Commvault Policy and Storage Pool Data Research

Research Summary

This document provides comprehensive research on how to pull and present **Storage Policy data** and **Storage Pool data** from the Commvault REST API.

Part 1: Storage Pool Data (Currently Being Pulled)

Current Status: ✅ **WORKING - Data is being pulled AND presented**

API Endpoint

```
GET /StoragePool
```

Current Implementation Location

- **Database Function**: `save_storage_pools_to_db()` - app.py:517
- **Fetch Route**: `/fetch_data` when `dtype == "storage_pools"` - app.py:863

- **View Route**: `/view/storage_pools` - app.py:996
- **Dashboard Display**: dashboard.html:167-199

Data Currently Being Saved to Database

Table: `storage_pools`

Column	Data Type	Description
storagePoolId	INTEGER (PK)	Unique identifier for storage pool
storagePoolName	TEXT	Display name of the pool
storagePoolType	TEXT	Type code (1, 4, etc.)
mediaAgentName	TEXT	Associated MediaAgent
totalCapacity	TEXT	Total pool capacity
freeSpace	TEXT	Available free space
dedupeEnabled	TEXT	Deduplication status
lastFetchTime	TEXT	Last data retrieval timestamp

JSON Response Structure

```
{
  "storagePoolList": [
    {
      "cloudStorageClassName": "Standard/Glacier (Combined Storage Tiers)",
      "numberOfNodes": 13,
      "poolRetentionPeriodDays": -1,
      "sizeOnDisk": 16691809,
      "totalCapacity": -1,
      "totalFreeSpace": -1,
      "wormStoragePoolFlag": 0,
      "reserved1": 1,
      "cloudStorageClassNumber": 9,
      "isArchiveStorage": 1,
      "libraryVendorType": 15,
      "storagePoolType": 1,
      "storageSubType": 0,
      "storageType": 2,
      "status": "Online",
      "statusCode": 0,
      "libraryList": [
        {
          "_type_": 9,
          "libraryId": 151
        }
      ],
      "storagePool": {
        "_type_": 28,
        "clientGroupId": 481,
        "clientGroupName": "ActiveScale_EXT"
      },
      "storagePolicyEntity": {
        "_type_": 17,
        "storagePolicyName": "ActiveScale_EXT",
        "storagePolicyId": 481,
        "entityInfo": {
          "companyId": 0,
          "companyName": "Commcell",
          "multiCommcellId": 0
        }
      },
      "storagePoolEntity": {
        "storagePoolName": "ActiveScale_EXT",
        "_type_": 160,
        "storagePoolId": 481
      }
    }
  ]
}
```

Additional Available Fields (Not Currently Saved)

These fields are available in the API response but not currently being saved:

1. **sizeOnDisk** - Actual disk space consumed (in MB)
2. **cloudStorageClassName** - Cloud storage tier/class name
3. **cloudStorageClassNumber** - Cloud storage tier code
4. **isArchiveStorage** - Boolean (0/1) indicating if it's archive storage
5. **libraryVendorType** - Vendor type code
6. **storageType** - Storage type code (1=disk, 2=cloud, etc.)
7. **storageSubType** - Sub-type classification
8. **status** - Pool status ("Online", "Offline", etc.)
9. **statusCode** - Numeric status code
10. **wormStoragePoolFlag** - WORM (Write Once Read Many) flag
11. **poolRetentionPeriodDays** - Retention period in days
12. **numberOfNodes** - Number of nodes in pool
13. **libraryList[]** - Array of associated libraries
14. **storagePolicyEntity** - Associated storage policy information
15. **dedupeFlags** - Deduplication settings (if present)

Part 2: Storage Policy Data

Current Status: ⚠️ **PARTIALLY IMPLEMENTED - Minimal data being saved**

API Endpoint

```
GET /StoragePolicy
```

Current Implementation

- **Database Function**: `save_storage_to_db()` - app.py:410
- **Fetch Route**: `/fetch_data` when `dtype == "storage"` - app.py:823
- **View Route**: `/view/storage` - app.py:988

Current Database Schema (MINIMAL)

Table: `storage_policies`

Column	Data Type	Description
storagePolicyId	INTEGER (PK)	Unique identifier
storagePolicyName	TEXT	Policy display name
lastFetchTime	TEXT	Last retrieval timestamp

Problem: Only stores ID and name - no operational data!

