

VMware vCenter Server Management Programming Guide

15 APR 2020

VMware vSphere 7.0

vCenter Server 7.0

VMware ESXi 7.0



vmware®

You can find the most up-to-date technical documentation on the VMware website at:

<https://docs.vmware.com/>

If you have comments about this documentation, submit your feedback to

docfeedback@vmware.com

VMware, Inc.
3401 Hillview Ave.
Palo Alto, CA 94304
www.vmware.com

Copyright © 2016-2020 VMware, Inc. All rights reserved. [Copyright and trademark information.](#)

Contents

About the vCenter Server Management Programming Guide 6

1 Introduction to the vCenter Server APIs 7

About vSphere 7

About ESXi 7

vCenter Server Management Overview 8

Limitations of Programming for vCenter Server 8

API Endpoints for Managing vCenter Server 9

Supplementing the vCenter Server API 9

Direct Console User Interface to vCenter Server 9

vCenter Server Management Interface 9

vCenter Server Appliance Bash Shell 9

vSphere Client and the vCenter Server 10

DCLI and vCenter Server 10

Quick Start with vCenter Server APIs 10

2 Retrieving Service Endpoints 11

Filtering for Predefined Service Endpoints 12

Filter Parameters for Predefined Service Endpoints 13

Connect to the Lookup Service and Retrieve the Service Registration Object 14

Java Example of Connecting to the Lookup Service and Retrieving the Service Registration Object 14

Python Example of Connecting to the Lookup Service and Retrieving a Service Registration Object 15

Retrieve Service Endpoints on vCenter Server Instances 16

Java Example of Retrieving a vSphere Automation Endpoint on a vCenter Server Instance 17

Python Example of Retrieving Service Endpoints on vCenter Server Instances 17

Retrieve a vCenter Server ID by Using the Lookup Service 18

Java Example of Retrieving a vCenter Server ID by Using the Lookup Service 18

Python Example of Retrieving a vCenter Server ID by Using the Lookup Service 19

Retrieve a vSphere Automation API Endpoint on a vCenter Server Instance 20

Java Example of Retrieving a vSphere Automation Endpoint on a vCenter Server Instance 20

Python Example of Retrieving a vSphere Automation Endpoint on a vCenter Server Instance 21

3 Authentication Mechanisms 22

vCenter Single Sign-On User Name and Password Authentication for vCenter Server 22

Authenticate with vCenter Single Sign-On Credentials and Create a Session 22

vCenter Single Sign-On Token Authentication for vCenter Server 26

- [Retrieve a SAML Token](#) 27
- [Create a vSphere Automation Session with a SAML Token](#) 29

4 Authorization Model for Administration of vCenter Server 32

- [Authorization Model Mapping to the vCenter Single Sign-On Domain](#) 32
- [Using the Operator Role](#) 33
- [Using the Admin Role](#) 33
- [Using the SuperAdmin Role](#) 33

5 Installing and Upgrading vCenter Server 34

- [Install Stage 2](#) 34
 - [Setting Up a Newly Installed vCenter Server Instance](#) 34
 - [HTTP Requests for Install Stage 2](#) 36
 - [Use HTTP Requests to Set Up a Newly Deployed vCenter Server Instance](#) 37
 - [Workflows for Install Stage 2](#) 40
- [Upgrade Stage 2](#) 42
 - [Upgrading a vCenter Server Instance](#) 43
 - [HTTP Requests for Upgrade Stage 2](#) 45
 - [Workflows for Upgrade Stage 2](#) 47
- [Historical Data Transfer](#) 49
 - [Deferred Import](#) 50
 - [HTTP Status Codes for Deferred Import](#) 52
 - [Historical Data Import Errors](#) 53
 - [Class Diagrams for Deferred Import](#) 53
 - [Use the Deferred Import Sample](#) 54
 - [Python Example of Pausing and Resuming the Deferred Import Process](#) 55

6 Monitoring vCenter Server 56

- [Health Monitoring of vCenter Server](#) 56
 - [Check Overall System Health of vCenter Server](#) 56
- [Capacity Monitoring of vCenter Server](#) 58
 - [Frequency and Retention of Statistics Collection in vCenter Server](#) 58
 - [Nature of Statistics in vCenter Server](#) 59
 - [Requesting Statistics from vCenter Server](#) 59
 - [Statistics Collection Times](#) 59
 - [Statistics Interval Adjustment in vCenter Server](#) 60
 - [Empty Data Values](#) 60
 - [Check Database Usage in vCenter Server](#) 61
 - [List Storage Consumption By Data Type in vCenter Server](#) 63

7 Maintenance of vCenter Server 67

- Backing up vCenter Server 67
 - Backup and Restore Protocols for vCenter Server 68
 - Calculate the Size Needed To Store the Backup File 68
 - Back up a vCenter Server Instance by Using the API 70
 - Schedule a Backup Job 74
- Restoring vCenter Server 76
 - Authentication When Restoring a vCenter Server Instance 76
 - Availability of Services While Restoring a vCenter Server Instance 76
 - Restore a vCenter Server Instance by Using the API 76
- Reconcile a vCenter Server Instance with Nodes in Embedded Linked Mode 80
- Managing System Logs 81
 - Configuring Syslog Forwarding 81
- Performing Infrastructure Profile Management Operations 86
 - HTTP Requests for Infrastructure Profile Management Operations 87

8 Planning vCenter Server Updates 88

- Performing Discovery and Planning Operations 88
 - HTTP Requests for Discovery and Planning Operations 90
 - cURL Examples of Performing Discovery and Planning Operations 91
- List Available Products and Manage Associated Products 93
 - Python Example of Listing Available Products and Managing Associated Products 94
- List Available Updates 94
 - Python Example of Listing Available Updates 95
- Retrieve a Report 95
 - Python Example of Retrieving a Report 96

9 Updating vCenter Server 97

- Applying vCenter Server Software Updates 97
 - vCenter Server Software Update Workflow 100
 - cURL Examples of Performing vCenter Server Software Update Operations 102

About the vCenter Server Management Programming Guide

The *vCenter Server Management Programming Guide* provides information about using APIs to work with vCenter Server, a turnkey solution for managing data centers featuring VMware[®] vCenter Server and VMware ESXi.

Intended Audience

This information is intended for anyone who wants to develop software to configure, monitor, and manage vCenter Server. The information is written for developers who have some experience with REST APIs, JavaScript, Java, or Python.

Introduction to the vCenter Server APIs

1

vCenter Server provides a fully packaged solution for data center management in a vSphere environment. You can use the APIs to configure, monitor, and maintain vCenter Server.

This chapter includes the following topics:

- [About vSphere](#)
- [About ESXi](#)
- [vCenter Server Management Overview](#)
- [Limitations of Programming for vCenter Server](#)
- [API Endpoints for Managing vCenter Server](#)
- [Supplementing the vCenter Server API](#)
- [Quick Start with vCenter Server APIs](#)

About vSphere

vSphere is the VMware software stack that implements private-cloud data center management and the on-premises component of hybrid-cloud deployments.

A vSphere installation includes one or more instances of vCenter Server configured to manage one or more virtual data centers. Each virtual data center includes one or more instances of VMware ESXi.

The vCenter Server Management API gives you programmatic access to manage the management elements of your data center.

About ESXi

Each instance of ESXi includes management agents and the VMware hypervisor layer, which runs several virtual machines. Each virtual machine contains a guest operating system, such as Windows or Linux, capable of running IT or user applications.

vCenter Server runs as a virtual machine on an ESXi host. vCenter Server provides an independent endpoint capable of handling API requests both for vCenter Server and for the vCenter Server Management API.

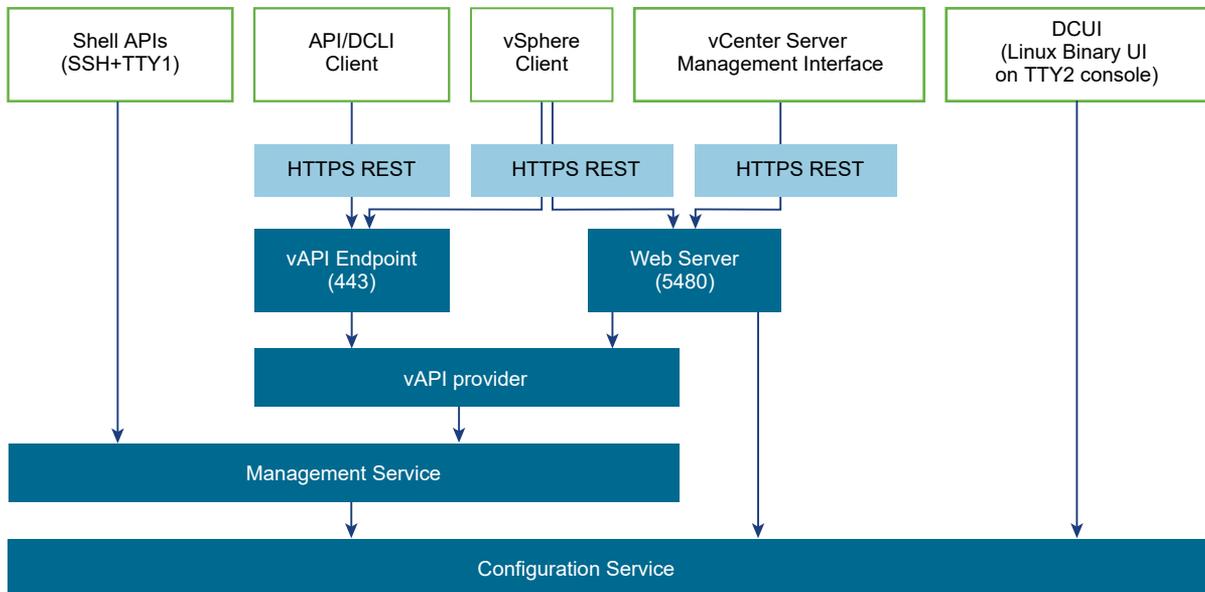
vCenter Server Management Overview

vCenter Server runs in a Photon OS™ guest operating system.

vCenter Server is a collection of services designed for managing and monitoring vSphere installations. vCenter Server responds to CLI commands, requests from the vSphere Client, and API requests from custom clients. API clients can be written in a choice of several software languages.

vCenter Server is managed by CLI, Web interfaces, or API requests. These requests help you manage vCenter Server configuration, monitor resource usage, or back up and restore the vCenter Server instance. You can also use API requests to check the health of vCenter Server. This programming guide explains how to use the APIs that are available to manage vCenter Server.

Figure 1-1. vCenter Server Management Connections



For more information about the capabilities of vCenter Server, see *vCenter Server Configuration*.

Limitations of Programming for vCenter Server

vCenter Server supports several programming interfaces for monitoring health and performance, managing network configuration, security hardening, and other functionalities. vCenter Server also supports several user interfaces, which offer overlapping sets of functionality.

You can use the vSphere Client to perform common operations. By using the API, you have access to more specific settings and operations.

However, the API cannot access all the capabilities. A few special features require direct shell access or special user interfaces. See [Supplementing the vCenter Server API](#).

API Endpoints for Managing vCenter Server

vCenter Server integrates with the vSphere Automation API endpoint that provides a common surface for exposing several vSphere services. When you use the vSphere Automation API endpoint, you establish a single session that provides access to virtual machine management, search and filter, Content Library, and other services for working with vSphere objects.

Other endpoints associated with vCenter Server include the Lookup Service and the vCenter Single Sign-On service. For more information about using the Lookup Service, see [Chapter 2 Retrieving Service Endpoints](#). For more information about using the vCenter Single Sign-On service, see [vCenter Single Sign-On Token Authentication for vCenter Server](#).

Supplementing the vCenter Server API

Some less common features of vCenter Server are not accessible by API. These features require direct shell access or specific user interfaces.

Direct Console User Interface to vCenter Server

The Direct Console User Interface provides access to basic operations for vCenter Server management and set up.

The DCUI provides access to a subset of management functions. It provides direct access to vCenter Server if the vSphere Client and the vCenter Server Management Interface become unavailable.

For an illustration showing vCenter Server connections, see the block diagram [Figure 1-1. vCenter Server Management Connections](#).

After the vCenter Server startup is complete, the DCUI displays basic CPU, memory, and network information on the operator console. The root user can use the DCUI screen to configure network interfaces, DNS, and super administrator password.

vCenter Server Management Interface

The vCenter Server Management Interface is an interface for configuring, monitoring, and patching vCenter Server.

The vCenter Server Management Interface runs in a browser that connects to port 5480 of vCenter Server. The vCenter Server Management Interface provides access to all the service APIs of vCenter Server.

For an illustration showing vCenter Server connections, see the block diagram [Figure 1-1. vCenter Server Management Connections](#).

vCenter Server Appliance Bash Shell

You can use the vCenter Server appliance Bash shell to access all vCenter Server commands and plugins that you use for monitoring, troubleshooting, and configuring the vCenter Server instance through the API.

For an illustration showing vCenter Server connections, see the block diagram [Figure 1-1. vCenter Server Management Connections](#).

For more information about the appliance Bash shell, see *vCenter Server Configuration*.

vSphere Client and the vCenter Server

The vSphere Client is a user interface for general management tasks.

For an illustration showing vCenter Server connections, see the block diagram [Figure 1-1. vCenter Server Management Connections](#).

DCLI and vCenter Server

The Data Center CLI (DCLI) is a CLI client of the VMware vSphere[®] Automation[™] SDK. Almost all methods that are available in the vSphere Automation SDKs are available as DCLI commands.

For an illustration showing vCenter Server connections, see the block diagram [Figure 1-1. vCenter Server Management Connections](#).

For more information about DCLI, see *DCLI User's Guide*.

Quick Start with vCenter Server APIs

You can start using the vCenter Server APIs without accessing the Lookup Service endpoint or the vCenter Single Sign-On endpoint. In a production environment, you might instead use centralized service registration and token authentication.

To use the vCenter Server APIs without the Lookup Service or token authentication, see [vCenter Single Sign-On User Name and Password Authentication for vCenter Server](#).

Retrieving Service Endpoints

2

To access services and resources in the virtual environment, client applications that use the vSphere Automation API must know the endpoints of vSphere Automation and vSphere services. Client applications retrieve service endpoints from the Lookup Service that runs on vCenter Server.

The Lookup Service provides service registration and discovery by using the vSphere Web Services API. By using the Lookup Service, you can retrieve endpoints of services on vCenter Server. The following endpoints are available from the Lookup Service.

- The vCenter Single Sign-On endpoint on vCenter Server. You can use the vCenter Single Sign-On service to get a SAML token and establish an authenticated session with a vSphere Automation API endpoint or a vCenter Server endpoint.
- The vSphere Automation API endpoint on vCenter Server. Through the vSphere Automation endpoint, you can make requests to vSphere Automation API services such as virtual machine management, Content Library, and Tagging.
- The vCenter Server endpoint. In case you want to retrieve service endpoints on a vCenter Server instance that is part of a vCenter Enhances Linked Mode group, use the vCenter Server endpoint to get the node IDs of all linked instances. You can use the node ID of the specific vCenter Server instance to retrieve service endpoints on that instance.
- The vSphere Web Services API endpoint and endpoints of other vSphere Web services that run on vCenter Server.

Workflow for Retrieving Service Endpoints

The workflow that you use to retrieve service endpoints from the Lookup Service might vary depending on the endpoints that you need and their number. Follow this general workflow for retrieving service endpoints.

- 1 Connect to the Lookup Service on vCenter Server and service registration object so that you can query for registered services.
- 2 Create a service registration filter for the endpoints that you want to retrieve.
- 3 Use the filter to retrieve the registration information for services from the Lookup Service.

- 4 Extract one or more endpoint URLs from the array of registration information that you receive from the Lookup Service.

This chapter includes the following topics:

- [Filtering for Predefined Service Endpoints](#)
- [Filter Parameters for Predefined Service Endpoints](#)
- [Connect to the Lookup Service and Retrieve the Service Registration Object](#)
- [Retrieve Service Endpoints on vCenter Server Instances](#)
- [Retrieve a vCenter Server ID by Using the Lookup Service](#)
- [Retrieve a vSphere Automation API Endpoint on a vCenter Server Instance](#)

Filtering for Predefined Service Endpoints

The Lookup Service maintains a registration list of vSphere services. You can use the Lookup Service to retrieve registration information for any service by setting a registration filter that you pass to the `List()` function on the Lookup Service. The functions and objects that you can use with the Lookup Service are defined in the `Lookup.wsdl` file that is part of the SDK.

Lookup Service Registration Filters

You can query for service endpoints through a service registration object that you obtain from the Lookup Service. You invoke the `List()` function on the Lookup Service to list the endpoints that you need by passing `LookupServiceRegistrationFilter`. `LookupServiceRegistrationFilter` identifies the service and the endpoint type that you can retrieve.

Optionally, you can include the node ID parameter in the filter to identify the vCenter Server instance where the endpoint is hosted. When the node ID is omitted, the `List()` function returns the set of endpoint URLs for all instances of the service that are hosted on different vCenter Server instances in the environment.

For example, a `LookupServiceRegistrationFilter` for querying the vSphere Automation service has these service endpoint elements.

Table 2-1. Service Registration Filter Parameters

Filter Types	Value	Description
LookupServiceRegistrationServiceType	<code>product= "com.vmware.cis"</code>	vSphere Automation namespace.
	<code>type="cs.vapi"</code>	Identifies the vSphere Automation service.
LookupServiceRegistrationEndpointType	<code>type="com.vmware.vapi.endpoint"</code>	Specifies the endpoint path for the service.
	<code>protocol= "vapi.json.https.public"</code>	Identifies the protocol that will be used for communication with the endpoint .

For information about the filter parameter of the available predefined service endpoints, see [Filter Parameters for Predefined Service Endpoints](#).

Filter Parameters for Predefined Service Endpoints

Depending on the service endpoint that you want to retrieve, you provide different parameters to the `LookupServiceRegistrationFilter` that you pass to the `List()` function on the Lookup Service. To search for services on a particular vCenter Server instance, set the node ID parameter to the filter.

Table 2-2. Input Data for URL Retrieval for the Lookup Service Registration Filter

Service	Input Data	Value
vCenter Single Sign-On	product namespace	com.vmware.cis
	service type	cs.identity
	protocol	wsTrust
	endpoint type	com.vmware.cis.cs.identity.sso
vSphere Automation Endpoint	product namespace	com.vmware.cis
	service type	cs.vapi
	protocol	vapi.json.https.public
	endpoint type	com.vmware.vapi.endpoint
vCenter Server	product namespace	com.vmware.cis
	service type	vcenterserver
	protocol	vmomi
	endpoint type	com.vmware.vim
vCenter Storage Monitoring Service	product namespace	com.vmware.vim.sms
	service type	sms
	protocol	https
	endpoint type	com.vmware.vim.sms
vCenter Storage Policy-Based Management	product namespace	com.vmware.vim.sms
	service type	sms
	protocol	https
	endpoint type	com.vmware.vim.pbm
vSphere ESX Agent Manager	product namespace	com.vmware.vim.sms
	service type	cs.eam
	protocol	vmomi
	endpoint type	com.vmware.cis.cs.eam.sdk

Connect to the Lookup Service and Retrieve the Service Registration Object

You must connect to the Lookup Service to gain access to its operations. After you connect to the Lookup Service, you must retrieve the service registration object to make registration queries.

Procedure

- 1 Connect to the Lookup Service.
 - a Configure a connection stub for the Lookup Service endpoint, which uses SOAP bindings, by using the HTTPS protocol.
 - b Create a connection object to communicate with the Lookup Service.
- 2 Retrieve the Service Registration Object.
 - a Create a managed object reference to the Service Instance.
 - b Invoke the `RetrieveServiceContent()` method to retrieve the `ServiceContent` data object.
 - c Save the managed object reference to the service registration object.

With the service registration object, you can make registration queries.

