

# MODERNIZE DATA PROTECTION WITH AUTOMATION



## Raghava Jainoje

Advisory Systems Engineer  
Dell Technologies

## Mike van der Steen

Principal Systems Engineer  
Dell Technologies

The Dell Technologies Proven Professional Certification program validates a wide range of skills and competencies across Dell's multiple technologies and products with both skill and outcome-based certifications.

Proven Professional exams cover concepts and principles which enable professionals working in or looking to begin a career in IT. With training and certifications aligned to the rapidly changing IT landscape, learners can take full advantage of the essential skills and knowledge required to drive better business performance and foster more productive teams.

Proven Professional certifications include skills and solutions such as:

- Data Protection
- Converged and Hyperconverged Infrastructure
- Cloud and Elastic Cloud
- Networking
- Security
- Servers
- Storage
- ...and so much more.

Courses are offered to meet different learning styles and schedules, including self-paced On Demand, remote-based Virtual Instructor-Led and in-person Classrooms.

Whether you are an experienced IT professional or just getting started, Dell Technologies Proven Professional certifications are designed to clearly signal proficiency to colleagues and employers.

## Table of Contents

Why Leverage Automation .....	4
Automation Protocols and Languages .....	5
Deep Dive into REST .....	6
REST API Response.....	7
Commonly Used Tool Sets.....	8
Automation with PowerProtect Data Manager .....	10
PPDM's REST API, Getting Started .....	12
Postman and Postman Collections.....	12
PowerShell.....	15
cURL.....	15
Ansible .....	16
Python .....	17
The Response from REST API Requests.....	18
Tool Set Summary .....	20
Backup Administrator Use Cases .....	21
Defining the Logical Process.....	21
Adding PowerProtect DD Series Appliance.....	21
Adding vCenter .....	22
Creating Protection Policies .....	22
Installing File System Agents .....	23
Adding Clients to a Protection Policy.....	23
Application Owner Automation Use-Cases .....	25
Adding Clients.....	25
Self-service Backup .....	25
Self-service Restore – VM restore to New VM .....	26
Advanced use case .....	27
VMware VM Backup Validation .....	27
End-to-end Solution.....	29
ServiceNow and other Self-Service Portals.....	29
External data loggers .....	30
Ticketing .....	31
Conclusion.....	32
Bibliography.....	33

## Why Leverage Automation

With time, every industry evolves. The Information Technology (IT) sector is no exception with current transformation efforts being heavily focused on digitalization. This transformation has most certainly accelerated over the last few years, in part due to the COVID-19 pandemic forcing organizations to change how they interact with their customers whilst needing to quickly adopt a digital approach. Over this period, many applications have and are still being modernized, leveraging new technologies and approaches. Using agile development methods that enable new updates or features to be developed, tested, and released to production more quickly, to leveraging containerization of applications vs the traditional server-based approach.

Application developers and Information Technology<sup>1</sup> (IT) administrators have limited resources available to them and maximizing the output is a key objective. Achieving this can be challenging, however elements of the IT environment can be optimized to help achieve such objectives. One such aspect is to reduce the effort spent on manual and repetitive tasks through automation. For example, automation can be used in testing and deployment of applications, management of systems, optimizing operational oversight and enabling self-service capabilities. The time and effort saved through automation allows for it to be re-invested towards high value projects and activities.

Regardless of an organization's size, there is a finite number of resources it can invest into IT environments and infrastructure teams are often overwhelmed with activities. Within the Infrastructure business unit, the Data Protection team is often understaffed and busy, spending a lot of its time on repetitive tasks, which include ad hoc backup and restore requests, configuration tasks and monitoring. Automation can also assist this team; however, it is typically the least automated aspect of any IT environment.

While automating repetitive tasks saves time, automation can also be used in a declarative approach for configuration changes to an existing application. For data protection, this can include configuration tasks which include installation of a backup agent, creation of a protection policy or registration of a new client. Taking this declarative approach, the process of configuration is performed through scripting, with variables required for the tasks being provided by an application owner or IT administrator. Automation of individual basic tasks can be easily achieved; however, their real value can be realized when they are made available as a self-service capability and further extended into an end-to-end process via web portals and ticketing applications.

Automation is a term easy to mention, but not always easy to implement. Often the difficulty of where or how to start can prevent it from being realized. In this article, an overview of automation is provided which includes details about industry leading languages and tools, how they can be used to achieve a tangible outcome through standalone commands or via a scripted approach. Sample scripts and use cases will be provided and will focus on Dell PowerProtect Data Manager<sup>2</sup> (PPDM) as it has an extensive RESTful<sup>3</sup> API framework that data protection administrators can leverage.

IT Infrastructure has moved from physical systems to virtual and is now treated as a programmable infrastructure platform where systems are deployed declaratively via code. To align with that trend, the backup systems need to adopt modern protocol support, such as REST API, to communicate with different systems and automate repetitive tasks like ad hoc backup & restore and commissioning and decommissioning of backup clients.

The aim of this article is to empower an IT administrator to have a better understanding of automation and how it can be implemented into their environment. Ultimately, the extent of automation introduced into an environment will vary between organizations, however, the use of automation will lead to a more efficient and modernized IT environment.



## Automation Protocols and Languages

The use of automation in IT environments is not new and it has been used by many IT professionals to allow for repetitive tasks to be performed either in an ad hoc or scheduled manner. This can be as simple as using a Linux Shell<sup>4</sup> or Windows Batch<sup>5</sup> scripts to run a set of commands on the local server or interact with remote servers. Running Shell and Batch scripts have served the IT community well; however, they are proprietary to the application and operating system, so the output is in an unstructured format.

These automation scripts are limited to Linux, Windows or MacOS servers and for clients within a data center and with the adoption of cloud-based services, this has an impact on adopting automation with infrastructure now spanning between data centers and across the internet to hosted services.

Automation with scripts that leverage command line interfaces needed to evolve. One such example of this is evident with Batch script when Microsoft introduced PowerShell<sup>6</sup>. Microsoft PowerShell is a management platform used to automate tasks primarily on Windows systems (when it was first released in 2006); however since 2016 it has been further developed and made available as an open-source project (called PowerShell Core) to provide support for cross platform systems including Linux<sup>7</sup> and MacOS<sup>8</sup>.

With an option to now script with a single language across multiple different operating systems, it addressed one of the core issues, however, this does not address organizations running their business using applications and services hosted across multiple environments, including those hosted in cloud computing<sup>9</sup> platforms.

A more consistent method of communicating between different applications and operating systems, across the Internet, and with a data center is needed. That is where Representational State Transfer<sup>13</sup> (REST) comes into play.

Applications hosted on cloud computing are accessible via a connection across the Internet<sup>10</sup> using Uniform Resource Locator<sup>11</sup> (URL) as a reference for the application. That is then used by a client computer to connect to it. Connecting to a URL enables more than just information to be rendered on a web browser, it can also be used by computers in a client-server<sup>12</sup> architecture to communicate leveraging REST. The REST API is a modern and flexible architecture for automation, compared to traditional command line interface style scripting as shown in Figure 1.

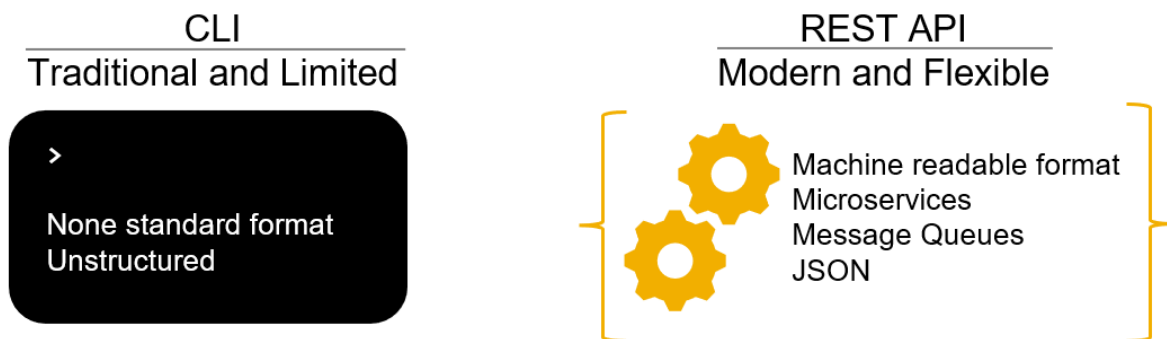


Figure 1 - Command line interface scripting vs REST API architecture

REST is a software architecture and is being widely adopted throughout the software industry as it provides a reliable way to communicate at scale, with high performance capability that is easily implemented and supported across a wide range of platforms. The information is shared in a JSON<sup>22</sup> format, which is machine readable and more importantly is compatible across different platforms. With the wide adoption of REST, it is ideal for automation solutions – it is easy to learn and interact with, hence the reason it is a large focus of this article.

## Deep Dive into REST

Before providing a deeper overview of REST, one additional component needs to be discussed which relates to the Application Programming Interface<sup>14</sup> (API). An API created by developers is exposed by the application and used to define the rules for communication to occur between a client computer and the application. APIs developed using the REST software architecture are referred to as “REST API,” and this enables communication between computers to be performed programmatically.

By leveraging REST APIs, IT professionals can leverage this protocol for automation purposes, whether the application being communicated with is located on the Internet or deployed within the organization’s own IT environment. Before diving into the details, a high-level overview of a REST API communication / architecture is provided in Figure 2.

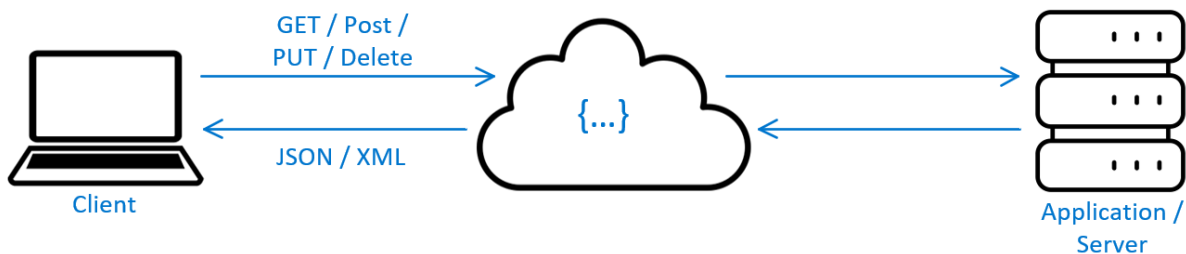


Figure 2 - High-level overview of REST API communication / architecture

There are several principles that relate to the REST architecture, including:

- **uniform Interface** – imposes that a request should identify the resource, that the client request contains sufficient information for the resource to process, that the client receives a self-descriptive response from the resource, and that this information contains all other related resources to complete a task
- **client-server** – with separation of the client and server in the design pattern, it enables each component to be developed and evolve independently
- **stateless** – every request sent to the resource includes all the information necessary to understand and fulfil the request
- **layered system** – the application is composed of hierarchical layers, each with limited component behavior that results in each component not seeing beyond the next immediate layer
- **cacheability** – requires that the response received be implicitly or explicitly labelled as being able to be cached for reuse or not
- **code on demand** – allows for clients receiving the response to extend their functionality by executing code in the form of applets or scripts

When a REST API call is made it requires that request to contain several components including the resource, method and HTTP headers. The server or application typically has multiple resources available via a REST API request and each resource has a unique identifier in the form of a URL. This URL is similar in format to a web site address that would be entered into a web browser and clearly specifies what the client requires. When the client initiates a request, a Hypertext Transfer Protocol<sup>15</sup> (HTTP) method is specified, and they include:

- **GET** – used to access the resource on the server and can be cached
- **POST** – used to send data to the server
- **PUT** – used to update existing resource on the server
- **DELETE** – used to request that a resource is removed from the server

The final component of a REST API request is the HTTP headers which include the metadata exchanged between the client and server. For example, this can include information about the format of the request and the corresponding response, along with data that is used for the corresponding HTTP method.