JSON Response Structure

```
{
  "policies": [
    {
      "numberOfStreams": 150,
      "storagePolicy": {
        "storagePolicyName": "Silver Plan (Server)",
        "storagePolicyId": 59
      }
    }
  ]
}
```

Available Fields from Basic Endpoint

From `GET /StoragePolicy`: 1. **storagePolicyId** - Unique identifier ✅ *Currently saved* 2. **storagePolicyName** - Policy name ✅ *Currently saved* 3. **numberOfStreams** -

Maximum concurrent streams ❌ *NOT saved*

Part 3: Enhanced Storage Policy Data (Detailed Endpoint)

API Endpoint for Detailed Information

```
GET /StoragePolicy/{storagePolicyId}
```

According to Commvault documentation, this endpoint provides comprehensive policy details.

Expected Detailed Response Structure

Based on research, the detailed endpoint should return:

Basic Policy Information

- **storagePolicyId** - Unique identifier
- **storagePolicyName** - Display name
- **description** - Policy description
- **type** - Policy type code
- **flags** - Various policy flags
- **isDefault** - Boolean indicating if it's the default policy

Copy Configuration

Each policy can have multiple copies (Primary, Secondary, Archive, etc.): - **copyId** - Copy identifier - **copyName** - Copy name (e.g., "Primary Copy") - **copyType** - Type code

(1=Primary, 2=Secondary, etc.) - **copyPrecedence** - Priority order - **isDefault** - Boolean for default copy - **active** - Boolean indicating if copy is active

Storage Configuration

- **storagePoolId** - Associated storage pool
- **storagePoolName** - Pool name
- **libraryId** - Associated library
- **libraryName** - Library name
- **mediaAgentId** - MediaAgent handling this copy
- **mediaAgentName** - MediaAgent name

Deduplication Settings

- **enableDeduplication** - Boolean
- **enableClientSideDedup** - Boolean
- **enableDASHFull** - Boolean for DASH Full
- **useGlobalDedupStore** - Boolean

Retention Rules

- **retainBackupDataForDays** - Days to retain backup data
- **retainBackupDataForCycles** - Number of cycles to retain
- **retainArchiverDataForDays** - Archive retention in days
- **enableDataAging** - Boolean for data aging
- **jobBasedRetention** - Boolean

Storage Type Information

- **storageType** - Storage type code
- 1 = Disk

- 2 = Cloud
- 3 = Tape
- **storagePoolType** - Pool type code
- **deviceType** - Device type code
- **isArchiveStorage** - Boolean

Advanced Features

- **wormStorageFlag** - WORM protection enabled
- **isSnapCopy** - Snapshot copy enabled
- **isMirrorCopy** - Mirror copy enabled
- **encryptionType** - Encryption configuration
- **compressionType** - Compression settings

Part 4: Plan Data (Modern Commvault Approach)

API Endpoint

```
GET /Plan
GET /Plan/{planId}
```

What are Plans?

Plans are Commvault's modern, simplified approach to backup configuration that combines: - Storage policies - Schedule policies - Retention rules - Security settings - Subclient configurations

Plans are replacing traditional Storage Policies in newer Commvault versions.

Current Status: ❌ NOT IMPLEMENTED

Plans are **not currently being pulled or stored** in the database.

JSON Response Structure

```
{
  "plans": [
    {
      "numCopies": 2,
      "description": "Silver (Server Plan)",
      "type": 2,
      "numDevices": 0,
      "storageTarget": "HSXPOOL",
      "subtype": 33554437,
      "isElastic": false,
      "numAssocEntities": 0,
      "restrictions": 0,
      "numCompanies": 1,
      "planStatusFlag": 0,
      "rpoInMinutes": 1380,
      "numUsers": 0,
      "supportedWorkloads": {},
      "storage": {
        "storagePolicy": {
          "storagePolicyId": 50
        },
        "copy": [
          {
            "copyType": 1,
            "deviceType": 1,
            "active": 1,
            "isArchiveStorage": false,
            "isDefault": 1,
            "isSnapCopy": 0,
            "isMirrorCopy": 0,
            "wormStorageFlag": 0,
            "poolRetentionPeriodDays": -1,
            "storageClass": "",
            "copyPrecedence": 1,
            "storagePoolType": 4,
            "storageType": 3,
            "dedupeFlags": {
              "enableDASHFull": 1,
              "enableDeduplication": 1,
              "enableClientSideDedup": 1
            },
            "retentionRules": {
              "retainBackupDataForCycles": 1,
              "jobs": 0,
              "retainBackupDataForDays": 10,
              "retentionFlags": {
                "enableDataAging": 1
              }
            },
            "StoragePolicyCopy": {
              "copyId": 1341,
              "copyName": "01 - Primary"
            },
            "library": {
              "libraryName": "DiskLib_HSXPOOL",
              "libraryId": 107
            }
          }
        ]
      },
      "plan": {
        "planId": 28,
        "planName": "Silver (Packet Hub Server Plan)"
      }
    }
  ]
}
```

