

Final project of “Distributed Systems” course

AdriaBOX

**A Decentralized Storage Matrix with Synchronous
Pipeline Replication and Zero-Knowledge
Encryption**



Sajmir Buzi
sajmir.buzi@studio.unibo.it



Massimiliano Pupillo
massimiliano.pupillo@studio.unibo.it



Alma Mater Studiorum - Università di Bologna

The big “Why?” of AdriaBOX.

1. Why Decentralized Storage Matrix?

- **What it is:** Separates HTTP control metadata from raw TCP storage workers across 12 autonomous node domains.
- **Why:** Eliminates the central server bandwidth bottleneck, preventing Single Points of Failure and enabling horizontal capacity scaling.

2. Why Synchronous Pipeline Replication?

- **What it is:** Slices large files into **64MB logical chunks**, then streams them in a daisy-chain cascade across 3 target nodes via **4KB micro-buffers** before committing write ACKs.
- **Why:** Guarantees strong sequential consistency (CP System), enforces O(1) RAM footprint, and saves 66% of client bandwidth by offloading replication to internal node pipes.

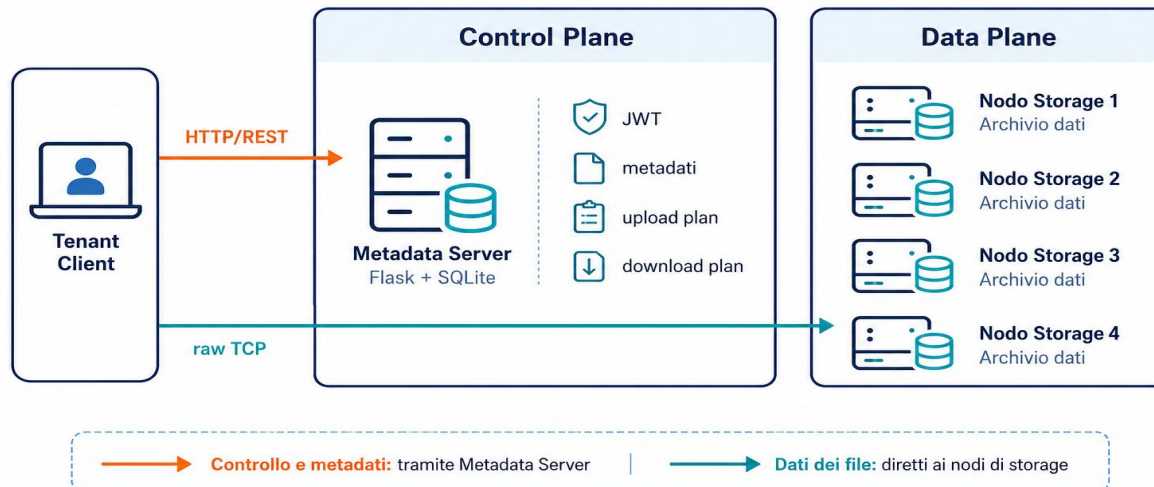
2. Why Zero-Knowledge Encryption?

- **What it is:** Derives 256-bit AES-256-CTR keys locally via PBKDF2 (100k iterations) using tenant password and salt.
- **Why:** Enforces absolute data privacy; plaintext files and decryption keys never touch the network or cloud infrastructure, keeping the server completely blind.

Architectural Separation & CAP Theorem Strategy

Control Plane vs. Data Plane Decoupling

- **Control Plane (Flask REST API):** Metadata orchestration, JWT auth, and storage planning; lightweight & stateless (zero binary file data flows through the master).
- **Data Plane (Raw TCP Fleet):** Decentralized 12-node worker fleet dedicated exclusively to high-throughput binary transfers.



CAP Theorem Position: Strict CP System (Consistency & Partition Tolerance)

- **What CAP Means:** Under a network partition (P), a distributed system must choose between absolute data correctness (C) or uninterrupted availability (A).
- **Our Trade-off Choice:** We strictly prioritize **Strong Sequential Consistency** over uncoordinated uptime.
- **Why (Practical Example):** In cloud storage, returning a corrupted or partially written file just to maintain "high availability" is catastrophic. It is far better to abort a transaction than to expose inconsistent or corrupted data to users.
- **Authoritative Control & Atomic Abort:** All metadata is validated by a single authoritative Master; if a replication pipeline breaks midway, the write transaction aborts instantly to prevent split-brain or orphan blocks.

