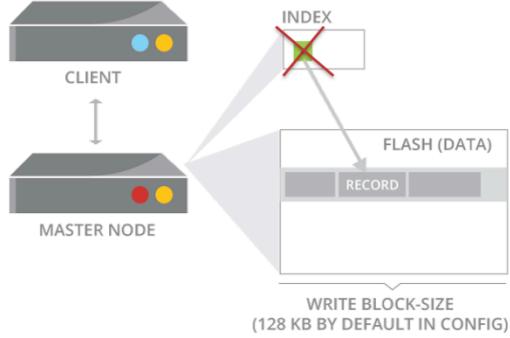
- 1) Client finds Master Node from partition map.
- 2) Client makes replace request to Master Node.
- 3) Master Node queues write of replaced record in a temporary write buffer
- 4) Master Node coordinates replace with replica nodes (not shown).
- 5) Master Node returns success to client.
- 6) Master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



## Accessing An Object In Aerospike

### Deleting A Standard Data Type Record With Flash

- 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.

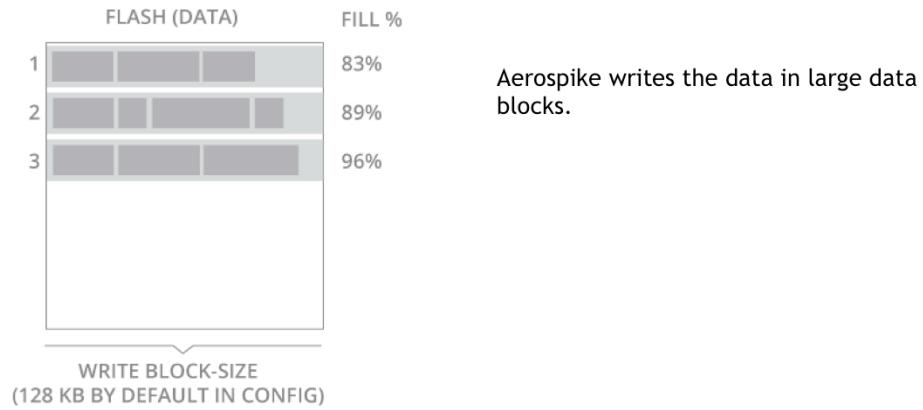


A

Defragmentation

## Accessing An Object In Aerospike

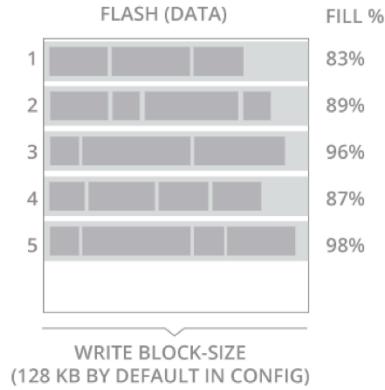
### How Space Is Freed Up



As each buffer gets filled, it will write the block onto Flash.

## Accessing An Object In Aerospike

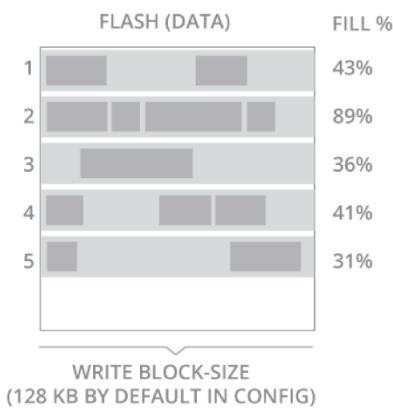
### How Space Is Freed Up



As new data is added to the disk, new blocks will be continually written to the flash device.

## Accessing An Object In Aerospike

### How Space Is Freed Up



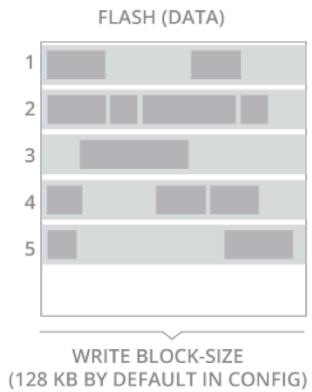
Over time, some records will be deleted or updated, resulting in fragmented usage on the flash/SSD disk. This unused space must be freed up.