Java Example of Connecting to the Lookup Service and Retrieving the Service Registration Object

The example is based on the code in the `LookupServiceHelper.java` sample file.

Note For a complete and up-to-date version of the Java sample code, see the `vsphere-automation-sdk-java` VMware repository at GitHub.

Note The connection code in the example disables certificate and host name checking for the connection for simplicity. For a production deployment, supply appropriate handlers. See the SDK sample file for a more detailed example of connection code.

```

...

String lookupServiceUrl;
LsService lookupService;
LsPortType lsPort;
ManagedObjectReference serviceInstanceRef;
LookupServiceContent lookupServiceContent;
ManagedObjectReference serviceRegistration;

//1 - Configure Lookup Service stub.
HostnameVerifier hostVerifier = new HostnameVerifier (){
    public boolean verify(String urlHostName, SSLSession session){
        return true;
    }
};

```

```

HttpsURLConnection.setDefaultHostnameVerifier(hostVerifier);
SslUtil.trustAllHttpsCertificates();

//2 - Create the Lookup Service stub.
LookupService = new LsService();
LsPort = new LsPortType.getLsPort();
((BindingProvider)lsProvider).getRequestContext().put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY,
LookupServiceUrl);

//4 - Create a predetermined management object.
serviceInstanceRef = new ManagedObjectReference();
serviceInstanceRef.setType("LookupServiceInstance");
serviceInstanceRef.setValue("ServiceInstance");

//5 - Retrieve the ServiceContent object.
lookupServiceContent = lsPort.retrieveServiceContent(serviceInstanceRef);

//6 - Retrieve the service registration
serviceRegistration = lookupServiceContent.getServiceRegistration();

...

```

Python Example of Connecting to the Lookup Service and Retrieving a Service Registration Object

The example is based on the code from the `lookup_service_helper.py` sample file.

Note For a complete and up-to-date version of the sample code, see the `vsphere-automation-sdk-python` VMware repository at GitHub.

```

...

# 1 - Create SOAP client object to communicate with the Lookup Service.
my_ls_stub = Client(url=wSDL_url, location=ls_url)

# 2 - Configure service & port type for client transaction.
my_ls_stub.set_options(service='LsService', port='LsPort')

# 3 - Manufacture a managed object reference.
managed_object_ref = \
    my_ls_stub.factory.create('ns0:ManagedObjectReference')
managed_object_ref._type = 'LookupServiceInstance'
managed_object_ref.value = 'ServiceInstance'

# 4 - Retrieve the ServiceContent object.
ls_service_content = \
    my_ls_stub.service.RetrieveServiceContent(managed_object_ref)

# 5 - Retrieve the service registration object.
service_registration = ls_service_content.serviceRegistration

```

Retrieve Service Endpoints on vCenter Server Instances

You can create a function that obtains the endpoint URLs of a service on all vCenter Server instances in the environment. You can modify that function to obtain the endpoint URL of a service on a particular vCenter Server instance.

Prerequisites

- Establish a connection with the Lookup Service.
- Retrieve a service registration object.

Procedure

- 1 Create a registration filter object, which contains the following parts:
 - A filter criterion for service information
 - A filter criterion for endpoint information

Option	Description
Omit the node ID parameter	Retrieves the endpoint URLs of the service on all vCenter Server instances.
Include the node ID parameter	Retrieves the endpoint URL of the service on a particular vCenter Server instance.

- 2 Retrieve the specified service information by using the `List()` function.

Results

Depending on whether you included the node ID parameter, the `List()` function returns one of the following results:

- A list of endpoint URLs for a service that is hosted on all vCenter Server instances in the environment.
- An endpoint URL of a service that runs on a particular vCenter Server instance.

What to do next

Call the function that you implemented to retrieve service endpoints. You can pass different filter parameters depending on the service endpoints that you need. For more information, see [Filter Parameters for Predefined Service Endpoints](#).

To retrieve a service endpoint on a particular vCenter Server instance, you must retrieve the node ID of that instance and pass it to the function. For information about how to retrieve the ID of a vCenter Server instance, see [Retrieve a vCenter Server ID by Using the Lookup Service](#).

Java Example of Retrieving a vSphere Automation Endpoint on a vCenter Server Instance

This example is based on the in the `LookupServiceHelper.java` sample file.

Note For a complete and up-to-date version of the Java sample code, see the `vsphere-automation-sdk-java` VMware repository at GitHub.

```
...

//1 - Determine management node ID.
String targetNodeId = getMgmtNodeId(targetNodeFqdn);

//2 - List filtered registration info.
List<LookupServiceRegistrationInfo> results =
    lookupSingleServiceUrl("com.vmware.cis",
                           "cs.vapi",
                           "vapi.json.https.public",
                           "com.vmware.vapi.endpoint",
                           targetNodeId);

//3 - Extract endpoint URL from registration info.
LookupServiceRegistrationInfo registrationInfo = results.get(0);
LookupServiceRegistrationEndpoint serviceEndpoint = registrationInfo.getServiceEndpoints().get(0);
String ssoUrl = serviceEndpoint.getUrl();

...
```

Python Example of Retrieving Service Endpoints on vCenter Server Instances

This example provides a common pattern for filtering Lookup Service registration data. This example is based on the code in the `lookup_service_helper.py` sample file.

Note For a complete and up-to-date version of the sample code, see the `vsphere-automation-sdk-python` VMware repository at GitHub.

```
def lookup_service_infos(prod, svc_type, proto, ep_type, node_id='*') :

    # 1 - Create a filter criterion for service info.
    filter_service_type = \
        my_ls_stub.factory.create('ns0:LookupServiceRegistrationServiceType')
    filter_service_type.product = prod
    filter_service_type.type = svc_type

    # 2 - Create a filter criterion for endpoint info.
    filter_endpoint_type = \
        my_ls_stub.factory.create('ns0:LookupServiceRegistrationEndpointType')
    filter_endpoint_type.protocol = proto
    filter_endpoint_type.type = ep_type
```

```

# 3 - Create the registration filter object.
filter_criteria = \
    my_ls_stub.factory.create('ns0:LookupServiceRegistrationFilter')
filter_criteria.serviceType = filter_service_type
filter_criteria.endpointType = filter_endpoint_type
if (node_id != '*') :
    filter_criteria.nodeId = node_id

# 4 - Retrieve specified service info with the List() method.
service_infos = my_ls_stub.service.List(service_registration,
                                       filter_criteria)

return service_infos

```

Retrieve a vCenter Server ID by Using the Lookup Service

You use the node ID of a vCenter Server instance to retrieve the endpoint URL of a service on that vCenter Server instance. You specify the node ID in the service registration filter that you pass to the `List()` function on the Lookup Service.

Managed services are registered with the instance name of the vCenter Server instance where they run. The instance name maps to a unique vCenter Server ID. The instance name of a vCenter Server system is specified during installation and might be an FQDN or an IP address.

Prerequisites

- Establish a connection with the Lookup Service.
- Retrieve a service registration object.

Procedure

- 1 List the vCenter Server instances.
- 2 Find the matching node name of the vCenter Server instance and save the ID.

Results

Use the node ID of the vCenter Server instance to filter subsequent endpoint requests. You can use the node ID in a function that retrieves the endpoint URL of a service on a vCenter Server instance. For information about implementing such a function, see [Retrieve Service Endpoints on vCenter Server Instances](#).

Java Example of Retrieving a vCenter Server ID by Using the Lookup Service

This example is based on the in the `LookupServiceHelper.java` sample file.

Note For a complete and up-to-date version of the Java sample code, see the `vsphere-automation-sdk-java` VMware repository at GitHub.

...

```

getMgmtNodeId(String targetNodeName)
{
    // 1 - List the vCenter Server instances.
    List<LookupServiceRegistrationInfo> serviceInfos =
        lookupServiceUrls("com.vmware.cis",
                        "vcenterserver",
                        "vmomi",
                        "com.vmware.vim");

    // 2 - Find the matching node name and save the ID.
    for (LookupServiceRegistrationInfo serviceInfo : serviceInfos) {
        for (LookupServiceRegistrationAttribute serviceAttr : serviceInfo.getServiceAttributes()) {
            if ("com.vmware.vim.vcenter.instanceName".equals(serviceAttr.getKey())) {
                if (serviceAttr.getValue().equals(targetNodeName)) {
                    return serviceInfo.getNodeId();
                }
            }
        }
    }
}
...

```

Python Example of Retrieving a vCenter Server ID by Using the Lookup Service

This example provides a common pattern for filtering Lookup Service registration data. This example is based on the code in the `lookup_service_helper.py` sample file.

Note For a complete and up-to-date version of the sample code, see the `vsphere-automation-sdk-python` VMware repository at GitHub.

```

def get_mgmt_node_id(node_instance_name) :

    # 1 - List the vCenter Server instances.
    mgmt_node_infos = lookup_service_infos(prod='com.vmware.cis',
                                         svc_type='vcenterserver',
                                         proto='vmomi', ep_type='com.vmware.vim',
                                         node_id='*')

    # 2 - Find the matching node name and save the ID.
    for node in mgmt_node_infos :
        for attribute in node.serviceAttributes :
            if attribute.key == 'com.vmware.vim.vcenter.instanceName' :
                if attribute.value == node_instance_name :
                    return node.nodeId

```

Retrieve a vSphere Automation API Endpoint on a vCenter Server Instance

Through the vSphere Automation API Endpoint, you can access other vSphere Automation services that run on vCenter Server, such as Content Library and Tagging. To use a vSphere Automation service, you must retrieve the vSphere Automation API Endpoint.

Prerequisites

- Establish a connection with the Lookup Service.
- Retrieve a service registration object.
- Determine the node ID of the vCenter Server instance where the vSphere Automation service runs.
- Implement a function that retrieves the endpoint URL of a service on a vCenter Server instance.

Procedure

- 1 Invoke the function for retrieving the endpoint URL of a service on a vCenter Server instance by passing filter strings that are specific to the vSphere Automation API endpoint.
- 2 Save the URL from the resulting single-element list.

Java Example of Retrieving a vSphere Automation Endpoint on a vCenter Server Instance

This example is based on the in the `LookupServiceHelper.java` sample file.

Note For a complete and up-to-date version of the Java sample code, see the `vsphere-automation-sdk-java` VMware repository at GitHub.

```
...

//1 - Determine management node ID.
String targetNodeId = getMgmtNodeId(targetNodeFqdn);

//2 - List filtered registration info.
List<LookupServiceRegistrationInfo> results =
    lookupSingleServiceUrl("com.vmware.cis",
        "cs.vapi",
        "vapi.json.https.public",
        "com.vmware.vapi.endpoint",
        targetNodeId);

//3 - Extract endpoint URL from registration info.
LookupServiceRegistrationInfo registrationInfo = results.get(0);
LookupServiceRegistrationEndpoint serviceEndpoint = registrationInfo.getServiceEndpoints().get(0);
String ssoUrl = serviceEndpoint.getUrl();

...
```

Python Example of Retrieving a vSphere Automation Endpoint on a vCenter Server Instance

This example provides a common pattern for filtering Lookup Service registration data. This example is based on the code in the `lookup_service_helper.py` sample file.

Note For a complete and up-to-date version of the sample code, see the `vsphere-automation-sdk-python` VMware repository at GitHub.

```
service_infos = lookup_service_infos(prod='com.vmware.cis',
                                     svc_type='cs.vapi',
                                     proto='vapi.json.https.public',
                                     ep_type='com.vmware.vapi.endpoint',
                                     node_id=my_mgmt_node_id)
my_vapi_url = service_infos[0].serviceEndpoints[0].url
```

Authentication Mechanisms

3

vCenter Server accepts several authentication methods. The authentication method that you choose depends on whether you choose token authentication, and on the state of the vCenter Server instance.

During normal operation, vCenter Server enables you to authenticate with vCenter Single Sign-On credentials. You have the option to use either token authentication or user name and password authentication. The user name and password must be recognized within the vCenter Single Sign-On domain.

However, during the process of restoring vCenter Server from a backup image, you must use a different authentication protocol. For more information, see [Restoring vCenter Server](#).

This chapter includes the following topics:

- [vCenter Single Sign-On User Name and Password Authentication for vCenter Server](#)
- [vCenter Single Sign-On Token Authentication for vCenter Server](#)

vCenter Single Sign-On User Name and Password Authentication for vCenter Server

You can authenticate with vCenter Server by using a user name and password known to the vCenter Single Sign-On service.

If you prefer to delegate the process of requesting a SAML token for your API client, you can present your vCenter Single Sign-On domain credentials to the vSphere Automation API endpoint and request a session ID. The endpoint process forwards your credentials to the vCenter Single Sign-On service and requests a SAML token on your behalf. In this case, you never deal with the token.

Authenticate with vCenter Single Sign-On Credentials and Create a Session

To establish a session with the vSphere Automation API endpoint in vCenter Server, you create a connection to the endpoint and authenticate with vCenter Single Sign-On credentials to receive a session ID.

Prerequisites

To perform this task, you must have the following items in place.

- The DNS name or IP address of vCenter Server
- A vCenter Single Sign-On domain account that has the requisite permissions for the operation that you intend to invoke

Procedure

- 1 Create a connection context by specifying the vSphere Automation API endpoint URL and the message protocol to be used for the connection.
- 2 Create the request options or stub configuration and set the specific security context to be used.
The security context contains the vCenter Single Sign-On user name and password that are used for authenticating to the vSphere Automation API endpoint.
- 3 Create an interface stub or a REST path that uses the stub configuration.
The interface stub corresponds to the interface containing the method to be invoked.
- 4 Invoke the session `create` method.
The service creates an authenticated session and returns a session identification cookie to the client.
- 5 Add the cookie to your request headers or to a security context for your client stub configuration.
- 6 Remove the basic authentication from your request headers or the security context of your client stub configuration.

Results

Subsequent method calls `authenticate` with the session cookie instead of the user name and password.

What to do next

Use the updated stub configuration with the session ID to create a stub for the interface that you want to use. Method calls on the new stub use the session ID to authenticate.

JavaScript Example of Creating a vSphere Automation API Session with vCenter Single Sign-On Credentials

This example shows the use of JavaScript with the vSphere Automation SDK for REST to send a session creation request to vCenter Server by using vCenter Single Sign-On credentials.

The example retrieves a session cookie for future authentication. The JavaScript code depends on the `Node.js` package, which allows it to run standalone.

This example depends on the following global variables.

- `my_host`
- `my_sso_username`
- `my_sso_password`

■ *my_http_options*

For clarity, this example specifies a complete set of HTTP options for the NodeJS request, rather than retaining and modifying an existing object.

```
var https = require('https');
var httpPort = 443;
var httpPath = '/rest/com/vmware/cis/session';
var httpMethod = 'POST';

// Prepare the HTTP request.
my_http_options = {
  host: my_host,
  port: httpPort,
  path: httpPath,
  method: httpMethod,
  rejectUnauthorized: false,
  requestCert: true,
  agent: false,
  auth: my_sso_username + ":" + my_sso_password
};

// Define the callbacks.
function callback(res) {
  console.log("STATUS: " + res.statusCode);
  res.on('error', function(err) { console.log("ERROR in SSO authentication: ", err) });
  res.on('data', function(chunk) {});
  res.on('end', function() {
    if (res.statusCode == 200) {
      // Save session ID authentication.
      var cookieValue = res.headers['set-cookie'];
      my_http_options.headers = {'Cookie': cookieValue};
      // Remove username-password authentication.
      my_http_options.auth = {};
    }
    console.log("Session ID:\n" + res.headers['set-cookie']);
  });
}

// Issue the session creation request.
https.request(my_http_options, callback).end();
```

Java Example of Creating a vSphere Automation API Session with User Credentials

This example is based on the code in the `VapiAuthenticationHelper.java` sample.

Note For a complete and up-to-date version of the Java sample code, see the `vsphere-automation-sdk-java` VMware repository at GitHub.

```
...

this.stubFactory = createApiStubFactory(server, httpConfig);
```

```

// Create a security context for username/password authentication
SecurityContext securityContext =
    SecurityContextFactory.createUserPassSecurityContext(
        username, password.toCharArray());

// Create a stub configuration with username/password security context
StubConfiguration stubConfig = new StubConfiguration(securityContext);

// Create a session stub using the stub configuration.
Session session =
    this.stubFactory.createStub(Session.class, stubConfig);

// Login and create a session
char[] sessionId = session.create();

// Initialize a session security context from the generated session id
SessionSecurityContext sessionSecurityContext =
    new SessionSecurityContext(sessionId);

// Update the stub configuration to use the session id
stubConfig.setSecurityContext(sessionSecurityContext);

/*
 * Create a stub for the session service using the authenticated
 * session
 */
this.sessionSvc =
    this.stubFactory.createStub(Session.class, stubConfig);

VM vmService = this.stubFactory.createStub(VM.class, stubConfig);

```

Python Example of Creating a vSphere Automation API Session with SSO Credentials

This example is based on code in the `vapiconnect.py` sample file.

This example uses the following global variables.

- `my_vapi_hostname`
- `my_sso_username`
- `my_sso_password`
- `my_stub_config`

Note For a complete and up-to-date version of the sample code, see the `vsphere-automation-sdk-python` VMware repository at GitHub.

```

import requests
from com.vmware.cis_client import Session
from vmware.vapi.lib.connect import get_requests_connector
from vmware.vapi.security.session import create_session_security_context
from vmware.vapi.security.user_password import create_user_password_security_context
from vmware.vapi.stdlib.client.factories import StubConfigurationFactory

```

```

# Create a session object in the client.
session = requests.Session()

# For development environment only, suppress server certificate checking.
session.verify = False

# Create a connection for the session.
vapi_url = 'https://' + my_vapi_hostname + '/api'
connector = get_requests_connector(session=session, url=vapi_url)

# Add username/password security context to the connector.
basic_context = create_user_password_security_context(my_sso_username,
                                                    my_sso_password)
connector.set_security_context(basic_context)

# Create a stub configuration by using the username–password security context.
my_stub_config = StubConfigurationFactory.new_std_configuration(connector)

# Create a Session stub with username–password security context.
session_stub = Session(my_stub_config)

# Use the create operation to create an authenticated session.
session_id = session_stub.create()

# Create a session ID security context.
session_id_context = create_session_security_context(session_id)

# Update the stub configuration with the session ID security context.
my_stub_config.connector.set_security_context(session_id_context)

```

vCenter Single Sign-On Token Authentication for vCenter Server

You can authenticate with vCenter Server by using a SAML token from the vCenter Single Sign-On service. The token can be either a bearer token or a holder-of-key token.

To use SAML token authentication, you issue a request to the vCenter Single Sign-On service, specifying the token type (bearer or holder-of-key), expected token lifetime, renewability, and other parameters. You also supply a user name and password combination that is valid in the vCenter Single Sign-On domain. These credentials must have an associated role with sufficient privilege for the operations that you intend to invoke with the Management API.

If the vCenter Single Sign-On service accepts your credentials, it responds with an XML message. The message contains a SAML assertion that your client can extract and present as an Authorization header in an HTTP request to the vSphere Automation API endpoint.

Retrieve a SAML Token

The vCenter Single Sign-On service provides authentication mechanisms for securing the operations that your client application performs in the virtual environment. Client applications use SAML security tokens for authentication.

Client applications use the vCenter Single Sign-On service to retrieve SAML tokens. For more information about how to acquire a SAML security token, see the *vCenter Single Sign-On Programming Guide* documentation.

Prerequisites

Verify that you have the vCenter Single Sign-On URL. You can use the Lookup Service on vCenter Server to obtain the endpoint URL. For information about retrieving service endpoints, see [Chapter 2 Retrieving Service Endpoints](#).

Procedure

- 1 Create a connection object to communicate with the vCenter Single Sign-On service.
Pass the vCenter Single Sign-On endpoint URL, which you can get from the Lookup Service.
- 2 Issue a security token request by sending valid user credentials to the vCenter Single Sign-On service on vCenter Server.

Results

The vCenter Single Sign-On service returns a SAML token.

