# Table of Contents

Lab Overview - HOL-1911-06-SDC - vSphere Automation and Development - API and SDK ........................................................................................................................................... 2  
Lab Guidance .................................................................................................................................................................................. 3  
Module 1 - Accessing the API Explorer (15 minutes) ...................................................................................................................... 9  
   Introduction .................................................................................................................................................................................. 10  
   Accessing the API Explorer ...................................................................................................................................................... 11  
Module 2 - REST API and Datacenter Command Line Interface (DCLI) (30 minutes) ...................................................... 16  
   Introduction .................................................................................................................................................................................. 17  
   Invoke vCenter REST API calls with Postman Client .................................................................................................................. 19  
   Exploring DCLI features and workflow .................................................................................................................................. 42  
Module 3 - Python and the vSphere Automation API (30 minutes) ...................................................................................... 51  
   Introduction .................................................................................................................................................................................. 52  
   Exploring the Python SDK Directory Structure ....................................................................................................................... 53  
   Managing Virtual Machine using Python and vSphere Automation API ................................................................................... 58
Lab Overview - HOL-1911-06-SDC - vSphere Automation and Development - API and SDK
Lab Guidance

Note: It may take more than 90 minutes to complete this lab. You should expect to only finish 2-3 of the modules during your time. The modules are independent of each other so you can start at the beginning of any module and proceed from there. You can use the Table of Contents to access any module of your choosing.

The Table of Contents can be accessed in the upper right-hand corner of the Lab Manual.

Rainpole Systems is an electronics manufacturer located in Palo Alto, CA. Rainpole designs and manufactures electronic devices for use in everything from aircraft instrumentation to home automation. Given the diversity of their product set, Rainpole needs to develop cloud-based IT services to support the increasing demands of the business units.

Rainpole Systems is planning to use vCenter API's to automate the build, test and deployment of their custom developed eCommerce application. They also plan to leverage the Python and Java SDK's for vCenter API's for IT DevOps to minimize the chance of errors impacting their production cloud environment.

VMware vCenter REST API's simplifies the automation of complex IT tasks and integrates with REST clients like POSTMAN and also comes integrated with Java and Python via SDKs to adapt and extend service delivery and operational management, effectively working with existing infrastructure, tools and processes. You have the most knowledge around existing processes and have been tasked with coming up to speed on VMware vCenter REST API's and starting down the path of automating existing processes.

As you work through the lab's use cases, you'll be assuming roles within a fictional company, Rainpole, and addressing their business and IT challenges.

You will take on the roles of a Rainpole Cloud Administrator, a Rainpole Developer and a Development Manager in this exciting lab exercise to experience how VMware vCenter API's can help make these goals a reality.

- **Module 1 - Accessing the API Explorer** (15 minutes) (Basic) In this module, we will access the API explorer and learn how it can be used to assist us in developing tools using the vSphere Automation API.
- **Module 2 - API and Datacenter Command Line Interface (DCLI)** (30 minutes) (Basic) In this module, we will explore the use of REST API calls using Postman and the Datacenter Command Line Interface (DCLI).
- **Module 3 - Python and the vSphere Automation API** (30 minutes) (Basic) In this module, we will explore the Python SDK structure and then we will use Python with the vSphere Automation API.
Lab Captains:

- Module 1 - Howard Shoobe, Professional Services, USA
- Module 2 - Howard Shoobe, Professional Services, USA
- Module 3 - Howard Shoobe, Professional Services, USA

This lab manual can be downloaded from the Hands-on Labs Document site found here:

http://docs.hol.vmware.com

This lab may be available in other languages. To set your language preference and have a localized manual deployed with your lab, you may utilize this document to help guide you through the process:


**Location of the Main Console**

1. The area in the RED box contains the Main Console. The Lab Manual is on the tab to the Right of the Main Console.
2. A particular lab may have additional consoles found on separate tabs in the upper left. You will be directed to open another specific console if needed.
3. Your lab starts with 90 minutes on the timer. The lab cannot be saved. All your work must be done during the lab session. But you can click the **EXTEND** to increase your time. If you are at a VMware event, you can extend your lab time twice, for up to 30 minutes. Each click gives you an additional 15 minutes.
Outside of VMware events, you can extend your lab time up to 9 hours and 30 minutes. Each click gives you an additional hour.

**Alternate Methods of Keyboard Data Entry**

During this module, you will input text into the Main Console. Besides directly typing it in, there are two very helpful methods of entering data which make it easier to enter complex data.

**Click and Drag Lab Manual Content Into Console Active Window**

You can also click and drag text and Command Line Interface (CLI) commands directly from the Lab Manual into the active window in the Main Console.

**Accessing the Online International Keyboard**

You can also use the Online International Keyboard found in the Main Console.

1. Click on the Keyboard Icon found on the Windows Quick Launch Task Bar.
Click once in active console window

In this example, you will use the Online Keyboard to enter the "@" sign used in email addresses. The "@" sign is Shift-2 on US keyboard layouts.

1. Click once in the active console window.
2. Click on the Shift key.

Click on the @ key

1. Click on the "@ key".

Notice the @ sign entered in the active console window.
Activation Prompt or Watermark

When you first start your lab, you may notice a watermark on the desktop indicating that Windows is not activated.

One of the major benefits of virtualization is that virtual machines can be moved and run on any platform. The Hands-on Labs utilizes this benefit and we are able to run the labs out of multiple datacenters. However, these datacenters may not have identical processors, which triggers a Microsoft activation check through the Internet.

