Disaster Recovery Strategy: Integrinow DMS v2

1. Strategic Continuity Framework

In the modern regulatory landscape, the value of a document management system is measured not merely by its storage capacity, but by the integrity of the data it holds. This Disaster Recovery (DR) Strategy for Integrinow DMS v2 is designed to maintain a **defensible chain of custody** for sensitive customer documents. It ensures that even in the event of a catastrophic system failure, the tamper-evident nature of the records remains intact and verifiable. The system's primary value proposition is its tamper-evident integrity, which must survive any restoration event.The Integrinow DMS v2 manages a rigorous "Document Lifecycle" governed by a **fixed state machine** . Documents transition from initial receipt (RECEIVED) through malware scanning (AV_SCANNING), validation, processing, and human review (NEEDS_REVIEW) until they reach a terminal state of APPROVED, REJECTED, or ARCHIVED. Protecting this lifecycle requires a recovery strategy that prevents any document from "skipping" a state or entering an invalid transition, ensuring the audit trail accurately reflects every automated and human decision.

Critical Dependencies by Role

During a recovery event, the following role-based dependencies must be prioritized to restore functional service:| Role | Critical Dependency during Recovery | Strategic Continuity Nuance || ------ | ------ | ------ || **Customer** | Access to download APPROVED or ARCHIVED documents. | Scoped strictly to their own org_id. || **Processor** | Access to upload queues and intake confirmation. | Acts as the "bridge" for intake if Customer-facing portals are degraded. || **Reviewer** | Access to the NEEDS_REVIEW queue. | Must be restored to prevent bottlenecks in the manual validation phase. || **Admin** | Full access to audit logs, chain verification, and keys. | Critical for confirming system state and managing restoration. |
This strategy moves beyond simple data redundancy, establishing technical layers of data durability designed to prevent loss before a disaster occurs and to ensure a "warm" state is achievable during restoration.

2. Data Durability & Tamper-Resistance Architecture

Effective disaster recovery begins with proactive durability. By embedding protection directly into the storage and logging layers, Integrinow DMS v2 minimizes the risk of data loss or unauthorized alteration that could compromise the system's defensibility.

Immutable Storage (The Vault)

The system utilizes a "Vault" bucket for long-term document storage. This bucket is configured with **Write Once, Read Many (WORM)** protection using MinIO's Object Lock feature. By operating in **GOVERNANCE mode** , the system prevents unauthorized deletion or overwriting of validated documents. While the system defaults to a **1-day retention window** , this is a strategic lever: administrators must configure this period to match specific regulatory requirements to ensure files cannot be purged, even by privileged users, during a crisis.

Audit Log Integrity

To ensure a defensible record of all system transitions, the Audit Log employs a **SHA-256 hash-chaining mechanism** . Every log entry contains a hash of its own data combined with the hash of the preceding entry. To prevent "chain forking" during high-concurrency recovery events—where multiple Celery workers might attempt to append logs simultaneously—the system utilizes **Postgres advisory transaction locks** . Without these locks, a system reboot or recovery could lead to race conditions that render the audit trail mathematically inconsistent and therefore indefensible.The presence of immutable storage and cryptographic audit trails ensures that upon recovery, the system can provide mathematical proof that the restored data is identical to the data as it existed prior to the disruption, transitioning now to the formal backup protocols.

3. Systematic Backup & Multi-Cloud Replication

To mitigate the risk of total site failure, this strategy shifts from local redundancy to geographic and provider-level diversity. The objective is to ensure that a copy of the system exists outside of the primary production environment at all times.

Full System Backup Procedure

The system performs a comprehensive backup by bundling the **PostgreSQL 16 database** (captured via pg_dump -Fc) and the **MinIO object storage** (Quarantine and Vault buckets). A critical security feature of this process is the use of an **in-memory tarball** . The backup components are bundled and encrypted in a BytesIO buffer, ensuring that plaintext data never touches the physical disk of the host, thereby eliminating the risk of forensic leakage.

Multi-Cloud Redundancy

While local retention maintains a **14-day cycle** , the system is designed for broad off-site replication. Supported through a JSON-based target list, the system utilizes a multi-cloud approach to replicate backups to:

- **AWS S3**
- **Google Cloud Storage**
- **Azure Blob StorageNote:** To maintain a high security posture, cloud credentials for these targets must be stored within **Vault (KV v2)** rather than as environmental variables, ensuring secrets are encrypted at rest and accessible only to authorized backup processes.