What to do next

You can present the SAML token to the vSphere Automation API endpoint or other endpoints, such as the vSphere Web Services endpoint. The endpoint returns a session ID and establishes a persistent session with that endpoint. Each endpoint that you connect to uses your SAML token to create its own session.

JavaScript Example of Retrieving a SAML Token

This example shows the use of JavaScript with the vSphere Automation SDK for REST to send a SAML token request to the vCenter Single Sign-On endpoint.

The example assumes that you have previously saved certain connection information in global variables. The JavaScript depends on the Node .js package, which allows it to run standalone.

This example depends on the following global variables.

- *my_sso_username*
- *my_sso_password*
- *my_host*

```
var https = require('https');
var fs = require('fs');

var httpPort = 443;
```

```

var tokenFilename = './token.xml';

// Create connection settings object.
my_http_options = {
    host: my_host,
    port: httpPort,
    path: '/sts/STSService/vsphere.local',
    method: 'POST',
    rejectUnauthorized: false,
    requestCert: true,
    agent: false,
    headers: {
        'Content-type': 'text/xml; charset="UTF-8"',
        'Content-length': 0,
        'User-Agent': 'VMware/jsSample',
        'Connection': 'keep-alive',
        'SOAPAction': "http://docs.oasis-open.org/ws-sx/ws-trust/200512/RST/Issue"
    }
};

// Set parameters for token lifetime.
var now = new Date();
var created = now.toISOString();
now.setHours(now.getHours() + 1);
var expires = now.toISOString();

// Build SOAP token request.
var requestXml = '<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"> \
<SOAP-ENV:Header xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"> \
  <ns5:Security \
    xmlns="http://docs.oasis-open.org/ws-sx/ws-trust/200512" \
    xmlns:ns2="http://www.w3.org/2005/08/addressing" \
    xmlns:ns3= \
      "http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd" \
    xmlns:ns4="http://www.rsa.com/namespace/2009/12/std-ext/WS-Trust1.4/advice" \
    xmlns:ns5= \
      "http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd"> \
<ns3:Timestamp> \
  <ns3:Created>' + created + '</ns3:Created> \
  <ns3:Expires>' + expires + '</ns3:Expires> \
</ns3:Timestamp> \
<ns5:UsernameToken> \
  <ns5:Username>' + my_sso_username + '</ns5:Username> \
  <ns5:Password>' + my_sso_password + '</ns5:Password> \
</ns5:UsernameToken> \
</ns5:Security> \
</SOAP-ENV:Header> \
<SOAP-ENV:Body xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"> \
  <RequestSecurityToken \
    xmlns="http://docs.oasis-open.org/ws-sx/ws-trust/200512" \
    xmlns:ns2="http://www.w3.org/2005/08/addressing" \
    xmlns:ns3= \
      "http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd" \
    xmlns:ns4="http://www.rsa.com/namespace/2009/12/std-ext/WS-Trust1.4/advice" \

```

```

xmlns:ns5= \
  "http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd"> \
<TokenType>urn:oasis:names:tc:SAML:2.0:assertion</TokenType> \
<RequestType>http://docs.oasis-open.org/ws-sx/ws-trust/200512/Issue</RequestType> \
<Lifetime> \
  <ns3:Created>' + created + '</ns3:Created> \
  <ns3:Expires>' + expires + '</ns3:Expires> \
</Lifetime> \
<Renewing Allow="true" OK="false" /> \
<Delegatable>>true</Delegatable> \
<KeyType>http://docs.oasis-open.org/ws-sx/ws-trust/200512/Bearer</KeyType> \
<SignatureAlgorithm>http://www.w3.org/2001/04/xmldsig-more#rsa-sha256</SignatureAlgorithm> \
</RequestSecurityToken> \
</SOAP-ENV:Body> \
</SOAP-ENV:Envelope>'

// Define callback to extract SAML assertion.
function extractToken(xmlResponse) {
  var token;
  token=xmlResponse.toString().match(/<\s\saml2:Assertion[\s\S]*\s\saml2:Assertion\s\s/>/m).toString();
  return token;
}

// Define request callback functions.
var callback = function(res) {
  str = '';
  res.on('error', function(err) {console.log("ERROR in SSO authentication", err)});
  res.on('data', function(chunk) {str += chunk});
  res.on('end', function() {
    console.log("SSO: Authenticated successfully");
    my_saml_token = extractToken(str);
    fs.writeFile(tokenFilename, my_saml_token, function(err){
      if (err) {
        console.log("Couldn't save SAML token to " + tokenFilename)
      } else {
        console.log("Saved SAML token to " + tokenFilename)
      }
    });
  });
};

// Issue security token request.
my_http_options.headers['Content-length'] = requestXml.length;
https.request(my_http_options, callback).end(requestXml);

```

Create a vSphere Automation Session with a SAML Token

To establish a vSphere Automation session, you create a connection to the vSphere Automation API endpoint and then you authenticate with a SAML token to create a session for the connection.

Prerequisites

- Retrieve the vSphere Automation endpoint URL from the Lookup Service.

- Obtain a SAML token from the vCenter Single Sign-On service.

Procedure

- 1 Create a connection by specifying the vSphere Automation API endpoint URL and the message protocol to be used for the connection.

Note To transmit your requests securely, use **https** for the vSphere Automation API endpoint URL.

- 2 Create the request options or stub configuration and set the security context to be used.
The security context object contains the SAML token retrieved from the vCenter Single Sign-On service. Optionally, the security context might contain the private key of the client application.
- 3 Create an interface stub or a REST path that uses the stub configuration instance.
The interface stub corresponds to the interface containing the method to be invoked.
- 4 Invoke the session create method.
The service creates an authenticated session and returns a session identification cookie to the client.
- 5 Create a security context instance and add the session ID to it.
- 6 Update the stub configuration instance with the session security context.

What to do next

Use the updated stub configuration with the session ID to create a stub for the interface that you want to use. Method calls on the new stub use the session ID to authenticate.

JavaScript Example of Creating a vSphere Automation API Session with a SAML Token

This example shows the use of JavaScript with the vSphere Automation SDK for REST to apply a SAML token to the vSphere Automation API endpoint and exchange it for a vSphere Automation API session ID.

The example assumes that you have previously saved certain connection information in global variables. The JavaScript depends on the Node .js package, which allows it to run standalone.

This example depends on the following global variables.

- *my_vapi_host*
- *my_vapi_port*
- *my_saml_token*
- *my_http_options*

```
// Import required libraries.
var sso = require('./sso');
var https = require('https');
var gzip = require('gzip-js');

// Configure HTTP request.
var sessionPath = '/rest/com/vmware/cis/session';
```

```

var httpMethod = 'POST';

// Base64 encode the token value for the security context.
var b64Token = new Buffer(gzip.zip(my_saml_token)).toString('base64');

// Build the Authorization header value.
var start = 0;
var bufSize = 3 * 1024;
var prefix = 'SIGN ';
var authArray = [];
while (start < b64Token.length) {
  var end = start + bufSize;
  authArray.push(prefix + 'token="' + b64Token.slice(start, end) + '"');
  start = end;
  prefix = '';
}

// Prepare the HTTP request.
my_http_options = {
  host: my_vapi_host,
  port: my_vapi_port,
  path: sessionPath,
  method: httpMethod,
  rejectUnauthorized: false,
  requestCert: true,
  agent: false,
  headers: {
    'Authorization': authArray
  }
};

// Define the callbacks.
function callback(res) {
  res.on('error', function(err) { console.log('Login error: ', err) });
  res.on('data', function(chunk) {});
  res.on('end', function() {
    var cookieValue = res.headers['set-cookie'];
    my_http_options[headers] = {'Cookie': cookieValue,
                               'Content-Type': 'application/json'} // Save the session ID.
  }
}

// Issue the login request.
https.request(my_http_options, callback).end();

```

Authorization Model for Administration of vCenter Server

4

There are three types of authorization levels in vCenter Server.

Table 4-1. Authorization Levels

Authorization Level	Description
operator	A user has read access to configuration settings.
administrator	A user has read and write access to configuration settings, but cannot manage user accounts.
super administrator	A user has all the capabilities of the other roles, and has the additional capabilities of creating local user accounts and accessing the local Bash shell.

This model applies to the API and all other interfaces to vCenter Server except when you use SSH and log in by using a local account.

This chapter includes the following topics:

- [Authorization Model Mapping to the vCenter Single Sign-On Domain](#)
- [Using the Operator Role](#)
- [Using the Admin Role](#)
- [Using the SuperAdmin Role](#)

Authorization Model Mapping to the vCenter Single Sign-On Domain

The three-level authorization model of vCenter Server maps to local roles and to vCenter Single Sign-On groups, depending on how the user authenticated. This model allows consistent security control regardless of operational context.

The authorization levels map to group and role.

Table 4-2. Authorization Mapping

Authorization Level	vCenter Single Sign-On Group	vCenter Server Local Role
operator	SystemConfiguration.Administrators	operator
administrator	SystemConfiguration.Administrators	admin
superAdministrator	SystemConfiguration.BashShellAdministrators	superAdmin

When a super administrator adds user accounts, the options available include a choice of the role to assign to the new user.

Using the Operator Role

The **operator** role is the most restricted of the authorization levels available to users who work with vCenter Server.

Operators are allowed to view information about vCenter Server. They are not allowed to alter its configuration. The **operator** role is suited for monitoring and reporting functions. For example, the **operator** role provides access to the following methods.

- `resources.system.health.get`
- `resources.storage.stats.list`
- `services.status.get`

Using the Admin Role

The **administrator** role provides an intermediate authorization level for users who manage vCenter Server.

An **administrator** role is required for users who alter the vCenter Server configuration, exercise control functions, or other operations that can affect regular users.

For example, the **administrator** role provides access the following methods.

- `networking.ip4v.renew`
- `networking.firewall.addr.inbound.add`
- `services.control`
- `shutdown.reboot`

Using the SuperAdmin Role

The **superAdmin** role is the most expansive authorization level for users who manage vCenter Server.

The **superAdmin** role allows unrestricted access to vCenter Server. This role is required for adding or altering user accounts and for using the Bash shell.

Installing and Upgrading vCenter Server

5

You can use the vCenter Server API to perform operations related to stage 2 of the installation and upgrade processes. You can also perform historical data transfer operations.

This chapter includes the following topics:

- [Install Stage 2](#)
- [Upgrade Stage 2](#)
- [Historical Data Transfer](#)

Install Stage 2

The vCenter Server API provides methods for performing stage 2 deployment operations on a newly installed vCenter Server instance.

The vCenter Server instance is deployed in two stages. With stage 1 of the deployment process, you deploy the OVA file, which is included in the installer. With stage 2 of the deployment process, you set up and start the services of the newly deployed vCenter Server instance.

To complete stage 1 of the deployment process, you can use the GUI installer or perform a CLI deployment. For details, see *vCenter Server Installation and Setup*. Alternatively, you can perform a deployment by using the VMware OVF Tool. See *OVF Tool User's Guide*.

Setting Up a Newly Installed vCenter Server Instance

You can use the API to set up a newly deployed vCenter Server instance.

After stage 1 of the deployment process completes successfully, the vCenter Server instance enters in an `INITIALIZED` state. If the instance is not initialized, you cannot run stage 2 of the deployment process. You can get the state of the vCenter Server instance by using the `vcenter deployment` service. The vCenter Server instance can enter six states during the deployment process.

Figure 5-1. Install Stage 2 State Diagram

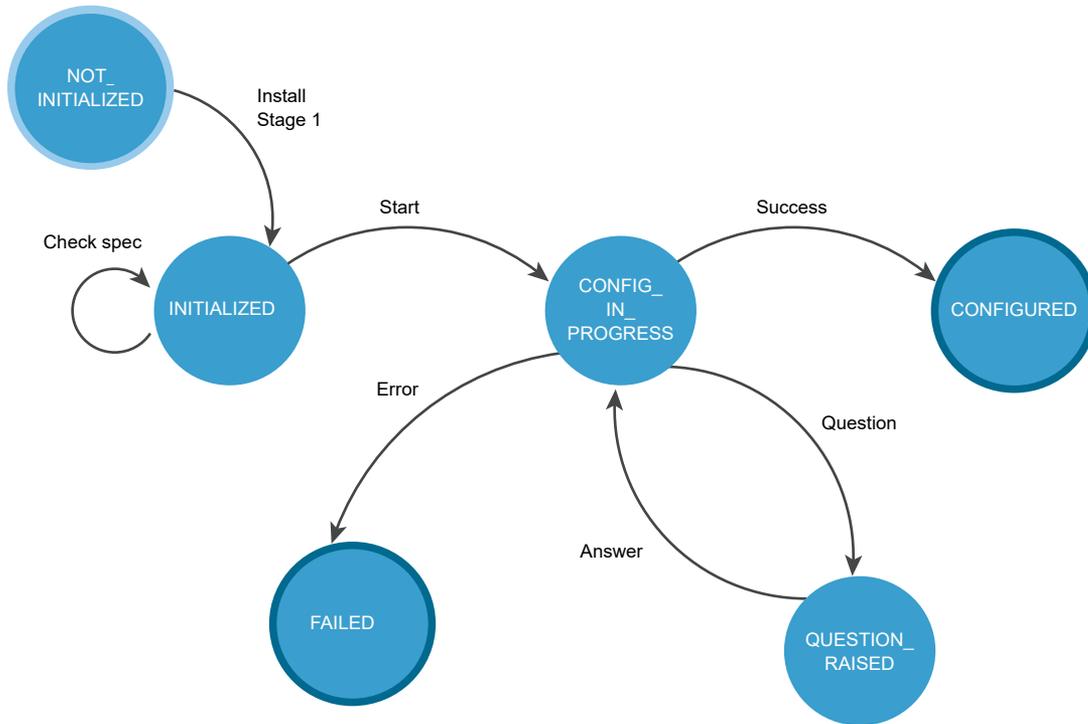


Table 5-1. vCenter Server Instance States During Install Stage 2

State	Description
NOT_INITIALIZED	The install stage 1 phase is in progress, not started, or failed.
INITIALIZED	The vCenter Server instance is deployed and ready for setup.
CONFIG_IN_PROGRESS	The setup process is in progress.
QUESTION_RAISED	You must answer the question to continue the setup process. The vCenter Server instance stays in the QUESTION_RAISED state until it receives the correct answer.
FAILED	Errors occurred during the setup process. You can check the errors, warnings, and info data structures.
CONFIGURED	The vCenter Server instance is installed and configured successfully.

FAILED and CONFIGURED are final states.

[Table 5-2. User Operations](#) lists operations that you can perform to set up your newly deployed vCenter Server instance.

Table 5-2. User Operations

Operation	Description
Get deployment information	You can retrieve information about the current deployment status. This operation is useful both before initiating stage 2 of the deployment and for monitoring the progress of the setup process.
Validate the configuration document	You can optionally verify whether your install spec is valid before starting the setup process.
Configure the vCenter Server instance	You can initiate the setup process by providing an install spec that defines the values for the settings that you want to configure.
Get question	You can retrieve a question raised during the setup process.
Answer question	You can provide an answer to the question raised during the setup process. The available answer values are YES, NO, OK, CANCEL, ABORT, RETRY, and IGNORE. The possible answer values depend on the type of the question. Note Each question has a default answer value. If you set questions to receive automatic answers in the install spec and a question is raised during the setup process, the default answer value is automatically provided as the answer to the question.

For information about the HTTP requests that you can use to perform the user operations, see [HTTP Requests for Install Stage 2](#).

HTTP Requests for Install Stage 2

You can use HTTP requests to set up a newly deployed vCenter Server instance.

HTTP Requests

After stage 1 of the deployment process completes successfully, you can perform setup by sending HTTP requests.

Note When you send the requests, you must authenticate with vCenter Server **root** credentials.

The following HTTP requests show the syntax that you can use to perform the available user operations.

- Get deployment information

```
GET https://<server>:5480/rest/vcenter/deployment
```

- Validate the install spec

```
POST https://<server>:5480/rest/vcenter/deployment/install?action=check
```

- Configure the vCenter Server instance

```
POST https://<server>:5480/rest/vcenter/deployment/install?action=start
```

- Get question

```
GET https://<server>:5480/rest/vcenter/deployment/question
```

- Answer question

```
POST https://<server>:5480/rest/vcenter/deployment/question?action=answer
```

For information about the content and syntax of the HTTP request body, see the *API reference* documentation.

HTTP Status Codes and Errors

[Table 5-3. HTTP Status Codes and Errors](#) lists the status codes that you can receive when you send HTTP requests.

Table 5-3. HTTP Status Codes and Errors

HTTP Status Code	Description	Operations that Return the Status Code
200	The operation is successful.	All operations. You can check the returned data in the results data structure.
400	You cannot perform the operation because the vCenter Server instance is in the current state. For information about the states of the vCenter Server instance, see Setting Up a Newly Installed vCenter Server Instance .	<ul style="list-style-type: none"> ■ Validate the install spec ■ Start the setup ■ Get the install spec ■ Get the raised question ■ Answer the question ■ Get the state of the vCenter Server instance
401	You use an invalid user name or password, or authentication has failed.	<ul style="list-style-type: none"> ■ Validate the install spec ■ Start the setup ■ Get the install spec ■ Get the raised question ■ Answer the question ■ Get the state of the vCenter Server instance
404	The state of the vCenter Server instance cannot be determined .	Get the vCenter Server instance state.
500	There is a <code>vapi_std</code> error. For information about the types of <code>vapi_std</code> errors, see <code>vapi_std.errors</code> in the <i>API reference</i> documentation.	Get the raised question.

If errors occur during the setup process, you can check the `results` data structure, the API log file, and download the vCenter Server instance support bundle from `https://<server>:443/appliance/support-bundle`.

Use HTTP Requests to Set Up a Newly Deployed vCenter Server Instance

You can send HTTP requests to complete stage 2 of the deployment process of a newly deployed vCenter Server instance.

You set up a newly deployed vCenter Server instance by providing configuration settings in the body of the HTTP request.

Prerequisites

- Verify that the newly deployed vCenter Server instance is reachable.
- Verify that you have the correct credentials for sending HTTP requests.

Procedure

- 1 Check whether the vCenter Server instance state is set to INITIALIZED.

```
GET https://<server>:5480/rest/vcenter/deployment
```

If the vCenter Server instance is in the correct state, you receive a message body that contains the following line and you can continue with the setup process.

```
...
  "state": "INITIALIZED",
  ...
```

- 2 (Optional) Validate the configuration settings that you provide in the body of the HTTP request.

```
POST https://<server>:5480/rest/vcenter/deployment/install?action=check
```

The following example shows syntax that you can use in the body of the HTTP request.

```
{
  "spec": {
    "vcsa_embedded": {
      "ceip_enabled": true,
      "standalone": {
        "sso_domain_name": "vsphere.local",
        "sso_admin_password": "<your_password>"
      }
    }
  },
  "auto_answer": true
}
```

If the input is valid, you receive the following response.

```
{
  "status": "SUCCESS"
}
```

- 3 Initiate the setup process by providing valid input in the body of the HTTP request.

```
POST https://<server>:5480/rest/vcenter/deployment/install?action=start
```

4 Monitor the progress of the setup process.

```
GET https://<server>:5480/rest/vcenter/deployment
```

The following example shows part of the response body when the setup process is ongoing.

```
{
  "progress": {
    "completed": 2,
    "message": {
      "id": "install.ciscommon.component.starting",
      "args": [
        "VMware Authentication Framework"
      ],
      "default_message": "Starting VMware Authentication Framework..."
    },
    "total": 3
  },
  "status": "RUNNING",
  "state": "CONFIG_IN_PROGRESS",
  "operation": "INSTALL",
  ...
}
```

The following example shows part of the response body when the setup process has completed successfully.

```
{
  "subtask_order": [
    "rpminstall",
    "validate",
    "firstboot"
  ],
  "cancelable": false,
  "progress": {
    "completed": 3,
    "total": 3,
    "message": {
      "default_message": "Task has completed successfully.",
      "id": "com.vmware.vcenter.deploy.task.complete.success",
      "args": []
    }
  },
  "status": "SUCCEEDED",
  "description": {
    "default_message": "Install vCenter Server.",
    "id": "com.vmware.vcenter.deploy.task.description.op.install",
    "args": []
  },
  "state": "CONFIGURED",
  ...
}
```

Results

You successfully configured the newly deployed vCenter Server instance.