For information to be exchanged, the client must authenticate the request. This is achieved using HTTP authentication, using basic authentication where the username and password are sent encoded with base64<sup>16</sup> or using a bearer authentication method where a token in the form of an encrypted string of characters is used. For highly secure requests, authentication can be achieved using Open Authorization<sup>21</sup> (oAuth), which combines passwords and tokens.

## REST API Response

When a REST API request is sent to the server, the response will contain three main components which include the status line, the message body, and headers. The status line contains a three-digit status code which informs the client of success or failure. The message body contains the response of the request in the format of JSON<sup>22</sup> or XML.<sup>23</sup> The headers provide more context about the response. Once the message body is received by the client it can then be processed for the following action or instruction as required.

To provide a very simple example of a REST API call response, a GET request was made to obtain the current time in Sydney, Australia using the end point of <http://worldtimeapi.org/api/timezone/australia/sydney> with Postman<sup>17</sup>, a response was received in a JSON format as detailed in Figure 3.

```
{
  "abbreviation": "AEDT",
  "client_ip": "xx.xx.xx.xx",
  "datetime": "2023-01-03T15:18:25.054647+11:00",
  "day_of_week": 2,
  "day_of_year": 3,
  "dst": true,
  "dst_from": "2022-10-01T16:00:00+00:00",
  "dst_offset": 3600,
  "dst_until": "2023-04-01T16:00:00+00:00",
  "raw_offset": 36000,
  "timezone": "Australia/Sydney",
  "unixtime": 1672719505,
  "utc_datetime": "2023-01-03T04:18:25.054647+00:00",
  "utc_offset": "+11:00",
  "week_number": 1
}
```

Figure 3 - JSON output from API call for the current time in Sydney

From an end user point of view, receiving the information in a JSON format is not practical and converting this information to a more friendly format is what applications perform. For example, a weather application on a mobile device uses REST API calls to require information from a weather system, process the information and display it in an easy to consume format on the mobile device as shown in Figure 4.

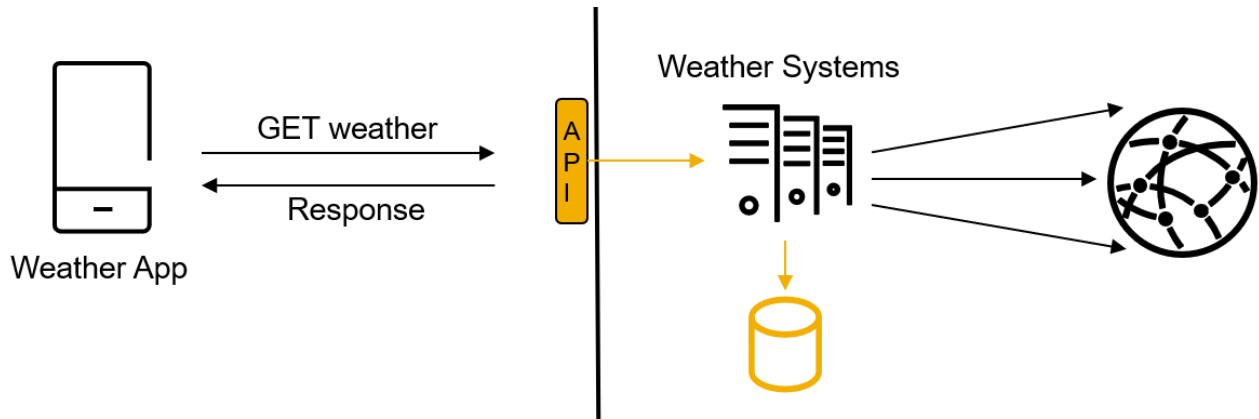


Figure 4 - An example data flow for a weather application on a mobile device

A more detailed look at the components and responses of a REST API request will become clear through the examples provided in the Automation with PowerProtect Data Manager section.

### Commonly Used Tool Sets

The choice of automation tool sets which are available for use can be overwhelming. Knowing which one to implement may not be easily determined. These tools sets provide the framework and management to enable automation scripts to be created, run, and maintained. Based on experience and what is commonly used in organizations, the following are some of the more commonly used tool sets as shown in Figure 5.

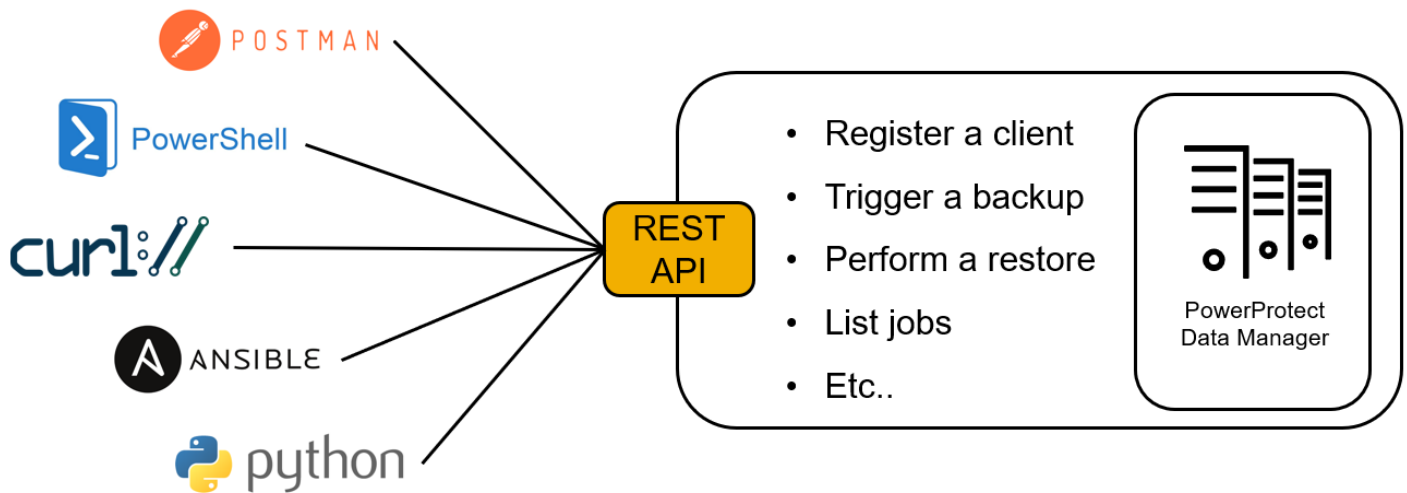


Figure 5 - Commonly used tool sets for REST API automation solutions

- **Postman** - Postman primarily is an application used for testing of API requests through a graphical user interface, and can also be used to help develop, design and document APIs. This application is not typically used for automation, however it is a fantastic tool that helps us to understand what is needed for a successful API request and the output provided from the server in an easily readable format
- **PowerShell** – PowerShell is a solution that provides cross-platform task automation through command line shell in the form of a scripting language
- **cURL** - cURL<sup>18</sup> is a command-line tool and application library used to transfer data between computers using URL syntax over HTTP or HTTPS connection and many other protocols. cURL by itself is not a complete automation tool set, but can be leveraged by scripts like Linux Shell and PowerShell
- **Ansible** - Ansible<sup>19</sup> is a powerful software automation tool which supports cross-platform computers and provides the ability to facilitate application deployment, updating of systems, configuration management and more, accomplished through simple scripts
- **Python** – Python<sup>20</sup> is a programming language used from software development to scripting and



supported on cross-platform systems. Scripting of an automated solution can be achieved by using Python code in a pure functional structure, creating the code using its object-orientated programming or a combination of both

Figure 6 provides an overview of various tools which can be leveraged for automation of Dell data protection solutions through REST APIs.

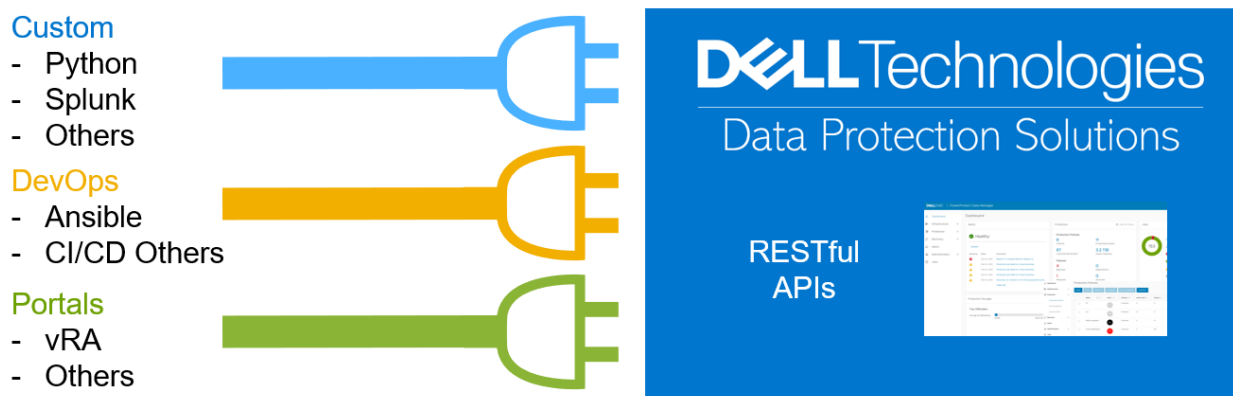


Figure 6 - Overview of some selected tools that can interface with Dell data protection solutions

In the Automation with PowerProtect Data Manager section, examples of how a REST API call is made for each of the tool sets are provided along with the corresponding output. Please note that while an example is provided for each tool set, most automation examples in this article are based on Python. Python is a relatively simple language to use, especially if it is written as functional only script and it is very portable between platforms.

## Automation with PowerProtect Data Manager

Dell provides information for the development community at <https://developer.dell.com/>. This site includes information about APIs for a wide range of Dell Technologies solutions, including data protection. With the focus of this article being on data protection automation and modernization, PowerProtect Data Manager will be used as the backup application for most examples provided. This backup application provides software defined data protection through its rich REST API feature set. Details of the resources, available methods, expected authentication, associated responses, and the format of schema used in the response can be found at <https://developer.dell.com/apis/4378/versions/19.12.0/docs/introduction.md>.

When determining the URL to be used for REST API calls, there is a defined structure. For PowerProtect Data Manager, the REST API end point is based on the following structured URL.

`https://{{ppdm-host}}:8443/api/<API-VERSION>/<RESOURCE>`

RESOURCE here refers to individual APIs provided by PowerProtect Data Manager. For example, login API is `api/v2/login` and storage metrics API is `api/v2/protection-storage-metrics`

For each of the below listed tool sets, the same REST API request will be made, and an overview of the process is provided in Figure 7.

First, a POST request is made to the login endpoint and when successful, PPDM will respond with the details of the bearer access token. This bearer access token can be used for the authorization type to initiate a GET request for the protection storage metric details.