Rest assured, VMware and the Hands-on Labs are in full compliance with Microsoft licensing requirements. The lab that you are using is a self-contained pod and does not have full access to the Internet, which is required for Windows to verify the activation. Without full access to the Internet, this automated process fails and you see this watermark.

This cosmetic issue has no effect on your lab.

Look at the lower right portion of the screen

![Lab Status and Network Adapter Image]
Please check to see that your lab is finished all the startup routines and is ready for you to start. If you see anything other than "Ready", please wait a few minutes. If after 5 minutes your lab has not changed to "Ready", please ask for assistance.
Module 1 - Accessing the API Explorer (15 minutes)
Introduction

In this module, you will learn about the API Explorer.

The API Explorer allows users to connect to the API endpoint and then browse and interact with available REST based APIs and their calls. This includes available parameters, expected responses, what response status codes may mean, and much more. If you’ve happened to use the Managed Object Browser, or MOB, think of that but on steroids!

This Module contains the following lessons:

- **Accessing the API Explorer**: In this lesson, you will learn access and navigate the API Explorer.
Accessing the API Explorer

In this module we will look at the API Explorer

Accessing the API Explorer is incredibly easy. It's available on any API endpoint whether that be a vCenter server (appliance or Windows based) or external PSC appliance.

Open API Explorer in Chrome

Launch Chrome

1. Launch Google Chrome by clicking the "Chrome" icon on the taskbar.

2. Type the URL of the vCenter Server or PSC followed by "'/apiexplorer/'" and hit enter.

https://vcsa-01a.corp.local/apiexplorer
After having loaded the API Explorers interface, its time to start looking around and seeing what's available. There are two items that should jump out immediately, the login option and the Select API dropdown.

- Appliance handles calls to the underlying VMware appliance
- CIS Common Infrastructure Services, handles calls involving tagging
- Content handles calls involves the Content Library
- vAPI vSphere API, handles calls regarding the API endpoint
- vCenter handles calls to involving the vCenter
1. Select vCenter from API Drop Down.
2. Click on **Cluster**.
3. Click on /vcenter/cluster.

4. Scroll all the way down until you see the "Try it out button"

5. Click the Try it out button.

6. We can see the path to make the call and the method. However, the API Explorer also gives some other information such as the expected response class,
parameters and potential response messages. Clicking on those items expands and collapses additional information. Scroll down to reveal more information.

We have now reached the end of this module. Before starting the next module, you may want to look around and become more familiar with the API explorer.
Module 2 - REST API and Datacenter Command Line Interface (DCLI) (30 minutes)
Introduction

In this module, you will learn how to build and and invoke vCenter REST API calls using Postman Client and DCLI.

If you have not already reviewed the Lab Introduction, it contains details about Rainpole Systems (our example company), as well as important information on text entry, multiple language support and on-screen keyboard configuration. If desired, you can click here to review this information again.

This Module contains the following lessons:

- **Invoke vCenter REST API calls with Postman Client**: Learn how to build and invoke vCenter REST API calls using the Postman client.
- **Exploring DCLI features and workflow**: Learn how to automate various vCenter administrative tasks using Datacenter Command Line Interface (DCLI).

Introducing VMware vCenter REST API

vCenter REST API is a developer friendly, simplified API which allows for Virtual Machine creation, modification and deletion via a consistent set of developer and automation tooling. This will allow for consistent interaction with the API via the tools which make sense to the developer community or someone automating VMware products.

The focus of this lab session will be on vCenter REST API Calls.

Overview/Benefits

VMware helps enterprise IT overcome cloud management challenges with the following differentiators:

- **Achieving Fastest Time-to-Cloud Value**: extensive out-of-the-box capabilities, extensible architecture and ability to leverage existing investments, enables you to create scalable private, public and desktop cloud services in just days
- **Rapid Ecosystem Integration**: a full spectrum of extensibility options that empowers IT personnel to enable, adapt, and extend their cloud to work within their existing IT infrastructure and processes, thereby eliminating expensive service engagements while reducing risk
- **Embracing Consumerization of IT**: unifies disparate systems and processes into a highly personalized, self-service experience governed by user and organizational awareness, ensuring the exact needs of the user are met in the context of the business
- **Boosting IT Efficiency**: VMware's advanced self-service and automation capabilities lead to a significant reduction in operational expenditures (OpEx),
while organizationally aware governance enables a multi-tenant infrastructure, improving hardware utilization and eliminating capital expenditures (CapEx)

- **Increasing Business Agility**: enterprise IT is empowered to quickly enable cloud services so that the business can react quickly to changing market demands and capture market share
- **Hybrid Cloud Automation**: agnostic provisioning of hardware platforms, operating systems, hypervisors, management tools, and public clouds; vRealize Automation allows customers to rapidly stand up cloud services to deliver quick time to value from your IT investments
Invoke vCenter REST API calls with Postman Client

vCenter REST API is a developer friendly, simplified API which allows for Virtual Machine creation, modification and deletion via a consistent set of developer and automation tooling. This allows for consistent interaction with the API via the tools which make sense to the developer community or someone automating VMware products, included in this are:

- Multiple SDKs
- PowerCLI
- Datacenter CLI (DCLI)
- vRealize Orchestrator

Featured uses of this API include:

- **Cluster** - Get/List
- **Datacenter** - Create/Get/List/Delete
- **Datastore** - Get/List
- **Folder** - List
- **Host** - Create/Get/List/Delete/Connect/Disconnect
- **Network** - List
- **Resource pool** - Get/List
- **VM** - Create/Get/List/Delete
  - **Power** - Get/Start/Stop/Suspend/Reset
  - **Hardware** - Get/Update/Upgrade
    - **Boot** - Get/Update
    - **Device** - Get/Set
    - **Disk** - Get/Set
  - **Cdrom** - Create/Get/List/Delete/Update/Connect/Disconnect
  - **Cpu** - Get/Update
  - **Disk** - Create/Get/List/Delete/Update
  - **Ethernet** - Create/Get/List/Delete/Update/Connect/Disconnect
  - **Floppy** - Create/Get/List/Delete/Update/Connect/Disconnect
  - **Memory** - Get/Update
  - **Parallel** - Create/Get/List/Delete/Update/Connect/Disconnect
  - **Serial** - Create/Get/List/Delete/Update/Connect/Disconnect
  - **Adapter**
    - **Sata** - Create/Get/List/Delete
    - **SCSI** - Create/Get/List/Delete/Update

In previous versions of vSphere the primary API for management has been SOAP based and used heavily by partners through SDKs and by Administrators via vCLI, PowerCLI or vRealize Orchestrator.
In today’s modern development world the API, Developers and Automation have become a focus of modern IT, the new vCenter REST API has been designed to fulfill the needs of a modern API by providing a simple, modern and expansive API surface and consistent tools and SDKs to access this.

During the workflows on the next pages you will realize that even if you are not a developer you will be able to make use of the API and tooling to automate some of your Virtual Machine tasks.

**Disable GPU for Postman**

In order to for Postman to work properly in our environment, we need to disable the ability for Postman to use the GPU.

1. Click on the Windows Start icon.

2. Type *System Variable* in the dialog box
3. Click on **Edit the system environment variables**
4. Click on the **Environment Variables...** button
5. Click the **New...** button to create a new System variable

6. Enter `POSTMAN_DISABLE_GPU` in the Variable name text box
7. Enter `true` in the Variable value text box
8. Click the OK button
9. Click the OK button

**Launching the Postman Client**

Postman is a free application which can be used to organize and send requests to a **REST** based **API**, you will be using this during the steps below to work with this new API.
1. Double-Click the Postman Icon on the desktop.

1. Once Postman launches click the "X" to close the pop-up Window
Become familiar with Postman UI to call the vCenter API

![Postman UI to call vCenter API](image)

1. Select the GET method.
2. Enter the API endpoint URL: `https://vcenter61.corp.local/rest/com/vmware/cis/session`.
3. Click the Send button to execute the request.

Disable SSL certificate verification

![Disabling SSL certificate verification](image)

1. Click on File.
2. Select Settings.

---

HOL-1911-06-SDC  Page 25
We are using self-signed certificates in our lab. As a result, it is necessary to disable SSL certificate verification in Postman.

1. Click on **File** on the menu bar.
2. Select **Settings** from the presented menu options.

3. Click the toggle switch next to **SSL certificate verification** to turn it off. The switch will be gray with the word **OFF** next to it.
4. Click the **X** to close the Settings window.
Login to vCenter Server to Establish our Session

The first call necessary is to establish log into our vCenter server and establish a HTTP session. This will be a POST request using Basic Authentication.

1. Use the REST method dropdown to select the desired REST method. As indicated, the login call is a **POST** call.
2. Click on the Authorization Tab to view authentication methods.
3. Use the Type dropdown to select the **Basic Auth** Authentication type.
4. Enter credentials
   - Username: administrator@corp.local
   - Password: VMware1!
5. Type the following REST api command in the URI field.

   https://vcsa-01a.corp.local/rest/com/vmware/cis/session
6. Click the “Send” button to send the API request.

7. Upon successful execution of the command, the session token will be displayed in the body area of the Postman REST client.

Note that the response to the API command is delivered in JSON format.
Add the Session ID to the Header

1. Change the Authorization type to **No Auth**

2. Click on the **Header** tab

3. Enter **vmware-api-session-id** in the header key
4. Copy the session ID value
5. Paste the session ID value into the Value box besides the header key entered above.

Request a Datacenter Summary

Now that we have established our session, let's get some information about what datacenters are in our vCenter environment.

1. Use the REST method dropdown to select the desired REST method. Because we are requesting information from the vCenter Server, this is a **GET** call.
2. Type the following REST api command in the URI field.
   
   ```
   https://vcsa-01a.corp.local/rest/vcenter/datacenter
   ```

4. Click the “Send” button to send the API request.
7. Upon successful execution of the command, the body of the response includes a JSON string. In our example, we have a datacenter named **RegionA01** with a datacenter ID of **datacenter-21**. We will use this datacenter ID for our next command. It may not be the same as the datacenter ID used in our example.
Show Datacenter Details

The following **GET** request will report the datacenter details for the **datacenter id** we made a note of in the previous step.

Now that we have established our session, let's get some information about what datacenters are in our vCenter environment.

1. As with our prior step, we are requesting information from the vCenter Server. Therefore, this is another **GET** call.
2. Type the following REST api command in the URI field. You can see that this is the same string we used before with the addition of the datacenter ID for which we want additional information. Use the datacenter ID returned in the prior step.