Workflows for Install Stage 2

You can use the `vcenter deployment` API to run the install stage 2 process of your vCenter Server instance.

[Figure 5-2. Install Workflow](#) and [Figure 5-3. Install Stage 2 Workflow](#) show example install workflows.

During stage 1, the vCenter Server instance is in a `NOT_INITIALIZED` state. After a successful deployment, the vCenter Server instance enters in an `INITIALIZED` state. If there are errors during stage 1, the vCenter Server instance stays in a `NOT_INITIALIZED` state and you must redeploy it.

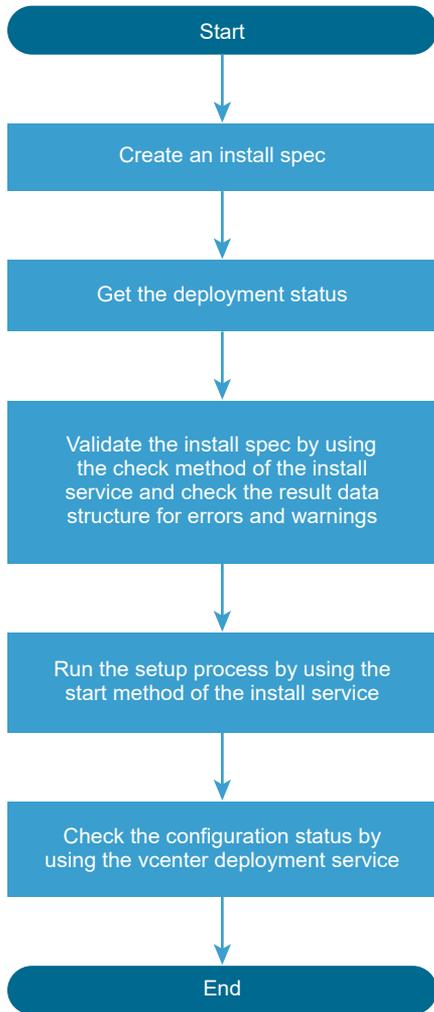
You can check the state of the vCenter Server instance before, during, and after the setup process. You can run the install stage 2 process if the vCenter Server instance is initialized. You can check the setup configuration before you initiate stage 2 by running pre-checks. If errors or warnings appear during the validation of the install specification, you must remove the causes and correct the specification.

During the setup process, the regular vCenter Server instance state is `CONFIG_IN_PROGRESS`. The vCenter Server instance can also enter in a `FAILED` or `QUESTION_RAISED` state. If a question appears during the setup, the vCenter Server instance enters in a `QUESTION_RAISED` state and stays in it until you provide an answer. You can set questions to receive automatic answers in the install spec and if a question is raised during the setup process, the default answer value is automatically provided as the answer to the question.

If errors occur during the setup process, the vCenter Server instance enters in a `FAILED` state and you must restart the setup after the causes are removed. If the setup is successful, the vCenter Server instance enters in a `CONFIGURED` state.

For information about the states of the vCenter Server instance and available operations, see [Setting Up a Newly Installed vCenter Server Instance](#).

Figure 5-3. Install Stage 2 Workflow



You can run the setup pre-checks and the install stage 2 process by creating and passing an `InstallSpec`. In `InstallSpec`, you define the setup configuration. See [Figure 5-3. Install Stage 2 Workflow](#). You can run the setup in silent mode by setting the `InstallSpec.auto_answer` to `true`. The default value of `InstallSpec.auto_answer` is `false` and the setup is in interactive mode, in which you must provide answers to the raised questions.

For information about the classes, variables, and default values, see the *API reference* documentation.

Upgrade Stage 2

You can upgrade your vCenter Server instance by using the API, CLI, or GUI.

For information about how to upgrade the vCenter Server instance by using CLI and GUI, see the *vCenter Server Upgrade* documentation.

Upgrading a vCenter Server Instance

You can use the API during stage 2 of the vCenter Server instance upgrade.

By using the API, you can upgrade your vCenter Server instance. For information about the upgrade process, its stages, supported configurations, upgrade paths, prerequisites for upgrading, and the sequence for upgrading a vSphere environment, see the *vCenter Server Upgrade* documentation.

After you deploy the vCenter Server instance on stage 1 by using the GUI or CLI, the instance enters in an INITIALIZED state. If the vCenter Server instance is not initialized, you cannot run stage 2 of the upgrade process. You can get the state of the vCenter Server instance by using the `vcenter deployment` service. There are six states during the upgrade process.

Figure 5-4. Upgrade Stage 2 State Diagram

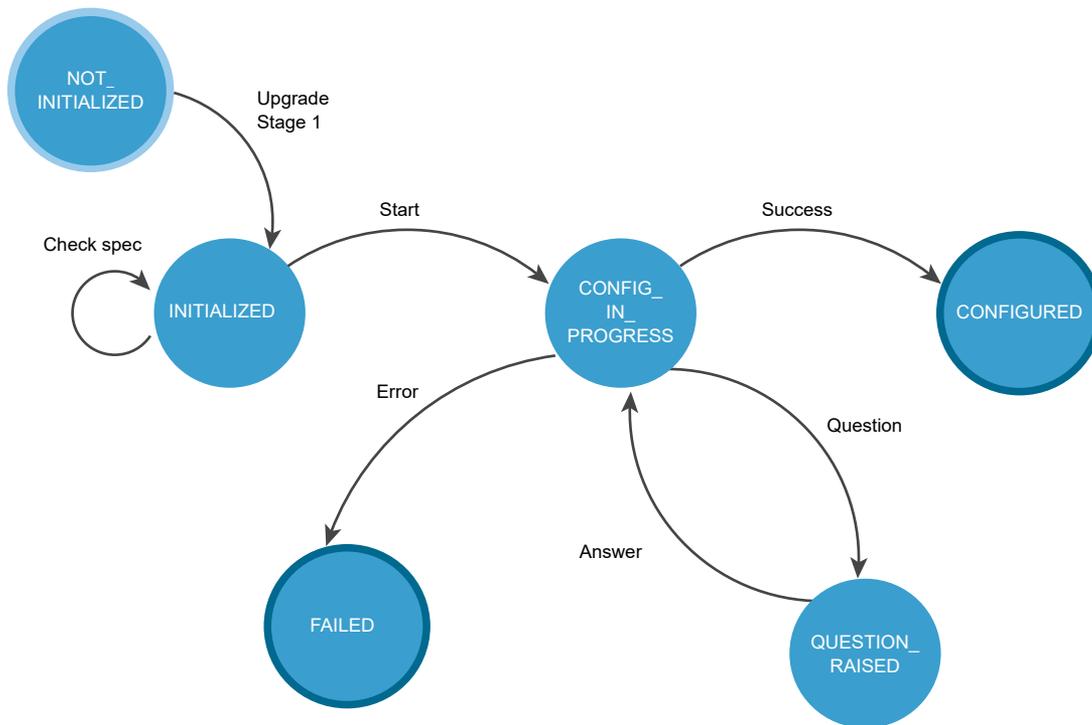


Table 5-4. vCenter Server Instance States During Upgrade Stage 2

State	Description
NOT_INITIALIZED	The upgrade stage 1 phase is in progress, not started, or failed.
INITIALIZED	The vCenter Server instance is deployed and ready for upgrading.
CONFIG_IN_PROGRESS	The upgrade process is in progress.
QUESTION_RAISED	You must answer the question to continue the upgrade process. The vCenter Server instance stays in the QUESTION_RAISED state until it receives the correct answer.

Table 5-4. vCenter Server Instance States During Upgrade Stage 2 (continued)

State	Description
FAILED	Errors appeared during the upgrade process. You can check the errors, warnings, and info data structures.
CONFIGURED	The vCenter Server instance is upgraded or configured successfully.

FAILED and CONFIGURED are final states.

You can roll back a vCenter Server instance upgrade by using the GUI. For information about how to roll back a vCenter Server instance, see the *vCenter Server Upgrade* documentation.

After the upgrade you can check the vCenter Server instance type, domain registration, services, their state and health status by using the API. For information about how to verify whether the upgrade of your vCenter Server instance is successful, see the *vCenter Server Upgrade* documentation.

[Table 5-5. User Operations](#) shows operations that you can perform to upgrade your vCenter Server instance.

Table 5-5. User Operations

Operation	Description
Operations for upgrading	
Get the state of the vCenter Server instance	You can get the state of the vCenter Server instance before, during and after the upgrade process. For information about the HTTP requests for diagnostic and deferring the transfer of historical data, see HTTP Requests for Upgrade Stage 2 .
Check	You can validate the upgrade spec before you run the upgrade process. If the vCenter Server instance is in the INITIALIZED state, you can run the validation. The operation runs upgrade pre-checks. You can check the errors, warnings, and status data structures before you run the upgrade process. For information about the HTTP requests for upgrading, see HTTP Requests for Upgrade Stage 2 .
Start	If the vCenter Server instance is in an INITIALIZED state, you can run the upgrade process. If errors appear during the upgrade, you can download the vCenter Server support bundle. For information about the HTTP requests for upgrading, see HTTP Requests for Upgrade Stage 2 .
Get	If the vCenter Server instance is in a CONFIGURED state, you can get the spec that is used for upgrading. For information about the HTTP requests for upgrading, see HTTP Requests for Upgrade Stage 2 .
Operations for getting and answering a question	

Table 5-5. User Operations (continued)

Operation	Description
Get	<p>You can get the raised question. If you set the <code>Upgrade.auto_answer</code> to <code>true</code>, the upgrade process will be in a <code>silent</code> mode and the vCenter Serverinstance does not generate questions. It uses default answers and you should not provide an answer.</p> <p>For information about the HTTP requests for getting and answering a question, see HTTP Requests for Upgrade Stage 2.</p>
Answer	<p>You can provide an answer to the raised question. The available answers for the upgrading are <code>OK</code>, <code>CANCEL</code>, <code>YES,NO</code>, <code>ABORT</code>, <code>RETRY</code>, and <code>IGNORE</code>. The answer depends on the type of the question. If you set the <code>Upgrade.auto_answer</code> to <code>true</code>, the upgrade process will be in a <code>silent</code> mode and the vCenter Serverinstance does not generate questions. It uses default answers and you should not provide an answer.</p> <p>For information about the HTTP requests for getting and answering a question, see HTTP Requests for Upgrade Stage 2.</p>

For information about the available operations in the API, see the `vcenter deployment`, `vcenter deployment upgrade`, `vcenter services`, and `vcenter system-config deployment type services` in the *API reference* documentation.

You can upgrade your vCenter Server instance by using HTTP requests. For information about the HTTP requests, see [HTTP Requests for Upgrade Stage 2](#).

HTTP Requests for Upgrade Stage 2

You can use HTTP requests or the API to upgrade your vCenter Server instance.

HTTP Requests

HTTP requests for upgrading

- Get the state of the vCenter Server instance

```
GET https://<IP_address_upgraded_or_target_instance>:5480/rest/vcenter/deployment
```

- Validate the upgrade spec

```
POST https://<IP_address_target_instance>:5480/rest/vcenter/deployment/upgrade?action=check
```

- Run the upgrading

```
POST https://<IP_address_target_instance>:5480/rest/vcenter/deployment/upgrade?action=start
```

- Get the upgrade spec used for upgrading

```
GET https:// https://<IP_address_upgraded_instance>:5480/rest/vcenter/deployment/upgrade
```

HTTP requests for getting and answering a question

- Get the raised question

```
GET https://<IP_address_upgraded_or_target_instance>:5480/rest/vcenter/deployment/question
```

- Answer to the question

```
POST https://<IP_address_upgraded_or_target_instance>:5480/rest/vcenter/deployment/question?
action=answer
```

For information about the content and syntax of the HTTP request body, see the *API reference* documentation.

Note When you send the requests, you must authenticate with vCenter Server **root** credentials.

HTTP Status Codes and Errors

Table 5-6. HTTP Status Codes and Errors

HTTP Status Code	Description	Operations That Return the Status Code
200	The operation is successful.	All operations. You can check the returned data in the results data structure.
400	You cannot perform the operation because the vCenter Server instance is in the current state. For information about the states of the vCenter Server instance, see the Upgrading a vCenter Server Instance topic.	<ul style="list-style-type: none"> ■ Get the state of the vCenter Server instance ■ Validate the upgrade spec ■ Run the upgrade ■ Get the upgrade spec ■ Get the raised question ■ Answer the question
401	You use invalid user name or password, or authentication is failed.	<ul style="list-style-type: none"> ■ Get the state of the vCenter Server instance ■ Validate the upgrade spec ■ Run the upgrade ■ Get the upgrade spec ■ Get the raised question ■ Answer the question
404	The state of the vCenter Server instance cannot be determined	Get the state of the vCenter Server instance.
500	There is a <code>vapi_std</code> error. For information about the types of <code>vapi_std</code> errors, see the <code>vapi_std.errors</code> API in the <i>API reference</i> documentation.	Get the raised question.

If errors appear during the upgrade process, you can check the `results` data structure, the API log file, and download the vCenter Server support bundle from `https://<server>:443/appliance/support-bundle`.

Workflows for Upgrade Stage 2

You can use the `vcenter deployment` API to run the upgrade stage 2 process of your vCenter Server instance.

[Figure 5-5. Upgrade Workflow](#) and [Figure 5-6. Upgrade Stage 2 Workflow](#) show example upgrade workflows.

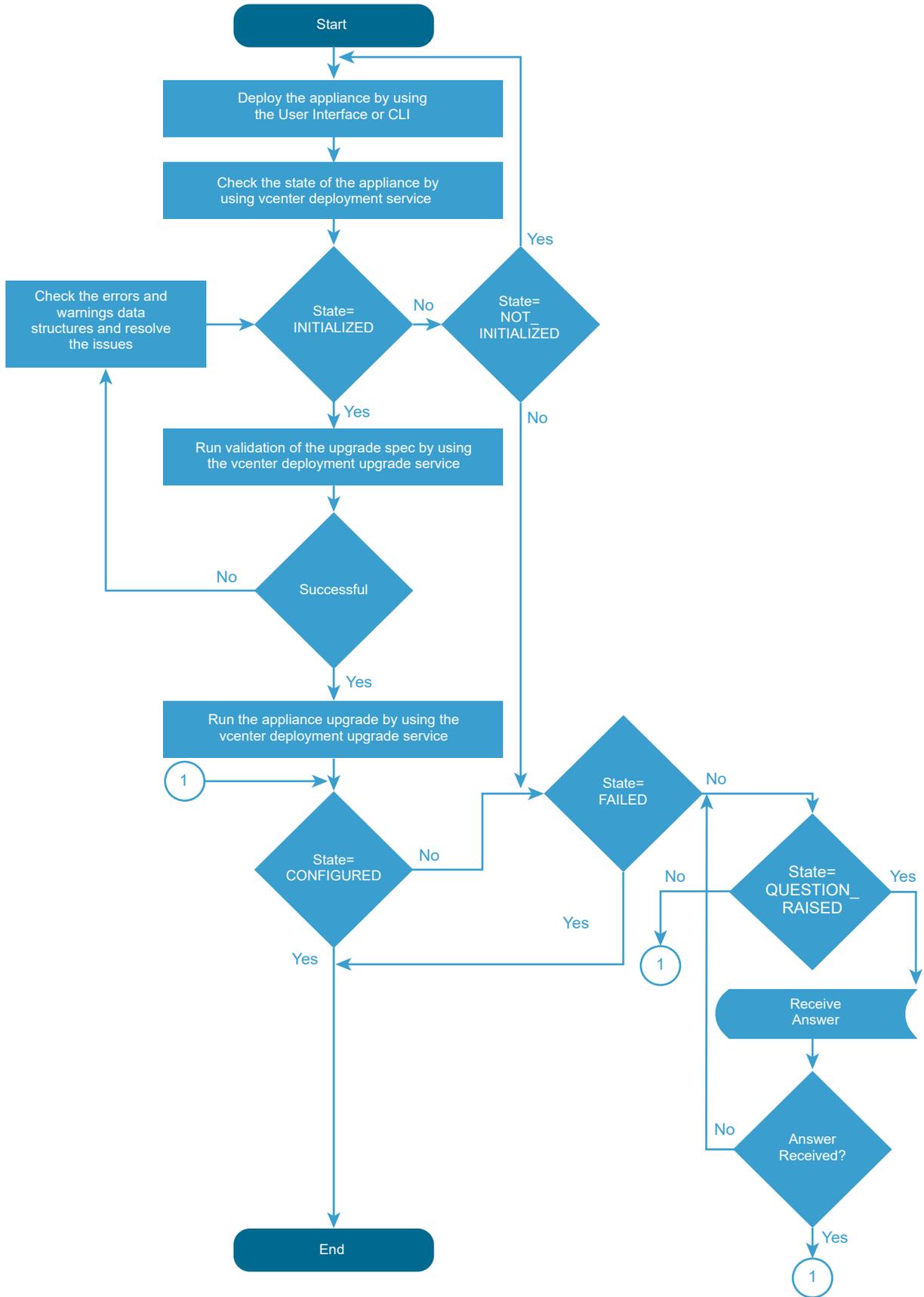
During stage 1, the vCenter Server instance is in a `NOT_INITIALIZED` state. After a successful deployment, the vCenter Server instance enters in an `INITIALIZED` state. If there are errors during stage 1, the vCenter Server instance stays in a `NOT_INITIALIZED` state and you must redeploy it.

You can check the state of the vCenter Server instance before, during, and after the upgrade process. You can run the upgrade stage 2 process if the vCenter Server instance is initialized. You can check the upgrade configuration before you run the upgrade by running pre-checks. If errors or warnings appear during the validation of the upgrade specification, you must remove the causes and correct the specification.

During the upgrade process, the vCenter Server instance can enter in a `FAILED` or `QUESTION_RAISED` state. If a question appears during the upgrade, the vCenter Server instance enters in a `QUESTION_RAISED` state and stays in it until you provide an answer. You can run the upgrade in silent mode, in which the vCenter Server instance does not generate questions, and uses default answers.

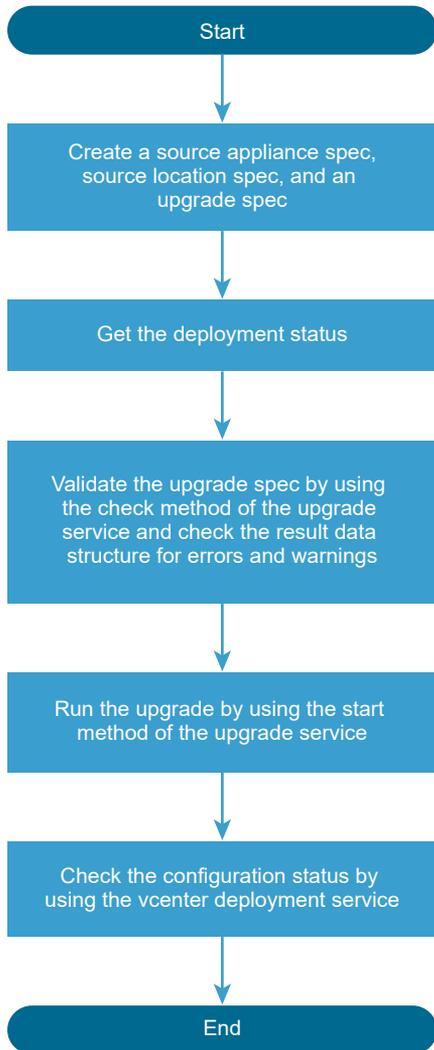
If errors appear during the upgrade, the vCenter Server instance enters in a `FAILED` state and you must remove the causes, redeploy the instance and restart the upgrade. If stage 2 of the upgrade process is successful, the vCenter Server instance enters in a `CONFIGURED` state. If the vCenter Server instance is configured, you can check its services and pause the historical data transfer.