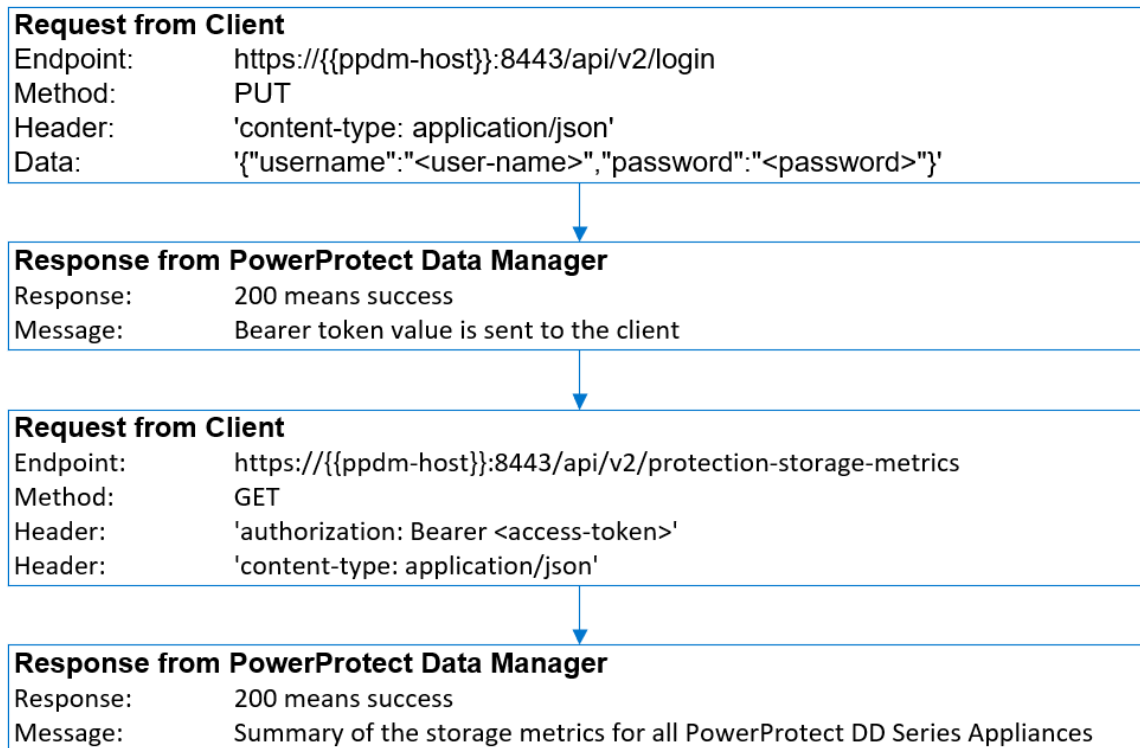


Figure 7 - Overview of sample API calls for each tool set

To confirm what information is required for a POST request or the type of authorization, the PPDM API documentation on the <https://developer.dell.com/> will be of assistance. For example, to get the bearer access token, the username and password need to be provided to the POST request as part of the body as shown in Figure 8.

POST

## Log in to your account

`/api/v2/login`

Logs in with user credentials. Once logged in, the user may perform operations granted to the user and defined by the user role.

### Servers

URL	Description
<a href="https://localhost:8443">https://localhost:8443</a>	

### Request Body

User login credentials.

Schema	
object {2}	User login credentials. Required
password string	The password to log in. Required
username string	The username to log in. Required

Figure 8 - API documentation for PPDM login endpoint from the Dell Developer website

Looking at the details provided for the protection-storage-metrics endpoint for PPDM in Figure 9, the documentation states that the authorization type is to be Bearer.

Therefore, to obtain the bearer access token, a request needs to first be sent to the login endpoint. Then the access token can be obtained so that a GET request can be made to the protection-storage-metrics endpoint.

GET

## Get all protection storage metrics

`/api/v2/protection-storage-metrics`

Retrieves aggregation of all protection storage metrics.

This endpoint supports execution by the following roles: Administrator, User, Backup Administrator, Restore Administrator, Security Administrator

This endpoint supports execution by the following roles: Administrator, User, Backup Administrator, Restore Administrator, Security Administrator

### Authorization


Key & Type	Details
BearerAuth 	For accessing the API, a valid JWT token must be passed in all the queries in the 'Authorization' header with the 'Bearer' HTTP authorization schema: 'Authorization: Bearer <JWT_token>' Scheme: bearer

Figure 9 - API documentation for PPDM protection storage metrics endpoint from the Dell Developer website

## PPDM's REST API, Getting Started

When making REST API requests to PowerProtect Data Manager, a bearer token is required for most resources, and this is obtained by first making a login request using basic authentication. The output of this JSON response can be seen in Figure 22, where the bearer token value is provided in the *access\_token* key and the duration that the token can be used is provided in the value of the *expires\_in* key in seconds.

Once the bearer token has been obtained, further REST requests can be sent to resources with that token only. If the token needs to be used beyond the time specified in the *expires\_in* key, it will need to be refreshed so that a new bearer token can be issued and used for REST API requests to resources.

The example commands provided for each of the tool sets are basic, that is, they do not include commands where a user is asked to enter in the required credentials, which would then be passed into the command. Nor, is there any error handling to verify if the command is successful with the REST API request.

These example commands show what is required to make a REST API request for each tool set. For these examples, credentials have been hard coded into the command and the password used has been obscured. Also, the bearer token value is a more than 1,100-character string and has been truncated in the sample codes snippets.

It is common knowledge that credentials should never be hard coded into a script and to build a script that is robust and contains error handling. The code/script used in the use cases for the backup administrator and application owner available from GitHub are built with error handling and request input from the administrator or user.

The REST API that is available with PowerProtect Data Manager provides options to implement automation for various use cases as shown in Figure 10. As the different tool sets are discussed, keep in mind the automation use case that is to be implemented and how one or more tool sets can achieve that goal.

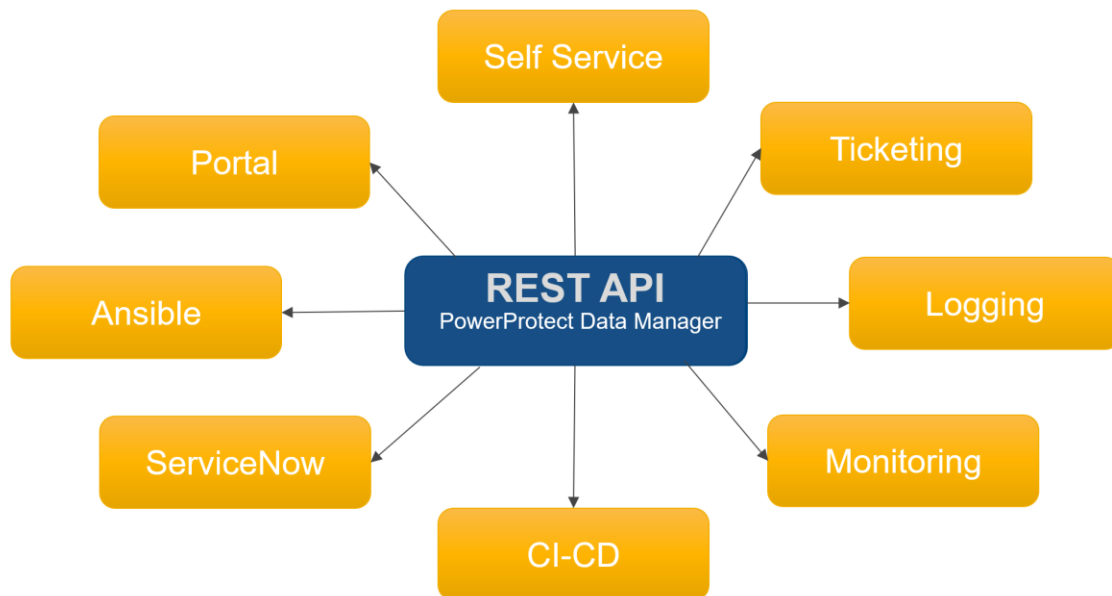


Figure 10 - Various automation use cases that can be developed for PowerProtect Data Manager

## Postman and Postman Collections

Postman is an application that enables API calls to be made and it is available through a graphical user interface or command line format on supported operating systems, including Windows, MacOS and Linux. It is also available using Postman on the web if there is a preference for not installing the application.

Using the Postman application is a great way to start exploring how to make API calls as it is very easy to use and provides the response in an easily readable format. The example screenshots provided have been taken from Postman installed on a Windows 2016 server.

Figure 11 shows the POST request using the Postman application which is required to make a REST API call to PPDM login endpoint to obtain the bearer token. While the login endpoint requires basic authentication, it is expecting this information in the body of the request. There is an option to provide the username and password under the Authorization tab when selecting the type of basic authorization, however, this will not work for PPDM. This option would work for other API requests, for example, it can be used with Dell NetWorker<sup>25</sup>. Content type should be JSON as shown in the picture.

Here is the blog with detailed instructions on how to use Postman to explore PPDM's REST API <https://www.dell.com/community/Blogs/Getting-started-with-PPDM-REST-API-using-Postman-Collections/ba-p/8424744>

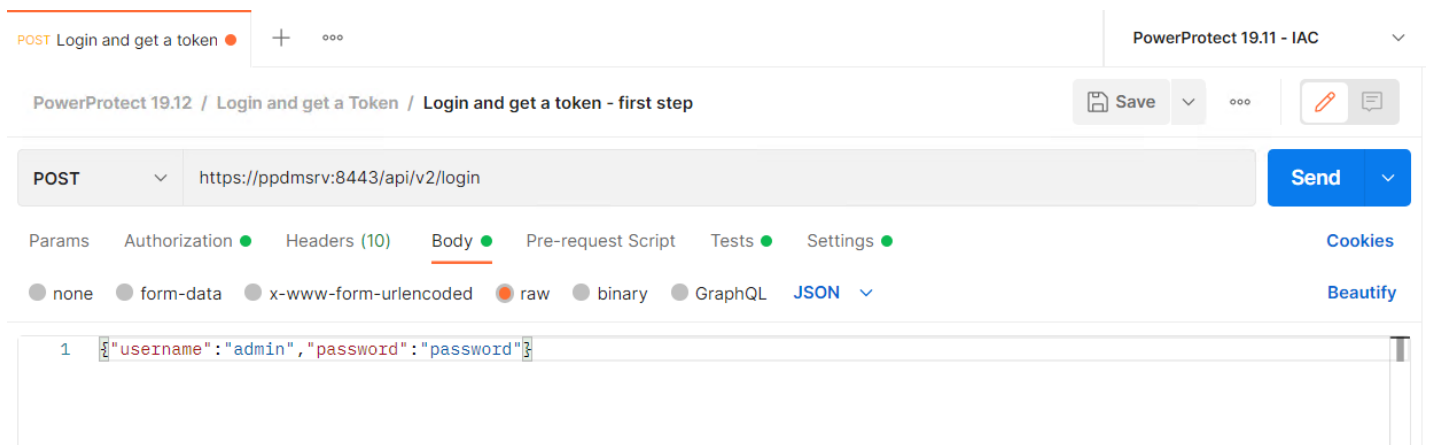


Figure 11 - Postman POST request to PPDM login endpoint using basic authentication

Upon receiving a successful response as per Figure 22, the bearer access token is displayed in the lower pane of Postman application. This token is copied and used for the subsequent API call to the protection-storage-metrics endpoint. Under the Authorization tab, select Bearer Token as the type and paste in the token in the input field to the right of the section. The method is changed from POST to GET for the protection-storage-metrics endpoint as shown in Figure 12 with a successful response providing output as per Figure 23.

