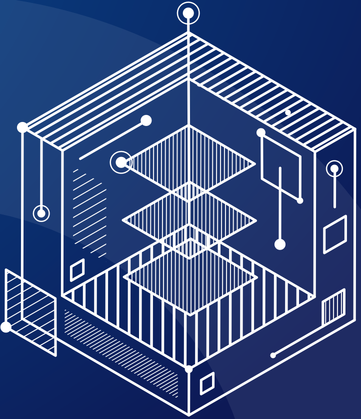
CHECKSUM

TECHNICHAL CAPABILITY SHEET

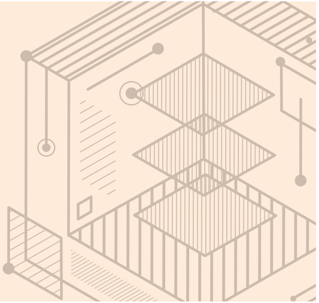


open DRIVES

Capability Definition

To ensure data integrity, Atlas uses a robust system of checksums to detect data corruption. A checksum value is derived by analyzing a larger block of data and creating a unique value based on the contents (also called a hash value). Between the time the data is written and when it is accessed, the value should not change. Atlas compares checksums created when data is written to checksums created when accessed to make sure they match exactly. Any difference in checksum values indicates data has somehow been corrupted — usually either through a data transfer error or bit-rot degradation of the data on disk. In the event we detect a data integrity error, Atlas will automatically initiate a recovery process to regain the uncorrupted data.

Note: Many different algorithms are available within Atlas for calculating checksums. The default setting is using the Fletcher-4 algorithm, a lightweight approach to error detection that does not require a level of resources that would affect storage performance.



Outcome & Value

Atlas' checksum capabilities sustain a high level of data integrity. Though highly unlikely, data could be corrupted during transmission through the I/O pipeline, or it could succumb to bit rot over time (depending on operating conditions). Bit rot is a result of some form of deterioration of the information stored on physical media, such as HDDs. This typically occurs over time, not through a catastrophic failure of the entire disk. By fully checksumming all data that enters our file system's I/O pipeline (ZIO pipeline), when written to or read from disk, Atlas is able to track data integrity levels and immediately launch remediation efforts upon a detected mismatch.

How It Works

Atlas uses the Fletcher algorithm to compute checksums in order to verify the integrity of data written to disk, read from disk, and/or transmitted across a network through an I/O pipeline. While other checksum algorithms are available as custom settings, the Fletcher algorithm is the default and recommended method because the Fletcher computation method provides greatly improved performance, especially with general-purpose processors.

When data enters the ZIO pipeline whether from SMB, NFS, or even the internal operating system, Atlas immediately performs a checksum operation to calculate a hash value. The same checksum process occurs when writing data to disk. For each transactional group being stored, Atlas calculates a checksum that accompanies both the group ID and data payload. When data is then read from disk, Atlas again creates a checksum that it compares to the previously calculated checksum value. If the two checksums do not perfectly match, we can assume a data integrity error.

OpenDrives uses the phrase "fully checksummed" to indicate instances when Atlas calculates a checksum for information already checksummed, thereby indicating recursive layers of data integrity preservation. Essentially, everything associated with data that's written to storage media, including checksums and parity bits, then has its own checksum value calculated within a tree-like structure. "Fully checksumming," therefore, describes our process of using a secondary checksummed roll-up of all checksums, across all data, to enhance data integrity.

Checksums are spread across volumes, much like the data itself, to to avoid writing both data payloads and corresponding checksums to the same physical disk. Data and checksums transact in parallel from different disks, rather than serially from a single disk, to drive latency reduction.

Automated data correction occurs on read operations only. Atlas reconstitutes data to match the calculated data checksum from either the parity bits (of which two exist for each written transaction) or from a recursive process if the parity option is not available or functional. In the latter case, we perform an automated bi-weekly scrub to verify and then roll up all checksums from disk to the file system.

Characteristics

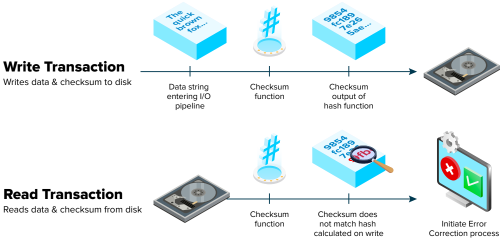
The checksum capability has the following characteristics:

- Ensures a high level of data integrity for data traversing the ZIO pipeline on write and read.
- Initiates in-flight data recovery when a checksum mismatch occurs at the moment data is being accessed.
- Leverages a default Fletcher algorithm, which creates a high level of data integrity without the potentially significant performance issues that could arise with other algorithms like SHA-256.
- Allows for different protection levels on the physical disks that make up the file system (e.g. double parity means multiple drives can fail across multiple chassis without risk of data loss).
- Automated scrubs occur every other week to verify and roll up all checksums.
- Atlas also checksums the checksum values and parity bits for multiple layers of redundancy.

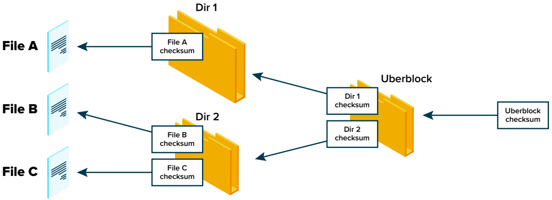
Further Reading

Checksum and snapshots capabilities are closely related. Please refer to the Snapshot Capability Sheet which explains in more detail the mechanics behind the way in which Atlas Core creates system snapshots for data integrity and recovery.

The following diagram illustrates the basic process of creating checksum values for data transactions that can then be compared for data integrity—data errors can be assumed when the checksum of data upon write and then read are not precisely the same:


Write Transaction
Writes data & checksum to disk
The quick brown fox...
9854 fc189 7e26 5ae...
Data string entering I/O pipeline
Checksum function
Checksum output of hash function
Read Transaction
Reads data & checksum from disk
9854 fc189 7e
Checksum function
Checksum does not match hash calculated on write
Initiate Error Correction process

To ensure data integrity further, Atlas creates a tree structure in which checksums themselves are checksummed, what is called a merkle tree allowing for very rapid validation of data integrity across large quantities of data. The following diagram illustrates a simplified version of this tree structure in which multiple recursive checksums are calculated along the entirety of the tree branch:


Dir 1
File A
File A checksum
Uberblock
Dir 1 checksum
Dir 2 checksum
Uberblock checksum
File B
Dir 2
File B checksum
File C
File C checksum