Figure 5-5. Upgrade Workflow



For information about the states of the vCenter Server instance and available operations, see [Upgrading a vCenter Server Instance](#).

Figure 5-6. Upgrade Stage 2 Workflow



You can run the upgrade pre-checks and the upgrade stage 2 process by creating and passing an `UpgradeSpec`. In `UpgradeSpec`, you define the upgrade configuration and specify the source vCenter Server instance and the source ESXi host in `SourceApplianceSpec` and `LocationSpec`. See [Figure 5-6. Upgrade Stage 2 Workflow](#). You can run the upgrading in silent mode by setting the `UpgradeSpec.auto_answer` to `true`. The default value of `UpgradeSpec.auto_answer` is `false` and the upgrading is in interactive mode, in which you must provide answers to the raised questions.

For information about the classes, variables, and default values, see the *API reference* documentation.

Historical Data Transfer

If you migrate vCenter Server for Windows, you can transfer the historical data of your source vCenter Server instance together with the core configuration data.

Deferred Import

The deferred import is a process of historical data transfer after the successful migration of a vCenter Server instance with an external database. The historical data includes statistics, events, and tasks.

By using the deferred import feature, you can postpone the historical data transfer after the migration process completes, so that you manage the downtime of your environment. You can select whether all historical data, or only events and tasks, will be migrated with the core data during the migration. The historical data transfer and deferred import of historical data are disabled by default. You can enable and configure the historical data transfer by using the API, vCenter Server Management Interface, vCenter Server installer, or CLI installer. A vCenter Server super administrator can run and control the migration and deferred import processes.

If you use the deferred import feature, the historical data is migrated with the core data and the historical data import process starts automatically after a successful migration and when the vCenter Server instance is running. You can pause the historical data import and resume it later.

For information about how to configure and run the migration and deferred import processes by using the vCenter Server Management Interface, see the *vCenter Server Upgrade* documentation.

By using the API, you can configure, control, and monitor the data transfer process. If you use the API to enable the deferred import feature, you must create a history migration spec and set `defer_import` to `true`. For information about how to configure the deferred import by using the API, see [Class Diagrams for Deferred Import](#) and the *API reference*.

The data import process has five states that you can check. If the historical data migration and the deferred import are configured, the historical data import starts automatically after a successful migration.

Figure 5-7. Deferred Import State Diagram

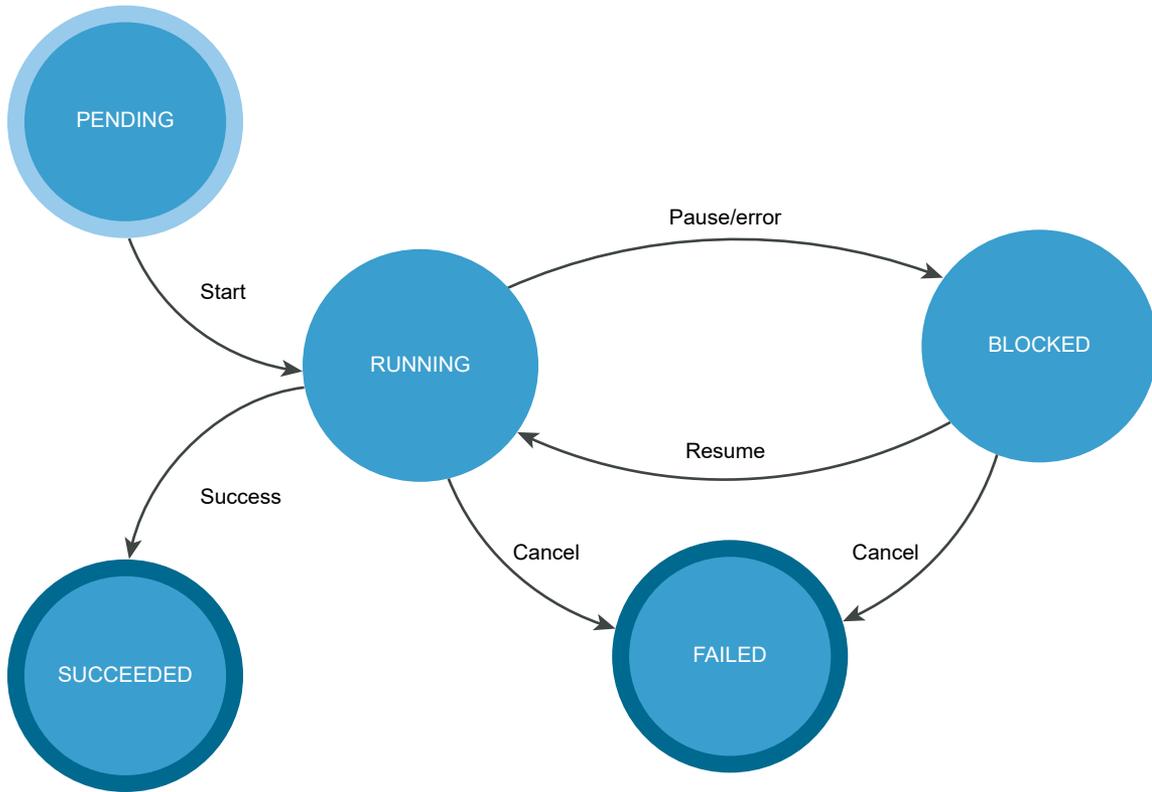


Table 5-7. User Operations

Operation	Description										
Pause	You can pause the running data transfer process. If the data transfer is paused, you can resume or cancel it.										
Cancel	You can cancel the data transfer process if it is in a RUNNING or BLOCKED state. Note If you cancel the data transfer, the process enters in a final FAILED state and you cannot resume the transfer.										
Resume	You can resume the stopped data transfer.										
Get status	You can retrieve the status of the data transfer process. There are five states.										
	<table border="0"> <tr> <td>PENDING</td> <td>The transfer is not started.</td> </tr> <tr> <td>RUNNING</td> <td>The transfer is started or resumed.</td> </tr> <tr> <td>BLOCKED</td> <td>The transfer is paused or there was a recoverable error, such as not enough disk space, during the import.</td> </tr> <tr> <td>SUCCEEDED</td> <td>The transfer is successful.</td> </tr> <tr> <td>FAILED</td> <td>The transfer is canceled.</td> </tr> </table>	PENDING	The transfer is not started.	RUNNING	The transfer is started or resumed.	BLOCKED	The transfer is paused or there was a recoverable error, such as not enough disk space, during the import.	SUCCEEDED	The transfer is successful.	FAILED	The transfer is canceled.
PENDING	The transfer is not started.										
RUNNING	The transfer is started or resumed.										
BLOCKED	The transfer is paused or there was a recoverable error, such as not enough disk space, during the import.										
SUCCEEDED	The transfer is successful.										
FAILED	The transfer is canceled.										

You can run the deferred import operations by using the API or sending an HTTP request.

Note When you send the requests, you must use an authentication.

If you send requests to port 5480, you must authenticate with vCenter Server **root** credentials. If you send requests to the vCenter Server reverse proxy port, you must authenticate with vCenter Single Sign-On credentials.

The following HTTP requests show the syntax that you can use to perform the available user operations.

```
https://<IP_of_migrated_instance>:5480/rest/vcenter/deployment/history?action=pause
```

```
https://<IP_of_migrated_instance>:5480/rest/vcenter/deployment/history?action=cancel
```

```
https://<IP_of_migrated_instance>:5480/rest/vcenter/deployment/history?action=resume
```

```
https://<IP_of_migrated_instance>:5480/rest/vcenter/deployment/history
```

```
https://<IP_of_migrated_instance>/rest/vcenter/deployment/history?action=pause
```

```
https://<IP_of_migrated_instance>/rest/vcenter/deployment/history?action=cancel
```

```
https://<IP_of_migrated_instance>/rest/vcenter/deployment/history?action=resume
```

```
https://<IP_of_migrated_instance>/rest/vcenter/deployment/history
```

For information about the status codes and historical data transfer errors, see [HTTP Status Codes for Deferred Import](#) and [Historical Data Import Errors](#).

If you pause the data transfer by using the API or an HTTP request, you can resume or cancel the process by using the API or the vCenter Server Management Interface.

Important If you cancel the transfer process, and want to transfer the historical data later, you must restart the migration process.

HTTP Status Codes for Deferred Import

Each deferred import operation returns an HTTP status code.

The following table lists the HTTP status codes that you can receive and the operations which can trigger each status code.

Table 5-8. HTTP Status Codes

HTTP Status Code	Description	Operations that return the status code
200	The operation is successful.	<ul style="list-style-type: none"> ■ pause ■ cancel ■ resume
400	You cannot perform the operation because the data import is in the current state. For information about the data import states and transitions between the states, see the Figure 5-7. Deferred Import State Diagram figure.	<ul style="list-style-type: none"> ■ pause ■ cancel ■ resume

Table 5-8. HTTP Status Codes (continued)

HTTP Status Code	Description	Operations that return the status code
401	You use invalid user name or password, or the authentication has failed.	<ul style="list-style-type: none"> ■ <code>pause</code> ■ <code>cancel</code> ■ <code>resume</code> ■ <code>getstatus</code>
403	You do not have permissions to perform the operation. The vCenter Server super administrator has permissions to run the operation.	<ul style="list-style-type: none"> ■ <code>pause</code> ■ <code>cancel</code> ■ <code>resume</code> ■ <code>getstatus</code>
500	There is a <code>vapi std</code> error. For information about the types of <code>vapi std</code> errors, see <i>vapi.std.errors</i> in the <i>API reference</i> .	<ul style="list-style-type: none"> ■ <code>pause</code> ■ <code>cancel</code> ■ <code>resume</code> ■ <code>getstatus</code>

Historical Data Import Errors

If an error appears during the data import, the import stops and the process enters in a `BLOCKED` state. You can resume the data import after you eliminate the cause.

You can check the errors, warnings, and info messages by reading the `info`, `status`, and `notifications` data structures.

If the information in the data structures is not enough, you can download the vCenter Server support-bundle from `<server>:443/appliance/support-bundle` and check the log files.

Table 5-9. Log Files

Log File	Path
API log file	<code>/var/log/vmware/applmgmt/applmgmt.log</code>
Backend log file	<code>/var/log/vmware/upgrade/upgrade-post-import.log</code>
Upgrade Runner log file	<code>/var/log/vmware/upgrade/deferredimport-upgrade-runner.log</code>
Deferred import log file	<code>/var/log/vmware/upgrade/DeferredImport_com.vmware.vcdb_<date_time>.log</code>

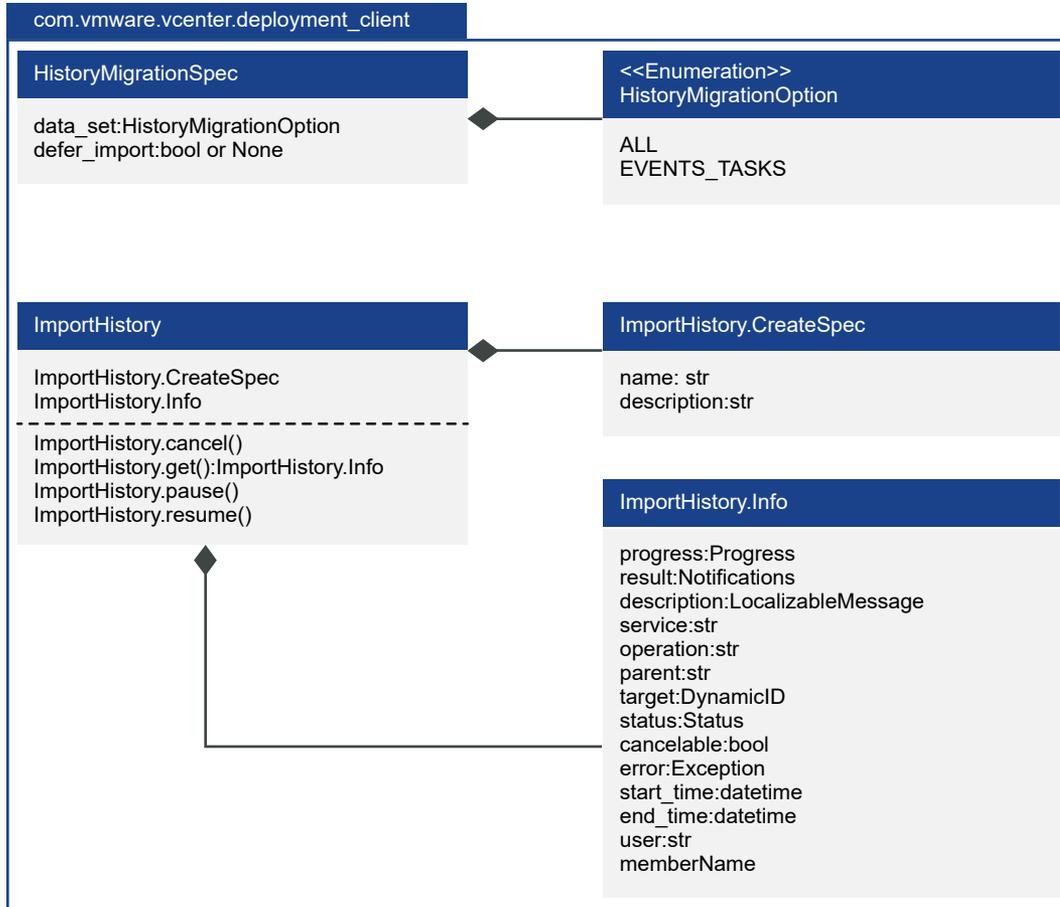
Class Diagrams for Deferred Import

The `vcenter deployment` API provides classes and interfaces that you can use for configuring and controlling the historical data import.

The historical data transfer during the vCenter Server migration is disabled by default and only the core data is migrated. You can enable the historical data transfer and the deferred import by creating a history migration spec and setting `defer_import` to `true`. For example, see [Figure 5-8. Python Class Diagrams for Deferred Import](#). You can change the historical data scope by using the `HistoryMigrationOption` enumeration. By default, the `data_set` is set to `EVENTS_TASKS`.

You can control the deferred import by creating an import history spec and calling the methods of the `ImportHistory` class. [Figure 5-8. Python Class Diagrams for Deferred Import](#) shows the classes that you can use to configure and control the deferred import.

Figure 5-8. Python Class Diagrams for Deferred Import



Use the Deferred Import Sample

You can use the vCenter Server Management Interface to migrate your vCenter Server instance and to run the `vc_import_history_sample.py` sample to pause and resume the historical data import.

Prerequisites

- Verify that you cloned or downloaded the vSphere Automation SDK for Python from <https://github.com/vmware/vsphere-automation-sdk-python>.
- Verify that you set up a test environment. For information about the prerequisites and how to set up a test environment, see the `README.md` file in the `deferhistoryimport` directory and the Quick Start guide for vSphere Automation SDK for Python at <https://github.com/vmware/vsphere-automation-sdk-python>.
- Verify that you have vCenter Server root credentials.

- Verify that you opened the vCenter Server Management Interface of your target vCenter Server instance.

Procedure

- 1 From the getting started page, run the **Migrate** wizard.
- 2 Select one of the options for **Configuration and historical data**.
You can import only events and tasks or import all historical data.
- 3 Select **Import historical data in the background** and complete all steps from the wizard.
- 4 Run the `vc_import_history_sample.py` sample after successful migration.

Use the IP address of your source vCenter Server instance and the vCenter Server administrator credentials when you run the sample. You can use or skip the verification of the vCenter Server certificate. For example, you can use the following command.

```
vc_import_history_sample.py --server <IP_of_migrated_instance> --username <admin_username> --password <admin_password> --skipverification
```

Python Example of Pausing and Resuming the Deferred Import Process

The example shows how you can pause and resume the deferred import process by using the API. The example is based on the `vc_import_history_sample.py` sample.

Note For a complete and up-to-date version of the sample code, see the `vsphere-automation-sdk-python` VMware repository at GitHub.

```
...
self.service_manager = ServiceManager(args.server,
                                      args.username,
                                      args.password,
                                      args.skipverification)

self.service_manager.connect()
...
# Using REST API service
import_history = ImportHistory(self.service_manager.stub_config)
...
# Change the status - either pause or resume it
start_status = get_defer_history_import_status(import_history)
if start_status == Status.RUNNING:
    print('Pausing Defer History Data Import.')
    import_history.pause()
    ...
elif start_status == Status.PAUSED:
    print('Resuming Defer History Data Import.')
    import_history.resume()
...

```

Monitoring vCenter Server

6

vCenter Server provides interfaces to check the current health of its components, and to report the system's history of resource consumption. You can use these interfaces to spot potential trouble areas or predict future shortages.

This chapter includes the following topics:

- [Health Monitoring of vCenter Server](#)
- [Capacity Monitoring of vCenter Server](#)

Health Monitoring of vCenter Server

The vCenter Server API offers health status indicators for several key components of the system. These indicators can be polled periodically to monitor the components for problems.

The health status indicators report graded values from green to red. The general meanings of the grades are as follows.

green	The component is healthy.
yellow	The component is healthy, but may have some problems.
orange	The component is degraded, and may have serious problems.
red	The component is unavailable, or will stop functioning soon.
gray	No health data is available.

Check Overall System Health of vCenter Server

vCenter Server provides a composite health indicator that enables you to test a single value that represents the health of all system components. This procedure shows how to test the composite health indicator.

The value of the overall system health indicator reflects the strongest trouble indication among the vCenter Server components. If any component has a red indicator, the overall system health indicator is red, else if any component has an orange indicator, the overall system health indicator is orange, and so on.

A gray value for any component indicates that data for the subsystem is unknown. If one or more components have a gray value, but all other subsystems are green, then the overall system health indicator is gray rather than green. However, if any component has a definite trouble indication, the overall system health indicator reflects the strongest trouble indication among the components.

Prerequisites

Verify that you have an active authenticated session with vCenter Server. This procedure assumes that the session ID is present in the security context of a stub configuration.

Procedure

- 1 Create an interface stub or REST path that uses the stub configuration.
- 2 Invoke the `health.system` method.
- 3 Format and display the resulting value.

JavaScript Example of Checking Overall System Health of vCenter Server

This example shows the use of JavaScript with the vSphere Automation SDK for REST to request the overall system health indicator for vCenter Server.

The example assumes a previously existing session with the vSphere Automation API endpoint. The JavaScript code depends on the `Node.js` package.

This example depends on the following global variables.

- `my_http_options`

```
var https = require('https');
var httpPort = 443;
var httpPath = '/rest/appliance/health/system';
var httpMethod = 'GET';

// Prepare the HTTP request.
my_http_options = session.my_http_options;
my_http_options.method = httpMethod;
my_http_options.path = httpPath;

// Define the callbacks.
function callback(res) {
  res.on('error',
    function(err) {console.log('ERROR checking system health: ', err)});
  res.on('data', function(chunk) {data = chunk.toString();});
  res.on('end', function() {
    if (res.statusCode == 200) {
      console.log('Overall system health status: ', JSON.parse(data).value);
    }
  })
}
```

```

    })
};

// Issue the request.
https.request(my_http_options, callback).end();

```

Python Example of Checking the Overall System Health of vCenter Server

This example shows the use of Python with the vSphere Automation SDK for Python to request the overall system health indicator for vCenter Server and the overall health indicator for management services. The example assumes a previously existing session with the vSphere Automation API endpoint.

This example depends on the following global variables.