   ```
   https://vcsa-01a.corp.local/rest/vcenter/datacenter/datacenter-21
   ```

3. Click the “Send” button to send the API request.
4. Upon successful execution of the command, the body of the response includes a **JSON** string. You can now see additional details about the datacenter.
Request a Datastore Summary

1. As with our prior step, we are requesting information from the vCenter Server. Therefore, this is another GET call.
2. Type the following REST api command in the URI field.
   
   https://vcsa-01a.corp.local/rest/vcenter/datastore

3. Click the “Send” button to send the API request.
4. Upon successful execution of the command, the body of the response includes a JSON string. In our example, we have a datastore named RegionA01-ISCSI01-COMP01 with a datacenter ID of datastore-37. For any datastore specific API calls, you would use this datastore ID to identify the datastore on which we want to act. We also see additional information including datastore type, total capacity and free space. Make note of the datastore ID because we will be using it later in this module.

Now let's get some information about what datastores are in our vCenter environment.

1. As with our prior step, we are requesting information from the vCenter Server. Therefore, this is another GET call.
2. Type the following REST api command in the URI field.

   https://vcsa-01a.corp.local/rest/vcenter/datastore
Request all Folders

1. GET
3. Send
4. JSON

```
>>= [ 
  { 
    "folder": "group-01", 
    "name": "Datacenters", 
    "type": "DATACENTER"
  }
```

```
  { 
    "folder": "group-02", 
    "name": "host", 
    "type": "HOST"
  }
```

```
  { 
    "folder": "group-03", 
    "name": "network", 
    "type": "NETSVM"
  }
```

```
  { 
    "folder": "group-04", 
    "name": "datastore", 
    "type": "DATASTORE"
  }
```

```
  { 
    "folder": "group-05", 
    "name": "vm", 
    "type": "VMDK"
  }
```

```
  { 
    "folder": "group-06", 
    "name": "virtualmachine", 
    "type": "VMMOM_VMCHREW" 
  }
```

```
Filter Folder Types

As we saw in our example above, we can easily list all the folders in our vCenter environment. However, if you have a large environment, you may want to look at only virtual machine folders. You can do this by applying a filter to our command.

1. As with our prior steps, we are requesting information from the vCenter Server. Therefore, this is another GET call.
2. Type the following REST API command in the URI field.


3. Click the “Send” button to send the API request.
4. Upon successful execution of the command, the body of the response includes a JSON string with information about only the virtual machine folders in the vCenter environment. Make note of the folder IDs because we will be using one later in this module.
Request all Resource Pools

Now let's get some information about what folders are in our vCenter environment.

1. As with our prior steps, we are requesting information from the vCenter Server. Therefore, this is another GET call.
2. Type the following REST api command in the URI field.

   https://vcsa-01a.corp.local/rest/vcenter/resource-pool

3. Click the “Send” button to send the API request.
4. Upon successful execution of the command, the body of the response includes a JSON string with information about all the resource pools in the vCenter environment. Make note of the resource_pool ID because we will be using it later in this module.
Create a simple VM

Now that we have retrieved necessary information from our vCenter server, we can provision a simple virtual machine.

1. For this API call, we will be sending or posting information to the API interface on our vCenter server. Therefore, we need to use the dropdown box to change the method to **POST**.
2. Type the following REST api command in the URI field. 
   
   ```
   https://vcsa-01a.corp.local/rest/vcenter/vm
   ```
3. We will be sending the necessary data for our VM request in the body. Therefore, we need to select the Body tab.
4. The data type we will be sending will be a JSON in **raw** text format. Therefore, we need to select raw for the data type.
5. As identified in the prior step, we will be using JSON format. Therefore in the format dropdown, we need to select **JSON(application/json)**
6. In many of our previous examples, the body field was only used for information retrieved from the vCenter server. In this case, we need to create a JSON object which contains the information necessary to provision a virtual machine. The datastore ID, folder ID and resource pool ID that we will be using are based on prior steps in this lesson. Type the following code into the body area.

   ```
   
   ```
7. Click the “Send” button to send the API request

8. Upon successful execution of the command, the body of the response includes a JSON string with the virtual machine ID of the newly created virtual machine.
Create a Fully Configured VM

In the prior step, we provisioned a basic virtual machine with just default configuration. Now, we will go a bit further and provision a virtual machine with a bit more configuration including memory capacity, number of vCPU, a CD ROM and an IDE drive.

Now that we have retrieved necessary information from our vCenter server, we can provision a simple virtual machine.

1. As in our prior example, we will be sending or posting information to the API interface on our vCenter server. Therefore, we need to use the dropdown box to change the method to **POST**.
2. Type the following REST api command in the URI field. This is the same command as our prior example.
3. As in our prior example, we will be sending the necessary data for our VM request in the body. Therefore, we need to select the Body tab.
4. As in our prior example, the data type we will be sending will be a JSON in raw text format. Therefore, we need to select raw for the data type.
5. As in our prior example, we will be using JSON format. Therefore in the format dropdown, we need to select JSON(application/json)
6. We will now modify the JSON from the prior example by providing some additional fields. Type the following code into the the body area.

