Commvault Job Schedule Data Research

Date: 2025-11-14 **Purpose:** Research how to pull job schedule data from Commvault REST API

⚠️ **IMPORTANT: READ-ONLY PROJECT** - This project is **explicitly READ-ONLY** - **ONLY GET requests** are allowed - **NO PUT, POST, PATCH, or DELETE operations** - Data retrieval and viewing ONLY

API Endpoints Discovered

Based on Commvault REST API documentation research, the following endpoints are available for retrieving schedule data:

1. Get All Schedules

Endpoint: `GET /Schedules`

Purpose: Returns all of the schedules for a client

Query Parameters: - `clientId` (optional) - Filter by specific client - `admin` (optional) - Include admin/system schedules (0 or 1) - `storagepolicyId` (optional) - Filter by storage policy

Example Request:

```
GET {baseUrl}/Schedules?admin=1
GET {baseUrl}/Schedules?storagepolicyId=3&admin=1
```

Response Structure:

```
{
  "taskDetail": [
    {
      "task": {
        "taskId": 0,
        "taskName": "string",
        "taskType": 0,
        "policyType": 0
      },
      "associations": [
        {
          "subclientId": 0,
          "storagePolicyId": 0,
          "copyId": 0,
          "clientId": 0,
          "clientGroupId": 0,
          "applicationId": 0,
          "libraryId": 0,
          "backupsetId": 0,
          "instanceId": 0,
          "workflowId": 0,
          "trackingPolicyId": 0
        }
      ],
      "subTasks": [
        {
          "subTask": {
            "subTaskName": "string",
            "subTaskType": 0,
            "operationType": 0,
            "options": {
              "backupOpts": {
                "backupLevel": 0
              }
            }
          }
```

```
        }
      ]
    }
  ]
}
```

2. Get Schedule Properties

Endpoint: `GET /Schedules/{scheduleId}`

Purpose: Retrieves detailed properties of a specific schedule

Path Parameters: - `scheduleId` (required) - The ID of the schedule

Example Request:

```
GET {baseUrl}/Schedules/121
```

Response Contains: - Schedule name and ID - Task type and operation type - Schedule pattern (frequency, time, days) - Associated entities (client, subclient, etc.) - Next run time - Schedule status (enabled/disabled)

3. Get Schedule Policies

Endpoint: `GET /SchedulePolicy`

Purpose: Returns a list of schedule policies (READ-ONLY)

Query Parameters: - `operationType` (optional) - Filter by operation type (backup, aux copy, etc.) - `entityType` (optional) - Filter by entity type - `scheduleId` (optional) - Get specific schedule policy

Example Request:

```
GET {baseUrl}/SchedulePolicy
GET {baseUrl}/SchedulePolicy?operationType=2
```

Note: This endpoint is for viewing schedule policies only. No modifications are supported in this read-only project.

Schedule Data Fields

Task Information

Field	Type	Description
taskId	Integer	Unique identifier for the scheduled task
taskName	String	Name of the scheduled task
taskType	Integer	Type of task (backup, aux copy, workflow, etc.)
policyType	Integer	Policy type code

Schedule Pattern Fields

Field	Type	Description
scheduleId	Integer	Unique schedule identifier
scheduleName	String	Name of the schedule
schedulePattern	Object	Contains frequency and timing information
freq_type	Integer	Frequency type (daily, weekly, monthly)
active_start_time	Integer	Start time of day (in seconds from midnight)
active_end_time	Integer	End time of day

Field	Type	Description
freq_interval	Integer	Interval between occurrences
freq_recurrence_factor	Integer	Recurrence factor

Association Fields

Field	Type	Description
clientId	Integer	Associated client ID
clientName	String	Associated client name
subclientId	Integer	Associated subclient ID
subclientName	String	Associated subclient name
backupsetId	Integer	Associated backup set ID
instanceId	Integer	Associated instance ID
storagePolicyId	Integer	Associated storage policy ID

Operation Types

Code	Description
1	Full Backup
2	Incremental Backup
3	Differential Backup
4	Synthetic Full

Code	Description
5	Auxiliary Copy
1001	Data Aging
1002	DDB Verification

Task Types

Code	Description
1	Backup
2	Restore
3	Auxiliary Copy
4	Workflow
5	Data Recovery
6	Data Aging
7	Report

Common Schedule API Patterns

Pattern 1: Get All Schedules for Infrastructure

```
GET /Schedules?admin=1
```

Returns all schedules including system/admin schedules.

Pattern 2: Get Schedules for Specific Client

```
GET /Schedules?clientId=123
```

Returns schedules associated with a specific client.

Pattern 3: Get Schedule Details

```
GET /Schedules/{scheduleId}
```

Gets detailed information about a specific schedule including next run time.

Pattern 4: Get Schedules by Storage Policy

```
GET /Schedules?storagepolicyId=50
```

Gets schedules associated with a specific storage policy.