- *my_stub_config*

```

from com.vmware.appliance import health_client

# This example assumes you have previously created a session
# and stored the session ID in my_stub_config.

# Issue request for overall system health.
System_stub = health_client.System( my_stub_config )
health = System_stub.get()
print( 'Overall system health: %s' % health )

# Issue request for applmgmt services health.
Applmgmt_stub = health_client.Applmgmt( my_stub_config )
health = Applmgmt_stub.get()
print( 'Applmgmt services health: %s' % health )

```

Capacity Monitoring of vCenter Server

vCenter Server keeps a history of statistics that you can use to monitor resources used by the vCenter Server instance.

You can use the statistics to spot peak usage demands or to monitor trends for advance warning of potential resource exhaustion.

Frequency and Retention of Statistics Collection in vCenter Server

vCenter Server collects statistics from the guest operating system at regular intervals and stores them in a database. Users can query the statistics in the database by selecting a time period and a roll-up function that vCenter Server applies to the statistics before returning them to the client.

After the monitoring service starts up, it begins requesting statistics from the guest operating system periodically, at a frequency that depends on the type of statistic. The service requests storage statistics once every 10 minutes, while it requests memory, CPU, and networking statistics once per minute. The collection times are fixed relative to the startup time of the monitoring service, rather than to clock time.

The monitoring service retains statistics approximately 13 months, by default. Older statistics are deleted by the service, creating a 13-month moving window within which you can query statistics. You can choose to delete statistics as needed to conserve storage resources.

Nature of Statistics in vCenter Server

vCenter Server supplies statistics of several types.

The guest operating system computes statistics either as rates, such as CPU cycles per second, or as snapshots of size, such as KB used for storage. Statistics stored as size snapshots are collected at the end of their sample periods. Statistics stored as rates are computed as averages of values sampled frequently during each sample period.

When you query the statistics database, the units are not returned with the data, but you can determine the units for any metric by requesting metadata for the metric with the `get()` method.

Requesting Statistics from vCenter Server

To request statistics, you must construct an appropriate request structure to filter statistics from the database.

To request data or metadata for a metric, you must supply the ID of the metric. You can get a list of metric IDs by using the `list()` method, which returns information on all available metrics.

When you query statistics, you provide a list of IDs to specify the metrics in which you are interested. You also supply a start time, an end time, a roll-up interval, and a roll-up function. These values interact as follows to determine the data returned to you.

- The response contains a list of data points for each metric ID you specified in the request.
- The start time and end time control the limits for the data you want in the response. The response contains data points only for statistics that have timestamps between those limits, inclusive of the endpoints. However, the start time is adjusted to a round number, in some cases. For more information, see [Statistics Interval Adjustment in vCenter Server](#).
- The roll-up interval enables you to control the granularity of the data points in the response. Rather than a response with a data point for every statistic between the start time and end time, you get a response with a number of data points equal to the number of intervals between the start and end times. Generally, you should specify a time period that is an even multiple of the interval, so that each data point in the response represents the same number of statistics.
- The roll-up function specifies how the response summarizes the statistics that fall within each interval. The resulting data point can be the maximum statistic value within collection interval, or the mean of the statistics values within the interval, and so on.

Statistics Collection Times

The actual time that a statistic was collected is not readily predictable.

The API does not enable you to determine the exact time that a statistic was collected. Furthermore, some statistics, such as those for storage metrics, might take seconds or minutes to collect, so that they are not available immediately at the time a request is made to the guest operating system.

However, because statistics are collected at regular intervals, and roll-up intervals for a request generally all have the same size, each data point in the response represents the same number of statistics as the others. See [Statistics Interval Adjustment in vCenter Server](#) for more information.

Statistics Interval Adjustment in vCenter Server

When you make a request for statistics, the monitoring service might adjust the specified roll-up interval times to improve the appearance of statistics graphs in a graphical interface.

The monitoring service adjusts the start time of a data collection request when it is not an exact multiple of the interval length. In these instances, the start time is rounded downward to the previous UTC time that is a multiple of the interval. All subsequent intervals of the data collection are also adjusted to align with the new start time.

For example, if the start time is 10:31 and the interval length is 1 hour, the monitoring service adjusts the start time to 10:00 and the roll-up intervals have the following continuous pattern.

- 10:00 to 10:59:59.999
- 11:00 to 11:59:59.999
- 12:00 to 12:59:59.999

The monitoring service does not adjust the end time of a data collection. Consequently, the response to a statistics query might contain one more data value than expected, or an incomplete final interval might be lengthened.

Empty Data Values

In some instances, you might encounter a response that reports an empty data value, or even a series of empty data values. This might manifest as a list of data values containing some numeric values alternating with empty values.

- Empty data values can happen when the report time period is too short to be certain of containing any statistics. For instance, a time period of 30 seconds is half the length of the sample period for network metrics, so you have only a 50% chance of finding a network statistic during any 30-second reporting period.
- Empty data values can also happen when the interval is shorter than the sample period for a metric you have requested. In this case, some data points are present in the list, while others are empty because no statistic was collected during those intervals. For instance, an interval of 5 minutes is only half the length of the sample period for storage metrics, so every second data value is empty.

- Empty data values can also happen when the monitoring service has not finished collecting and writing the last sample to the database, even if the nominal sample timestamp falls within the report time period. For example, calculation of storage used can delay writing a storage statistic to the database. A request for the statistic during that delay time produces an empty data point in the response.

When a response contains an empty data value, this indicates that no statistics were collected during a collection interval. An appropriate action for the client in such a case depends on how the client is using the data. For example, if you are graphing a resource usage trend, you might choose to interpolate for the missing value to produce a smooth line.

Check Database Usage in vCenter Server

vCenter Server contains a database of all objects managed by the vCenter Server instance. In addition to inventory objects, the database includes vCenter Server statistics, events, alarms, and tasks. You can calculate the database storage consumption by adding the sizes of all data categories.

You need to monitor storage consumption in vCenter Server.

Prerequisites

This task assumes you have previously authenticated and created a client session.

Procedure

- 1 Prepare a request for database usage statistics.
Include metric IDs both for `vcdb_core_inventory` and `vcdb_seat`. The name `vcdb_seat` refers to Statistics, Events, and Tasks in the vCenter Server database.
- 2 Issue the request to the API endpoint.
- 3 Process the resulting data points as needed.
- 4 Format and print the results.

Results

The result of this procedure shows the storage used in the vCenter Server database, which includes storage overhead used for indexes and other optimizations beyond the actual size of the data.

JavaScript Example of Checking Database Usage in vCenter Server

This example shows the use of JavaScript with the vSphere Automation SDK for REST to view recent statistics for the vCenter database usage in vCenter Server.

The example assumes a previously existing session with the vSphere Automation API endpoint. The JavaScript code depends on the `Node.js` package.

This example requests statistics at 30-minute intervals for a recent 2-hour report period. The example requests the storage used by the inventory component and the storage used by the combination of statistics, events, alarms, and tasks. The example adds the two values to calculate the vCenter Server database usage in each 30-minute roll-up interval, and then reports the maximum size found over the 2-hour report period.

This example depends on the following global variables.

- *my_http_options*

```
var https = require('https');
var httpPort = 443;
var httpPath = '/rest/appliance/monitoring/query';
var httpMethod = 'GET';

// Prepare the HTTP request.
my_http_options = session.my_http_options;
my_http_options.method = httpMethod;
var query = '?item.names.1=storage.used.filesystem.vcdb_core_inventory';
query += '&item.names.2=storage.used.filesystem.vcdb_seat';
query += '&item.function=MAX&item.interval=MINUTES30';
var d_now = new Date();
var ms = d_now.getTime();
var min_15 = 15*60*1000;
var endTime = new Date( d_now - min_15 ).toISOString();
var startTime = new Date( d_now - 9*min_15 ).toISOString();
query += '&item.start_time=' + startTime;
query += '&item.end_time=' + endTime;
my_http_options.path = httpPath + query;

// Define the callbacks.
function callback(res) {
  res.on('error', function(err) {console.log('ERROR querying database size: ', err)});
  res.on('data', function(chunk) {data = chunk.toString();});
  res.on('end', function() {
    if (res.statusCode === 200) {
      var results = JSON.parse(data).value;
      var coreSizes = results[0].data;
      var seatSizes = results[1].data;

      // Eliminate empty data points and process remaining data.
      var highest = coreSizes.filter(function(s){return s !== ''}).
        map(function(n,i){return parseInt(n) + parseInt(seatSizes[i])}).
        reduce(function(t,n){return Math.max( t,parseInt(n) )});
      console.log('vCenter database inventory + stats, events, alarms, tasks: (max) size = ',
        highest);
    }
  });
};
// Issue the request.
https.request(my_http_options, callback).end();
```

Python Example of Checking Database Usage in vCenter Server

This example shows the use of Python with the vSphere Automation SDK for Python to view recent statistics for the vCenter database usage in vCenter Server. The example assumes a previously existing session with the vSphere Automation API endpoint.

This example requests statistics at 30-minute intervals for a recent 2-hour report period. The example requests the storage used by the inventory component and the storage used by the combination of statistics, events, alarms, and tasks. The example adds the two values to calculate the vCenter Server database usage in each 30-minute roll-up interval, and then reports the maximum size found over the 2-hour report period.

```
from com.vmware import appliance_client
import datetime

# This example assumes you have previously created a session
# and stored the session ID in my_stub_config.

# Issue request for core inventory and 'SEAT' (stats, events, & alarms) usage.
req = appliance_client.Monitoring.MonitoredItemDataRequest()
req.names = ['storage.used.filesystem.vcdb_core_inventory',
            'storage.used.filesystem.vcdb_seat']
req.interval = appliance_client.Monitoring.IntervalType.MINUTES30
req.function = appliance_client.Monitoring.FunctionType.MAX
d_now = datetime.datetime.utcnow()
req.start_time = d_now - datetime.timedelta( minutes=135 )
req.end_time = d_now - datetime.timedelta( minutes=15 )
Monitoring_stub = appliance_client.Monitoring( my_stub_config )
resp = Monitoring_stub.query( req )

# Extract resulting arrays.
core_sizes = resp[0].data
seat_sizes = resp[1].data
# Remove empty data points:
core_sizes = filter( (lambda x: x != ''), core_sizes )
seat_sizes = filter( (lambda x: x != ''), seat_sizes )

# Add the usage stats for each interval, and display maximum usage.
highest = max( map( (lambda a,b: int(a) + int(b)),
                  core_sizes, seat_sizes ) )
print( 'vCenter database inventory + stats, events, alarms, tasks:' +
      ' (max) size = {0} KB'.format( highest ) )
```

List Storage Consumption By Data Type in vCenter Server

vCenter Server provides statistics on several types of storage.

For example, you can query statistics about inventory storage, transaction log, and vCenter Server tasks. Many of these statistics are available both for storage consumed and storage available.

This task provides data for system administrators who need to monitor storage consumption in the guest operating system of vCenter Server.

Prerequisites

Verify that you have authenticated and created a client session.

Procedure

- 1 Prepare a request for database usage statistics.
Include metric IDs for each data type you wish to monitor.
- 2 Issue the request to the API endpoint.
- 3 Process the resulting data points as needed.
- 4 Format and print the results.

JavaScript Example of Listing Storage Consumption By Data Type in vCenter Server

This example shows the use of JavaScript with the vSphere Automation SDK for REST to break down storage consumption by type in vCenter Server.

The example assumes a previously existing session with the vSphere Automation API endpoint. The JavaScript code depends on the `Node.js` package.

This example requests the most recent statistics for several categories of storage. The example requests both current usage and total storage available for each category, then calculates the percentage used in each category.

This example depends on the following global variables.

■ *my_http_options*

```

var https = require('https');
var httpPort = 443;
var httpPath = '/rest/appliance/monitoring/query';
var httpMethod = 'GET';
// Prepare the HTTP request.
my_http_options = session.my_http_options;
my_http_options.method = httpMethod;
var query = '?item.function=MAX&item.interval=MINUTES30';
var d_now = new Date();
var ms = d_now.getTime();
var min_30 = 30*60*1000;
var startTime = new Date( d_now - min_30 ).toISOString();
var endTime = d_now.toISOString();
query += '&item.start_time=' + startTime;
query += '&item.end_time=' + endTime;
// Array of monitoring points with output labels:
var mon = [ ['VCDB core: ',
            'storage.used.filesystem.vcdb_core_inventory',
            'storage.totalsize.filesystem.vcdb_core_inventory'],
            ['VCDB SEAT: ',
            'storage.used.filesystem.vcdb_seat',
            'storage.totalsize.filesystem.vcdb_seat'],
            ['Log: ',

```

```

        'storage.used.filesystem.log',
        'storage.totalsize.filesystem.log'] ];
for (var i=0, j=0; i<mon.length; i++) {
    query += '&item.names.' + (++j).toString() + '=' + mon[i][1];
    query += '&item.names.' + (++j).toString() + '=' + mon[i][2];
}
my_http_options.path = httpPath + query;
// Define the callbacks.
function callback(res) {
    res.on('error', function(err) {
        console.log('ERROR retrieving storage sizes: ', err));
    });
    res.on('data', function(chunk) {data = chunk.toString();});
    res.on('end', function() {
        if (res.statusCode == 200) {
            var results = JSON.parse(data).value;
            console.log(results);
            for (var i=0, j=0; i<mon.length; i++) {
                // Discard empty data points:
                var u='', t;
                while (u == '') {
                    u = results[j].data.pop();
                    t = results[j+1].data.pop();
                }
                j += 2;
                mon[i].push(u, t, (u == 0 ? 0 : Math.floor(10000 * u / t)) / 100);
            }
            for (i=0; i<mon.length; i++) {
                console.log(mon[i][0], mon[i][3], '/', mon[i][4], ' (' , mon[i][5], '%)');
            }
        }
    });
}
};
// Issue the request.
https.request(my_http_options, callback).end();

```

Python Example of Listing Storage Consumption By Data Type in vCenter Server

This example shows how to use the Monitoring interface to break down database usage by data type. The example requests the individual data types that you can also query as a composite metric for all storage used by Alarms, Statistics, Events, and Tasks in the vCenter Server instance.

```

from com.vmware import appliance_client
import datetime

# This example assumes you have previously created a session
# and stored the session ID in my_stub_config.

# Prepare request for chosen data types.
req = appliance_client.Monitoring.MonitoringItemDataRequest()
req.interval = appliance_client.Monitoring.IntervalType.MINUTES30
req.function = appliance_client.Monitoring.FunctionType.MAX
d_now = datetime.datetime.utcnow()
req.start_time = d_now - datetime.timedelta( minutes=30 )

```

```

req.end_time = d_now
mon = {'storage.totalsize.directory.vcdb_hourly_stats' :
      'Hourly stats',
      'storage.totalsize.directory.vcdb_daily_stats' :
      'Daily stats',
      'storage.totalsize.directory.vcdb_monthly_stats' :
      'Monthly stats',
      'storage.totalsize.directory.vcdb_yearly_stats' :
      'Yearly stats',
      'storage.totalsize.directory.vcdb_events' :
      'Events',
      'storage.totalsize.directory.vcdb_alarms' :
      'Alarms',
      'storage.totalsize.directory.vcdb_tasks' :
      'Tasks'}

req.names = []
for item in mon.keys() :
    req.names.append( item )

# Issue request.
Monitoring_stub = appliance_client.Monitoring( my_stub_config )
resp = Monitoring_stub.query( req )

# Assemble data from response.
out = {}
for metric in resp :
    # Discard empty data points:
    stat = ''
    while (stat == '') :
        stat = metric.data.pop()
    stat = int(stat)
    out[mon[metric.name]] = stat

# Format and print statistics.
for label in sorted( out.keys() ) :
    print( '{0:15s}: {1:8d} KB'.format( label, out[label] ) )

```

Maintenance of vCenter Server



The vCenter Server Management API facilitates backup and restore operations.

You can create a backup file that includes the database and configuration of the vCenter Server instance. You can also use the API to restore the backup file into a freshly deployed vCenter Server instance.

This chapter includes the following topics:

- [Backing up vCenter Server](#)
- [Restoring vCenter Server](#)
- [Reconcile a vCenter Server Instance with Nodes in Embedded Linked Mode](#)
- [Managing System Logs](#)
- [Performing Infrastructure Profile Management Operations](#)

Backing up vCenter Server

The vCenter Server Management API supports backing up key parts of the vCenter Server instance. This allows you to protect vCenter Server data and to minimize the time required to restore data center operations.

The backup process collects key files into a tar bundle and compresses the bundle to reduce network load. To minimize storage impact, the transmission is streamed without caching in the vCenter Server instance. To reduce total time required to complete the backup operation, the backup process handles the different components in parallel.

You have the option to encrypt the compressed file before transmission to the backup storage location. When you choose encryption, you must supply a password which can be used to decrypt the file during restoration.

The backup operation always includes the vCenter Server database and system configuration files, so that a restore operation has all the data needed to re-create an operational vCenter Server instance. Current Alarms are included as well. You also have the option to specify additional data sets, called parts. In this release, you can specify a data set that includes Statistics, Events, and Tasks.

Backup and Restore Protocols for vCenter Server

The vCenter Server backup and restore feature supports a number of plug-in communication protocols.

Choose one of these protocols as the backup location type when you invoke the operation.

- FTP
- FTPS
- SCP
- HTTP
- HTTPS
- NFS
- SMB

The value `PATH` for the location type field indicates a locally mounted volume.

Note If you specify the `SCP` protocol, you must specify an absolute path as the value of the location type field when you create the backup job.

Calculate the Size Needed To Store the Backup File

When you prepare to do a backup of a vCenter Server instance, you can use the API to calculate the storage space needed for the backup file.

You can do this task when you are choosing a backup storage location or whenever your existing storage location may be approaching full capacity.

Prerequisites

- Verify that you have a vCenter Server instance running.
- Verify that you are familiar with authentication methods. See [Chapter 3 Authentication Mechanisms](#).

Procedure

- 1 Authenticate to the vSphere Automation API endpoint and establish a session.
- 2 Request a list of backup parts available.
- 3 For each available backup part, request the size of the backup file.

The backup process calculates the compressed size of each backup part.

- 4 Choose which parts to include in the backup, and sum their sizes.

The backup storage server must have sufficient space to contain the chosen parts.

What to do next

After you choose which backup parts you will store, and verify that the backup storage server has sufficient free space, you can launch a backup job. For information, see [Back up a vCenter Server Instance by Using the API](#).

Bash Example of Calculating the Size Needed To Store the Backup File

This example shows how to invoke `curl` from a bash script to collect the information you use to calculate the size needed to store a backup file of the vCenter Server instance.

This example depends on certain variables that specify the address of the vCenter Server instance and credentials to access the `appliance.recovery` API. For simplicity, the variables are hard-coded at the start of the bash script.

```
#!/bin/bash
##### EDITABLE BY USER to specify vCenter Server instance and credentials. #####
VC_ADDRESS=vcsa_ip
VC_USER=sso_user
VC_PASSWORD=sso_pass
#####

# Authenticate with basic credentials.
curl -u "$VC_USER:$VC_PASSWORD" \
  -X POST \
  -k --cookie-jar cookies.txt \
  "https://$VC_ADDRESS/rest/com/vmware/cis/session"
echo ''

# Issue a request to list the backup file parts.
curl -k -s --cookie cookies.txt \
  -H 'Accept:application/json' \
  "https://$VC_ADDRESS/rest/appliance/recovery/backup/parts" \
  >response.txt

# Extract IDs of backup file parts.
IDs=$(awk '{for (s=$0; match(s,/"id":"\w+"/); s=substr(s,RSTART+RLENGTH)) \
  print substr(s,RSTART+6,RLENGTH-7);}' \
  response.txt)

# Request sizes of parts.
echo Backup space required, by part ID:
let "total=0"
for ID in $IDs ; do
  curl -k -s --cookie cookies.txt \
    -H 'Accept:application/json' \
    "https://$VC_ADDRESS/rest/appliance/recovery/backup/parts/$ID" \
    >response.txt
  size=$(awk '{if (match($0,/"value":\w+/)) \
    print substr($0,RSTART+8,RLENGTH-8);}' \
    response.txt)
  printf "  %-8s - %5dMB\n" "$ID" "$size"
  let "total += $size"
done
```

```

echo ''
echo "Complete backup file size:  ${total}MB"

# Clean up temporary files.
echo ''
rm -f response.txt
rm -f cookies.txt

```

Python Example of Calculating the Size Needed To Store the Backup Image

This example shows how to use Python to collect the information you need to calculate the size needed to store a backup image of the vCenter Server instance.

```

from com.vmware.appliance.recovery.backup_client import Parts

# This example assumes you have previously created a session
# and stored the session ID in my_stub_config.

# Issue a request to list the backup image parts.
Parts_stub = Parts( my_stub_config )
parts = Parts_stub.list()

# Extract IDs of backup image parts.
sizes = {}
total = 0
for part in parts :
    size = Parts_stub.get( part.id )
    sizes[part.id] = size
    total += size

# Show the result.
print( 'Backup image parts:' )
for part_id in sizes.keys() :
    print( ' part {0} = {1}KB'.format( part_id, sizes[part_id] ) )
print( 'Total size: {0}KB'.format( total ) )

```

Back up a vCenter Server Instance by Using the API

You can use the Management API to create a backup of the vCenter Server database and key components of the vCenter Server instance.