The amount of space actually used is called the disk used percent. However the percent available on disk is actually represented by 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.

## Accessing An Object In Aerospike

### How Space Is Freed Up

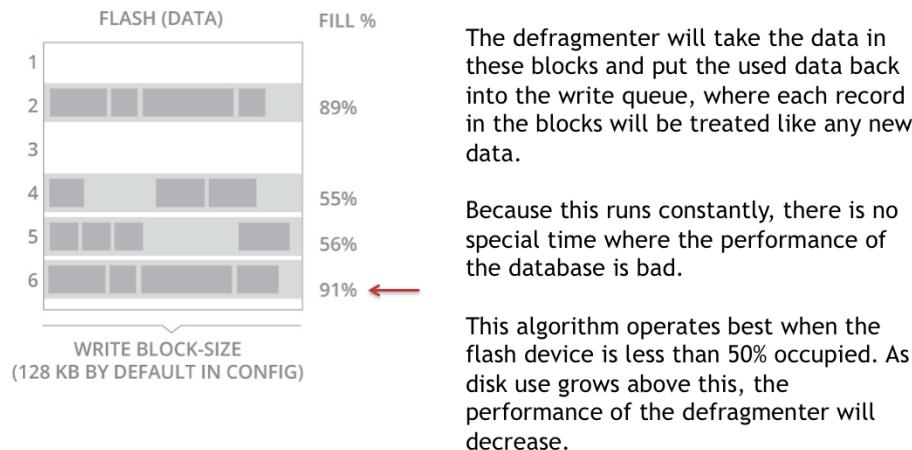


Some databases use a nightly process called “compaction,” which is an intensive process. Aerospike continuously (as writes are happening) checks for blocks below some level of use (called the [defrag-lwm-pct](#)).

In this example, if the defrag low watermark is 50%, blocks 1 and 3 to the left are below 50% used (as well as blocks 4 and 5) and will be put on the defrag queue.

## Accessing An Object In Aerospike

### How Space Is Freed Up



A

Data Hygiene

## Data Hygiene

Every database has a maximum capacity. Actually hitting this maximum capacity may result in lost data. Aerospike provides you with a few tools to manage the behavior of the database to prevent this.

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

In most cases the DBA is responsible for tracking the policies on what happens when space gets tight. Each of these are policies that will help you control the use of space within your database.

Ideally, you have sized your namespace correctly, and either the data is naturally expiring while data usage is still within the namespace's limits or the records are set to not expire and also stay within the namespace's limits. But you may have a burst of writes, or just under estimated the avg. record size or the number of records the namespace would hold. Aerospike has some built in mechanisms to handle such situations.

## TTL and Expiration

Aerospike has a mechanism for automatically expiring data if it has not been changed in a while.

In the configuration of the namespace you can set a default TTL (time-to-live). When a record has reached its TTL limit, it will be automatically deleted from the database.

Correct use of the TTL is imperative for the health of the database.

Expiration is often used when there is a policy for the data.

## TTL and Expiration

This works using the following logic:

- When a record is first written, the server will take the current time and add the TTL. The server will use the default TTL as configured on the server or, if supplied, an overriding TTL from the client.
- If the record is updated the TTL will be extended as if it were a new write. You can override this from the client.
- A “touch” from the client has the same impact, but does not change the data.
- Reads do not affect the expiration time.
- When the expiration time has been reached a background job will delete the data (nsup thread).

It is very important to remember that the logic for expiring data is based on writes.

If a value is very popularly read, but not written, it will still expire.

## TTL and Expiration

Although there is a default TTL value, the application can always override it. The exact behavior depends on the new TTL value.

Client TTL Value	Result
0 or [not set]	Update the TTL with the current default value for the namespace. This is normally set in the aerospike.conf configuration file, but can be dynamically changed. Be sure to check the value on the server.
> 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.

The exact behavior of how TTLs are set is important to understand.

Most importantly, if you decide that you do not want a TTL, the database will not evict data.