Available Plan Fields

Plan Summary

- **planId** - Unique plan identifier

- **planName** - Plan display name
- **description** - Plan description
- **type** - Plan type (2 = Server Plan, etc.)
- **subtype** - Detailed subtype code
- **planStatusFlag** - Status flag
- **restrictions** - Restriction flags

Operational Metrics

- **numCopies** - Number of backup copies
- **numAssocEntities** - Number of entities using this plan
- **numUsers** - Number of users assigned
- **numDevices** - Number of devices
- **numCompanies** - Number of companies using plan
- **rpoInMinutes** - Recovery Point Objective in minutes
- **slaInMinutes** - Service Level Agreement target

Storage Configuration

- **storageTarget** - Primary storage target name
- **storagePolicy.storagePolicyId** - Associated policy ID
- **storage.copy[]** - Array of copy configurations with full details

Feature Flags

- **isElastic** - Elastic plan capability
- **supportedWorkloads** - Workload types supported

Part 5: Recommendations for Implementation

Priority 1: Enhance Storage Pool Data Collection ⭐⭐⭐

Current: Basic info only (name, type, capacity, free space) **Enhancement**: Capture all available fields

```
def save_storage_pools_to_db_enhanced(db, pools_json):
    """Enhanced Storage Pools data collection"""
    cur = db.cursor()
    fetch_time = datetime.now().isoformat()

    pools_list = pools_json.get("storagePoolList", [])

    for pool_entry in pools_list:
        pool_info = pool_entry.get("storagePoolEntity", {})
        pool_id = pool_info.get("storagePoolId")
        name = pool_info.get("storagePoolName", "")

        # Basic info
        storage_type = pool_entry.get("storageType", "")
        storage_subtype = pool_entry.get("storageSubType", "")
        pool_type = pool_entry.get("storagePoolType", "")

        # Capacity info
        total_cap = pool_entry.get("totalCapacity", -1)
        free_space = pool_entry.get("totalFreeSpace", -1)
        size_on_disk = pool_entry.get("sizeOnDisk", -1)

        # Status
        status = pool_entry.get("status", "Unknown")
        status_code = pool_entry.get("statusCode", -1)

        # Features
        is_archive = pool_entry.get("isArchiveStorage", 0)
        worm_flag = pool_entry.get("wormStoragePoolFlag", 0)
        num_nodes = pool_entry.get("numberOfNodes", 0)
        retention_days = pool_entry.get("poolRetentionPeriodDays", -1)

        # Cloud-specific
        cloud_class_name = pool_entry.get("cloudStorageClassName", "N/A")
        cloud_class_num = pool_entry.get("cloudStorageClassNumber", -1)
        library_vendor = pool_entry.get("libraryVendorType", -1)
        # Associated policy
        policy_entity = pool_entry.get("storagePolicyEntity", {})
        policy_name = policy_entity.get("storagePolicyName", "")
        policy_id = policy_entity.get("storagePolicyId", None)

        cur.execute("""
            REPLACE INTO storage_pools (
                storagePoolId, storagePoolName, storageType, storageSubType,
                storagePoolType, totalCapacity, freeSpace, sizeOnDisk,
                status, statusCode, isArchiveStorage, wormStorageFlag,
                numberOfNodes, retentionPeriodDays, cloudStorageClass,
                cloudStorageClassNumber, libraryVendorType,
                storagePolicyName, storagePolicyId, lastFetchTime
            ) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
        """, (
            pool_id, name, storage_type, storage_subtype, pool_type,
            total_cap, free_space, size_on_disk, status, status_code,
            is_archive, worm_flag, num_nodes, retention_days,
            cloud_class_name, cloud_class_num, library_vendor,
            policy_name, policy_id, fetch_time
        ))
```