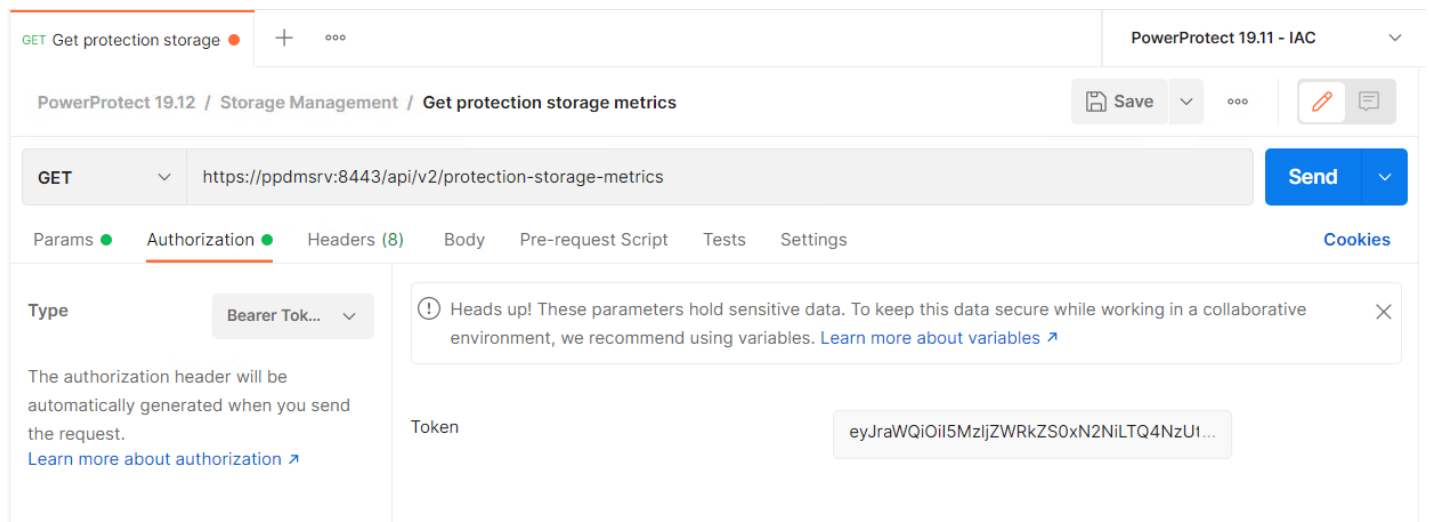


Figure 12 - Postman GET request to PPDM protection-storage-metrics endpoint using access token



Running the odd request is easy enough, but this can be cumbersome if requests are made on a regular occurrence. To address this issue, Postman allows for collections of requests to be saved and grouped to make it easy keep the workspace organized. When a collection is created, it can also be shared with others - this avoids the needed to manually create, save, and organize requests for every individual using Postman.

To enhance the efficiency of making requests with Postman, variables can be leveraged. This avoids the need to hard code usernames, passwords, tokens, and server names, hence, making the requests more efficient to use. For each collection, a set of variables can be defined and updated as needed. For example, make a single PPDM login request, copy the token, and update the token variable. Now any other request can be made to PPDM for the next eight hours.

As most of the examples provided are based on PPDM, Postman collection has been created containing commonly used API requests and available for download from <https://github.com/rjainoje/ppdm-automation>

The GitHub<sup>28</sup> repo contains the Postman collection and environment files, which need to be imported into the Postman application. The collection file contains information about the endpoint REST calls as shown in Figure 13, while the environments file contains variables that need to be populated with the PPDM server, username, password, vCenter and several other environment variables for the environment. With the environment details set, requests can be sent from the saved endpoints in the collection. This is handy for administrators starting out and for programmers to see the JSON response for individual API calls before code is written to process the data.

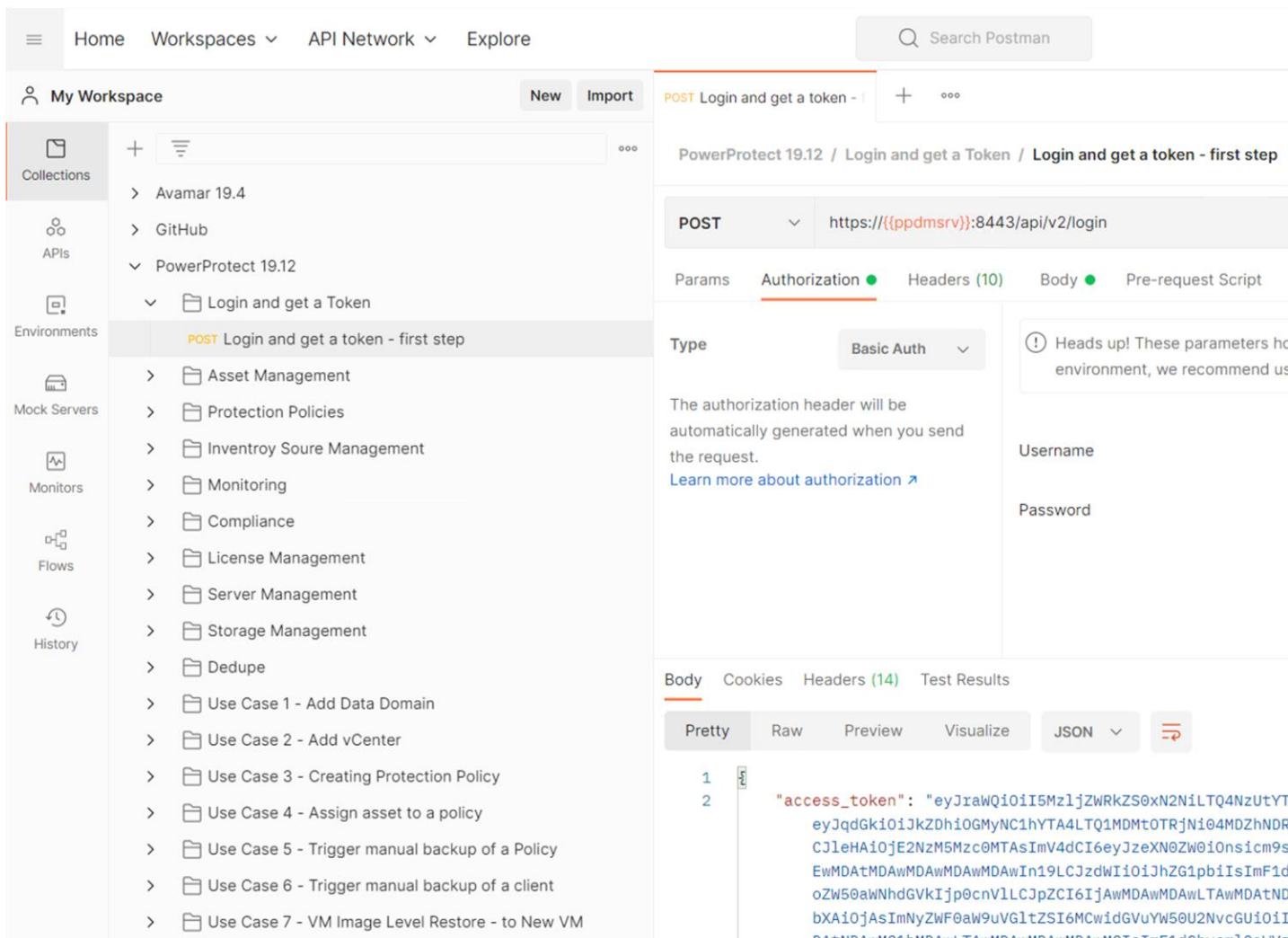


Figure 13 - View of collections available for Postman for PowerProtect Data Manager

## PowerShell

PowerShell uses a collection of modules that form a cmdlet, which is run from the PowerShell interface to make a REST API request. The cmdlet that is used is called Invoke-RestMethod. Before invoking the method, an additional security protocol type may be needed to allow for a Secure Socket Layer<sup>24</sup> (SSL) or a Transport Layer Security<sup>24</sup> (TLS) channel to be created. For a successful request to be made to PPDM an additional command was added before the Invoke-RestMethod is run.

Figure 14 provides the PowerShell POST request command that is required to make a REST API call to the PPDM login endpoint and obtain the bearer token to run on a Windows 2016 server.

```
$Body = @{
    username = "admin"
    password = "xxxxxxx"
}
$Parameters = @{
    Method = "POST"
    Uri = "https://ppdmsrv01:8443/api/v2/login"
    Body = ($Body | ConvertTo-Json)
    ContentType = "application/json"
}
[Net.ServicePointManager]::SecurityProtocol = [Net.SecurityProtocolType]::Tls12
Invoke-RestMethod @Parameters
```

Figure 14 – PowerShell POST request to PPDM login endpoint using basic authentication

Upon a successful response from PPDM, the bearer access token is provided in the message body as shown in Figure 22. The token is then used to make a GET request API call as shown in Figure 15 to obtain the details of the protection storage metrics as per Figure 23.

```
$Token =
"eyJraWQiOiIwMzEzNWExNy05NjczLTQ3YTctODYyNy1hYmY1M2Mx....."
$Headers = @{
    Authorization = "Bearer $Token"
    ContentType = "application/json"
}
$Parameters = @{
    Method = "GET"
    Uri = "https://ppdmsrv01:8443/api/v2/protection-storage-metrics"
    Headers = $Headers
}
[Net.ServicePointManager]::SecurityProtocol = [Net.SecurityProtocolType]::Tls12
Invoke-RestMethod @Parameters
```

Figure 15 – PowerShell GET request to PPDM protection-storage-metrics endpoint using access token

## cURL

The cURL library package can be leveraged on many platforms and the examples provided are as a single command on a Linux client. As a lab environment is being used, the certificate used by PowerProtect Data Manager has been marked as not trusted by the user. As a result, the (-k) or (- insecure) option can be used to turn off curl's verification of the certificate. In a production environment, this is not recommended, and trusted certificates should be issued, which would avoid this and remove the need to turn off verification of certificates.

Figure 16 provides the cURL POST request command that is required to make a REST API call to PPDM login endpoint to obtain the bearer token.

```
curl -k --request POST \  
  --url https://ppdmsrv01:8443/api/v2/login \  
  --header 'content-type: application/json' \  
  --data '{"username":"admin","password":"xxxxxxx"}'
```

*Figure 16 – cURL POST request to PPDM login endpoint using basic authentication*

Upon a successful response from PPDM, the bearer access token is provided in the message body as shown in Figure 22. The token is then used to make a GET request API call as shown in Figure 17 to obtain the details of the protection storage metrics as per Figure 23.

```
curl -k --request GET \  
  --url https://ppdmsrv01:8443/api/v2/protection-storage-metrics \  
  --header 'content-type: application/json' \  
  --header 'authorization: Bearer  
eyJraWQiOiIwM2EzNWExNy05NjcZLTQ3YTctODYyNy1hYmY1M2Mx.....'
```

*Figure 17 – cURL GET request to PPDM protection-storage-metrics endpoint using access token*

## Ansible

Ansible is an open-source management tool and needs to be installed on a server prior to it being able to run a playbook. A playbook is a file containing the information required to complete a task or series of tasks. As a lab environment is used, the certificate used by PowerProtect Data Manager has been marked as not trusted and as a result verification of the certificate has been set to “no” in the playbook.