```json
{
    "spec": {
        "name": "Full Configured VM",
        "guest_OS": "RHEL_7_64",
        "placement": {
            "datastore": "datastore-37",
            "folder": "group-v22",
            "resource_pool": "resgroup-27"
        },
        "memory": {
            "size_MiB": 16,
            "hot_add_enabled": true
        },
        "cpu": {
            "hot_remove_enabled": true,
            "count": 1,
            "hot_add_enabled": true,
            "cores_per_socket": 1
        },
        "cdroms": [
            {
                "type": "IDE"
            }
        ],
        "disks": [
            {
                "new_vmdk": {},
                "type": "IDE",
                "ide": {}
            }
        ]
    }
}
```
7. Click the “Send” button to send the API request

8. Upon successful execution of the command, the body of the response includes a JSON string with the virtual machine ID of the newly created virtual machine.
Now that we have created a couple of virtual machines, let's execute an API call to get a listing of all VM's.

1. As with some of our earlier examples, we are requesting information from the vCenter Server. Therefore, this is another **GET** call.
2. Type the following REST api command in the URI field.
   
   ```
   https://vcsa-01a.corp.local/rest/vcenter/vm
   ```

3. Click the “Send” button to send the API request.
4. Upon successful execution of the command, the body of the response includes a JSON string with information about all the virtual machines in the vCenter environment.
Exploring DCLI features and workflow

Datacenter Command Line Interface (DCLI) is a command line designed to provide automation and a troubleshooting for features exposed via the vSphere REST API. This is just one of the interfaces which can be used to work with this API, others include PowerCLI, vRealize Orchestrator and various SDKs.

During the following steps you will learn how the knowledge of the API will easily transfer to the CLI and vice-versa, you will see that this CLI can be used as a multi platform CLI to work with vSphere and achieve basic automation and troubleshooting workflows.

DCLI comes by default with the VCSA, vCenter for Windows and is available as part of vCLI.

Connect to the vCenter server using DCLI and Windows Command Prompt.
To access the DCLI, we are going to open a command prompt for the DCLI.

1. Click on the Windows Start Button
2. Click on All Programs

3. On the all programs menu, select **VMware -> VMware vSphere CLI** and then click on the **Command Prompt** icon.
Connect to the vCenter server using DCLI and Windows Command Prompt Continued.

1. Type in the following command to open DCLI in interactive mode and hit the enter key.

```
dcli +server vcsa-01a +skip-server-verification +interactive
```

**Note:** DCLI can be used in both interactive and single command mode
Explore DCLI help

1. Type in the following command to explore DCLI help and hit the enter key.

   --help

2. You will see that the options can be used by providing a + to the DCLI command, these include the ability to output in many formats, run interactively, provide a server to connect to and many more options, familiarize yourself with some of the functionality.
List VMs using DCLI

Our first command using the DCLI will be to list the virtual machines under the control of our vCenter, vcsa-01a.

1. Type in the following command to list VMs and hit enter key. As you are typing each word in the command, you will notice that the DCLI provides suggestions for valid options. This command directly maps to GET /rest/vcenter/vm. This will return a list of vms which are being managed by the vCenter server.

   vmware vcenter vm list

2. Because we have not yet provided credentials for our vCenter server, we are prompted to enter Username then Password on the execution of our first command. Enter administrator@corp.local and VMware1! as username and password respectively.
3. In order to avoid having to enter these credentials again during this session, select yes to save the credentials.
4. Now that we have logged in, we see a list of the virtual machines under management by our vCenter.
Report VM Data in different formats

By adding the +formatter following our command, we can select different format styles for our output.

1. Type in the following command to display our vm list in "simple" mode

```
vmware vcenter vm list +formatter simple
```
2. Alternatively, you might want to display your vm list in "json" format. To do so, type in the following command.

```
vmware vcenter vm list +formatter json
```

**List Folders, Datastores and ResourcePools with DCLI**

Type in the following commands to report a list of folders, datastores and resourcepools available in the cluster using DCLI.

1. Type in the following command to report a list of folders

```
vmware vcenter folder list
```

2. Type in the following command to report a list of datastores

```
vmware vcenter datastore list
```

3. Type in the following command to report a list of resourcepools

```
vmware vcenter resourcepool list
```

**NOTE:** For our next step, keep track of the virtual machine folder ID, datastore ID and resource pool ID.
Create VMs using DCLI

We can use the DCLI to create virtual machines as we did with the REST API.

1. To create a simple virtual machine, enter the following code in at the command prompt. You will need to use the virtual machine folder ID, datastore ID and resource pool ID identified in the prior step.

```bash
vmware vcenter vm create --guest-os CENTOS_7_64 --placement-folder group-v22
--placement-datastore datastore-37 --placement-resource-pool resgroup-27
```

2. Upon successful execution of the command, the DCLI will return the virtual machine ID for the newly created virtual machine.

3. For a virtual machine with some additional configuration, we can add a few additional switches to specify vCPUs, memory etc.