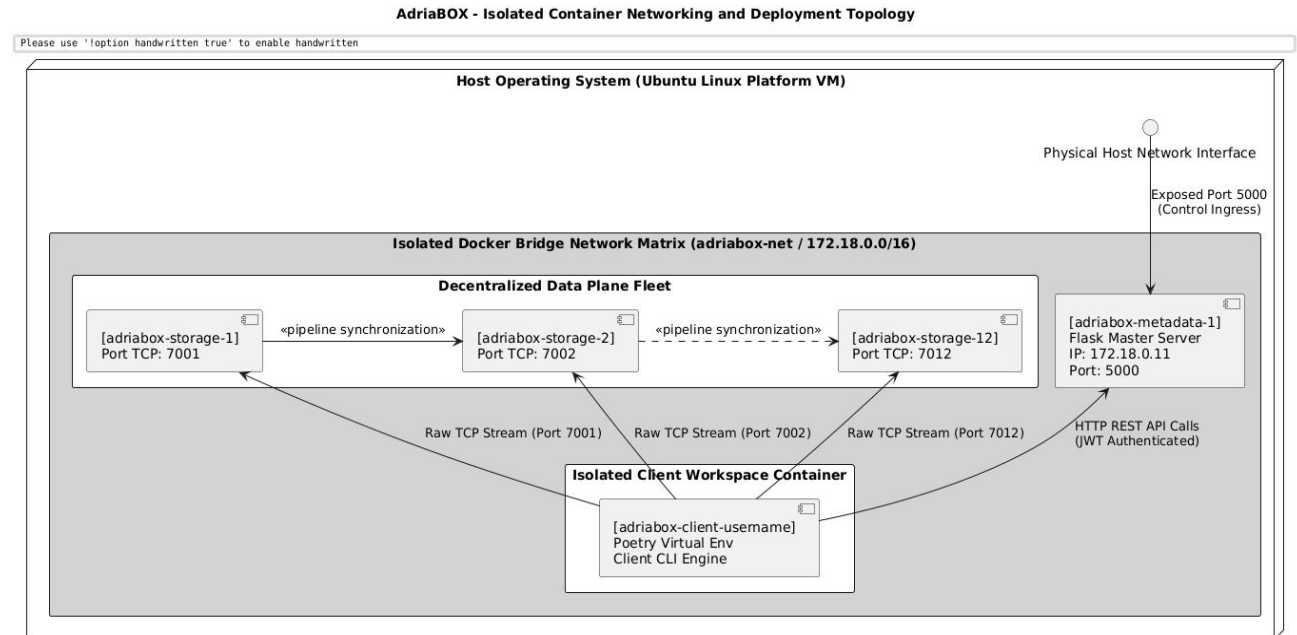
Real-World Multi-Container Orchestration (Dockerization)

Simulated Heterogeneous Environment:

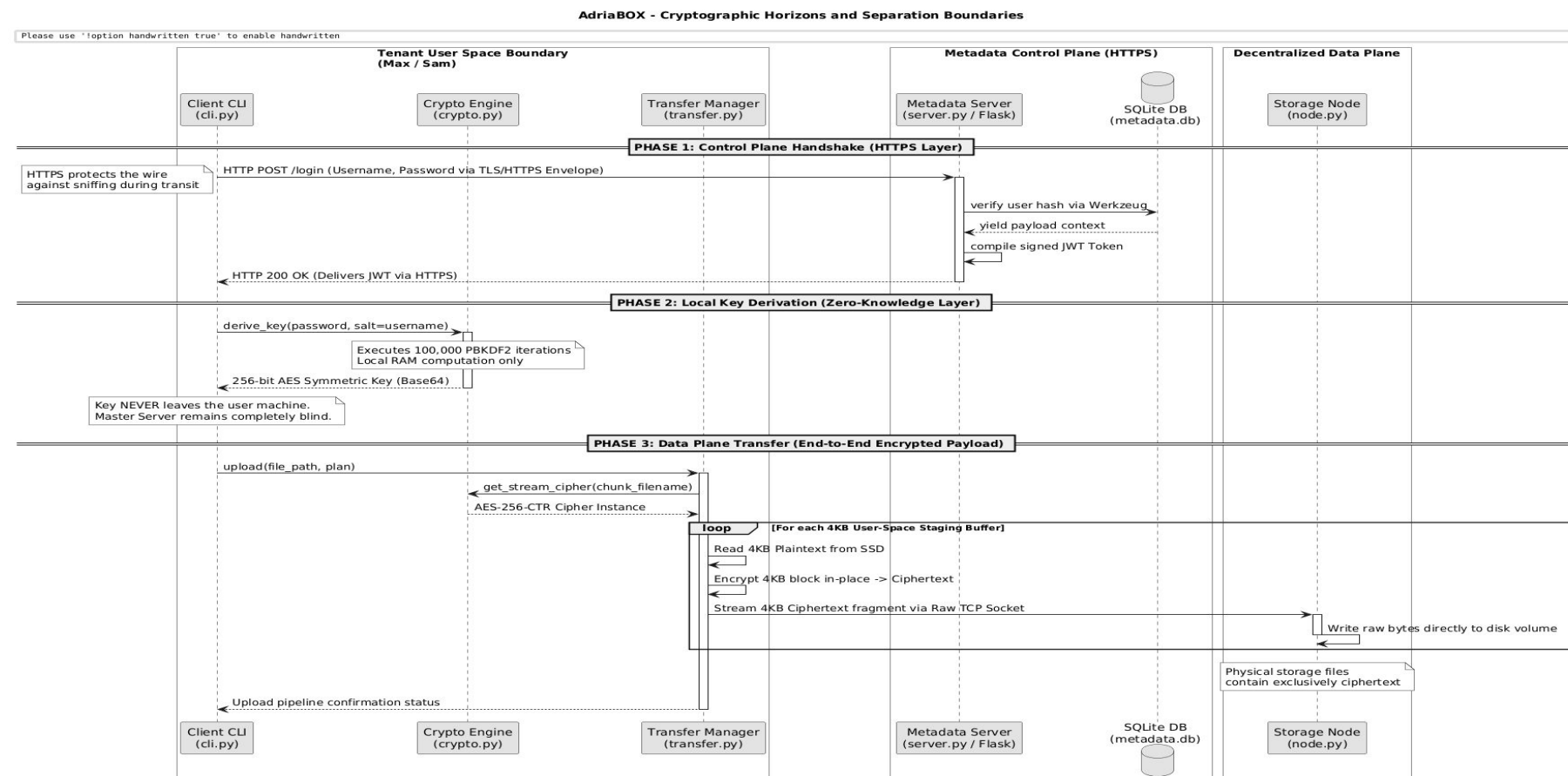
- Fully isolated environments using **Docker Containers**: 1 Master + 12 Storage Nodes + Client Workspaces.
- Virtualized Layer-2 Bridge Network (**adriabox-net 172.18.0.0/16**) simulating geographically distributed hardware.

Admin Out-of-Band Elevation (Zero-Trust Security):

- No public API endpoint allows role promotion.
- Privilege elevation to **Admin** occurs strictly **out-of-band** via direct SQLite database queries on the server container.



Data Isolation & Zero-Knowledge Client-Side Encryption



1 - Dual-Layer Protection:

- Control Plane: Protected via standard **TLS/HTTPS** against wire sniffing.
- Data Plane: End-to-End **Zero-Knowledge Encryption**.

2 - Local Key Derivation (PBKDF2HMAC):