Security and Encryption

Backups utilize **Envelope Encryption** via the **HashiCorp Vault transit engine** . Each backup is secured with a fresh Data Encryption Key (DEK), which is itself "wrapped" by Vault.**Crucial Limitation:** These backups are completely unusable without the associated cryptographic keys. Therefore, the restoration of the key management service is the critical path in any recovery operation.

4. Vault Recovery & Secret Management

HashiCorp Vault is the linchpin of the Integrinow DMS v2 security architecture. Because Vault manages the keys for both document and backup decryption, its restoration is the highest priority. A "Sealed" Vault represents a total block to system functionality.

Unsealing Methodologies

Operators may utilize two distinct methods to bring the Vault into an operational state:

1. **Manual Shamir Unseal:** Requires a **3-of-5 threshold** of key shares.
2. **Cloud KMS Auto-Unseal:** The recommended production configuration. By using **AWS KMS, GCP CKMS, or Azure Key Vault** , Vault automatically decrypts its master key at boot, enabling zero-human-intervention recovery.

Access Policy Hygiene

Once Vault is unsealed, strict access control is enforced through distinct policies:

- **Root Token Hygiene:** The root token is revoked immediately after initial setup; any further use requires a logged regeneration event via Shamir shares.
- **Scoped Policies:** The application uses a least-privilege dms-transit-policy for file operations. This is distinct from the dms-integrity-verify-policy used for code validation; the recovery of one does not grant access to the other.A "warm" Vault state must be confirmed before any data restoration can begin, as data remains inaccessible while the Vault is sealed.

5. Emergency Restoration Protocols

Data restoration is a destructive process. To prevent accidental data overwrites during the chaos of a disaster, the system requires explicit administrative confirmation.

Technical Protocol for Restoration

1. **Manifest Selection:** Select the desired backup and verify SHA-256 checksums.
2. **Administrative Confirmation:** The operator must type the **"RESTORE" confirmation string** into the API/UI to acknowledge the destruction of current data.
3. **Database Reconstruction:** The system executes pg_restore --clean, dropping and recreating tables to ensure a clean state.
4. **Connection Disposal:** Immediately post-restore, the system must call db.engine.dispose(). This prevents the application from using stale SQLAlchemy connections to tables that were recreated out-of-band.

Deployment Topology Considerations

The restoration method is dictated by security constraints:

- **Containerized/K8s (direct):** Uses the direct method because the containerized app has no access to the host's Docker socket—a critical security boundary that prevents containers from gaining host-root equivalent access.
- **Host-run (docker_exec):** Utilizes standard Docker shell-outs for Postgres operations.**Warning (Lesson Learned):** Restoration requires strict version parity.

Previous incidents revealed that postgresql-client-17 generates dumps that a v16 server rejects. Consequently, the environment pins **postgresql-client-16** to ensure restoration compatibility.

6. Source Integrity & Service Validation

Strategic recovery is not complete until the integrity of the application code is verified. This "Source Integrity Verification" ensures the recovery environment has not been compromised at the logic level.

The verify_at_startup() Process

Upon initialization, the system fingerprints the source tree and compares it against a **Vault-signed manifest** . This process uses Ed25519 asymmetric signing. The application holds only the verify key, ensuring that even if the running process is compromised, it cannot forge a passing manifest for modified code.

Tiered Severity of Integrity Failures

- **CRITICAL:** Mismatches in core logic files (.py, .sh) result in an immediate **system exit** .
- **WARNING:** Mismatches in static assets (CSS, images) trigger a log alert but allow the system to start.

Defensible Recovery

The separation of signing and verification keys (using the dms-integrity-verify-policy) is essential for a "defensible" recovery. Note that the current protection boundary extends only to the **source tree** ; it does not yet cover pip dependencies or the base Docker image.This Disaster Recovery Strategy is a living framework. To remain effective, it must be validated periodically through the system's documented test suites (unit, integration, and Playwright e2e), ensuring that every backup and recovery path remains functional as the system evolves.