```bash
vmware vcenter vm create --guest-os CENTOS_7_64 --placement-folder group-v22
--placement-datastore datastore-37 --placement-resource-pool resgroup-27 --cpu-count 1
--memory-size-mib 16 --cpu-hot-add-enabled true --memory-hot-add-enabled true
```

4. Upon successful execution of the command, the DCLI will return the virtual machine ID for the newly created virtual machine.

NOTE: For our next step, keep track of the virtual machine ID.
Retrieve virtual machine information in the DCLI

Now that we have created a virtual machine, let's use the virtual machine ID to request details about our new virtual machine through DCLI.

1. Type the following code on the command line being sure to replace the virtual machine ID with a virtual machine ID from one of the virtual machines you created.

   ```bash
   vmware vcenter vm get --vm vm-202
   ```

2. Upon successful execution of the command, the DCLI will return the details about the virtual machine.

3. You can now close the DCLI command line window.
Module 3 - Python and the vSphere Automation API (30 minutes)
Introduction

In this module, you will learn how to work with Python and vSphere API.

If you have not already reviewed the Lab Introduction, it contains details about Rainpole Systems (our example company), as well as important information on text entry, multiple language support and on-screen keyboard configuration. If desired, you can click here to review this information again.

This Module contains the following lessons:

- **Exploring the Python SDK Directory Structure**: In this lesson you will explore and learn about the contents of the Python SDK directory.
- **Managing Virtual Machine using Python and vSphere Automation API**: Learn how to automate Virtual Machine management using Python and vSphere Automation API.

Overview/Benefits

VMware helps enterprise IT overcome cloud management challenges with the following differentiators:

- **Achieving Fastest Time-to-Cloud Value**: extensive out-of-the-box capabilities, extensible architecture and ability to leverage existing investments, enables you to create scalable private, public and desktop cloud services in just days
- **Rapid Ecosystem Integration**: a full spectrum of extensibility options that empowers IT personnel to enable, adapt, and extend their cloud to work within their existing IT infrastructure and processes, thereby eliminating expensive service engagements while reducing risk
- **Embracing Consumerization of IT**: unifies disparate systems and processes into a highly personalized, self-service experience governed by user and organizational awareness, ensuring the exact needs of the user are met in the context of the business
- **Boosting IT Efficiency**: VMware's advanced self-service and automation capabilities lead to a significant reduction in operational expenditures (OpEx), while organizationally aware governance enables a multi-tenant infrastructure, improving hardware utilization and eliminating capital expenditures (CapEx)
- **Increasing Business Agility**: enterprise IT is empowered to quickly enable cloud services so that the business can react quickly to changing market demands and capture market share
- **Hybrid Cloud Automation**: agnostic provisioning of hardware platforms, operating systems, hypervisors, management tools, and public clouds; vRealize Automation allows customers to rapidly stand up cloud services to deliver quick time to value from your IT investments
Exploring the Python SDK Directory Structure

The VMware vCloud Suite SDK for Python enables programmatic access to vSphere. Access to capabilities exposed through the vSphere Web Services API is done using the open-source pyVim and pyVmomi projects. The SDK includes python libraries for accessing new features like Content Library and existing features like Tagging which did not support a public API prior to latest release of vSphere. The SDK contains samples for the features mentioned above as well as samples to demonstrate how to interoperate with vSphere APIs. If you want to look at the vSphere API samples,

The VMware vCloud Suite SDK for Python is intended for the following audiences:

- Developers who want to write new application(s) for future releases of vSphere.
- Developers migrating an existing vSphere application to use new features.

The vCloud Suite Client SDK for Python provides client bindings, documentation, samples, and WSDL files. The following table shows the client directories and their contents.

In this module we will explore the content of the Python SDK directory structure.

Exploring the Python SDK directory structure

1. Click the Windows Explorer icon on the taskbar
Exploring the Directory Structure for Python packages

1. Enter the following path in the Windows Explorer address bar:

   C:\Users\Administrator\AppData\Local\Programs\Python\Python36-32\Lib\site-packages

2. The two packages in which we are interested are in the pyVim and pyVmomi directories. In the next steps, we will explore these two directories.
3. The pyVim directory contains the connect.py package which includes a class supporting creation and dissolution of connection to our vSphere environment.

4. The pyVmomi package includes the VMware Object Model consisting of a number of files and classes.
Exploring the Directory Structure for Python SDK

1. Enter the following path in the Windows Explorer address bar. This will bring you to the directory contains all of the folders and files available in the Python SDK.

   C:\labfiles\HOL-1911\vsphere-automation-sdk-python-master

2. The **doc** folder contains documentation in html format for vSphere, NSX, and VMware Cloud.

3. The **samples** folder contains folders of example Python scripts for both vSphere and VMware Cloud.
Exploring the Directory Structure for Python SDK - vSphere Samples
Managing Virtual Machine using Python and vSphere Automation API

In this module, we use the python interactive console to illustrate the usage of Python with vSphere API. The vSphere Python API allows for Virtual Machine creation, modification and deletion via a consistent set of developer and automation tooling. This allows for consistent interaction with the API via the tools which make sense to the developer community or someone automating VMware products.

Modify the Path Variable

Before we begin this module using Python, we need to add the python directory to the Path variable.

1. Open the command line by clicking on the icon in the taskbar.

2. Enter the following at the command prompt and then press Enter.

```plaintext
setx path "%path%;C:\Users\Administrator\AppData\Local\Programs\Python\Python36-32"

```

2. Enter the following at the command prompt and then press Enter.
setx path "%path%;C:\Users\Administrator\AppData\Local\Programs\Python\Python36-32"

3. After the path variable value is saved, click the "x" to close the cmd window.

**Lauch Python Interactive Console and load the VMware SDK**

1. Double-click the Powershell icon on the Control Center desktop.
2. To start the Python interactive console enter python at the command prompt and then click Enter.

```
python
```

3. Now we will import the VMware SDK by typing the following at the Python command prompt and then click Enter.

```
import pyVim
from pyVim import connect
import pyVmomi
from pyVmomi import vim
```
4. In order to connect to our vCenter, we will import the connect package. Type the following at the Python command prompt and then click Enter.

```python
from pyVim import connect
```

5. A second important portion of the VMware Python SDK is the object model. This is imported by typing the following at the Python command prompt and then clicking Enter.

```python
import pyVmomi
```

6. Lastly we will import the vim package. Type the following at the Python command prompt and then click Enter.

