Commvault Data Aging Policy Research

Executive Summary

Aging policies in Commvault are NOT standalone entities - they are embedded as **retention rules** within: 1. **Storage Policy Copies** - Each copy has its own retention rules 2. **Plans** - Modern Commvault approach with retention at the copy level

There is NO separate `/AgingPolicy` endpoint. Aging data is retrieved as part of Storage Policy and Plan data.

What is Data Aging in Commvault?

Overview

Data Aging is Commvault's retention and lifecycle management feature that: 1. **Marks jobs as aged** when they exceed configured retention periods 2. **Prunes (deletes) eligible data** from storage when all retention conditions are met 3. **Runs automatically** (default: daily at 12:00 PM)

Key Concepts

Retention Logic: Both Days AND Cycles must be met - **Days**: Calendar days to retain data - **Cycles**: Number of full backup cycles to keep - Commvault uses **AND logic** - the

longer value determines actual retention

Retention Cycle Definition:

> *A complete full (or synthetic full) backup followed by all subsequent incremental, differential, or transactional log backups that depend on that full backup*

How Aging Data is Stored in Commvault API

Location 1: Storage Policy Copy Retention Rules

Each storage policy copy contains retention rules:

```
{
  "copy": {
    "StoragePolicyCopy": {
      "copyId": 1341,
      "copyName": "01 - Primary"
    },
    "retentionRules": {
      "retainBackupDataForCycles": 1,
      "retainBackupDataForDays": 10,
      "retainArchiverDataForDays": -1,
      "jobs": 0,
      "retentionFlags": {
        "enableDataAging": 1,
        "jobBasedRetention": 0
      }
    }
  }
}
```

Location 2: Plan Storage Retention

Plans contain similar retention at the copy level:

```
{
  "plan": {
    "planId": 28,
    "planName": "Silver Plan",
    "storage": {
      "copy": [
        {
          "copyName": "Primary",
          "retentionRules": {
            "retainBackupDataForCycles": 1,
            "retainBackupDataForDays": 30,
            "retentionFlags": {
              "enableDataAging": 1
            }
          }
        }
      ]
    }
  }
}
```

Retention Rules Field Reference

Core Retention Fields

Field	Type	Description	Values
retainBackupDataForDays	Integer	Days to retain backup data	-1 (infinite), 0+, default: 30

Field	Type	Description	Values
retainBackupDataForCycles	Integer	Number of backup cycles to retain	-1 (infinite), 0+, default: 1
retainArchiverDataForDays	Integer	Days to retain archived data	-1 (infinite), 0+
jobs	Integer	Job-based retention count	Usually 0

Retention Flags

Field	Type	Description	Values
enableDataAging	Integer (Boolean)	Whether data aging is enabled	0 (disabled), 1 (enabled)
jobBasedRetention	Integer (Boolean)	Use job-based retention	0 (time-based), 1 (job-based)
enableArchiving	Integer (Boolean)	Archive retention enabled	0/1

Extended Retention Rules

Some policies may have extended retention for specific backup types:

```
{
  "extendedRetentionRules": {
    "firstExtendedRetentionRule": {
      "retainBackupDataForDays": 90,
      "retainBackupDataForCycles": 3,
      "isInfiniteRetention": 0
    },
    "secondExtendedRetentionRule": {
```

```
      "retainBackupDataForDays": 365,
      "retainBackupDataForCycles": 12,
      "isInfiniteRetention": 0
    }
  }
}
```

API Endpoints for Aging Data

Primary Endpoints

Endpoint	Method	Purpose	Contains Aging Data
`/StoragePolicy`	GET	List all policies	❌ No (IDs only)
`/StoragePolicy/{id}`	GET	Get policy details	✅ Yes (retention rules per copy)
`/Plan`	GET	List all plans	⚠️ Partial (basic info)
`/Plan/{id}`	GET	Get plan details	✅ Yes (full retention rules)

Response Structure

GET /StoragePolicy/{id} response includes:

```
{
  "storagePolicy": {
    "storagePolicyId": 50,
    "storagePolicyName": "Silver Plan",
    "copy": [
      {
        "StoragePolicyCopy": {
          "copyId": 1341,
```

```
          "copyName": "Primary"
        },
        "retentionRules": {
          "retainBackupDataForDays": 30,
          "retainBackupDataForCycles": 1,
          "retentionFlags": {
            "enableDataAging": 1,
            "jobBasedRetention": 0
          }
        },
        "extendedRetentionRules": {...}
      }
    ]
  }
}
```

Calculating Effective Retention

Formula

```
Effective Retention = MAX(
    retainBackupDataForDays,
    retainBackupDataForCycles * Average_Cycle_Duration
)
```

Examples

Example 1: Days = 30, Cycles = 1, Avg Cycle = 7 days - Days retention: 30 days - Cycles retention: 1 × 7 = 7 days - **Effective**: MAX(30, 7) = **30 days**