This procedure explains the sequence of operations you use to create a backup file of the vCenter Server instance. You can do this as part of a regular maintenance schedule.

Prerequisites

- Verify that the vCenter Server instance is in a ready state. All processes with start-up type automatic must be running.
- Verify that no other backup or restore jobs are running.
- Verify that the destination storage location is accessible to the backup process.
- Verify that the path to the destination directory already exists, as far as the parent directory.

- If the destination directory does not exist, the backup process will create it. If the directory does exist, verify that it is empty.
- Verify that the destination storage device has sufficient space for the backup file. For information about how to calculate the space needed for the backup file, see [Calculate the Size Needed To Store the Backup File](#).

Procedure

- 1 Authenticate to the vSphere Automation API endpoint and establish a session.
- 2 Create a backup request object to describe the backup operation.

The request specifies several attributes, especially the backup location, the protocol used to communicate with the storage server, the necessary authorization, and which optional parts of the database you want to back up. The core inventory data and Alarms are always backed up, but you can choose whether or not to back up Statistics, Events, and Tasks. Collectively, this optional part of the backup is referred to as `seat`.

- 3 Issue a request to start the backup operation.
- 4 From the response, save the unique job identifier of the backup operation.
- 5 Monitor progress of the job until it is complete.
- 6 Report job completion.

Bash Example of Backing up a vCenter Server Instance

This example shows how to invoke `curl` from a bash script to back up the vCenter Server instance. A bash script can be invoked regularly as a cron job in the vCenter Server instance.

This example depends on certain variables that specify the source and destination for the backup operation. For simplicity, the variables are hard-coded at the start of the bash script.

This script does not encrypt the backup file.

```
#!/bin/bash
##### EDITABLE BY USER to specify vCenter Server instance and backup destination. #####
VC_ADDRESS=vc_server_ip
VC_USER=sso_user
VC_PASSWORD=sso_pass
FTP_ADDRESS=storage_server
FTP_USER=ftpuser
FTP_PASSWORD=ftpuser
BACKUP_FOLDER=backup
#####

# Authenticate with basic credentials.
curl -u "$VC_USER:$VC_PASSWORD" \
  -X POST \
  -k --cookie-jar cookies.txt \
  "https://$VC_ADDRESS/rest/com/vmware/cis/session"

# Create a message body for the backup request.
```

```

TIME=$(date +%Y-%m-%d-%H-%M-%S)
cat << EOF >task.json
{ "piece":
  {
    "location_type":"FTP",
    "comment":"Automatic backup",
    "parts":["seat"],
    "location":"ftp://$FTP_ADDRESS/$BACKUP_FOLDER/$TIME",
    "location_user":"$FTP_USER",
    "location_password":"$FTP_PASSWORD"
  }
}
EOF

# Issue a request to start the backup operation.
echo Starting backup $TIME >>backup.log
curl -k --cookie cookies.txt \
  -H 'Accept:application/json' \
  -H 'Content-Type:application/json' \
  -X POST \
  --data @task.json 2>>backup.log >response.txt \
  "https://$VC_ADDRESS/rest/appliance/recovery/backup/job"
cat response.txt >>backup.log
echo '' >>backup.log

# Parse the response to locate the unique identifier of the backup operation.
ID=$(awk '{if (match($0,/"id":"\w+-\w+-\w+"/)) \
  print substr($0, RSTART+6, RLENGTH-7);}' \
  response.txt)
echo 'Backup job id: '$ID

# Monitor progress of the operation until it is complete.
PROGRESS=INPROGRESS
until [ "$PROGRESS" != "INPROGRESS" ]
do
  sleep 10s
  curl -k --cookie cookies.txt \
    -H 'Accept:application/json' \
    --globoff \
    "https://$VC_ADDRESS/rest/appliance/recovery/backup/job/$ID" \
    >response.txt
  cat response.txt >>backup.log
  echo '' >>backup.log
  PROGRESS=$(awk '{if (match($0,/"state":"\w+"/)) \
    print substr($0, RSTART+9, RLENGTH-10);}' \
    response.txt)
  echo 'Backup job state: '$PROGRESS
done

# Report job completion and clean up temporary files.
echo ''
echo "Backup job completion status: $PROGRESS"
rm -f task.json
rm -f response.txt

```

```
rm -f cookies.txt
echo '' >>backup.log
```

Python Example of Backing Up a vCenter Server Instance

This example specifies that the backup image should include Statistics, Events, and Tasks as well as the core inventory and alarm data. The value for `req.parts` indicates the optional data part for Statistics, Events, and Tasks.

This example uses the following global variables.

- `my_storage_server`
- `my_backup_folder`
- `my_scp_user`
- `my_scp_password`
- `my_stub_config`

When you back up the vCenter Server instance, you need two sets of authentication credentials. The API client needs to authenticate to the vCenter Server instance, and the backup service needs to authenticate to the backup storage server.

The example assumes that your API client has already authenticated the connection to the vCenter Server instance, and the security context is stored in `my_stub_config`.

In the backup request, you need to specify the folder that will contain the backup image. The folder name must be specified as a path name relative to the home directory of the user that authenticates with the storage server.

```
from com.vmware.appliance.recovery.backup_client import Job
import time

# This example assumes you have previously created a session
# and stored the session ID in my_stub_config.

# Create a backup request object.
req = Job.BackupRequest()
# Include optional backup part for Statistics, Events, and Tasks.
req.parts = ['seat']
req.location_type = Job.LocationType.SCP
req.comment = 'On-demand backup'
req.location = my_storage_server + ':/home/scpuser/' + my_backup_folder \
    + '/' + time.strftime('%Y-%m-%d-%H-%M-%S')
req.location_user = my_scp_user
req.location_password = my_scp_password

# Issue a request to start the backup operation.
backup_job = Job( my_stub_config )
job_status = backup_job.create( req )
job_id = job_status.id

# Monitor progress of the job until it is complete.
```

```

while (job_status.state == Job.BackupRestoreProcessState.INPROGRESS) :
    print( 'Backup job state: {} ({}%)'.format( job_status.state, \
                                                job_status.progress ) )

    time.sleep( 10 )
    job_status = backup_job.get( job_id )

# Report job completion.
print( 'Backup job completion status: {}'.format( job_status.state) )

```

Schedule a Backup Job

You can automate the backup process by creating a schedule that runs backup jobs at specific times.

You can keep existing backups on the backup server. The retention policy defines the maximum number of backups that the server keeps. You can also specify whether the backup job should run once, or on a recurring basis. The recurrence policy defines the days of the week and specific times at which the backup job is scheduled to run.

Prerequisites

- Verify that you can access the backup server and you have read and write permissions.
- Verify that you have established a connection to the vAPI services.

Procedure

- 1 Create a Schedules object.
- 2 Specify the retention and recurrence information.
- 3 Create a schedule by passing the backup location, user credentials to access the location, retention, and recurrence information.
- 4 Create an UpdateSpec and pass the updated information.
- 5 Get a backup schedule by passing a schedule ID.

What to do next

Run the backup job by using the schedule.

Python Example of Scheduling a Backup Job

This example shows how you can schedule a backup job, update the schedule, and get a schedule. The example is based on the `backup_schedule.py` sample.

Note For a complete and up-to-date version of the sample code, see the `vsphere-automation-sdk-python` VMware repository at GitHub.

```

...
# Connect to vAPI services
self.stub_config = vapiconnect.connect(
    host=args.server,
    user=args.username,

```

```

        pwd=args.password,
        skip_verification=args.skipverification)

self.schedule_client = Schedules(self.stub_config)
...

def create_schedule(self):
    retention_info = Schedules.RetentionInfo(self.max_count)
    recurrence_info = Schedules.RecurrenceInfo(
        days=self.days,
        hour=self.hour,
        minute=self.minute)
    create_spec = Schedules.CreateSpec(
        location=self.location,
        location_user=self.location_user,
        location_password=self.location_password,
        recurrence_info=recurrence_info,
        retention_info=retention_info)

    self.schedule_client.create(self._schedule_id, create_spec)

def update_schedule(self):
    retention_info = Schedules.RetentionInfo(self.max_count)
    recurrence_info = Schedules.RecurrenceInfo(
        days=self.days,
        hour=self.hour,
        minute=self.minute)
    update_spec = Schedules.UpdateSpec(
        location=self.location,
        location_user=self.location_user,
        location_password=self.location_password,
        recurrence_info=recurrence_info,
        retention_info=retention_info)

    self.schedule_client.update(self._schedule_id, update_spec)

def get_schedule(self):
    self.schedule_client = Schedules(self.stub_config)
    schedule_spec = self.schedule_client.get(self._schedule_id)

    recurrence_info = schedule_spec.recurrence_info
    retention_info = schedule_spec.retention_info

    table = []
    data = [self._schedule_id,
            "{}:{}".format(recurrence_info.hour, recurrence_info.minute),
            " ".join(recurrence_info.days),
            retention_info.max_count]
    table.append(data)
    headers = ["Schedule ID", "Time", "Days", "Retention"]
    print(tabulate(table, headers))

...

```

Restoring vCenter Server

The vCenter Server Management API supports restoring a vCenter Server instance from a backup copy. The API simplifies the process by unifying the handling of various components of vCenter Server in a single operation.

The process of restoring a vCenter Server instance from a backup has two phases.

- 1 Deploy a new vCenter Server instance. OVF deployment is described in the *vSphere Automation SDKs Programming Guide*.
- 2 Invoke the restore operation from the Management API to apply configuration settings and load the vCenter Server database from the backup file.

Note You cannot specify optional parts for the restore operation. The restore operation includes all optional parts, such as Events and Tasks, that were specified at the time when the backup file was created.

Authentication When Restoring a vCenter Server Instance

During the process of restoring a vCenter Server instance from a backup image, you cannot use vCenter Single Sign-On authentication. You must use local authentication until the vCenter Server instance is fully configured.

When you restore your vCenter Server instance from a backup file, it begins in an unconfigured state. During this time, you must use local authentication to access the Management API. When you use local authentication, do not use the vSphere Automation API endpoint. Instead, you must connect your client to port 5480 of the vCenter Server instance.

When you use local authentication you must pass user name and password credentials with each method invocation. Use credentials that are known to the guest operating system of the vCenter Server instance.

Availability of Services While Restoring a vCenter Server Instance

During the process of restoring the vCenter Server backup file, services in the vCenter Server instance must restart. While they are restarting, your API client receives an error message.

You can write your client to trap the error, but you have no way to know when the vCenter Server services are running again. To determine when the restore process is complete, you must retry the API connection until it succeeds, then request the status of the job.

Restore a vCenter Server Instance by Using the API

You can use the Management API of to restore a vCenter Server instance from a backup file containing the vCenter Server database and key components of the vCenter Server instance.

Prerequisites

- Verify that the backed up vCenter Server instance is powered off.

- A new vCenter Server instance must be deployed in an unconfigured state, except that it must have a fully qualified domain name or IP address that matches the old one.
- Verify that the new vCenter Server instance has the same build number as the one in the backup file.
- Verify that the new vCenter Server instance has a size equal to or greater than the old one. If the old vCenter Server instance was customized to exceed the largest template size, the new one must be customized to the same size.
- Verify that no other backup or restore jobs are running.
- Verify that the destination storage location is accessible to the vCenter Server restore process.

Procedure

- 1 Create a restore request object to describe the restore operation.
- 2 Issue a request to start the restore operation.
- 3 Monitor progress of the job until it is complete.
- 4 Report job completion.

What to do next

After the vCenter Server instance is fully configured by the restore operation, you can resume using the vSphere Automation API endpoint for subsequent operations.

Bash Example of Restoring a vCenter Server Instance

This example shows how to invoke `curl` from a bash script to restore a vCenter Server instance. This operation is the second phase of restoring the vCenter Server instance from a backup file.

The example uses local user name and password authentication because the vSphere Automation API endpoint is not yet running when you restore the vCenter Server instance. The client must connect to port 5480 for this operation.

This example depends on certain variables that specify the source and destination for the restore operation. For simplicity, the variables are hard-coded at the start of the bash script.

This example assumes the backup image is not encrypted.

```
#!/bin/bash
##### EDITABLE BY USER to specify vCenter Server instance and backup location. #####
VC_ADDRESS=vc_server_ip
VC_USER=sso_user
VC_PASSWORD=sso_pass
FTP_ADDRESS=storage_server
FTP_USER=ftpuser
FTP_PASSWORD=ftpuser
BACKUP_FOLDER=backup
BACKUP_SUBFOLDER=2016-07-08-09-10-11
#####

# Create a message body for the restore request.
cat << EOF > task.json
```

```

{ "piece":
  {
    "location_type":"FTP",
    "location":"ftp://$FTP_ADDRESS/$BACKUP_FOLDER/$BACKUP_SUBFOLDER",
    "location_user":"$FTP_USER",
    "location_password":"$FTP_PASSWORD"
  }
}
EOF

# Issue a request to start the restore operation.
TIME=$(date +%Y-%m-%d-%H-%M-%S)
echo Starting restore $TIME $VC_B64_PASS >> restore.log
curl -k -u "$VC_USER:$VC_PASSWORD" \
  -H 'Accept:application/json' \
  -H 'Content-Type:application/json' \
  -X POST \
  "https://$VC_ADDRESS:5480/rest/com/vmware/appliance/recovery/restore/job" \
  --data @task.json 2>restore.log >response.txt
cat response.txt >> restore.log
echo '' >> restore.log

# Monitor progress of the operation until it is complete.
STATE=INPROGRESS
PROGRESS=0
until [ "$STATE" != "INPROGRESS" ]
do
  echo "Restore job state: $STATE ($PROGRESS%)"
  sleep 10s
  curl -s -k -u "$VC_USER:$VC_PASSWORD" \
    -H 'Accept:application/json' \
    -H 'Content-Type:application/json' \
    -X POST -d '' \
    "https://$VC_ADDRESS:5480/rest/com/vmware/appliance/recovery/restore/job?~action=get" \
    >response.txt
  cat response.txt >> restore.log
  echo '' >> restore.log
  PROGRESS=$(awk \
    '{if (match($0,/"progress":\w+/)) print substr($0, RSTART+11, RLENGTH-11);}' \
    response.txt)
  STATE=$(awk \
    '{if (match($0,/"state":\w+/)) print substr($0, RSTART+9, RLENGTH-10);}' \
    response.txt)
done
# Report job completion and clean up temporary files.
echo ''
echo Restore job completed.
rm -f task.json
rm -f response.txt
echo '' >> restore.log

```

Python Example of Restoring a vCenter Server Instance

This example shows how to use Python to restore a vCenter Server instance. This operation is the second phase of restoring the vCenter Server instance from a backup image.

This example uses the following global variables.

- *my_vcsa_hostname*
- *my_vcsa_username*
- *my_vcsa_password*
- *my_backup_name*
- *my_storage_server*
- *my_scp_user*
- *my_scp_password*
- *my_backup_folder*

When you restore the vCenter Server instance from a backup image, you need two sets of authentication credentials. The API client needs to authenticate to the vCenter Server instance, and the vCenter Server backup service needs to authenticate to the backup storage server.

The example uses local user name and password authentication for the connection to the vCenter Server instance because the vSphere Automation API endpoint is not yet running when you restore the vCenter Server instance. The client must connect to port 5480 for this operation.

In the restore request, you need to specify the folder that contains the backup image. The folder name is the same name that was specified in the backup request. It must be specified as a path name relative to the home directory of the user that authenticates with the storage server.

This example assumes the backup image is not encrypted.

```
import requests
from vmware.vapi.lib.connect import get_requests_connector
from vmware.vapi.security.user_password import create_user_password_security_context
from vmware.vapi.stdlib.client.factories import StubConfigurationFactory
from com.vmware.appliance.recovery.restore_client import (Job)
import time

# Create a session object in the client.
session = requests.Session()

# For development environment only, suppress server certificate checking.
session.verify = False
from requests.packages.urllib3 import disable_warnings
from requests.packages.urllib3.exceptions import InsecureRequestWarning
disable_warnings(InsecureRequestWarning)

# Create a connection to port 5480.
local_url = 'https://%s:5480/api' % my_vcsa_hostname
connector = get_requests_connector(session=session, url=local_url)
```

```

# Add username/password security context to the connector.
basic_context = create_user_password_security_context(my_vcsa_username, my_vcsa_password)
connector.set_security_context(basic_context)

# Create a stub configuration by using the username–password security context.
local_stub_config = StubConfigurationFactory.new_std_configuration(connector)

# Create a restore request object.
req = Job.RestoreRequest()
req.location_type = Job.LocationType.SCP
req.location = my_storage_server + ':/home/scpuser/' + my_backup_folder + '/' + my_backup_name
req.location_user = my_scp_user
req.location_password = my_scp_password

# Issue a request to start the restore operation.
restore_job = Job( local_stub_config )
job_status = restore_job.create( req )

# Monitor progress of the job until it is complete.
while (job_status.state == Job.BackupRestoreProcessState.INPROGRESS) :
    print( 'Restore job state: {} ({}%)'.format( job_status.state,
job_status.progress ) )
    time.sleep( 10 )
    job_status = restore_job.get()

# Report job completion.
print( 'Restore job completion status: {}'.format( job_status.state) )

```

Reconcile a vCenter Server Instance with Nodes in Embedded Linked Mode

You can run the reconciliation process after you successfully restored your vCenter Server instance. By using the API or HTTP requests, you can reconcile vCenter Server nodes that work in an embedded linked mode and are connected in a ring or daisy-chain.

Reconciliation is a post-restore process that checks whether the vCenter Server partners in embedded linked mode are available, synchronizes the vCenter Server data and services with the partners, and runs the vCenter Server services. The processes of restore and reconciliation depend on the topology and if there are changes in the topology between the backup and restore, you cannot restore the embedded linked mode. If the replication partners are not available and you try to restore the first node, you must ignore the warnings. In this case, any changes that are made in the topology or infrastructure after the backup will be lost. If you restore a node different from the first one, you must add it to the domain of the first node. If you use a daisy-chain topology, you must first restore the first node, and after that to restore the second, link it to the first one, and apply the same to the following nodes.

You can use the reconciliation API after a file-based and an image-based restore. After an image-based restore, you can run the reconciliation process by using the API or UI. After a file-based restore, you can monitor the reconciliation process by using the GET <https://<server>:5480/rest/appliance/recovery/reconciliation/job> HTTP request. For information about how to restore a vCenter Server instance from an image or a file by using the UI, see the *vCenter Server Installation and Setup* documentation.

