Commvault Data Retrieval Web Application

A Flask-based web application that connects to the Commvault REST API and stores retrieved data in an SQLite database. This self-contained tool provides a simple web interface for configuring API connections and fetching various types of Commvault data.

Features

- **Configurable Connection**: Enter Commvault Web Service URL, username, and password via web form or configuration file
- **Token-Based Authentication**: Securely authenticates with Commvault API using Base64-encoded credentials
- **Multiple Data Types**: Retrieve Clients, Jobs, Plans (Policies), and Storage Policies
- **Infrastructure Visibility**: NEW! View MediaAgents, Storage Pools, Libraries, Hypervisors, and Storage Arrays
- **Interactive Dashboard**: NEW! Visual infrastructure overview with health status and capacity monitoring
- **SQLite Storage**: All data is stored locally in a structured SQLite database
- **Web Interface**: Clean, modern UI for easy data retrieval and viewing
- **Data Preview**: View retrieved data directly in the browser
- **Database Viewer**: Browse stored data organized by type

Project Structure

```
Commvault_API/
│
├── app.py                  # Main Flask application
├── config.ini             # Configuration file (API credentials)
├── requirements.txt       # Python dependencies
├── README.md             # This file
│
├── Database/
│   └── commvault.db      # SQLite database (created on first run)
│
└── templates/
    ├── base.html         # Base template with styling
    ├── index.html        # Home page with configuration form
    ├── results.html      # Data retrieval results page
    └── view.html         # Database viewer page
```

Prerequisites

- Python 3.7 or higher
- Network access to Commvault Web Server
- Valid Commvault credentials

Installation

1. **Clone or download this project**
2. **Install dependencies** `bash pip install -r requirements.txt`
3. **Configure connection settings** (optional)

Edit `config.ini` with your Commvault details: ```ini [commvault] base_url = http://your-server:81/SearchSvc/CVWebService.svc username = your_username@domain.com password = Base64EncodedPassword==

[database] db_path = Database/commvault.db ```

Note: Password can be plaintext (will be Base64-encoded automatically) or pre-encoded.

Usage

Starting the Application

Run the Flask development server:

```
python app.py
```

The application will start on `http://localhost:5000`

Using the Web Interface

1. **Open your browser** and navigate to `http://localhost:5000`
2. **Enter connection details**:
3. Commvault Base URL (e.g., `http://commvaultweb01.jhb.seagatestoragecloud.co.za:81/SearchSvc/CVWebService.svc`)
4. Username (e.g., `guys@storvault.co.za`)
5. Password (plaintext or Base64-encoded)
6. **Select data types** to retrieve:

Basic Data: - **Clients**: All client machines in the CommCell - **Jobs**: Backup and restore jobs - **Plans**: Backup plans and policies - **Storage**: Storage policy configurations

Infrastructure & Hardware (NEW!): - **MediaAgents**: Backup infrastructure servers with capacity info - **Storage Pools**: Disk/tape storage pools with deduplication status - **Libraries**: Tape and disk libraries - **Hypervisors**: VM infrastructure (VMware, Hyper-V, etc.) - **Storage Arrays**: Physical storage hardware

1. **Click "Fetch Data"** to retrieve and store the data
2. **View results** in the browser or navigate to view pages:
3. **Infrastructure Dashboard**: Visual overview of all infrastructure components
4. View Clients

5. View Jobs (latest 100)
6. View MediaAgents
7. View Storage Pools
8. View Libraries
9. View Hypervisors

Commvault API Integration

API Endpoints Used

The application connects to the following Commvault REST API endpoints:

Data Type	Endpoint	Description
Authentication	`POST /Login`	Authenticates and retrieves auth token
Clients	`GET /Client`	Lists all clients in CommCell
Jobs	`GET /Job`	Lists backup/restore jobs
Plans	`GET /Plan`	Lists all plans/policies
Storage	`GET /V2/StoragePolicy`	Lists storage policies

Authentication Flow

1. Password is Base64-encoded (if not already)
2. POST request to `/Login` with credentials
3. Token extracted from response
4. Token used in `Authtoken` header for subsequent requests

Example API Call

```
# Login
POST http://server:81/SearchSvc/CVWebService.svc/Login
{
    "username": "user@domain.com",
    "password": "Base64EncodedPassword"
}

# Get Clients
GET http://server:81/SearchSvc/CVWebService.svc/Client
Headers: {
    "Accept": "application/json",
    "Authtoken": "retrieved_token"
}
```

Database Schema

The SQLite database contains four main tables:

clients

- `clientId` (INTEGER PRIMARY KEY)
- `clientName` (TEXT)
- `hostName` (TEXT)
- `clientGUID` (TEXT)
- `lastFetchTime` (TEXT)

jobs

- `jobId` (INTEGER PRIMARY KEY)
- `clientId` (INTEGER)
- `clientName` (TEXT)
- `jobType` (TEXT)
- `status` (TEXT)
- `startTime` (TEXT)

- `endTime` (TEXT)
- `backupSetName` (TEXT)
- `lastFetchTime` (TEXT)

plans

- `planId` (INTEGER PRIMARY KEY)
- `planName` (TEXT)
- `planType` (TEXT)
- `lastFetchTime` (TEXT)

storage_policies

- `storagePolicyId` (INTEGER PRIMARY KEY)
- `storagePolicyName` (TEXT)
- `lastFetchTime` (TEXT)

Application Routes

Route	Method	Description
`/`	GET	Home page with configuration form
`/fetch`	POST	Fetch data from Commvault API
`/view/<data_type>`	GET	View stored data by type

Features in Detail

Error Handling

- Connection errors are caught and displayed to the user

- Authentication failures show clear error messages
- API timeout after 30 seconds
- Database errors are handled gracefully

Data Management

- Uses `REPLACE INTO` for upserts (prevents duplicates)
- Tracks last fetch time for each record
- Automatic database creation on first run
- Database connection pooling via Flask's `g` object

Security Considerations

- Password can be stored as Base64 in config (not plaintext)
- Secret key should be changed in production
- Consider using environment variables for sensitive data
- Do not commit `config.ini` with real credentials to version control

Troubleshooting

Common Issues

1. Authentication Failed - Verify the base URL is correct and accessible - Check username and password are valid - Ensure network connectivity to Commvault server

2. No Data Retrieved - Confirm the selected data types exist in your CommCell - Check API permissions for your user account - Review error messages in the browser

3. Database Errors - Ensure write permissions for `Database/` directory - Delete `commvault.db` to reset database - Check disk space availability

Development

Running in Debug Mode

The application runs in debug mode by default when using `python app.py`. For production, use a WSGI server:

```
pip install gunicorn
gunicorn -w 4 -b 0.0.0.0:5000 app:app
```

Modifying the Schema

If you need to add fields to the database:

1. Update the `init_db()` function in `app.py`
2. Modify the corresponding `save_*_to_db()` function
3. Delete the existing database or use SQL ALTER statements

API Documentation References

- Commvault REST API Documentation
- Available Web Services
- REST API Authentication
- GET Client API
- GET Job API

License

This project is provided as-is for educational and development purposes.

Support

For issues related to: - **Commvault API**: Consult Commvault documentation or support - **This Application**: Review error messages and check configuration

Author

Created as a self-contained tool for Commvault data retrieval and analysis.

Note: This application is designed for local development and testing. For production use, implement additional security measures, proper authentication, HTTPS, and production-ready database solutions.