Figure 15 provides the Ansible playbook required to run a POST request for the REST API call to the PPDM login endpoint so that the bearer token can be obtained.

```
- name: Authenticate to PowerProtect DM Rest API  
  hosts: localhost  
  connection: local  
  tasks:  
  - name: Making the login request  
    uri:  
      url: https://ppdmsrv01:8443/api/v2/login  
      method: POST  
      validate_certs: no  
      return_content: yes  
      body_format: json  
      body:  
        username: "admin"  
        password: "xxxxxxx"  
      status_code: 200  
    register: data  
  - name: Show the access token value from the json response  
    debug:  
      var: data.json.access_token
```

*Figure 18 – Ansible Playbook for POST request to PPDM login endpoint using basic authentication*

Upon a successful response from PPDM, the bearer access token is provided in the message body as shown in Figure 22. The token is then used to make a GET request API call via a playbook as shown in Figure 19 to

obtain the details of the protection storage metrics as per Figure 23.

```
- name: Get Protection storage metrics from PowerProtect DM Rest API
hosts: localhost
connection: local
vars:
  token: " eyJraWQiOiIwM2EzNWExNy05NjczLTQ3YTctODYyNy1hYmY1M2Mx....."
tasks:
- name: Making the login request
  uri:
    url: https:// ppdmsrv01:8443/api/v2/protection-storage-metrics
    method: GET
    validate_certs: no
    return_content: yes
    body_format: json
    headers:
      Authorization: "Bearer {{token}}"
      Content-Type: "application/json"
    status_code: 200
  register: data
- name: Show the access token value from the json response
  debug:
    var: data.content
```

Figure 19 – Ansible Playbook for GET request to PPDM protection-storage-metrics endpoint using access token

## Python

There are many programming languages available in the market and one of the more popular languages is Python. While Python is an object orientated programming language, it is easy to learn and powerful for the development of web and software applications. In addition, it offers the ability for system scripting applications leveraging the ability of the language to be used in a pure functional construct. In other words, it is not necessary to write code as objects, but rather a function that performs an action and returns a result, making it ideal for automation.

The samples in this section for Python are not written as a function but as commands run from PyCharm.<sup>26</sup> PyCharm is one of many Python Integrated Development Environments<sup>27</sup> (IDE), a code editor tool and the best editor normally comes down to personal preference.

Figure 20 provides the Python POST request commands that are required to make a REST API call to the PPDM login endpoint to obtain the bearer token.

```
import requests
from urllib3.exceptions import InsecureRequestWarning

disable_warnings(InsecureRequestWarning)

uri = 'https:// ppdmsrv01:8443/api/v2/login'
headers = {'Content-Type': 'application/json'}
payload = '{"username": "admin ", "password": "xxxxxxx"}'
response = requests.post(uri, data=payload, headers=headers, verify=False)
print(response.json())
```

Figure 20 – Python POST request to PPDM login endpoint using basic authentication





dictionaries. While at first it may be somewhat challenging to extract the information from the JSON response programmatically, it will become easier with experience.

The response from PPDM from the protection-storage-metrics endpoint is provided in a JSON format. To extract the information programmatically, a script is used to parse the JSON output, look for certain keys and read the corresponding value for processing.

```

{
  "systemsBySpaceUtilization": [
    {
      "usedPercentage": 34,
      "availablePercentage": 66,
      "totalSize": 1.890593538048E12,
      "usedSize": 6.34598719488E11,
      "availableSize": 1.25599481856E12,
      "systemName": "ddve01"
    },
    {
      "usedPercentage": 16,
      "availablePercentage": 84,
      "totalSize": 1.17757181952E11,
      "usedSize": 1.8459131904E10,
      "availableSize": 9.9298050048E10,
      "systemName": "ddve02"
    }
  ],
  "criticalSystemsCount": 0,
  "nonCriticalSystemsCount": 2
}

```

Figure 23 - Response from GET request to PPDM protection-storage-metrics endpoint

The protection-storage-metrics end point is listed under the resource of Storage Management and is one of many resources available via REST. A high-level list of the resources available for PPDM is shown in Figure 24 and accessible at <https://developer.dell.com/apis/4378/versions/19.12.0/docs/introduction.md>.

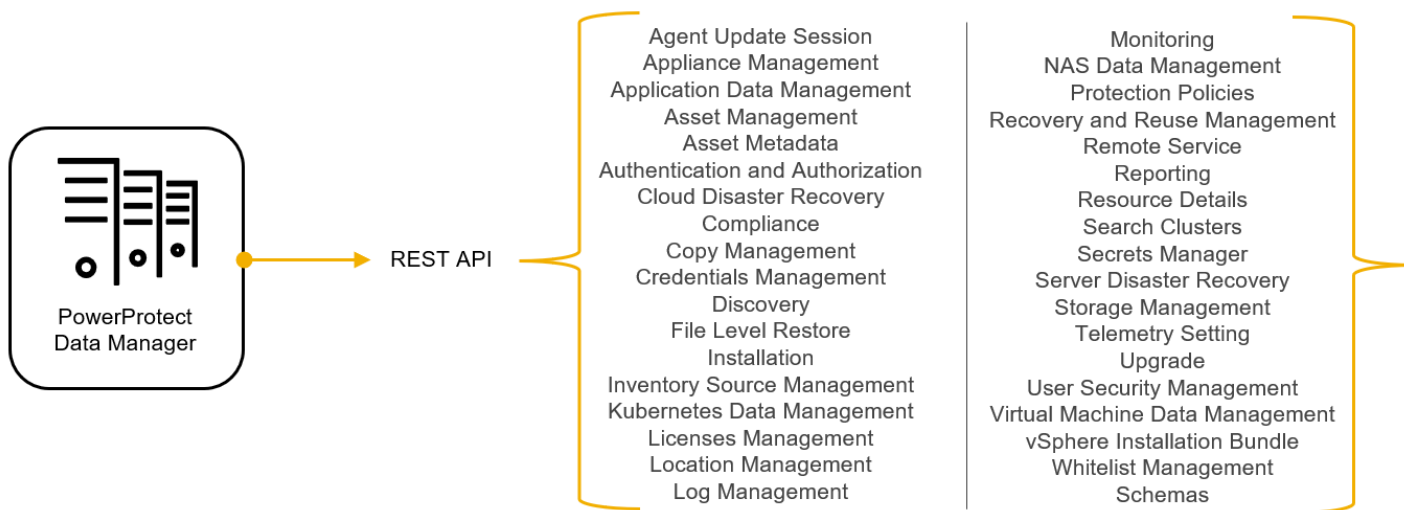


Figure 24 - List of PowerProtect Data Manager resources available via REST

## Tool Set Summary

As mentioned previously, these have been written in their most basic form and provide a glimpse of the code structure. How to get started with automation in your environment may be challenging and finding out if the organization has a desire for a particular tool set or language would be a good start. If, however, getting into automation is a personal skill development goal or there is no set direction from the organization, then it is recommended to get started with Postman first.

With Postman, making requests to an API endpoint is quite simple and the output from the response is easily understood, especially when the response has been prettified. Take the time to review the response and understand the value of the information it contains. Then, start thinking about how that information can be used for subsequent requests or how it can be used in day-to-day operations. For example, can API requests be used to capture failed backups every day and alert both the backup and application owners, or be sent to a ticketing system for action? Ideally, think about how the data in the responses can provide value back to the organization or application owners in terms of improved insights, self service capabilities, reducing errors with system configuration, etc.

Having an idea of how to leverage the APIs to achieve a given outcome, it is time to determine how to best automate it and how administrators or application owners will interact with it. A good next step is to start small and leverage a simple scripting language like a Linux Shell or PowerShell script. For example, if there is a desire to use Shell scripts, leverage cURL to make the API requests, get input from the user, perform conditional checks on the data and return a result. Using a Shell script is easy enough to create, maintain and does not need any applications or tool sets to be introduced into an environment.

The next step up from a scripting approach is to use Ansible and is ideal for configuration management, orchestration of infrastructure elements and provisioning. It will require some level of upskilling and Ansible would need to be installed on a server so it can run the playbooks.

If successful small scale automation tasks have been created using Shell or PowerShell, then there may be a desire to expand the automation scope, which may result in needing to leverage more advanced tool sets like Python. It may require administrators to invest some time to upskill to become efficient in writing code in Python. The benefit to using Python is that it will allow for more complex automation solutions to be created and it allows for web based or graphical interfaces to be built, making it easier for the organization or applications to consume.

The good news is that there are lots of people that have created snippets of code in Python and shared that on the internet or via GitHub for others to use. Chances are someone has already written code that can be reused or adapted as part of the automation solution. GitHub is a platform for hosting code for version control and enables collaboration, and depending on scale of automation, it may be a platform that is used as a repository for the automation solutions being created. Code uploaded to GitHub can be shared among other members, shared publicly, or remain visible to the owner only.

Ultimately, automation of elements in an IT environment may consist of both Python and Ansible - it all depends on what the automation end state needs to achieve.

The use case code examples provided in this paper for both the backup administrator and application owner are publicly available on GitHub, enabling code to be downloaded, modified, and used for the automation solutions to be implemented.

## Backup Administrator Use Cases

The focus of this paper is automation examples related to data protection using PowerProtect Data Manager. This section will provide several tasks which have been automated with Python. Using automation for the deployment of data protection elements or configuration changes to the backup application reduces potential misconfigurations. For example, there may be a particular naming standard that is used when creating a data protection policy and through automation, this can be enforced. Based on the input supplied at the start of the automation process, that input is then used to create the overall name of the policy.

The following are some typical use cases for backup administrators.

- Add PowerProtect Data Domain to PPDM
- Add vCenter
- Install backup agent on a client
- Create a protection policy
- Add client to a protection policy
- Decommission backup client

The frequency of which automation use cases are used will depend on many factors, including the growth of the environment being protected, if test environments are spun up, used and then retired, or the number of different security-based environments which need to be configured and maintained.

A summary of the API requests with a brief description is provided and often the information used by an API request would require information extracted from the previous API request. This may not be stated in the brief description, but this will be evident in the code samples.

### Defining the Logical Process

Typically, a plan is developed before implementing a new system, upgrading elements of the IT environment, or determining what the future direction of the environment could look like. Creating a plan or an architecture diagram is no different for a script or bit of code. Starting without one will lead to issues, inefficient code, complexity, or total failure in not completing the desired task. With each of the backup administrator use cases, a high-level logical diagram is provided, and this is important as it lays out the structure of the code and how it will flow.

This diagram will also help in getting started with writing the code necessary for the automation solution, as it can be daunting thinking of the code needed to complete the script in its entirety. It may even prevent the automation solution from being created.

By breaking the architecture flow of the solution into each element, it allows for snippets of code to be written and tested before moving to the next section. For example, write a bit of code that allows a user to provide the username, password and server address and display those variables via a simple print statement. Then with that completed, create the code needed to make an API request which can be completed by passing in the variables obtained by the input section of code. With that working, write the code that takes in the JSON response and extract the required value from a key or set of keys. This process continues until the script is working as required. In this approach, starting with small sections of code and making sure that they work before moving to the next section, it will greatly improve the chance that the automation solution or task will be created.

### Adding PowerProtect DD Series Appliance

While a PowerProtect DD series appliance is not something that is added very often, having the ability to add a DD series appliance is useful, especially for deployments at remote sites or for use in test environments. Once the script is developed, it will ensure consistent registration of the protection storage with PPDM.

When adding a PowerProtect DD series appliance to PPDM, a POST request to the endpoint of /api/v2/inventory-sources needs to be performed and the required information is provided in the body of the request. The logical process and the API endpoints requests used are as follows.

