Integrinow DMS v2: System Component Primer

1. Introduction: The Mission of Defensible Chain of Custody

The Integrinow Document Management System (DMS) v2 is built upon a single, uncompromising architectural requirement: the maintenance of a **defensible chain of custody** for every document ingested. In high-stakes regulatory environments, "defensible" means more than just storage; it implies a mathematically verifiable and tamper-evident history of every byte from the moment of upload through to final archival.Before the system even accepts a file, it verifies its own "Source Integrity." Using Ed25519 digital signatures, the system fingerprints its own codebase at startup to ensure no unauthorized modifications have occurred in the execution environment. This establishes a "root of trust" that extends to the roles of **Processors** (internal staff who upload on behalf of customers) and **Reviewers** (officials who grant final approval).**Core Principle** No document reaches durable storage without successfully navigating a gauntlet consisting of mandatory memory-streamed malware scanning, SHA-256 integrity recording, envelope encryption, and manual human review. No document is considered "on file" until this chain is complete.While the mission is focused on trust, the execution relies on a layered defense of specialized tools orchestrated by a central state machine.

2. The Gatekeeper: ClamAV Malware Scanner

The entry point to the pipeline is the **ClamAV** malware scanner. Integrinow treats raw uploads as "untrusted" until proven otherwise.To maximize security, the Celery worker utilizes the **INSTREAM protocol** via the clamd daemon. We explicitly avoid path-based scanning—a method where a scanner accesses a file already sitting on a disk—to mitigate a known historical regression where files could potentially execute or be accessed by other processes before being cleared. By streaming the file's bytes directly to the scanner's memory, we ensure that malicious code never touches our internal "Vault" storage in its raw form.

Scanner Logic

Result,System Action
Infected,Immediate REJECTED state. File is deleted from quarantine; malware signature is logged; uploader is notified.
Clean,State progresses to VALIDATED . A SHA-256 integrity record is generated to lock the document's identity.
Once a file is proven safe, its unique SHA-256 signature is recorded. This ensures that any subsequent "DNA" change, no matter how small, is immediately detectable.

3. The Storage Strategy: MinIO Object Storage

The system utilizes **MinIO** object storage, but treats the movement of data as a **promotion of trust** . We divide storage into two logically and physically distinct buckets:

- **The Quarantine Bucket:** A transient, low-trust area for initial uploads.

- **The Vault Bucket:** The high-security home for processed documents, configured with **WORM (Write Once Read Many)** protections.

The Significance of WORM and Governance

The Vault bucket is configured with **Object Lock** in **GOVERNANCE mode** , with a default retention period of **1 day** . This prevents documents from being overwritten or deleted by standard users or automated processes once they reach the vault.**Key Properties of the Vault Bucket:**

- **Object Lock:** Enforces immutability for the duration of the retention period.
- **Versioning:** Maintains a permanent history of the object, preventing data loss.
- **Retention:** Governance mode provides a safety valve for administrators while blocking accidental or malicious deletions from standard service accounts.While storage ensures the object's persistence, the next layer ensures its confidentiality through advanced cryptography.

4. The Security Heart: HashiCorp Vault & Envelope Encryption

To protect sensitive data, Integrinow employs **Envelope Encryption** managed by **HashiCorp Vault** . We utilize a unique key for every document, ensuring that a single key compromise does not jeopardize the entire repository.

The Encryption Flow

1. **Generate DEK:** The system generates a local **Data Encryption Key (DEK)** using the **Fernet** (AES-128 in CBC mode with HMAC) algorithm.
2. **Encrypt File:** The file content is encrypted using this Fernet DEK.
3. **Wrap DEK:** The DEK is sent to the HashiCorp Vault **Transit Engine** . Vault encrypts the DEK using a **Key Encryption Key (KEK)** .**The Security Insight:** The system only persists the **wrapped DEK** (ciphertext) in the PostgreSQL database. Because the KEK never leaves the Vault's internal memory and the Vault itself is often kept in a **sealed state** requiring manual unsealing, an attacker with full database access cannot decrypt a single document. They would lack the cleartext DEK and the ability to unwrap it without a least-privilege token authorized by the Vault.

5. The Paper Trail: PostgreSQL & Tamper-Evident Audit Logs

The database records every state change, but unlike standard logs, our **AuditLogEntry** table uses a cryptographic **hash-chain** . Each record is linked to the one preceding it via a prev_hash field.

Anatomy of an Audit Log Entry

```
{
  "id": 1024,
  "prev_hash": "a3f28b...88e1",
  "entry_hash": "7e49c2...f102",
  "actor_id": "user_uuid",
```

```
  "action": "DOCUMENT_APPROVED",
  "resource_type": "Document",
  "resource_id": "doc_uuid",
  "event_metadata": {
    "notes": "Verified identity and signature."
  }
}
```

Architectural Insight: Preventing Chain Forks Because the system uses multiple Celery workers to process documents concurrently, there is a risk that two workers might attempt to append to the chain at the exact same microsecond. To prevent the chain from "forking"—which would create a false-positive tamper signal—the system utilizes a **Postgres advisory transaction lock** (specifically key 725_001_001). This serializes the "read-compute-append" operation, ensuring the integrity of the linear chain across all concurrent workers.

6. Orchestration: The Async Pipeline (Celery & Redis)

The system's "brain" is the **Celery** worker pool and **Redis** broker. This pipeline ensures that no document can skip a security gate. Each stage is a mandatory progression through the state machine.

Document Lifecycle Mapping

Document Status,Key Component Active,Responsibility
AV_SCANNING,ClamAV / Celery,In-memory malware verification.
VALIDATED,Integrity Service,SHA-256 Fingerprinting.
PROCESSING,Transformation Engine,Placeholder application/OCR preparation.
ENCRYPTING,HashiCorp Vault,DEK generation and wrapping (Transit Engine).
NEEDS_REVIEW,Reviewer / Review Queue,Final human-in-the-loop validation.
APPROVED,Storage Service,Document made available for authorized download.
Note that the transition to **NEEDS_REVIEW** only occurs *after* the document is encrypted and safely committed to the Vault bucket. At that point, the document sits in a "sealed" state awaiting a **Reviewer's** decision.

7. Summary: The Layered Defense Checklist

By combining these specialized components, the Integrinow DMS v2 ensures that security is not a single point of failure, but an emergent property of the entire stack.

Security Checklist

Before a document is considered "On File," the system confirms:

- **Source Integrity Verified:** The system has verified its own code signatures (Ed25519) before starting the pipeline.
- **Malware Free:** Scanned via ClamAV INSTREAM protocol to prevent disk-based execution.

- **Integrity Locked:** A SHA-256 integrity record is stored before any transformations occur.
- **Privacy Protected:** File is encrypted with a unique Fernet DEK, wrapped by a Vault KEK.
- **Durable & Immutable:** Stored in a WORM-protected MinIO Vault bucket with a 1-day governance lock.
- **Accountability Proven:** All state changes are etched into a hash-chained audit log protected by advisory transaction locks.
- **Human Verified:** A designated **Reviewer** has manually inspected and approved the document.