Priority 2: Implement Storage Policy Details Collection ⭐⭐⭐

Current: Only ID and name **New**: Full policy configuration

```
def fetch_storage_policy_details(base_url, headers, policy_id):
    """Fetch detailed information for a specific storage policy"""
    try:
        response = requests.get(
            f"{base_url}/StoragePolicy/{policy_id}",
            headers=headers,
            timeout=30
        )
        if response.status_code == 200:
            return response.json()
        return None
    except Exception as e:
        print(f"Error fetching policy {policy_id}: {e}")
        return None

def save_storage_policies_enhanced(db, base_url, headers):
    """Enhanced storage policy collection with details"""
    # First, get list of all policies
    response = requests.get(f"{base_url}/StoragePolicy", headers=headers)
    policies_json = response.json()

    policies_list = policies_json.get("policies", [])

    for policy_entry in policies_list:
        storage_policy = policy_entry.get("storagePolicy", {})
        policy_id = storage_policy.get("storagePolicyId")
        policy_name = storage_policy.get("storagePolicyName", "")
        num_streams = policy_entry.get("numberOfStreams", 0)

        # Fetch detailed information
        details = fetch_storage_policy_details(base_url, headers, policy_id)

        # Extract copy information
        if details and "copy" in details:
            for copy in details["copy"]:
                # Save each copy configuration
                save_policy_copy_to_db(db, policy_id, copy)

        # Save main policy info
        cur.execute("""
            REPLACE INTO storage_policies (
                storagePolicyId, storagePolicyName, numberOfStreams,
                description, type, isDefault, lastFetchTime
            ) VALUES (?, ?, ?, ?, ?, ?, ?)
        """, (policy_id, policy_name, num_streams, ...))
```

Priority 3: Implement Plan Data Collection ⭐⭐

Plans are the modern approach and should be collected alongside policies.

```
def save_plans_to_db(db, plans_json):
    """Save Plan data to database"""
    cur = db.cursor()
    fetch_time = datetime.now().isoformat()

    plans_list = plans_json.get("plans", [])

    for plan_entry in plans_list:
        plan_info = plan_entry.get("plan", {})
        plan_id = plan_info.get("planId")
        plan_name = plan_info.get("planName", "")

        # Summary info
        description = plan_entry.get("description", "")
        plan_type = plan_entry.get("type", 0)
        subtype = plan_entry.get("subtype", 0)
        num_copies = plan_entry.get("numCopies", 0)
        num_entities = plan_entry.get("numAssocEntities", 0)
        rpo_minutes = plan_entry.get("rpoInMinutes", 0)
        storage_target = plan_entry.get("storageTarget", "")

        # Flags
        is_elastic = plan_entry.get("isElastic", False)
        status_flag = plan_entry.get("planStatusFlag", 0)

        # Associated storage policy
        storage = plan_entry.get("storage", {})
        storage_policy_info = storage.get("storagePolicy", {})
        storage_policy_id = storage_policy_info.get("storagePolicyId", None)

        cur.execute("""
            REPLACE INTO plans (
                planId, planName, description, type, subtype,
                numCopies, numAssocEntities, rpoInMinutes,
                storageTarget, storagePolicyId, isElastic,
                statusFlag, lastFetchTime
            ) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
        """, (
            plan_id, plan_name, description, plan_type, subtype,
            num_copies, num_entities, rpo_minutes, storage_target,
            storage_policy_id, is_elastic, status_flag, fetch_time
        ))
```

Part 6: Database Schema Updates Needed

Enhanced Storage Pools Table

```
CREATE TABLE IF NOT EXISTS storage_pools (
    storagePoolId INTEGER PRIMARY KEY,
    storagePoolName TEXT,
    storageType INTEGER,              -- NEW: 1=disk, 2=cloud, 3=tape
    storageSubType INTEGER,           -- NEW
    storagePoolType INTEGER,
    totalCapacity INTEGER,            -- Changed to INTEGER for MB
    freeSpace INTEGER,                -- Changed to INTEGER for MB
    sizeOnDisk INTEGER,               -- NEW: Actual disk usage
    status TEXT,                      -- NEW: Online/Offline
    statusCode INTEGER,               -- NEW
    isArchiveStorage INTEGER,         -- NEW: Boolean 0/1
    wormStorageFlag INTEGER,          -- NEW: WORM enabled 0/1
    numberOfNodes INTEGER,            -- NEW
    retentionPeriodDays INTEGER,      -- NEW
    cloudStorageClass TEXT,           -- NEW
    cloudStorageClassNumber INTEGER,  -- NEW
    libraryVendorType INTEGER,        -- NEW
    storagePolicyName TEXT,           -- NEW: Associated policy
    storagePolicyId INTEGER,          -- NEW: Policy ID
    lastFetchTime TEXT
);
```