1. POST - /api/v2/login  
(request to login API to authenticate and get bearer token)
2. GET / PUT - /api/v2/certificates  
(Get server certificate and use it in the next call)
3. PUT /api/v2/certificates/{certId}  
(Trust the certificate)
4. POST - api/v2/datadomain-system-validation  
(determine if PowerProtect DD appliance is supported)
5. POST - /api/v2/credentials  
(create credentials for PowerProtect DD system)
6. POST - /api/v2/inventory-sources  
(add PowerProtect DD server)
7. GET - /api/v2/storage-systems  
(verify if the PowerProtect DD has been added)

The above-mentioned use case is available as a Postman collection <https://github.com/rjainoje/ppdm-automation/tree/main/ppdm-postman-collections> and Python code <https://github.com/rjainoje/ppdm-automation/tree/main/ppdm-python-restapi>

## Adding vCenter

To protect virtual machines hosted on VMware, the vCenter must first be added as an asset source to PPDM and once completed, the discovered virtual machines can be protected.

1. POST - /api/v2/login  
(request to login API to authenticate and get bearer token)
2. GET - /api/v2/certificates  
(Get server certificate and use it in the next call)
3. PUT /api/v2/certificates/{certId}  
(Trust the certificate)
4. POST - /api/v2/credentials  
(create credentials for vCenter system)
5. POST - /api/v2/inventory-sources  
(This will add the vCenter, look for more information on Git repo)

The above-mentioned use case is available as a Postman collection <https://github.com/rjainoje/ppdm-automation/tree/main/ppdm-postman-collections> and Python code <https://github.com/rjainoje/ppdm-automation/tree/main/ppdm-python-restapi>

## Creating Protection Policies

Creating a protection policy should be a task configured before clients are added and something that is not changed very often, if at all. Having the ability to create policies ensures consistent naming conventions, retention policies and clone activities. Following is an example of how to create a policy by leveraging PPDM's REST API using Postman collection as well as using Python.

1. POST - /api/v2/login  
(request to login API to authenticate and get bearer token)
2. GET - /api/v2/storage-systems  
(get the PowerProtect DD ID)

3. POST - (/api/v2/protection-policies  
(provided in the body of the request the storage system ID, retention policies, schedules, retention lock option, etc.)
4. Get - /api/v2/protection-policies  
(verify recently created policy)

The above-mentioned use case is available as a Postman collection <https://github.com/rjainoje/ppdm-automation/tree/main/ppdm-postman-collections> and Python code <https://github.com/rjainoje/ppdm-automation/tree/main/ppdm-python-restapi>

## Installing File System Agents

To protect file systems hosted on physical servers, an agent needs to be installed and registered with PPDM before it can be protected via a protection policy. To accomplish this, there are two requirements - first the agent needs to be installed on the client and, in this use case, Ansible will be used. Then once installed, the client or asset needs to be approved from PPDM before it can be added to a protection policy.

Installing the file agent on a server is completed using the Ansible playbook and the code in the playbook, and information about how to run it is available from the following GitHub link <https://github.com/rjainoje/ppdm-automation/tree/main/ppdm-ansible-playbooks>.

Once the agent is installed, the client or asset must be approved by making the following calls to PPDM.

1. POST - /api/v2/login  
(request to login API to authenticate and get bearer token)
2. GET - /api/v2/whitelist  
(Get whitelist ID of the client you are planning to register)
3. PATCH - /api/v2/whitelist/{whitelistId}  
(Register the client i.e whitelist the client using whitelist ID from the earlier call)
4. GET - /api/v2/hosts?filter=type eq "APP\_HOST" and attributes.appHost.appServerTypes in ("FS")  
(Get file system host ID to be used in the next call)
5. POST - /api/v2/discoveries  
(Discover file system host by file system host ID from an earlier call)
6. GET - /api/v2/assets?filter=type eq "FILE\_SYSTEM"  
(Verify that the file system host is discovered)

The above-mentioned use case is available as a Postman collection <https://github.com/rjainoje/ppdm-automation/tree/main/ppdm-postman-collections> and Python code <https://github.com/rjainoje/ppdm-automation/tree/main/ppdm-python-restapi>

## Adding Clients to a Protection Policy

Adding clients programmatically to a Protection Policy is one of the most common use cases. Clients are referred to as assets in PPDM and they can be added to an existing policy with the following API calls. The following use case is available as part of Postman collection

1. POST - /api/v2/login  
(request to login API to authenticate and get bearer token)
2. GET - /api/v2/assets  
(this API provides a list of assets and their associated IDs)
3. GET - /api/v2/protection-policies?filter=name eq "VMBackups"  
(this API provides the policy-id to be used when adding a VMware VM asset)
4. POST - /api/v2/protection-policies/{{policy-id}}/asset-assignments



(this API request assigns the identified asset to a particular protection policy)

The above-mentioned use case is available as a Postman collection <https://github.com/rjainoje/ppdm-automation/tree/main/ppdm-postman-collections> and Python code <https://github.com/rjainoje/ppdm-automation/tree/main/ppdm-python-restapi>

## Application Owner Automation Use-Cases

Empowering the application owner through automation not only reduces the effort that administrators supporting the application need to expend, but it also improves the time to value for the application owner. From a data protection context, allowing the application owners to have their applications automatically added to data protection policies through a self-service portal, or as part of the provisioning of the server itself, ensures business data is protected and recoverable. Enabling self-service backups allows an application owner to backup data just prior to an upgrade, minimizing the recovery point objective. Furthermore, self-service backups can allow for faster recovery of data too by reducing the associated operational process.

The following use cases can be delivered through ServiceNow<sup>31</sup> or a custom portal by leveraging the organization's choice of programming language to invoke REST API calls.

### Adding Clients

Application owners should be able to include their servers for backup protection by providing minimal input such as protection type category (e.g., Platinum, Gold, Silver, etc.) and the code would then add the asset to the respective policy.

1. POST - `{{ppdm-host}}/api/v2/login`  
(request to login API to authenticate and get bearer token)
2. GET - `{{ppdm-host}}/api/v2/assets`  
(this API provides a list of assets and their associated IDs)
3. GET - `{{ppdm-host}}/api/v2/protection-policies?filter=name eq "VMBackups"`  
(this API provides the policy-id to be used when adding a VMware virtual machine asset)
4. POST - `{{ppdm-host}}/api/v2/protection-policies/{{policy-id}}/asset-assignments`  
(this API request assigns the identified asset to a particular protection policy)

The above-mentioned use case is available as a Postman collection <https://github.com/rjainoje/ppdm-automation/tree/main/ppdm-postman-collections> and Python code <https://github.com/rjainoje/ppdm-automation/tree/main/ppdm-python-restapi>

All the above API calls can be combined into a program and can be executed.

### Self-service Backup

Running an ad hoc backup is one of most common requests from application / server owners, as it is used to take a backup copy before an upgrade or maintenance task is performed. Providing self-service backup capability to application owners speeds up the development cycle and reduces the burden on the backup team. The following API calls help trigger a manual backup of a client.

1. POST - `{{ppdm-host}}/api/v2/login`  
(request to login API to authenticate and get bearer token)
2. GET - `{{ppdm-host}}/api/v2/assets?filter=name eq "client-name"`  
(this API will fetch asset-id and policy-id)
3. GET - `{{ppdm-host}}/api/v2/protection-policies/{{policy-id}}`  
(this API will fetch policy-id)
4. POST - `{{ppdm-host}}/api/v2/protection-policies/{{policy-id}}/protections`  
(this API will trigger the backup of a client)

The above-mentioned use case is available as a Postman collection <https://github.com/rjainoje/ppdm-automation/tree/main/ppdm-postman-collections> and Python code <https://github.com/rjainoje/ppdm-automation/tree/main/ppdm-python-restapi>

## Self-service Restore – VM restore to New VM

This use case is slightly more involved than the previous examples as REST API calls are made to PPDM as well as vCenter. The first part of the example communicates with PPDM to extract information about the VM backups, then information is captured from vCenter and finally, a POST request is made to initiate the recovery of the VM.

1. POST - `{{ppdm-host}}/api/v2/login`  
(request to login API to authenticate and get bearer token)
2. GET - `{{ppdm-host}}/api/v2/assets?filter=name eq "client-name"`  
(asset-id of the VM to restore)
3. GET - `{{ppdm-host}}/api/v2/assets/{{source-asset-id}}/copies`  
(get backup copies of the asset to restore)
4. GET - `{{ppdm-host}}/api/v2/copies/{{copy-id}}`  
(verify the copy details to restore)
5. GET - `{{ppdm-host}}/api/v2/inventory-sources?filter=type eq "VCENTER"`  
(get vCenter details to restore the VM to)

The next few API calls will gather vCenter information that needs to be supplied to make a final restore call

6. POST - `{{vcenter-host}}:443/rest/com/vmware/cis/session`  
(login to vCenter)
7. GET - `{{vcenter-host}}/rest/vcenter/datacenter`  
(get Data Center details)
8. GET - `{{vcenter-host}}/rest/vcenter/cluster`  
(get cluster details)
9. GET - `{{vcenter-host}}/rest/vcenter/host`  
(get host details)
10. GET - `{{vcenter-host}}/rest/vcenter/datastore`  
(get datastore details)

The final API call to PPDM is to restore the VM by providing input gathered from the above nine calls in the body of the below API call. The complete details are in the GitHub repository.

11. POST - `{{ppdm-host}}/api/v2/restored-copies`

The above-mentioned use case is available as a Postman collection <https://github.com/rjainoje/ppdm-automation/tree/main/ppdm-postman-collections> and Python code <https://github.com/rjainoje/ppdm-automation/tree/main/ppdm-python-restapi>

## Advanced use case

The backup administrator and application owner use cases provided all relate to PowerProtect Data Manager and, while this is a straightforward way to get started with automation, the power of automation can further be realized when it includes other systems. For example, having a script which interacts with one system, obtains information, and then uses that information to automate tasks on a different system. Keeping with the data protection theme, a more advanced scripted automation solution has been created to validate virtual machine (VM) image-based backups as detailed in the following section.

### VMware VM Backup Validation

Some organizations require regular validation of data protected by a backup application to ensure data is recoverable in the event of a disaster. Validation of data can be achieved by restoring the data from the protection storage and then verifying that the data has been successfully recovered. Performing this activity is typically a manual process and requires time that administrators and application owners may not have.

Automation can assist in the process and a scripted solution has been created to validate a VM image backup from VMware ESXi<sup>29</sup> hypervisor taken with PowerProtect Data Manager and NetWorker where the protection storage is the PowerProtect DD Series appliance. This scripted approach performs an Instant Access recovery of a VM by mounting the backup directly on the PowerProtect DD, starting the VM and waiting for the operating system of the VM to start and load VMware tools<sup>30</sup>. If successful, the result is recorded in a file, the VM is shut down and the process starts again for the following VM to be validated. Programmatically, an overview of the automation workflow as performed by the Python script is shown in Figure 25.