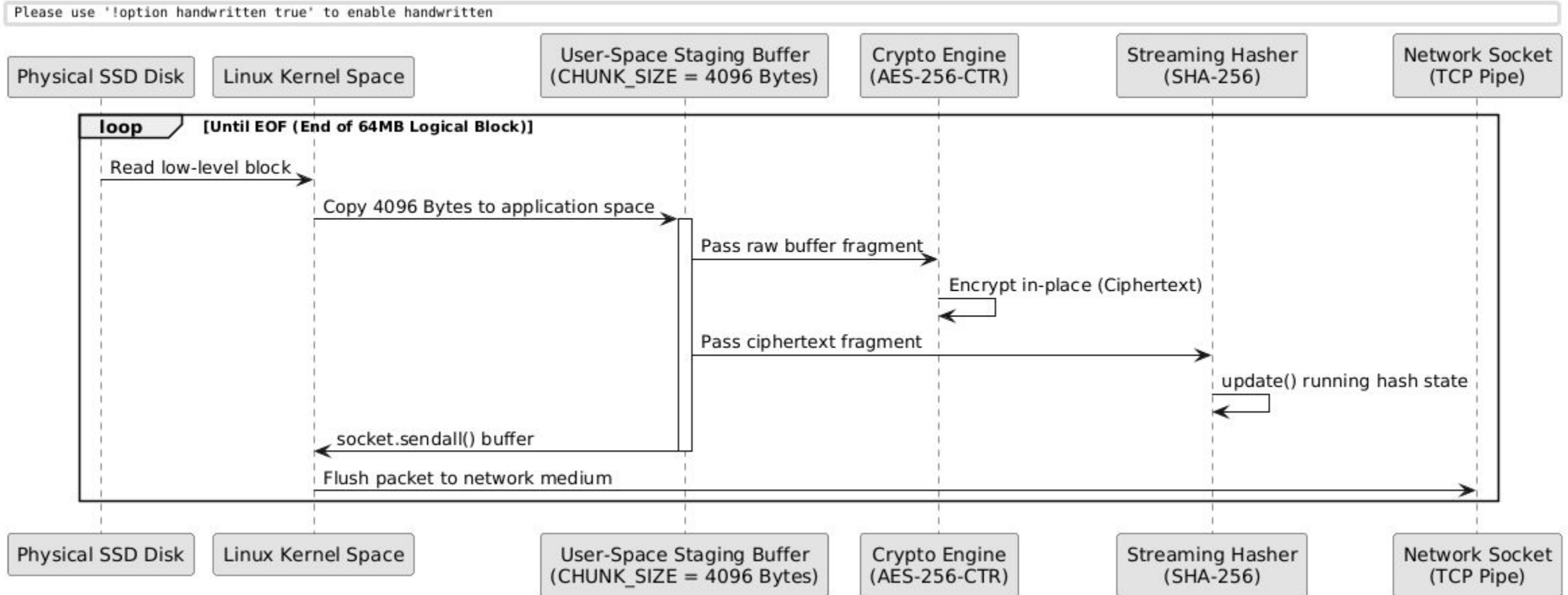
- Password derived locally with 100,000 iterations using **username** as salt. Keys never touch the network.

3 - In-Place Stream Ciphering (AES-256-CTR):

- Per-chunk unique Initialization Vectors (Nonces) derived via SHA-256 filename digests. Data stored on nodes is 100% unreadable ciphertext.

Memory Footprint Optimization & 4KB Buffered Streaming

User-Space Buffered Streaming, Online Encryption, and Hashing Pipeline Engine



1 - The RAM Exhaustion Bottleneck:

- Ciphering and transmitting large assets (e.g., 20 GB) in a single memory block triggers Out-Of-Memory (OOM) kernel panics on resource-constrained devices.

2 - The 4KB Staging Buffer Architecture:

- Fixed **4096-Byte user-space staging buffer** matching the native Linux Kernel Physical Page Size.
- *In-Place Processing Pipeline:* SSD (DMA) -> 4KB RAM Buffer -> AES-256-CTR Mutation (CPU) -> SHA-256 Hasher -> TCP Socket (DMA).

3 - Guaranteed O(1) MemoryFootprint:

- RAM footprint remains static at O(1) complexity (~4 KB buffer space) regardless of asset size (10 MB or 20 GB).