```python
from pyVmomi import vim
```

**Create a connection to our vCenter Server**

```
my_vCenter = connect.Connect("vcsa-01a.corp.local",443,"administrator@corp.local","VMware1!")
```

1. We will use the Connect class in the connect package to create our vCenter connection object by typing the following at the command prompt.
To get the best insight into how we will work with the Python SDK, we will be using the managed object browser in conjunction with our Python Code. We already have our Powershell Window open and need to open a browser Window as well to look at the MOB.

1. Double click on the Chrome icon on the control center desktop.
2. Enter the following URL in the address bar and hit Enter. A dialog box will be displayed for you to enter both Username (administrator@corp.local) and Password (VMware1!)

   https://vcsa-01a.corp.local/mob

3. The vCenter server will respond with the root of the MOB. This corresponds to the my_vCenter object we created with our connection in the prior step. This object has three properties and five Methods. The method we are going to investigate is Current Time.
4. In your PowerShell/Python window, type the following at the command line and press Enter. The vCenter will respond to this API method with the current vCenter Time.
print(my_vCenter.CurrentTime())
5. Now we will dig a little deeper into the MOB by clicking the `content` property.
<table>
<thead>
<tr>
<th>NAME</th>
<th>TYPE</th>
<th>ABOUTInfo</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>accountManager</td>
<td>ManagedObjectReference:HostLocalAccountManager</td>
<td>about</td>
<td>Unset</td>
</tr>
<tr>
<td>alarmManager</td>
<td>ManagedObjectReference:AlarmManager</td>
<td>AboutInfo</td>
<td>AlarmManager</td>
</tr>
<tr>
<td>authorizationManager</td>
<td>ManagedObjectReference:AuthorizationManager</td>
<td>certificateManager</td>
<td>AuthorizationManager</td>
</tr>
<tr>
<td>certificateManager</td>
<td>ManagedObjectReference:CertificateManager</td>
<td>clusterProfileManager</td>
<td>CertificateManager</td>
</tr>
<tr>
<td>complianceManager</td>
<td>ManagedObjectReference:ProfileComplianceManager</td>
<td>customFieldsManager</td>
<td>ComplianceManager</td>
</tr>
<tr>
<td>cryptoManager</td>
<td>ManagedObjectReference:CryptoManager</td>
<td>complianceManager</td>
<td>CryptoManager</td>
</tr>
<tr>
<td>customFieldsManager</td>
<td>ManagedObjectReference:CustomFieldsManager</td>
<td>datastoreNamespaceManager</td>
<td>CustomFieldsManager</td>
</tr>
<tr>
<td>datastoreNamespaceManager</td>
<td>ManagedObjectReference:DatastoreNamespaceManager</td>
<td>datastoreManager</td>
<td>CustomizationSpecManager</td>
</tr>
<tr>
<td>diagnosticManager</td>
<td>ManagedObjectReference:DiagnosticManager</td>
<td>datastoreManager</td>
<td>Diaglog</td>
</tr>
<tr>
<td>dvSwitchManager</td>
<td>ManagedObjectReference:DistributedVirtualSwitchManager</td>
<td>eventManager</td>
<td>DVSwitchManager</td>
</tr>
<tr>
<td>eventManager</td>
<td>ManagedObjectReference:EventManager</td>
<td>about</td>
<td>EventManager</td>
</tr>
<tr>
<td>extensionManager</td>
<td>ManagedObjectReference:ExtensionManager</td>
<td>failoverClusterConfigurator</td>
<td>ExtensionManager</td>
</tr>
<tr>
<td>failoverClusterConfigurator</td>
<td>ManagedObjectReference:FailoverClusterConfigurator</td>
<td>failoverClusterManager</td>
<td>FailoverClusterConfigurator</td>
</tr>
<tr>
<td>failoverClusterManager</td>
<td>ManagedObjectReference:FailoverClusterManager</td>
<td>fileManager</td>
<td>FailoverClusterManager</td>
</tr>
<tr>
<td>fileManager</td>
<td>ManagedObjectReference:FileManager</td>
<td>guestOperationsManager</td>
<td>FileManager</td>
</tr>
<tr>
<td>guestOperationsManager</td>
<td>ManagedObjectReference:GuestOperationsManager</td>
<td>healthUpdateManager</td>
<td>guestOperationsManager</td>
</tr>
<tr>
<td>healthUpdateManager</td>
<td>ManagedObjectReference:HealthUpdateManager</td>
<td>hostProfileManager</td>
<td>healthUpdateManager</td>
</tr>
<tr>
<td>hostProfileManager</td>
<td>ManagedObjectReference:HostProfileManager</td>
<td>hostSpecManager</td>
<td>HostProfileManager</td>
</tr>
<tr>
<td>hostSpecManager</td>
<td>ManagedObjectReference:HostSpecificationManager</td>
<td>ioFilterManager</td>
<td>HostSpecManager</td>
</tr>
<tr>
<td>ioFilterManager</td>
<td>ManagedObjectReference:IoFilterManager</td>
<td>licenseManager</td>
<td>IoFilterManager</td>
</tr>
<tr>
<td>licenseManager</td>
<td>ManagedObjectReference:LicenseManager</td>
<td>localizationManager</td>
<td>IpPoolManager</td>
</tr>
<tr>
<td>localizationManager</td>
<td>ManagedObjectReference:LocalizationManager</td>
<td>overheadMemoryManager</td>
<td>licenseManager</td>
</tr>
<tr>
<td>overheadMemoryManager</td>
<td>ManagedObjectReference:OverheadMemoryManager</td>
<td>ovfManager</td>
<td>localizationManager</td>
</tr>
<tr>
<td>ovfManager</td>
<td>ManagedObjectReference:OvfManager</td>
<td>perfManager</td>
<td>OverheadMemoryManager</td>
</tr>
<tr>
<td>perfManager</td>
<td>ManagedObjectReference:PerformanceManager</td>
<td>propertyCollector</td>
<td>OvfManager</td>
</tr>
<tr>
<td>propertyCollector</td>
<td>ManagedObjectReference:PropertyCollector</td>
<td>rootFolder</td>
<td>propertyCollector</td>
</tr>
<tr>
<td>rootFolder</td>
<td>ManagedObjectReference:Folder</td>
<td>scheduledTaskManager</td>
<td>rootFolder</td>
</tr>
<tr>
<td>scheduledTaskManager</td>
<td>ManagedObjectReference:ScheduledTaskManager</td>
<td>searchIndex</td>
<td>scheduledTaskManager</td>
</tr>
<tr>
<td>searchIndex</td>
<td>ManagedObjectReference:SearchIndex</td>
<td>serviceManager</td>
<td>searchIndex</td>
</tr>
<tr>
<td>serviceManager</td>
<td>ManagedObjectReference:ServiceManager</td>
<td>sessionManager</td>
<td>serviceManager</td>
</tr>
<tr>
<td>sessionManager</td>
<td>ManagedObjectReference:SessionManager</td>
<td>setting</td>
<td>sessionManager</td>
</tr>
<tr>
<td>setting</td>
<td>ManagedObjectReference:OptionManager</td>
<td>snmpSystem</td>
<td>setting</td>
</tr>
<tr>
<td>snmpSystem</td>
<td>ManagedObjectReference:SnmpSystem</td>
<td>storageResourceManager</td>
<td>SnmpSystem</td>
</tr>
<tr>
<td>storageResourceManager</td>
<td>ManagedObjectReference:StorageResourceManager</td>
<td>taskManager</td>
<td>storageResourceManager</td>
</tr>
<tr>
<td>taskManager</td>
<td>ManagedObjectReference:TaskManager</td>
<td>taskManager</td>
<td>SysInfo</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>vCenter</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>vCenter</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>vCenter</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>vCenter</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>vCenter</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>vCenter</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>vCenter</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>vCenter</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>vCenter</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>vCenter</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>vCenter</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>vCenter</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>vCenter</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>vCenter</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>vCenter</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>vCenter</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>vCenter</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>vCenter</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>vCenter</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>vCenter</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>vCenter</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>vCenter</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>vCenter</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>vCenter</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>vCenter</td>
</tr>
</tbody>
</table>
6. This page shows us all the properties of the content object under the rootFolder. We'll continue to dig into the MOB by clicking into the rootFolder of our vCenter. In our example, the ID of the root folder is **group-d1** and the name is **Datacenters**.

![Image of the Managed Object Browser](image)

7. As we explore the rootFolder, we see that our datacenter(s), **RegionA01**, is a child entity of the rootFolder.
8. We also see that there are a number of methods which we can use at the rootFolder level.

Now that we have reviewed the rootFolder in the MOB browser, let's use some Python commands to retrieve the information programatically.

9. First, we will create an object, `my_datacenter`, with the datacenter object we saw above. We will use this object later in this module. Type the following at the command prompt and click Enter. Note that we have specified only the first childEntity of the root folder. If there were multiple datacenters, we retrieve an array of datacenter objects.