## High Watermarks and Eviction

A high watermark is a threshold for RAM or disk use. When the watermark has been breached, the server will begin to evict data. It will start by evicting data closest to its TTL. This can be thought of as an advanced expiration. **This will only happen one of the watermarks (disk or memory) has been breached.**

Records will be grouped by buckets. Eviction will expire first all records in the first bucket, then will move on to the second one, etc... There are 100 buckets. The "width" of each bucket is **MaxTTL/100** where MaxTTL: ttl of record with longest ttl in the namespace.



Important note: If you have data with wildly different TTLs, such as 100 days and 5 minutes, the data with the 5 minute TTL will all be in the first bucket to expire. The server does not discriminate between the data in the same bucket. So be careful with your management of data.

The high watermark is an optional safety parameter often used with Aerospike.

In essence you are setting what the value for "full" is for RAM or disk. The method for evicting data was chosen to delete the data thought to be least valuable.

If you choose to not use these watermarks, the administrator is responsible for maintaining the data volume.

The server puts all data into 100 bins of time. Any data in the bin nearest to its void time will be evicted until enough room has been cleared or the server will move to the next bin.

For example:

A namespace has a 100 day TTL. It will automatically expire data that is closest to expiration, the 99 day bin.

## Read-Only Mode

In order to prevent data loss, 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).**

This would happen in the following situations:

- Data with no expiration, but keep growing. (So no evictions).
- Evictions not keeping up. (Database not able to evict data fast enough).
- Defragmentation not keeping up.

A

## Configuring Storage

## 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)

These will look like this:

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

Check the configuration reference manual:

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

## Aerospike Configuration File Notes

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

- Located at `/etc/aerospike/aerospike.conf` on each node.
- NOT centrally managed by Aerospike.
- Most variables can be changed dynamically while the Aerospike node is up.
- If you wish for changes to the file to be persistent, you must edit the configuration file manually.
- You may choose to use shorthand (K, M, G) to represent large numbers. For example 4 gigabytes can be represented as 4G, which is mathematically  $4 \times 1024 \times 1024 \times 1024$ .

### 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)

Each of these parameters will exist for all namespaces. While the values may not exist in the configuration file, there are default values.

## 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)	<code>high-water-memory-pct</code> <code>high-water-disk-pct</code> <code>stop-writes-pct</code>
Notes	These features do not exist in most other databases, so some care should be taken to understand how these work. If either the high-water-memory-pct or high-water-disk-pct, the server will begin to evict data closest to its time-to-live (TTL). If the either RAM or disk should exceed the stop-writes-pct, the server will no longer accept write requests for new records (note that it will accept updates to existing records).
Change dynamically	Yes
Best practices	<ul style="list-style-type: none"><li>Recommended settings are:<ul style="list-style-type: none"><li>high-water-memory-pct 60</li><li>high-water-disk-pct 50</li><li>stop-writes-pct 90</li></ul></li><li>You should always account for changes in node could within the cluster. So consider what happens if you were to lose a node.</li><li>These settings are dynamically changeable, if you do want to increase these values temporarily, makes sure to take the appropriate corrective action.</li><li>In order to properly defragment Flash/SSD namespaces, keep the high-water-disk at 50.</li></ul>

## Stop Writes

Description	When the database 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	For most cases the default of 90 is fine. The only time you may want to increase this is if you must do so temporarily. <b>Be very careful about increasing this to greater than 90.</b>

## Minimum Available Percent (for Flash/SSD)

Description	If you are using a Flash/SSD namespace, you may also run out of space on your drives. 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	For most cases the default of 5 is fine. Decreasing this should only be done if you need the space temporarily. <b>Do not set to below 5 permanently.</b>

## RAM Allocation

Description	All namespaces (whether RAM or Flash/SSD) require RAM for at least the index. This controls how much RAM is allocated.
Context location	namespace
Config parameters (defaults)	<code>memory-size (4G)</code>
Notes	Aerospike does not grab all of this memory when the node starts up. The minimum size for this is 128 MB. Aerospike uses exactly 64 bytes of memory for each record. This will be multiplied by the replication factor. If you are using a RAM namespace, all data will also be stored in RAM. Aerospike does not cache data.
Change dynamically	Yes
Best practices	Always be aware that the amount of RAM in use will increase if the cluster loses a node. The amount of memory set here should also take into account the high water marks for memory above.