Enhanced Storage Policies Table

```
CREATE TABLE IF NOT EXISTS storage_policies (
    storagePolicyId INTEGER PRIMARY KEY,
    storagePolicyName TEXT,
    numberOfStreams INTEGER,          -- NEW
    description TEXT,                 -- NEW
    type INTEGER,                     -- NEW
    isDefault INTEGER,                -- NEW: Boolean 0/1
    numCopies INTEGER,                -- NEW
    flags INTEGER,                    -- NEW
    lastFetchTime TEXT
);
```

New Table: Storage Policy Copies

```
CREATE TABLE IF NOT EXISTS storage_policy_copies (
    copyId INTEGER PRIMARY KEY,
    storagePolicyId INTEGER,          -- Foreign key
    copyName TEXT,
    copyType INTEGER,                 -- 1=Primary, 2=Secondary
    copyPrecedence INTEGER,
    active INTEGER,                   -- Boolean 0/1
    isDefault INTEGER,                -- Boolean 0/1
    isArchiveStorage INTEGER,         -- Boolean 0/1
    isSnapCopy INTEGER,               -- Boolean 0/1
    isMirrorCopy INTEGER,             -- Boolean 0/1
    wormStorageFlag INTEGER,          -- Boolean 0/1
    storagePoolId INTEGER,
    storagePoolName TEXT,
    libraryId INTEGER,
    libraryName TEXT,
    mediaAgentName TEXT,
    enableDeduplication INTEGER,      -- Boolean 0/1
    enableClientSideDedup INTEGER,    -- Boolean 0/1
    retainBackupDataForDays INTEGER,
    retainBackupDataForCycles INTEGER,
    enableDataAging INTEGER,          -- Boolean 0/1
    lastFetchTime TEXT,
    FOREIGN KEY(storagePolicyId) REFERENCES storage_policies(storagePolicyId)
);
```

New Table: Plans

```
CREATE TABLE IF NOT EXISTS plans (
    planId INTEGER PRIMARY KEY,
    planName TEXT,
    description TEXT,
    type INTEGER,                     -- Plan type code
    subtype INTEGER,                  -- Plan subtype code
    numCopies INTEGER,
    numAssocEntities INTEGER,         -- Entities using this plan
    numUsers INTEGER,
    numDevices INTEGER,
    rpoInMinutes INTEGER,             -- Recovery Point Objective
    slaInMinutes INTEGER,             -- Service Level Agreement
    storageTarget TEXT,               -- Primary storage target
    storagePolicyId INTEGER,          -- Associated policy
    isElastic INTEGER,                -- Boolean 0/1
    statusFlag INTEGER,
    restrictions INTEGER,
    lastFetchTime TEXT,
    FOREIGN KEY(storagePolicyId) REFERENCES storage_policies(storagePolicyId)
);
```

Part 7: UI Presentation Recommendations

Storage Pools - Enhanced View

Create a dedicated Storage Pools page similar to MediaAgents:

File: `templates/storagepools.html`

Features: - **Two-column layout**: List + detail panel (like MediaAgents) - **Clickable rows**: Select pool to view full details - **Visual indicators**: - Online/Offline status badges - Capacity bar graphs (used vs total) - Cloud storage tier badges - WORM protection indicator - Archive storage flag - **Detail panel shows**: - Pool ID and name - Storage type (Disk/Cloud/Tape) - Capacity metrics with visual bars - Node count - Associated policy - Retention period - Cloud storage class (if applicable) - Deduplication status

Storage Policies - New View

Create a comprehensive Storage Policy view:

File: `templates/storagepolicies.html`

Features: - **Master-detail layout**: - Left: List of all policies - Right: Selected policy details - **Policy list shows**: - Policy name - Number of copies - Number of streams - Default policy indicator - **Detail panel shows**: - Policy configuration - Copy breakdown (Primary, Secondary, Archive) - Retention rules for each copy - Deduplication settings - Associated storage pools - Libraries used - MediaAgents involved