Implementation Approach

Option 1: Collect All Schedules

Pros: - Complete picture of all scheduled jobs - Can correlate with clients, policies, plans

Cons: - May be large dataset - Includes system schedules that may not be relevant

Implementation:

```
def fetch_all_schedules(base_url, headers):
    response = requests.get(f"{base_url}/Schedules?admin=1", headers=headers)
    return response.json()
```

Option 2: Collect Schedules Per Client

Pros: - More targeted data collection - Easier to associate with specific clients

Cons: - Requires iterating through all clients - More API calls needed

Implementation:

```
def fetch_client_schedules(base_url, headers, client_id):
    response = requests.get(f"{base_url}/Schedules?clientId={client_id}", headers=hea
    return response.json()
```

Option 3: Collect Schedule Policies (Recommended)

Pros: - Policy-based view (aligns with Plans/Policies) - Less granular, more manageable - Shows schedule configuration patterns

Cons: - May not show all individual schedule instances

Implementation:

```
def fetch_schedule_policies(base_url, headers):
    response = requests.get(f"{base_url}/SchedulePolicy", headers=headers)
    return response.json()
```

Database Schema Design

Schedules Table

```
CREATE TABLE IF NOT EXISTS schedules (
    scheduleId          INTEGER PRIMARY KEY,
    scheduleName        TEXT,
    taskId              INTEGER,
    taskName            TEXT,
    taskType            INTEGER,
    operationType       INTEGER,

    -- Association IDs
    clientId            INTEGER,
    clientName          TEXT,
    subclientId         INTEGER,
    subclientName       TEXT,
    backupsetId         INTEGER,
    instanceId          INTEGER,
    storagePolicyId     INTEGER,

    -- Schedule Pattern
    freqType            INTEGER,
    activeStartTime     INTEGER,
    activeEndTime       INTEGER,
    freqInterval        INTEGER,

    -- Status
    enabled             INTEGER,
    nextRunTime         TEXT,
    lastRunTime         TEXT,

    -- Metadata
    lastFetchTime       TEXT,

    FOREIGN KEY (clientId) REFERENCES clients(clientId),
    FOREIGN KEY (storagePolicyId) REFERENCES storage_policies(storagePolicyId)
);
```

Schedule Policies Table (Alternative)

```
CREATE TABLE IF NOT EXISTS schedule_policies (
    policyId            INTEGER PRIMARY KEY AUTOINCREMENT,
    scheduleName        TEXT,
```

```
    taskType            INTEGER,
    operationType       INTEGER,
    policyType          INTEGER,

    -- Pattern
    schedulePattern     TEXT,       -- JSON string with full pattern
    frequency           TEXT,       -- Daily/Weekly/Monthly
    startTime           TEXT,       -- HH:MM format

    -- Associations count
    clientCount         INTEGER,
    subclientCount      INTEGER,

    -- Metadata
    lastFetchTime       TEXT
);
```

Integration with Existing Data

Link Schedules to Plans

Plans often have associated schedules. When collecting Plan data, we can extract schedule information:

```
# In save_plans_to_db() function
plan_schedule = storage.get("schedule", {})
if plan_schedule:
    # Extract schedule details
    schedule_pattern = plan_schedule.get("schedulePattern", {})
    # Save to schedules table
```

Link Schedules to Clients

Schedules are typically associated with clients/subclients:

```
# Query to find all schedules for a client
SELECT * FROM schedules WHERE clientId = ?
```

Link Schedules to Storage Policies

Schedules can be filtered by storage policy:

```
# Query schedules by policy
SELECT * FROM schedules WHERE storagePolicyId = ?
```

Testing Approach

Step 1: Test Basic Endpoint

```
# Test if /Schedules endpoint is available
import requests
response = requests.get(f"{base_url}/Schedules?admin=1", headers=headers)
print(response.status_code)
print(response.json())
```

Step 2: Test with Filters

```
# Test with client filter
response = requests.get(f"{base_url}/Schedules?clientId=123", headers=headers)

# Test with storage policy filter
response = requests.get(f"{base_url}/Schedules?storagepolicyId=50", headers=headers)
```

Step 3: Test Schedule Details

```
# Get specific schedule details
response = requests.get(f"{base_url}/Schedules/121", headers=headers)
print(response.json())
```

Next Steps

1. ✅ **Research Complete** - Documented available endpoints and data structure
2. ⏳ **Test Endpoints** - Test /Schedules endpoint with current Commvault setup
3. ⏳ **Analyze Response** - Save test response and analyze actual data structure
4. ⏳ **Design Database Schema** - Create appropriate tables for schedule data
5. ⏳ **Implement Collection** - Add schedule data collection to app.py
6. ⏳ **Create UI Views** - Display schedules in web interface
7. ⏳ **Link to Existing Data** - Associate schedules with clients, policies, plans

---

Useful Resources

- **Commvault API Docs:** https://api.commvault.com/
- **Schedule Operations:** https://api.commvault.com/docs/SP38/api/cv/ScheduleandSchedulePolicyOperations/
- **REST API Reference:** https://documentation.commvault.com/v11/essential/rest_api_reference.html

---

Known Limitations

1. **Version Dependency:** Schedule API structure may vary by Commvault version
2. **Permissions:** May require specific API permissions to view schedules
3. **System Schedules:** admin=1 flag needed to see system-created schedules
4. **Data Volume:** Large environments may have thousands of schedules

---

Schedule Data Use Cases

1. Schedule Compliance Reporting

- Identify clients without scheduled backups
- Find backup windows and frequencies
- Verify backup coverage

2. Schedule Conflict Detection

- Identify overlapping backup windows
- Find resource contention issues
- Optimize backup scheduling

3. Schedule Analysis

- Most common backup times
- Schedule patterns by client type
- Backup frequency distribution

4. Operational Monitoring

- Next scheduled jobs
- Overdue schedules

- Disabled schedules requiring attention

Status: Research Complete ✅ **Next Action:** Test /Schedules endpoint with actual Commvault environment