```python
my_datacenter = my_vCenter.content.rootFolder.childEntity[0]
```

10. To see what was returned from the command above, type the following at the command line. We will see the object ID returned.

```python
print(my_datacenter)
```

11. We can also see the name of our datacenter by typing the following at the command line.

```python
print(my_datacenter.name)
```
1. Now let's find what clusters we have in our vSphere environment. To create an array of cluster objects, type the following at the command line and click Enter.

```powershell
my_Clusters = my_datacenter.hostFolder.childEntity
```

2. We want to be able to loop through our my_Clusters array so type the following at the command line. Click Enter after each line. Click enter twice after x=x+1.

```powershell
x=0
for cluster in my_Clusters:
    print(my_Clusters[x].name)
    print(my_Clusters[x])
x=x+1
```

3. The output of this small loop will be a list of Cluster names and object IDs.
Request all VMs

1. Now let's find what clusters we have in our vSphere environment. To create an array of cluster objects, type the following at the command line and click Enter.

   ```python
   my_VMs = my_datacenter.vmFolder.childEntity
   ```

2. We want to be able to loop through our my_Clusters array so type the following at the command line. Click Enter after each line. Click enter twice after x=x+1.

   ```python
   x=0
   for vm in my_VMs:
   print(my_VMs[x].name)
   x=x+1
   ```

   Type tab before each of the following lines

   ```python
   print(my_VMs[x].name)
   x=x+1
   ```

3. The output of this small loop will be a list of Virtual Machines and VM folders in the datacenter.

```
Discovered virtual machine
kms-01a
core-01a
app-01a
template-01a
```
Conclusion

Thank you for participating in the VMware Hands-on Labs. Be sure to visit http://hol.vmware.com/ to continue your lab experience online.

Lab SKU: HOL-1911-06-SDC

Version: 20200210-210119