Plans - New View

File: `templates/plans.html`

Features: - **Card-based layout**: Each plan as a card - **Plan card shows**: - Plan name and description - Plan type (Server, Laptop, etc.) - RPO/SLA targets - Number of entities using plan - Number of copies configured - Storage target - Status indicators - **Click to expand**: Full plan configuration - **Visual elements**: - Progress bars for RPO compliance - Badge for plan type - Entity count - Copy configuration diagram

Dashboard Enhancements

Add to Infrastructure Dashboard:

1. **Storage Policies Summary Card**
2. Total policies count
3. Default policy name
4. Link to policies page
5. **Plans Summary Card**
6. Total plans count
7. Most used plan
8. Entities covered by plans
9. Link to plans page
10. **Storage Pool Capacity Chart**

11. Visual chart showing pool usage
12. Total capacity vs used capacity
13. Top pools by usage percentage

Part 8: Implementation Steps

Step 1: Update Database Schema

```
# Add new columns to storage_pools table
# Create storage_policy_copies table
# Create plans table
# Run schema updates
```

Step 2: Enhance Storage Pool Data Collection

```
# Update save_storage_pools_to_db() function
# Add all new fields from JSON response
# Test with existing data
```

Step 3: Implement Policy Details Collection

```
# Create fetch_storage_policy_details() function
# Update save_storage_to_db() to call details API
# Create save_policy_copy_to_db() for copy configurations
# Add to fetch_data route
```

Step 4: Implement Plans Collection

```
# Create save_plans_to_db() function
# Add /Plan endpoint to fetch_data route
# Add plans checkbox to home page
# Test API call
```

Step 5: Create UI Views

```
# Create templates/storagepools.html (enhanced)
# Create templates/storagepolicies.html
# Create templates/plans.html
# Add routes to app.py
# Update navigation links
```

Step 6: Update Dashboard

```
# Add policy and plan summaries
# Add storage capacity visualizations
# Update infrastructure_dashboard route
```

Part 9: API Endpoints Reference

Endpoint	Method	Purpose	Status
`/StoragePool`	GET	List all storage pools	✅ Implemented
`/StoragePool/{id}`	GET	Get pool details	⚠️ Possible enhancement
`/StoragePolicy`	GET	List all policies	✅ Implemented (basic)
`/StoragePolicy/{id}`	GET	Get policy details	❌ Not implemented

Endpoint	Method	Purpose	Status
`/Plan`	GET	List all plans	❌ Not implemented
`/Plan/{id}`	GET	Get plan details	❌ Not implemented
`/V4/StoragePool`	GET	V4 API pools (may not be available)	❌ Not available

Part 10: Expected Data Volumes

Based on your test data:

- **Storage Pools**: ~8-50 pools (typical environment)
- **Storage Policies**: ~100-500 policies
- **Plans**: ~50-200 plans
- **Policy Copies**: ~200-1000 copies (2-3 per policy average)

Database Impact: Minimal - all data combined should be <10MB

Conclusion

Current State Summary

Component	Data Pulled	Data Saved	Data Presented	Quality
Storage Pools	✅ Yes	✅ Yes	✅ Yes	⚠️ Basic

Component	Data Pulled	Data Saved	Data Presented	Quality
Storage Policies	✅ Yes	⚠️ Minimal	✅ Yes	❌ Poor
Plans	❌ No	❌ No	❌ No	❌ Not implemented

Recommendations Priority

1. 🔴 **HIGH**: Enhance Storage Pool data collection (add all available fields)
2. 🔴 **HIGH**: Implement Storage Policy details collection
3. 🟡 **MEDIUM**: Implement Plans collection
4. 🟡 **MEDIUM**: Create dedicated UI views for policies and plans
5. 🟢 **LOW**: Add capacity visualization charts

Next Actions

Choose one of these paths:

Path A - Quick Enhancement (1-2 hours): - Enhance storage pool database schema - Update save_storage_pools_to_db() to capture all fields - Update storage pools view to display new fields

Path B - Comprehensive Implementation (4-6 hours): - Implement all database schema changes - Add policy details collection - Add plans collection - Create new UI views for all components

Path C - Incremental (ongoing): - Phase 1: Storage pools enhancement - Phase 2: Storage policy details - Phase 3: Plans implementation - Phase 4: UI improvements