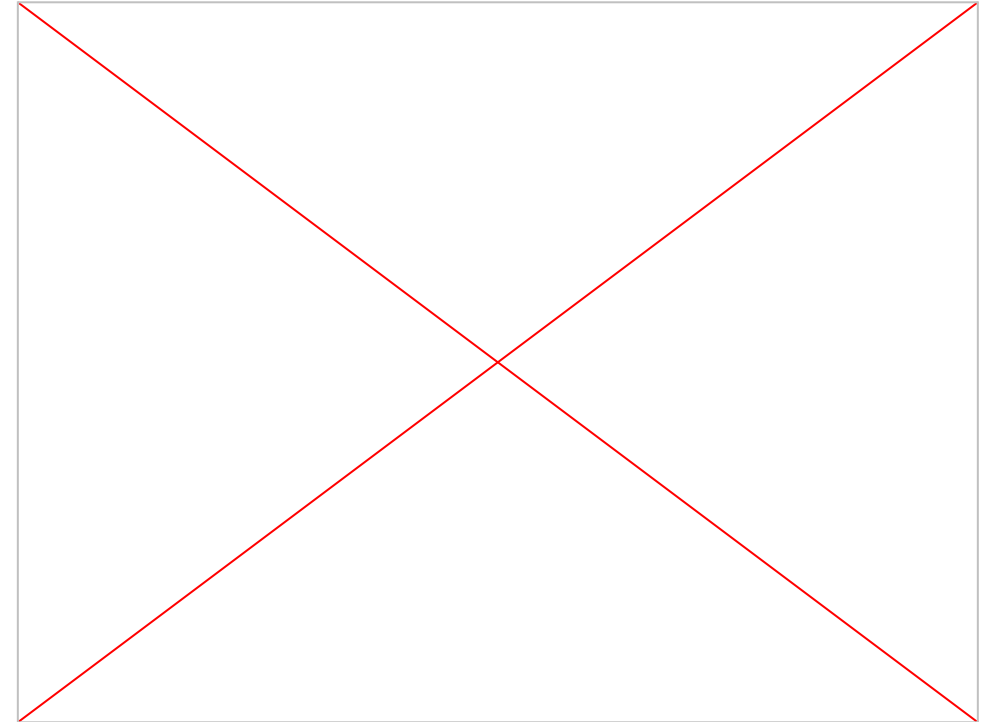
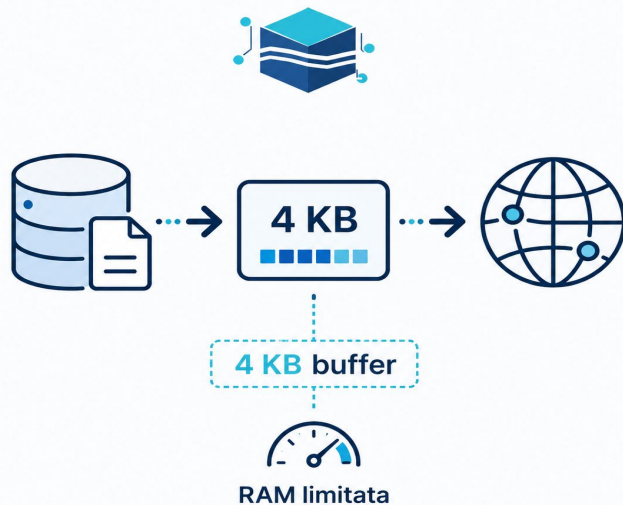
Memory Footprint Optimization & 4KB Buffered Streaming (video demo)

Test Case Parameters:

- Volumetric file asset: **5.0 Gigabytes** (5GB.zip).
- Test execution target: Isolated `adriabox-client-max` container.

Observed Metrics during Active Upload:

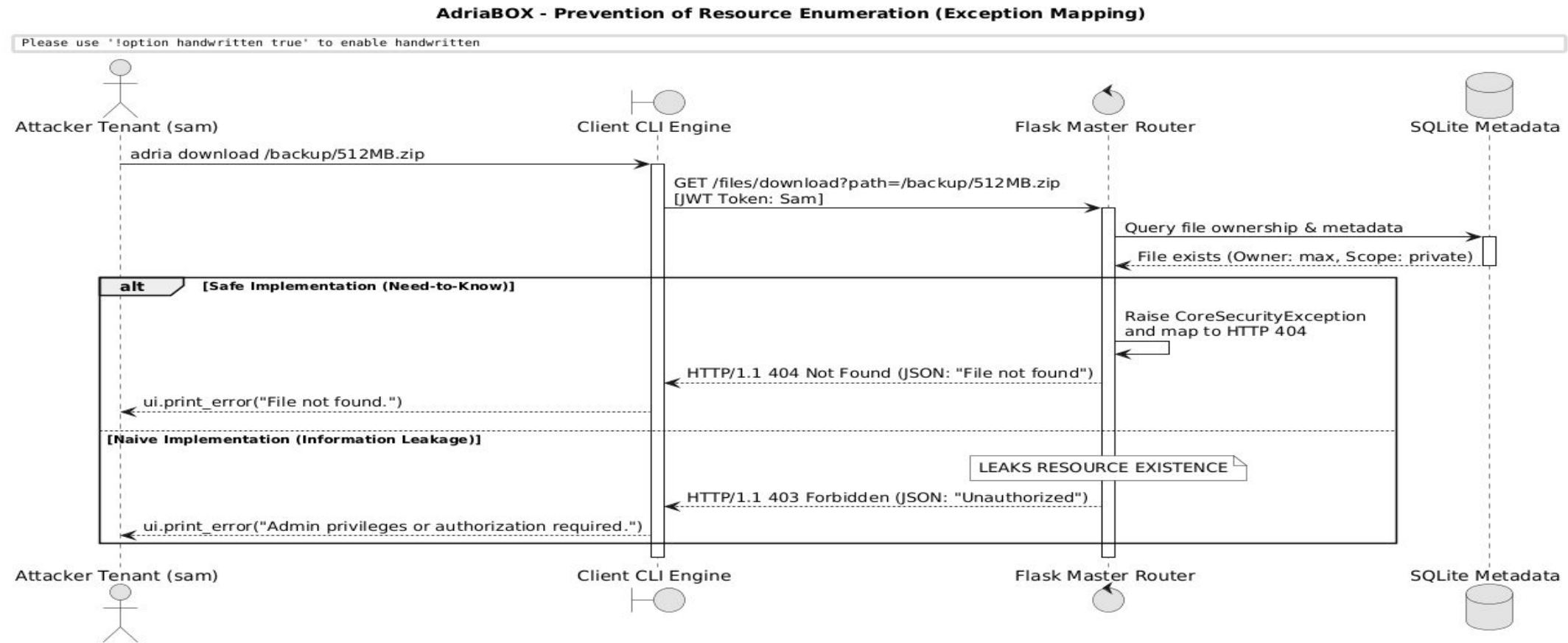
- **File Capacity:** 5,120 Megabytes (80 x 64MB logical chunks).
- **Client RAM Usage:** Flat and bounded at **< 60MB** total process memory.
- **Memory Exhaustion Risk:** 0% (Zero memory leaks or page swapping).



Theory vs. Practice Synthesis:

- While the core cryptographic pipeline operates on a strict **4KB staging buffer**, the total ~100MB footprint accounts for Python interpreter overhead, Rich UI rendering, and Linux Kernel TCP socket buffers.
- **Invariance Boundary:** Memory usage remains strictly bounded and flat at ~100 MB whether uploading 5 GB or 500 GB, completely eliminating Out-Of-Memory (OOM) risks.

Privacy Mitigation: Resource Enumeration Shielding



1 - Cross-Tenant Metadata Isolation:

- Multi-tenant separation enforced via strict relational SQL scoping (`WHERE owner_id = tenant_id`).

2 - The 403 Forbidden Leakage Threat:

- Returning **403 Forbidden** when an unauthorized user requests a file confirms the asset exists, enabling resource enumeration attacks.

3 - Opaque Path Hiding Interception:

- Custom `CoreSecurityException` interceptors catch unauthorized accesses and map them to a generic **HTTP 404 Not Found** ("File not found").

Custom TCP Protocol

1 - 64 MB Logical Chunk Slicing:

- **Architectural Rationale:** Inspired by Google File System (GFS) and HDFS standards to minimize Master database index overhead and optimize I/O throughput.
- **Slicing Ceiling Function:** Files exceeding 64 MB are mathematically partitioned into $N = \max(1, \text{FileSize} / 64\text{MB})$ logical fragments.

Custom 5-Region Binary Framing Protocol:

- Overcomes raw TCP stream boundary limitations: **OpCode (1B) | Meta Length (4B Big-Endian) | Payload Size (4B Big-Endian) | Meta Envelope (JSON) | Raw Payload.**

Campo Header	Dimensione	Tipo / Formato	Descrizione Funzionale
Op Code	1 Byte	Unsigned Char	Specifica il comando (es. WRITE_CHUNK, READ_CHUNK, REPLICATE).
Meta Length	4 Byte	Big-Endian uint32	Lunghezza espressa in byte dell'header JSON dei metadati.
Payload Size	4 Byte	Big-Endian uint32	Dimensione esatta del payload binario da trasferire.
JSON Metadata	Variabile	UTF-8 String	Dati di controllo (Chunk ID, Auth Token, Checksum SHA-256).
Binary Payload	Variabile	Raw Bytes	I byte effettivi del file cifrato trasferiti in streaming.