## 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)	<b>default-ttl</b> (2592000 / 30 days)
Notes	0 means do not expire (lives forever). Can be overridden 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  
    ...  
}
```

There are some variables we cover here that will not be in the default config file. What is shown above are the values that the database will use as defaults if they are not in the file.

While you may want to adjust the settings for the memory high water mark, it is highly recommended to not adjust the settings for disk.

## Data Storage In RAM (No persistence)

RAM namespaces with no persistence stores all data and indexes in RAM.

The configuration parameters are:

- Replication factor\*
- Watermarks\*
- RAM allocation\*
- Time-to-live (TTL)\*
- Storage Engine

\*See the Data Storage Common Parameters above.

The only additional parameter you need to set for this is the storage engine.

## Storage Engine for 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.

## 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 memory  
}
```

Some of these are not in the default config file, but the shown values are the default if they are not specified.

## Data Storage In RAM + HDD

RAM namespaces with persistence (RAM +HDD) stores all data and indexes in RAM, but stores the data on disk. Note that although we say “HDD,” you may also use Flash (SSDs).

The configuration parameters are:

- All the values from Data Storage in RAM above
- Storage Engine
- Persistence File
- Whether or not to store data in memory

## Storage Engine for RAM + HDD (With Persistence)

Description	Sets how data will be stored.
Context location	namespace
Config parameters (defaults)	storage-engine device
Notes	<p>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.</p> <p>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.</p>
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)	<b>file or device</b> <b>filesize</b>
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	<ul style="list-style-type: none"><li>Aerospike recommends setting the size of the persistence file at 4x the amount of RAM allocated in the “memory-size” of the namespace.</li><li>You may use more than one file or device, but the server will balance between them.</li></ul>

## 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
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
Best practices	<ul style="list-style-type: none"><li>In principle it is possible to use rotational disk to store the data and use RAM only for the index. Aerospike has found that performance with this configuration is very poor. Aerospike does not recommend or support this configuration.</li></ul>

## Namespace Configuration: RAM + HDD

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
    }
}
```

## Data Storage In Flash/SSD

Flash/SSD namespaces store primary indexes in RAM and all data on Flash/SSD.

The configuration parameters are:

- Common Data Storage Parameters
- Storage Engine
- Flash/SSD Devices
- Whether or not to store data in memory

\*See the Data Storage Common Parameters above.

## Storage Engine for Flash/SSD

Description	Sets how data will be stored.
Context location	namespace
Config parameters (defaults)	storage-engine device
Notes	The server will use RAM for index only and Flash/SSD for data.
Change dynamically	No
Best practices	<ul style="list-style-type: none"><li>• Be sure to take into consideration what will happen if you lose one or more nodes. See the Storage Watermark section above.</li><li>• 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.</li></ul>

## 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	<ul style="list-style-type: none"><li>• You may use more than one file or device, but the server will balance between them. Do not use heterogeneously sized devices.</li><li>• You may use SATA, SAS, or PCIe devices.</li></ul>

## 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
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
Best practices	<ul style="list-style-type: none"><li>• 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.</li><li>• The drives must be cleared or “dd”ed prior to use in the database.</li><li>• Be sure to test the drives in the servers using the Aerospike ACT test tool (<a href="http://github.com/aerospike/act/">http://github.com/aerospike/act/</a>).</li></ul>

## 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.

You can increase the block size, but cannot decrease the block size.

## Namespace Configuration: Flash/SSD

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
    }
}
```

## Namespace 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 you have certain kinds of data that belongs in a single bin, you can turn on the single bin optimizations. 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  
}
```

## 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
    }
}
```

Using data in index means you will only need 64 bytes of RAM for every record (multiplied by the replication factor) for every record in the cluster.

If you use the single-bin option and persistence and are using the Enterprise Edition, you can also use Faststart. This means that the index and the included data will be stored in shared memory. If you do not power down, you can restart the database, which will reconnect to the shared memory and be able to get going without having to read data from persistence.

## Summary

What we have covered:

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