Example 2: Days = 10, Cycles = 5, Avg Cycle = 7 days - Days retention: 10 days - Cycles retention: 5 × 7 = 35 days - **Effective**: MAX(10, 35) = **35 days**

Example 3: Days = -1 (infinite) - **Effective**: **Infinite** (data never ages)

Current Implementation Status

Storage Pools ✅ Fixed

- Parser corrected to use `storagePoolEntity`
- Data now being saved correctly to database

Storage Policies ⚠️ Basic Only

- **Currently**: Only ID and name saved
- **Missing**: Copy retention rules not captured
- **Solution**: Implement detailed policy collection

Plans ❌ Not Implemented

- **Currently**: Not being collected at all
- **Contains**: Full retention rules per copy
- **Priority**: HIGH - Plans are modern Commvault approach

Implementation Plan

Phase 1: Database Schema for Retention Data

Create new table to store retention rules:

```
CREATE TABLE IF NOT EXISTS retention_rules (
    ruleId INTEGER PRIMARY KEY AUTOINCREMENT,
    entityType TEXT NOT NULL,              -- 'policy_copy' or 'plan_copy'
```

```
    entityId INTEGER NOT NULL,             -- copyId or planId
    entityName TEXT,                       -- Copy name or Plan name
    parentId INTEGER,                      -- storagePolicyId or planId
    parentName TEXT,                       -- Policy name or Plan name

    -- Core retention
    retainBackupDataForDays INTEGER,      -- -1 for infinite
    retainBackupDataForCycles INTEGER,    -- -1 for infinite
    retainArchiverDataForDays INTEGER,    -- -1 for infinite, NULL if N/A

    -- Flags
    enableDataAging INTEGER,              -- 0/1
    jobBasedRetention INTEGER,            -- 0/1

    -- Extended retention (optional)
    firstExtendedRetentionDays INTEGER,
    firstExtendedRetentionCycles INTEGER,
    secondExtendedRetentionDays INTEGER,
    secondExtendedRetentionCycles INTEGER,
    thirdExtendedRetentionDays INTEGER,
    thirdExtendedRetentionCycles INTEGER,

    -- Metadata
    lastFetchTime TEXT,

    UNIQUE(entityType, entityId)
);
```

Phase 2: Data Collection Functions

Function 1: Extract Retention from Policy Details

```
def extract_retention_rules(copy_data):
    """Extract retention rules from storage policy copy or plan copy"""
    retention_rules = copy_data.get("retentionRules", {})

    return {
        'retainBackupDataForDays': retention_rules.get('retainBackupDataForDays', -1)
        'retainBackupDataForCycles': retention_rules.get('retainBackupDataForCycles',
        'retainArchiverDataForDays': retention_rules.get('retainArchiverDataForDays',
```

```
        'enableDataAging': retention_rules.get('retentionFlags', {}).get('enableDataA
        'jobBasedRetention': retention_rules.get('retentionFlags', {}).get('jobBasedR
    }
```

Function 2: Save Retention Rules to Database

```
def save_retention_rules_to_db(db, entity_type, entity_id, entity_name, parent_id, pa
    """Save retention rules to database"""
    cur = db.cursor()
    fetch_time = datetime.now().isoformat()

    cur.execute("""
        REPLACE INTO retention_rules (
            entityType, entityId, entityName, parentId, parentName,
            retainBackupDataForDays, retainBackupDataForCycles,
            retainArchiverDataForDays, enableDataAging, jobBasedRetention,
            lastFetchTime
        ) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
    """, (
        entity_type, entity_id, entity_name, parent_id, parent_name,
        retention_data['retainBackupDataForDays'],
        retention_data['retainBackupDataForCycles'],
        retention_data['retainArchiverDataForDays'],
        retention_data['enableDataAging'],
        retention_data['jobBasedRetention'],
        fetch_time
    ))
```

Function 3: Fetch Storage Policy Details

```
def fetch_and_save_policy_retention(db, base_url, headers, policy_id, policy_name):
    """Fetch detailed policy info and extract retention rules"""
    try:
        # Get policy details
        response = requests.get(
            f"{base_url}/StoragePolicy/{policy_id}",
            headers=headers,
            timeout=30
        )
```

```
        if response.status_code != 200:
            return 0

        policy_data = response.json()

        # Extract copies and their retention rules
        copies = policy_data.get("copy", [])
        count = 0

        for copy in copies:
            copy_info = copy.get("StoragePolicyCopy", {})
            copy_id = copy_info.get("copyId")
            copy_name = copy_info.get("copyName", "")

            # Extract retention rules
            retention_data = extract_retention_rules(copy)

            # Save to database
            save_retention_rules_to_db(
                db, 'policy_copy', copy_id, copy_name,
                policy_id, policy_name, retention_data
            )
            count += 1

        return count

    except Exception as e:
        print(f"Error fetching policy {policy_id}: {e}")
        return 0
```