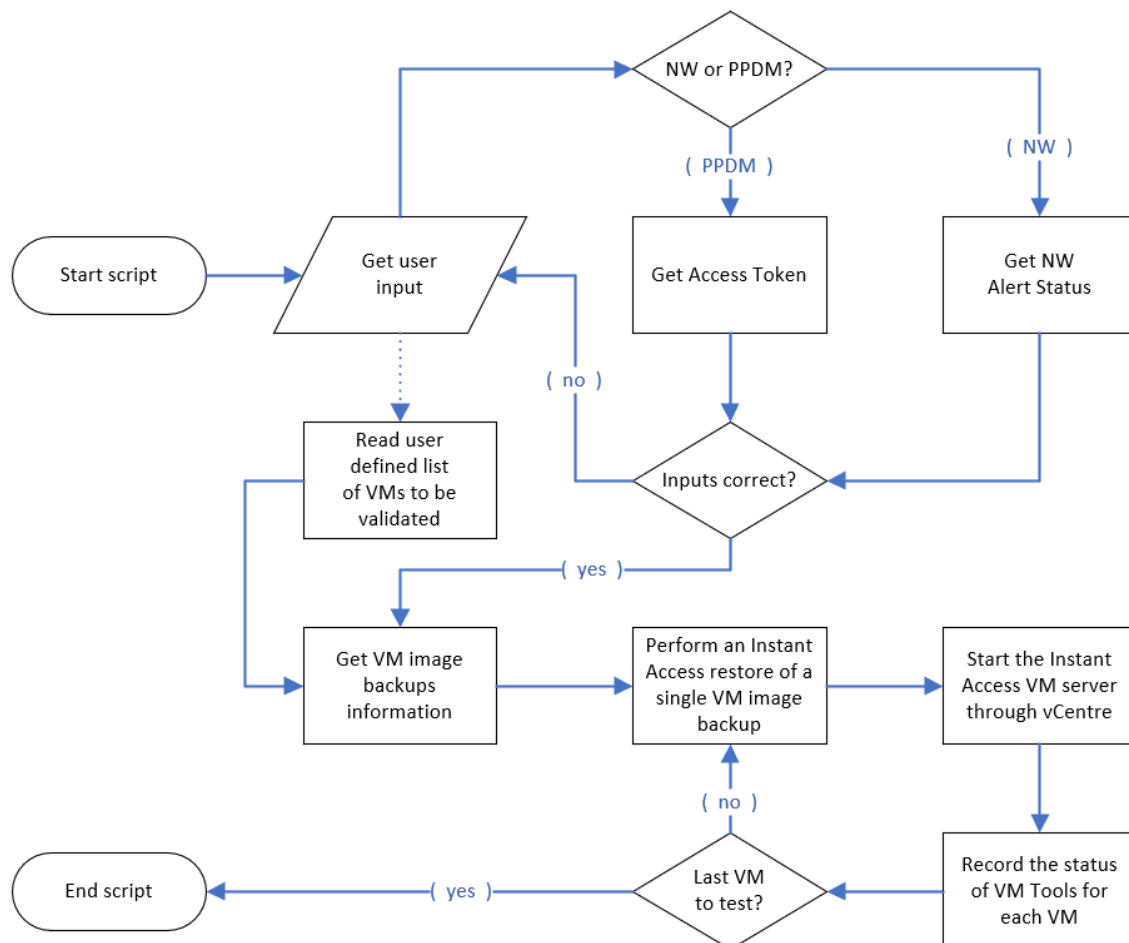


Figure 25 - Logical process of Python script for the VMware VM validation advanced use case

This scripted approach greatly reduces manual effort, provides the validation results for each VM and, with the Instant Access feature of the PowerProtect DD Series appliance, ensures for fast validation without sending large volumes of data across the network.

Note that VMware Tools is used to validate if the VM has successfully started or not, and as such, it is a requirement that VMware Tools is installed on the VM for it to be validated in this advanced automation use case. Most of the automation use cases provided have been written in a pure functional based Python script, making it easy to use and maintain for simple automation tasks.

However, for this VM validation use case, the Python code is a little more advanced and introduces objects to store information about the backup and VMware server, plus the code is written and stored in separate files. These Python files are imported and within each file, there are numerous functions that take input, process the data, and return the output. The idea behind this more advanced writing style is to make code reusable without having to modify or re-write it, so it may be used by another scripted automation use case. This does take a little bit more effort and forward planning, but the reward will be evident through re-use with other scripted solutions.

The Python code and a Windows distribution in the form of a self-containing executable file for this VM backup validation use case is available for download from <https://github.com/msteen2010/VM-validation>.



## End-to-end Solution

Ultimately, the benefit of automation is to provide end to end solutions for application owners and end users of IT systems and services. As automation solutions are created, they may be provided as a simple script that is run by an individual or by interaction through a basic web site. For example, allowing an application owner to perform an ad hoc backup can be achieved by running a script and, while this achieves the desired outcome, there is little to no oversight. In addition, having scripts shared and stored across the environment makes for a management nightmare. Storing them in a single location, with access to them enabled via a web site, is far more efficient.

One of the primary benefits of creating end to end solutions is that it allows for it to be integrated into business applications or portals, providing oversight and easy management. Plus, it also ensures a centralized method in obtaining IT related services and avoids bespoke ad hoc processes being scattered throughout the organization.

Ansible AWX<sup>35</sup> is a web-based user interface to manage ansible playbooks and job templates. It provides REST API endpoints to call individual job templates, where each job template will have one or more Ansible playbooks to execute. AWX can also be referred to as an API gateway, an element that forms a part of the automation solution and provides efficiency, especially for larger organizations.

Figure 26 shows an example of a custom portal, developed to provide self-service to application and server owners along with backup admins and executives. When a user selects a service on the portal, it will call the corresponding job template on AWX, which in turn executes playbooks that will call respective API on the PowerProtect Data Manager and other Dell data protection solutions.

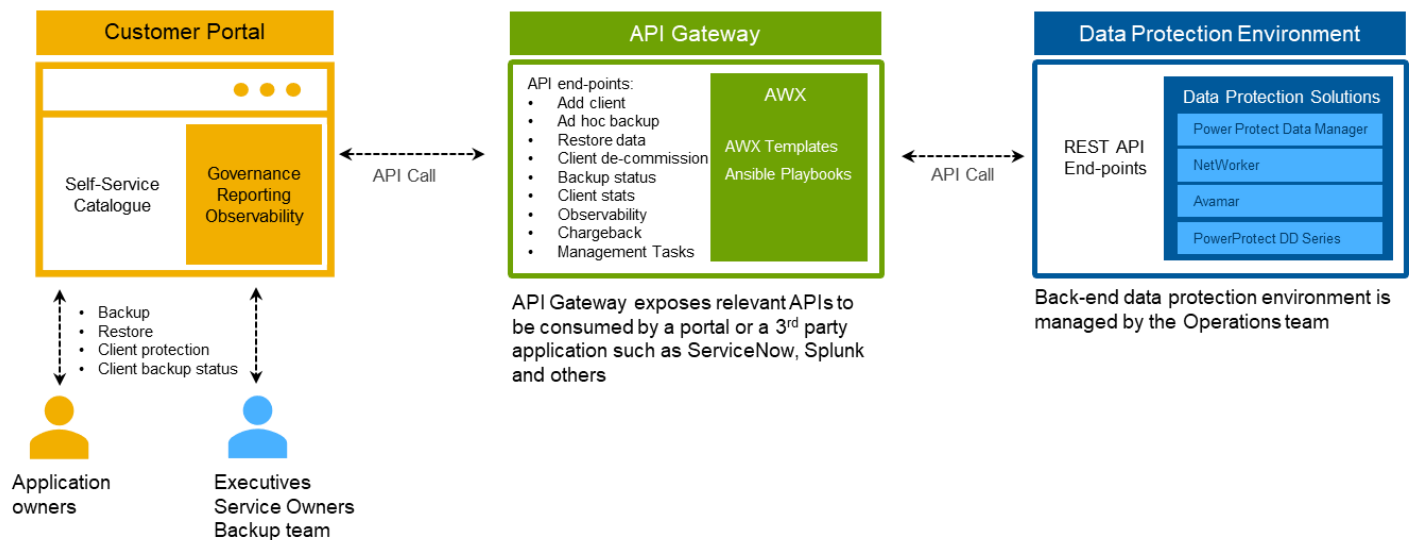


Figure 26 - End to end solutions, integrated into business applications or portals

By developing a custom portal with API gateway, it enables organizations to provide self-service capabilities to their end-users and application owners. The backup team can also utilize the portal for performing tasks programmatically, while executives and management can utilize the portal for reporting purposes.

## ServiceNow and other Self-Service Portals

Most organizations leverage a centralized catalogue for IT services, requests, and asset management, with the use of a self-developed portal or a commercial off the shelf solution, like ServiceNow. A portal provides that single point of contact for administrators, application owners and end users of IT services and is the perfect location to interface with automation solutions and tasks.

Several scripting languages and tools have been mentioned in this article and, while these require administrators or developers to create them, there are also pre-built 3<sup>rd</sup> party plugins/tools that can be used as part of the automated solutions. For example, VMware has a suite of automation platforms that help automate the life cycle of VMs managed and includes platforms like Aria Automation<sup>32</sup> (formally known as vRealize Automation), vRealize Orchestration<sup>33</sup>, and vCenter Director<sup>34</sup>. Vendors like Dell provide plug-ins that integrate with the automation stack of VMware, allowing organizations to accelerate their automation journey, especially for virtualized environments.

From a data protection point of view, Dell makes available through the support portal the Data Protection Extensions plug-in for each of the VMware automation platforms. With the plug-in installed, it provides the workflow for automation of data protection operations for virtual machines.

ServiceNow and self-service portals can call APIs via an API gateway. The API gateway in this example is AWX. AWX provides a web-based interface to execute relevant AWX templates, which in turn executes PPDM's REST API via its task engine as shown in Figure 27. In this example, there are three APIs exposed via API gateway, they include: register a client, run an ad hoc backup, and perform a restore.

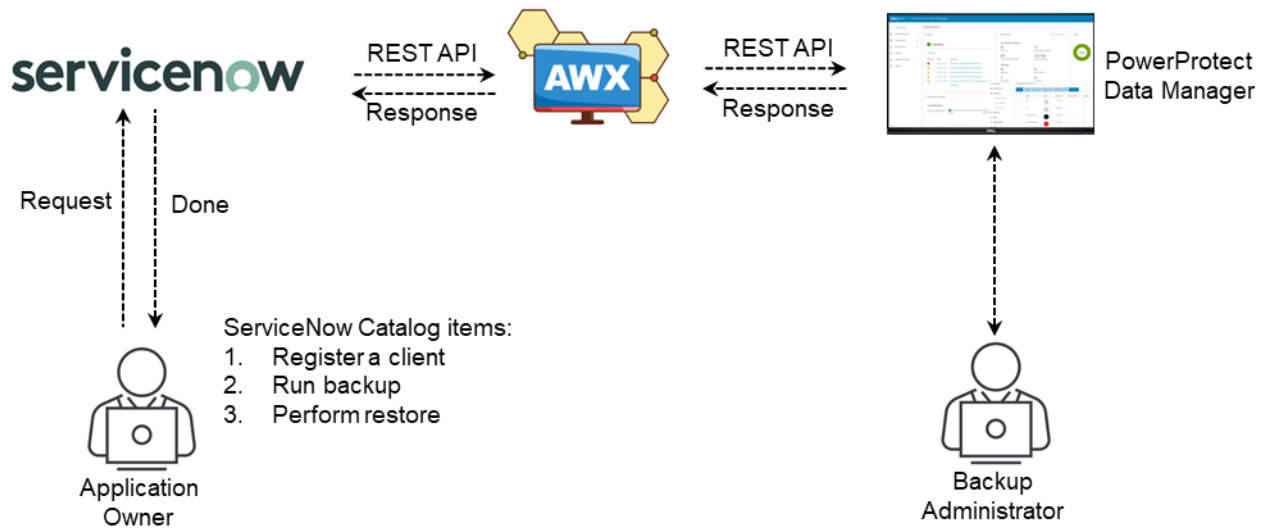


Figure 27 - Overview of an end-to-end data protection automation solution

Application and server owners can login into ServiceNow and request for an ad hoc backup or perform a restore or register a new client without needing to interact with the backup administrator. For more detailed information about this use case, read the following blog post at <https://www.dell.com/community/Blogs/Self-service-backup-with-PPDM-and-ServiceNow/ba-p/8280572>

### External data loggers

Customers can leverage REST API to pull data from PPDM and other Dell data protection solutions into a central repository for reporting, analytics, auditing, and alerting. While there are many commercial solutions available, Splunk is widely used. An example of how to integrate it into the environment through automation is shown in Figure 28. Just like PPDM, Splunk has a rich set of REST API resources and end points available at <https://docs.splunk.com/Documentation/Splunk/9.0.3/RESTREF/RESTprolog> and can be used in the automation solution.