Chunk Allocation, Scheduling & Metadata Tracking



Multilevel Scheduling & Placement Strategy

- **Chained Round-Robin Rotation:** The Master's scheduling logic distributes primary and replica chunks across distinct storage nodes to prevent disk and network I/O hotspots.
- **Multi-Failure Domain Isolation:** Each 64 MB chunk is assigned to 3 physically isolated nodes ($R = 3$), ensuring zero data loss if a node drops.

Atomic Metadata Tracking & Failover Readiness

- **Global Catalog Mapping:** The Master records the logical chunk layout (`FileID_ChunkIndex`) and the ordered array of target nodes in the relational database.
- **Metadata-Driven Client Recovery:** During downloads, the client fetches the execution plan containing primary and backup addresses.
- **Transparent Failover:** If a primary node fails, the client uses the metadata mapping to automatically shift its socket connection pointer to a backup replica without interrupting the user.

S3–Style Flat Namespace

- Derived from **Amazon S3**, which stands for **Simple Storage Service** (the three "S"s form the acronym).
- Launched by AWS in 2006, it revolutionized cloud data persistence and introduced the Object Storage paradigm.

User View (Logical Hierarchy)

- Traditional POSIX/Linux file systems store assets in deeply nested tree structures. Amazon S3 replaced this with a **Flat Namespace**, where directories do not exist as physical entities on disk—they are merely textual string prefixes within the object key

```
/documents/pizzeria/file.avi
```

Physical Storage on Nodes

Local node disks store files within a flat namespace:

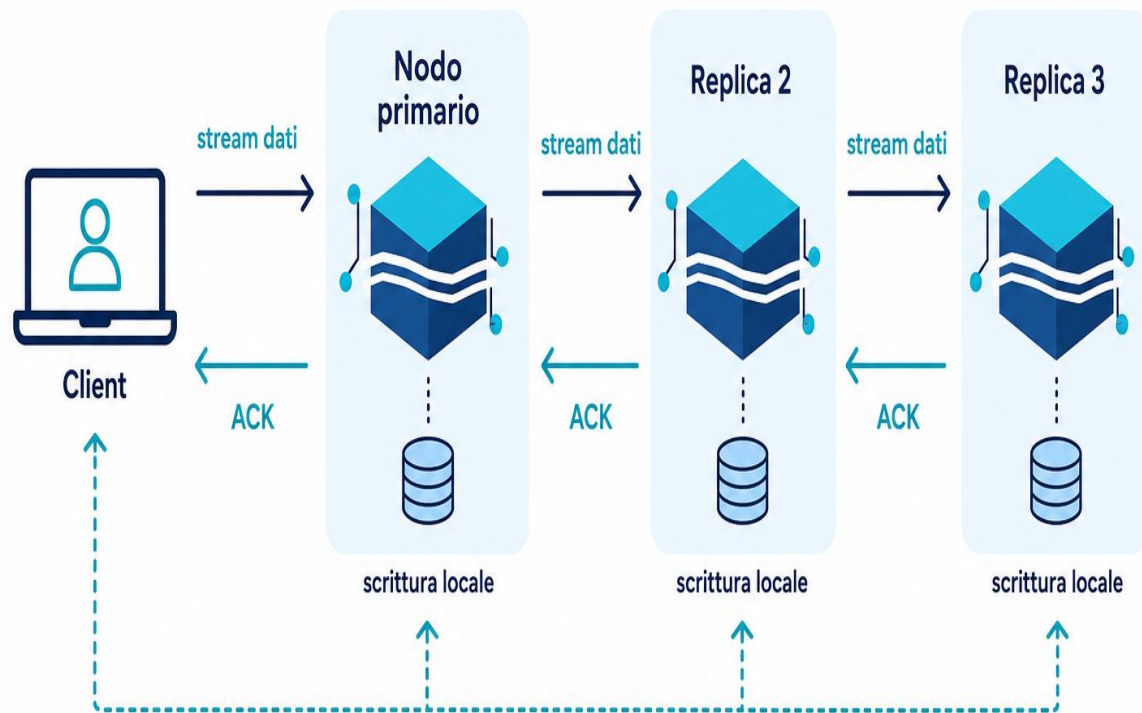
```
42_0_file.chunk
```

- File naming based on the **FileID_ChunkIndex** structural key pattern.
- Zero overhead caused by traditional hierarchical filesystem tree traversals.