Phase 3: UI Display

Aging Policy Summary View

Create: `templates/retention_summary.html`

Features: - **Summary Statistics**: - Total policies with aging enabled - Average retention period - Policies with infinite retention - Shortest/longest retention periods

- **Retention Rules Table**:
- Policy/Plan name
- Copy name
- Days retention
- Cycles retention
- Effective retention (calculated)
- Aging enabled status
- **Visual Elements**:
- Color-coded retention periods (green=short, yellow=medium, red=long)
- Infinite retention warning badges
- Aging disabled warnings

Detailed Retention View

Add to existing policy/plan views: - Expandable retention details - Extended retention rules display - Aging job history link

---

Quick Implementation (Minimal Viable Product)

Step 1: Add Retention Column to Existing Tables

Instead of new table, add retention summary to existing tables:

```
-- Add to storage_policies table
ALTER TABLE storage_policies ADD COLUMN avgRetentionDays INTEGER;
ALTER TABLE storage_policies ADD COLUMN agingEnabled INTEGER;
```

```
-- Add to plans table (when created)
ALTER TABLE plans ADD COLUMN avgRetentionDays INTEGER;
ALTER TABLE plans ADD COLUMN agingEnabled INTEGER;
```

Step 2: Quick Display on Dashboard

Add aging summary to dashboard:

```
# In infrastructure_dashboard route
cur.execute("""
    SELECT
        COUNT(*) as total,
        AVG(avgRetentionDays) as avg_retention,
        SUM(CASE WHEN agingEnabled = 1 THEN 1 ELSE 0 END) as aging_enabled_count
    FROM storage_policies
""")
aging_stats = cur.fetchone()
```

Display:

```
<div class="summary-card">
    <h3>{{ aging_stats.avg_retention }} days</h3>
    <p>Average Retention Period</p>
</div>
<div class="summary-card">
    <h3>{{ aging_stats.aging_enabled_count }}/{{ aging_stats.total }}</h3>
    <p>Policies with Aging Enabled</p>
</div>
```

Recommendations

Priority 1 (CRITICAL): Fix Storage Pools ✅ DONE

- Parser fixed to use correct JSON path

- Data now saves correctly

Priority 2 (HIGH): Implement Plan Collection

- Plans contain full retention data
- Modern Commvault approach
- Easier to parse than policies

Priority 3 (HIGH): Add Policy Details Collection

- Fetch `/StoragePolicy/{id}` for each policy
- Extract retention rules from each copy
- Store in retention_rules table

Priority 4 (MEDIUM): Create Retention Summary View

- Dashboard widget showing aging statistics
- Dedicated retention rules page
- Visual retention timeline

Priority 5 (LOW): Advanced Features

- Aging job history tracking
- Retention compliance reports
- Aging predictions based on data growth

Example Usage Scenarios

Scenario 1: Compliance Audit

Question: "What is our data retention policy for client X?"

Answer via UI: 1. Navigate to Retention Summary 2. Filter by client name 3. View effective retention period 4. Export compliance report

Scenario 2: Storage Optimization

Question: "Which policies are keeping data the longest?"

Answer via UI: 1. Navigate to Retention Summary 2. Sort by "Effective Retention" descending 3. Identify policies with infinite retention 4. Review and optimize retention settings

Scenario 3: Data Aging Monitoring

Question: "Is data aging running successfully?"

Answer via UI: 1. Check "Aging Enabled" count on dashboard 2. View last aging job status 3. Review aging job history

---

API Update Capabilities

Updating Retention Rules

Endpoint: `PUT /Plan/{planId}/Storage/Modify`

```
{
  "primaryRetentionInDays": 30,
  "secondaryRetentionInDays": 90
}
```

Endpoint: `POST /StoragePool?Action=copyEdit`

```
{
  "retentionRules": {
    "retainBackupDataForDays": 30,
    "retainBackupDataForCycles": 1
  }
}
```

Note: API write operations require appropriate permissions and should be implemented with caution.

Conclusion

Aging policies in Commvault are **not standalone entities** but are **retention rules embedded in Storage Policies and Plans**.

Current Status: - ✅ Storage Pools: Fixed and working - ⚠️ Storage Policies: Basic data only (needs enhancement) - ❌ Plans: Not implemented (high priority) - ❌ Retention Rules: Not extracted or displayed

Next Steps: 1. Implement Plan collection (/Plan endpoint) 2. Enhance Storage Policy collection to fetch details 3. Extract and store retention rules 4. Create UI to display aging/retention data 5. Add dashboard widgets for aging statistics

Data Location: Retention rules are found in: - `/StoragePolicy/{id}` response → `copy[].retentionRules` - `/Plan/{id}` response → `storage.copy[].retentionRules`