Once the data has been ingested into Splunk or the central repository, it is analyzed, indexed, and made available to another system and interested parties through visualization in the form of dashboards or via alerts for urgent related issues that have been detected.

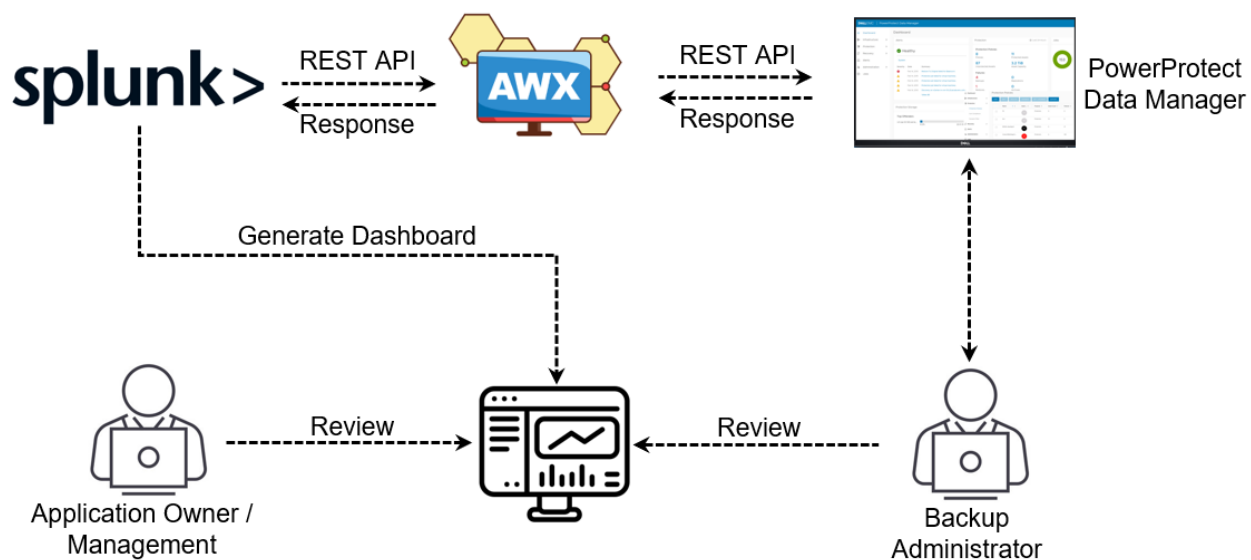


Figure 28 - Dashboards generated by Splunk by collecting and processing data from data protection application

### Ticketing

Addressing backup failures in a timely manner improves backup success rate, reduces the risk related to data unavailability, and improves data recovery. Organizations use different ticketing systems to create service requests for incidents, however, like most modern applications they are also likely to support REST API protocol. The example in Figure 29 shows a ticketing system receiving alert related data from PPDM and creating incident requests based on a severity of the incident. The backup administrator and/or application owner can then review the request and resolve the issue. By using a ticketing system, the organization has a historical view of all issues related to the backup software, who actioned the request and the root cause of the issue.

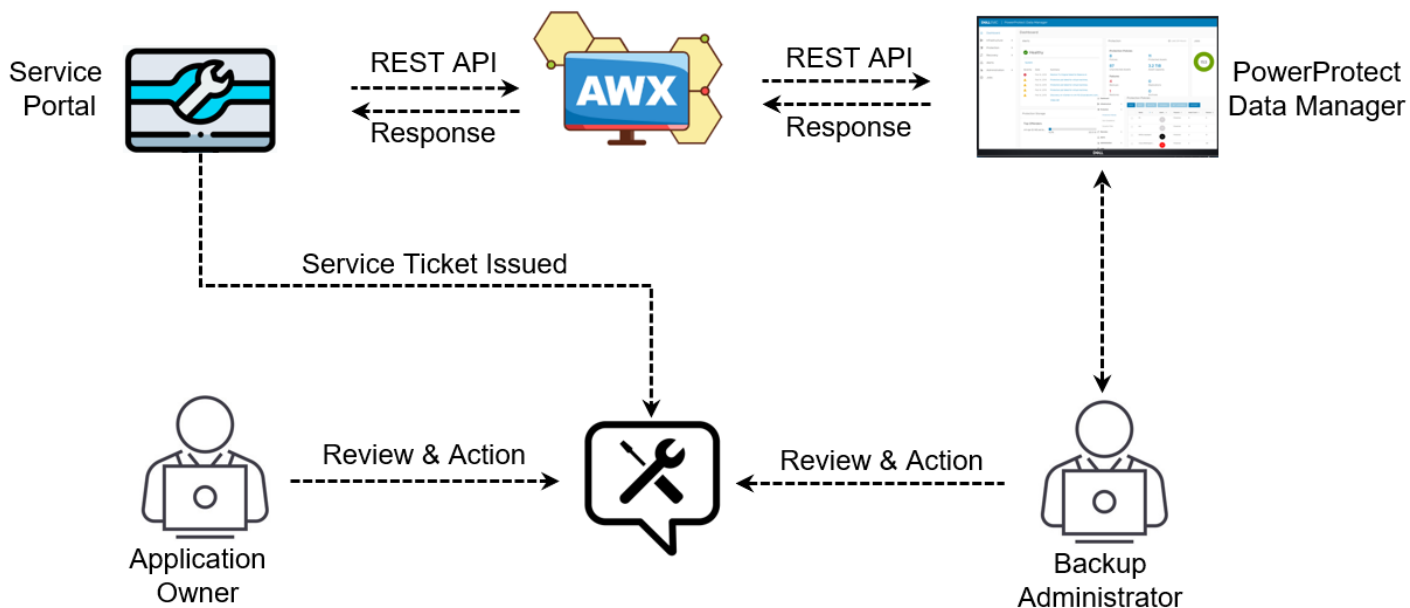


Figure 29 - Service request tickets issued for faults detected with data protection operations

In the three end to end use cases, an API gateway has been used and while it is not necessary to use an API gateway, it provides a central repository for all automation solutions.

## Conclusion

Whether automation is to be used for isolated use cases to reduce the effort spent on repetitive tasks, to ensure consistency of configurations, empower applications owners or provide the organization a cloud like service, automation is the key component as organizations continue with their digitization journey.

For those administrators new to automation and who may not necessarily know where to start, this paper provides information to start the automation journey. With background into the different automation tool sets, what the REST API is and how it is used, coded examples of several smaller automation use cases, there is nothing to prevent the automation journey from starting.

A single, more advanced automation example has been provided to highlight the business benefits that automation can provide through VM backup validation testing. While it may take additional effort to have automation interacting with multiple systems, it will provide more benefit to the organization, the IT administrators, application owners and consumers of the IT services.

Automation is extremely relevant today and a cornerstone for organizations in their digitization journey, so if automation is not being used today in your organization, get started with individual use cases and expand from there.

## Bibliography

1. [https://en.wikipedia.org/wiki/Information\\_technology](https://en.wikipedia.org/wiki/Information_technology)
2. <https://www.dell.com/en-us/dt/data-protection/powerprotect-data-manager.htm>
3. [https://en.wikipedia.org/wiki/Representational\\_state\\_transfer](https://en.wikipedia.org/wiki/Representational_state_transfer)
4. [https://en.wikipedia.org/wiki/Shell\\_script](https://en.wikipedia.org/wiki/Shell_script)
5. [https://en.wikipedia.org/wiki/Batch\\_file](https://en.wikipedia.org/wiki/Batch_file)
6. <https://en.wikipedia.org/wiki/PowerShell>
7. <https://en.wikipedia.org/wiki/Linux>
8. <https://en.wikipedia.org/wiki/MacOS>
9. [https://en.wikipedia.org/wiki/Cloud\\_computing](https://en.wikipedia.org/wiki/Cloud_computing)
10. <https://en.wikipedia.org/wiki/Internet>
11. <https://en.wikipedia.org/wiki/URL>
12. [https://en.wikipedia.org/wiki/Client%E2%80%93server\\_model](https://en.wikipedia.org/wiki/Client%E2%80%93server_model)
13. [https://en.wikipedia.org/wiki/Representational\\_state\\_transfer](https://en.wikipedia.org/wiki/Representational_state_transfer)
14. <https://en.wikipedia.org/wiki/API>
15. [https://en.wikipedia.org/wiki/Hypertext\\_Transfer\\_Protocol](https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol)
16. <https://en.wikipedia.org/wiki/Base64>
17. <https://www.postman.com/>
18. <https://en.wikipedia.org/wiki/CURL>
19. <https://www.ansible.com/>
20. <https://www.python.org/>
21. <https://en.wikipedia.org/wiki/OAuth>
22. <https://en.wikipedia.org/wiki/JSON>
23. <https://en.wikipedia.org/wiki/XML>
24. [https://en.wikipedia.org/wiki/Transport\\_Layer\\_Security](https://en.wikipedia.org/wiki/Transport_Layer_Security)
25. <https://www.dell.com/en-us/dt/data-protection/data-protection-suite/networker-data-protection-software.htm>
26. <https://www.jetbrains.com/pycharm/>
27. <https://realpython.com/python-ides-code-editors-guide/>
28. <https://github.com/>
29. <https://www.vmware.com/au/products/esxi-and-esx.html>
30. <https://docs.vmware.com/en/VMware-Tools/11.3.0/com.vmware.vsphere.vmwaretools.doc/GUID-28C39A00-743B-4222-B697-6632E94A8E72.html>
31. <https://www.servicenow.com/>
32. <https://www.vmware.com/au/products/aria-automation.html>
33. <https://docs.vmware.com/en/vRealize-Orchestrator/index.html>
34. <https://www.vmware.com/au/products/cloud-director.html>
35. <https://github.com/ansible/awx>
36. <https://docs.splunk.com/Documentation/Splunk/9.0.3/RESTREF/RESTprolog>

Dell Technologies believes the information in this publication is accurate as of its publication date. The information is subject to change without notice.

Disclaimer: The views, processes or methodologies published in this article are those of the authors. They do not necessarily reflect Dell Technologies' views, processes, or methodologies.

THE INFORMATION IN THIS PUBLICATION IS PROVIDED "AS IS." DELL TECHNOLOGIES MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WITH RESPECT TO THE INFORMATION IN

THIS PUBLICATION, AND SPECIFICALLY DISCLAIMS IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Use, copying and distribution of any Dell Technologies software described in this publication requires an applicable software license.

Copyright © 2023 Dell Inc. or its subsidiaries. All Rights Reserved. Dell Technologies, Dell, EMC, Dell EMC and other trademarks are trademarks of Dell Inc. or its subsidiaries. Other trademarks may be trademarks of their respective owners.