Why We Adopted the S3 Model in AdriaBOX:

- **Folder Operation Efficiency:** In a traditional file system, moving or renaming (**mv**) a folder containing 50,000 files requires recursively traversing the tree and updating directory nodes for every single file.
- **Guaranteed $O(1)$ Time Complexity:** In AdriaBOX, renaming a folder with 50,000 files requires a single atomic SQL update statement on the Master node to modify the string prefix. It executes in constant time **$O(1)$** without touching a single physical byte across the storage node fleet.

Cascading Pipeline Replication & Client-Side Failover



12 Storage Nodes & R=3 Replication Factor:

- Deterministic Chained Round-Robin allocation across failure domains.

Daisy-Chain Pipeline (Bandwidth Efficiency):

- **Client -> Node_Primary -> Node_Replica2 -> Node_Replica3.**
- **66% Client Bandwidth Saved:** Client streams bytes **only once**; nodes replicate data downstream across the internal cluster network.

Client-Side High Availability Failover:

- If a primary node drops, the client catches the socket exception and seamlessly redirects the connection pointer to a backup replica.

```
wearemassive@wearemassive: ~/AdriaBOX
- Installing mdurl (0.1.2)
- Installing packaging (26.2)
- Installing pluggy (1.6.0)
- Installing pycparser (3.0)
- Installing pygments (2.20.0)
- Installing pyjwt (2.12.1)
- Installing pytest (9.0.3)
- Installing requests (2.33.1)
- Installing rich (13.9.4)
- Installing urllib3 (2.6.3)

Installing the current project: adriabox (0.1.0)
=== Provisioning Complete. Dropping into interactive shell ===
root@221f7ab6aebb:/app# poetry run adria register sam 456

:anchor: AdriaBOX

A compact CLI for the AdriaBOX project.
Registration successful. Please login.
root@221f7ab6aebb:/app# poetry run adria login sam 456

:anchor: AdriaBOX

A compact CLI for the AdriaBOX project.
Login successful. Session active.
root@221f7ab6aebb:/app# poetry run upload 200MB.zip
Command not found: upload
root@221f7ab6aebb:/app# poetry run adria upload 200MB.zip

wearemassive@wearemassive: $
```

AI-Assisted Testing Methodology & Validation

AI-Assisted Software Engineering Paradigm:

- Leveraged AI as a high-throughput productivity multiplier under strict human architectural oversight.

2-Phase Test Suite Engineering:

1. *Human Architecture*: Designed mock interception filters using `@patch` on `unittest.mock` to isolate network calls in memory.
2. *AI Matrix Expansion*: Fed class interfaces and TCP protocol specifications to AI prompts to generate 112 targeted edge-case & boundary test checkpoints.

Empirical Validation Results:

- **112/112 Tests Passed in 1.63 Seconds** (unit, crypto, control, data, and database tiers).

Future Works: Raft Consensus Metadata Cluster

Single Point of Failure (SPOF) Elimination:

- Current SQLite Master is a SPOF. Future scaling mandates State Machine Replication (SMR).

Distributed Raft Consensus Mechanics:

- Cluster of $N = 3$ nodes with Quorum bound $Q = N/2 + 1$ to prevent split-brain.
- Randomized term leader elections, log replication, and snapshotting.
- Transition from SQLite to distributed engines like **RocksDB** or **rqlite**.

Ephemeral Soft-Deletion:

- 30-day trash-bin retention with asynchronous background garbage collection.

Thank you for your attention!

AdriaBOX: A Distributed, Resilient, Scalable, and Secure Object Storage Matrix.

Sajmir Buzi
sajmir.buzi@studio.unibo.
it

Massimiliano Pupillo
massimiliano.pupillo@studio.unibo.it