Prerequisites

- Verify that you successfully restored your node from an image.
- Verify that the replication partners are available.
- Verify that you restored your nodes in the correct order, if you use a daisy-chain topology.
- Verify that you have administrator's credentials to your Single Sign-On domain.
- Verify that there is no running or failed reconciliation job.

Procedure

- 1 Create a `Job.CreateSpec` object, specify user name and password of Single Sign-On administrator, and set the `ignore_warnings` field to `true`.

The default value of `ignore_warnings` is `false`. If you do not set `ignore_warnings` to `true`, the reconciliation fails due to the validation warnings.

- 2 Run a reconciliation job by using the `create(Job.CreateSpec)` method.

You can check the result of the operation by reading the `Job.Info` object. `Job.Info` contains information about the job such as description, status, progress, error, start and end time.

- 3 Get the status of the job by calling the `get()` method.

The possible states are `NONE`, `RUNNING`, `FAILED`, and `SUCCEEDED`.

Managing System Logs

You can automate the forwarding of vCenter Server system log messages to remote logging servers by using the vCenter Server Management API.

You can configure the syslog forwarding by using the API or user interface. For information about how to manage the syslog by using the user interface, see the *vSphere Monitoring and Performance* documentation.

Configuring Syslog Forwarding

You can use the vCenter Server Management API or HTTP requests to configure the forwarding of vCenter Server syslog messages and test the connection between the vCenter Server instance and remote servers.

[Table 7-1. User Operations](#) lists operations that you can perform to manage the forwarding of syslog messages to remote logging servers.

Table 7-1. User Operations

Operation	Description
Get forwarding configuration	You can retrieve information about the log forwarding configuration. See Figure 7-1. Forwarding Class Diagram for Python and HTTP Requests for Configuring Syslog Forwarding .
Test forwarding configuration	You can validate the current log forwarding configuration. Optionally, you can send a test diagnostic log message from the vCenter Server instance to all configured logging servers to allow manual end-to-end validation. See Figure 7-1. Forwarding Class Diagram for Python and HTTP Requests for Configuring Syslog Forwarding .
Set forwarding configuration	You can change the log forwarding configuration. See Figure 7-1. Forwarding Class Diagram for Python and HTTP Requests for Configuring Syslog Forwarding .

The forwarding configuration includes the IP or FQDN of the remote server, the remote port for receiving syslog information, and the communication protocol. The remote server must be a server with running `rsyslog`, for example, another vCenter Server instance. The API supports the TCP, UDP, TLS, and RELP protocols. For information about the supported TLS versions, see KB article [2147469](#). By creating a `Forwarding.Config` object, you specify the connection with a remote server. For information about the Forwarding class and its methods, see the *API Reference* documentation, [Figure 7-1. Forwarding Class Diagram for Python](#), and [Figure 7-2. Example Configuration Workflow](#).

You can use several remote servers by creating a list with `Forwarding.Config` objects and passing it to the `set` method. The maximum number of remote servers is three. You can validate the forwarding configuration by using the `test` method. The returned `Forwarding.ConnectionStatus` object shows the status of the connection between the vCenter Server instance and a remote server. The `State` enumeration shows whether the vCenter Server instance can reach the remote server. `State` can be `UP`, `DOWN`, or `UNKNOWN`. If the state is `DOWN` or `UNKNOWN`, the vCenter Server instance cannot access the remote server and you must check the remote server and its settings such as network ports, firewall, supported protocols, and syslog configuration.

Note If you use UDP, the connection status is always `UNKNOWN`.

Figure 7-1. Forwarding Class Diagram for Python

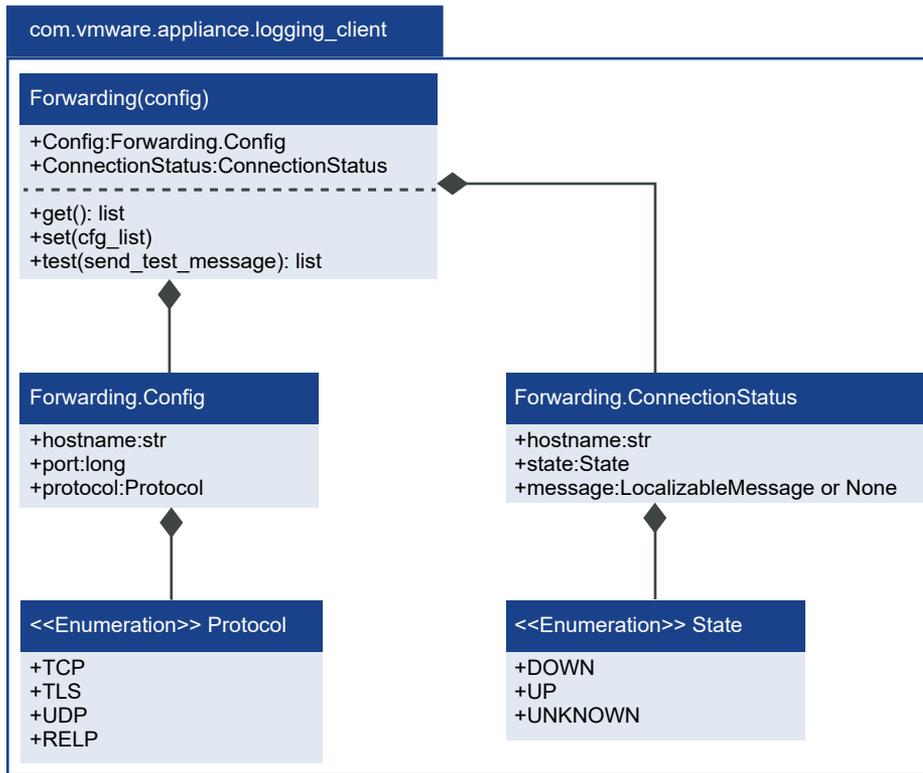
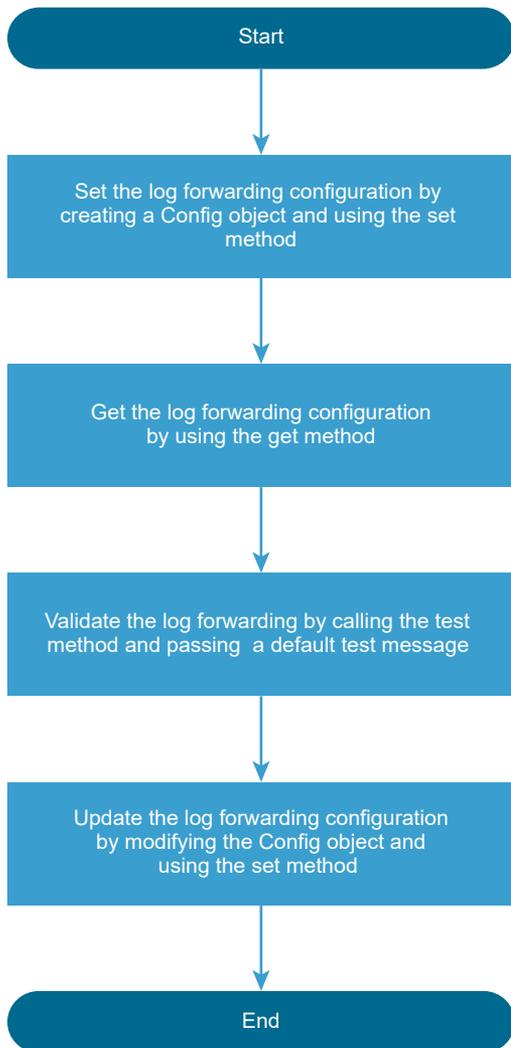


Figure 7-2. Example Configuration Workflow

For a code example of configuring the syslog forwarding, see [Python Example of Configuring Syslog Forwarding](#).

HTTP Requests for Configuring Syslog Forwarding

By using the API or HTTP requests, you can set and get the forwarding configuration, check the connection with the remote server or servers, and exchange test messages with them.

The following HTTP requests show the syntax that you can use to perform the available user operations.

Note When you send the requests, you must authenticate with vCenter Server **root** credentials.

```
GET https://<server>:5480/rest/appliance/logging/forwarding
```

```
POST https://<server>:5480/rest/appliance/logging/forwarding?action=test
```

```
PUT https://<server>:5480/rest/appliance/logging/forwarding
```

For information about the body of each HTTP request, see the *REST API Reference* documentation.

Table 7-2. HTTP Status Codes

HTTP Status Code	Description	Operations That Return the Status Code
200	The operation is successful.	All available operations. You can check the returned data in the results data structure.
400	You use an invalid argument. For example, a protocol that it is not supported, invalid port number, or the number of configurations is greater than 3.	Set forwarding configuration.
401	You use invalid user name or password, or authentication is failed.	All available operations.
500	There is a <code>vapi.std</code> error. For information about the types of <code>vapi.std</code> errors, see the <code>vapi.std.errors</code> API in the <i>API reference</i> documentation.	Set forwarding configuration.

Python Example of Configuring Syslog Forwarding

This example shows how you can configure and test the forwarding of a vCenter Server syslog by using the API. The example is based on the `log_forwarding.py` sample.

Note For a complete and up-to-date version of the sample code, see the `vsphere-automation-sdk-python` VMware repository at GitHub.

```
...
    self.log_forwarding_client = Forwarding(self.stub_config)
...

def set_log_forwarding(self):
    log_forwarding_config = [Forwarding.Config(hostname=self.loghost,
                                              port=self.port,
                                              protocol=self.protocol)]
    self.log_forwarding_client.set(log_forwarding_config)
```

```

def get_log_forwarding(self):
    configs = self.log_forwarding_client.get()

    print("\nLog forwarding configurations:")
    table = [[cfg.hostname, cfg.port, cfg.protocol] for cfg in configs]
    headers = ["Loghost", "Port", "Protocol"]
    print(tabulate(table, headers))

def test_log_forwarding(self):
    test_response = self.log_forwarding_client.test(True)

    print("\nLog forwarding test response:")
    table = [[resp.hostname,
              resp.state,
              resp.message.default_message if resp.message else None]
             for resp in test_response]
    headers = ["Loghost", "State", "Message"]
    print(tabulate(table, headers))

def update_log_forwarding(self):
    # Read log forwarding configuration
    log_forwarding_config = self.log_forwarding_client.get()

    # Delete the newly added configuration
    log_forwarding_config = list(filter(
        lambda cfg: cfg.hostname != self.loghost,
        log_forwarding_config))

    # Apply the modified log forwarding configuration
    self.log_forwarding_client.set(log_forwarding_config)

...

```

Performing Infrastructure Profile Management Operations

You can export an existing vCenter Server configuration and import it to other vCenter Server instances.

You can export multiple configuration profiles at once. The exported data can contain general configuration settings and user content. You can replicate the same configuration across all vCenter Server instances in your environment by importing the same data package. You can also use the exported data as a backup if you need to revert to the last known good configuration. To avoid configuration issues, you can validate the exported data before importing it to a vCenter Server instance.

The following table lists the operations that you can perform to manage the configuration profiles in your infrastructure.

Table 7-3. User Operations

Operation	Description
List configuration profiles	You can retrieve a list of all configuration profiles that are registered with vCenter Server.
Export configuration profiles	You can export specific vCenter Server configuration profiles.

Table 7-3. User Operations (continued)

Operation	Description
Validate configuration profiles	You can validate the exported vCenter Server configuration profiles. The validation process examines the configuration file for possible errors and conflicts and returns output. This operation can help avoid configuration issues or loading the wrong configuration file.
Import configuration profiles	You can import specific vCenter Server configuration profiles into another vCenter Server instance.

You can run infrastructure profile management operations by using the API or sending an HTTP request. For information about the HTTP requests that you can use to perform the user operations, see [HTTP Requests for Infrastructure Profile Management Operations](#).

HTTP Requests for Infrastructure Profile Management Operations

You can use HTTP requests to perform infrastructure profile management operations such as exporting, validating, and importing vCenter Server configuration profiles.

The following HTTP requests show the syntax that you can use to perform the available user operations.

- List configuration profiles

```
GET https://<server>:443/api/appliance/infraprofile/configs
```

- Export configuration profiles

```
POST https://<server>:443/api/appliance/infraprofile/configs?action=export
```

- Validate configuration profiles

```
POST https://<server>:443/api/appliance/infraprofile/configs?vmw-task=true&action=validate
```

- Import configuration profiles

```
POST https://<server>:443/api/appliance/infraprofile/configs?vmw-task=true&action=import
```

For information about the content and syntax of the HTTP request body for each operation, see the *API reference* documentation.

Planning vCenter Server Updates



The vCenter Server API provides operations that can help you plan the life cycle of vCenter Server instances in your environment.

You can use the operations to discover VMware products that can be associated with vCenter Server, list associated products, and manage product associations. You can also get details about available vCenter Server updates, perform pre-checks, and produce reports. The reports can contain interoperability or pre-check information. Interoperability reports contain information about the interoperability between the associated products and a specific vCenter Server version. Pre-check reports contain information about the compatibility of the current vCenter Server version with a pending update version. You can plan to perform vCenter Server updates based on the information gathered in the produced reports.

This chapter includes the following topics:

- [Performing Discovery and Planning Operations](#)
- [List Available Products and Manage Associated Products](#)
- [List Available Updates](#)
- [Retrieve a Report](#)

Performing Discovery and Planning Operations

You can retrieve information about VMware products associated with vCenter Server, list available vCenter Server updates, and produce reports. The findings can help you plan vCenter Server updates in your environment.

The life cycle management API provides operations that are grouped in the discovery, update, and reports categories. The discovery functionality of the API consists of the operations in the discovery category. The planning functionality of the API consists of the operations in the update category. Both functionalities can produce reports that you can download by using the reports category.

The discovery category provides operations for listing VMware products that can be associated with vCenter Server, managing products associations, and creating interoperability reports. The update category provides operations for listing all available updates and upgrades for vCenter Server and generating a pre-check compatibility report. The reports category provides an operation for downloading reports generated by interoperability and pre-check operations. By using the retrieved information, you can select one of the available patches and can plan an actual patch or upgrade for a specific vCenter Server version.

The following table lists the operations that are available in the discovery category.

Table 8-1. Discovery User Operations

Operation	Description
Get product catalog	You can retrieve a list of all VMware products that can be associated with vCenter Server.
List associated products	You can retrieve a list of all VMware product deployments in the environment that are associated with vCenter Server. Note The list contains both product deployments discovered automatically and deployments registered manually through the API.
Get associated product information	You can retrieve detailed information about a product associated with vCenter Server.
Create product association	You can manually associate a VMware product with vCenter Server.
Update product association	You can modify a manually added VMware product that is associated with vCenter Server. Note You cannot modify VMware products that are discovered automatically.
Delete product association	You can delete or dissociate a manually added VMware product that is associated with vCenter Server. Note You cannot delete or dissociate VMware products that are discovered automatically.
Create interoperability report	You can create an interoperability report between a vCenter Server release version and all products registered with the vCenter Server instance.

The following table lists the operations that are available in the update category.

Table 8-2. Update User Operations

Operation	Description
List updates	You can retrieve a list of all available vCenter Server updates. The list can contain minor, in-place, updates and major, migration-based, upgrades.
Get update info	You can retrieve detailed vCenter Server information about a specific update or upgrade.
Create pre-check report	You can create a vCenter Server pre-update compatibility check report for a pending update version. Note You can export and download the report in CSV format.

The following table lists the operations that are available in the reports category.

Table 8-3. Reports User Operations

Operation	Description
Get report	You can download the report generated by the interoperability and pre-check operations. For information about downloading the report, see the <i>API reference</i> documentation.

You can run life cycle management operations by using the API or sending an HTTP request. For information about the HTTP requests that you can use to perform the user operations, see [HTTP Requests for Discovery and Planning Operations](#). In addition to sending HTTP requests, you can also run cURL commands to perform operations. See [cURL Examples of Performing Discovery and Planning Operations](#).

HTTP Requests for Discovery and Planning Operations

You can use HTTP requests to perform discovery and planning operations and download reports.

The following HTTP requests show the syntax that you can use to perform the available user operations.

Note The default port for sending HTTP requests is 443. If the vCenter Server instance is configured to use a custom port, you must replace 443 with the custom port number when sending HTTP requests.

Discovery Operations

- Get product catalog

```
GET https://<server>:443/api/vcenter/lcm/discovery/product-catalog
```

- List associated products

```
GET https://<server>:443/api/vcenter/lcm/discovery/associated-products
```

- Get associated product information

```
GET https://<server>:443/api/vcenter/lcm/discovery/associated-products/<product>
```

- Create product association

```
POST https://<server>:443/api/vcenter/lcm/discovery/associated-products
```

- Update product association

```
PATCH https://<server>:443/api/vcenter/lcm/discovery/associated-products/<product>
```

- Delete product association

```
DELETE https://<server>:443/api/vcenter/lcm/discovery/associated-products/<product>
```

- Create interoperability report

```
POST https://<server>:443/api/vcenter/lcm/discovery/interop-report
```

Update Operations

- List updates

```
GET https://<server>:443/api/vcenter/lcm/update/pending
```

- Get update info

```
GET https://<server>:443/api/vcenter/lcm/update/pending/<version>
```

- Create pre-check report

```
POST https://<server>:443/api/vcenter/lcm/update/pending/<version>/precheck-report
```

Reports Operations

- Get report

```
GET https://<server>:443/api/vcenter/lcm/reports/<report>
```

For information about the content and syntax of the HTTP request body for each operation, see the *API reference* documentation.

cURL Examples of Performing Discovery and Planning Operations

The following cURL command examples show the syntax for performing life cycle operations such as discovering and managing product associations, listing updates, performing pre-checks, and retrieving reports.

Example: List Product Catalog

This example lists all compatible versions of VMware products that can be associated with vCenter Server.

```
curl -X GET -u administrator@vsphere.local:<password> 'https://<server>:443/api/vcenter/lcm/discovery/product-catalog'
```

Example: List Products Registered with vCenter Server

This example lists all VMware products associated with vCenter Server including manually added products.

```
curl -X GET -u administrator@vsphere.local:<password> 'https://<server>:443/api/vcenter/lcm/discovery/associated-products'
```

Example: Add a New Product

This example associates a VMware product with vCenter Server.

```
curl -H "Content-Type: application/json" -X POST -d '{"name" : "vRealize Automation", "version": "6.8.9"}' -u administrator@vsphere.local:<password> 'https://<server>:443/api/vcenter/lcm/discovery/associated-products' -k -i
```

Example: Get Product Details

This example retrieves details about a product associated with vCenter Server.

```
curl -X GET -u administrator@vsphere.local:<password> 'https://<server>:443/api/vcenter/lcm/discovery/products/com.vmware.vRA-6.8.9'
```

Example: Modify a Product

This example updates a VMware product associated with vCenter Server. You can modify only manually added products.

```
curl -H "Content-Type: application/json" -X PATCH -u administrator@vsphere.local:<password> 'https://<server>:443/api/vcenter/lcm/discovery/associated-products/com.vmware.vRealOrche_7.3.1' -k -d '{"spec":{"deployments": ["<ip_address>"]}]}'
```

Example: Delete a Product

This example delete or dissociates a VMware product associated with vCenter Server. You can delete or dissociate only manually added products.

```
curl -X DELETE -u administrator@vsphere.local:<password> 'https://<server>:443/api/vcenter/lcm/discovery/products/com.vmware.vRA-6.8.9'
```

Example: Create an Interoperability Report

This example generates an interoperability report for all VMware products associated with vCenter Server against a specific vCenter Server version.

```
curl -X POST -u administrator@vsphere.local:<password> 'https://<server>:443/api/vcenter/lcm/discovery/interop-report?vmw-task=true' -k -d '{"spec":{"target_version":"6.7.0.2000"}}' -H "Content-Type: application/json"
```

Example: List Available Updates

This example lists all available and applicable updates for vCenter Server versions discovered in your environment. The list can contain minor, in-place, updates and major, migration-based, upgrades. The operation calculates which updates or upgrades are applicable based on the current vCenter Server version.

```
curl -X GET -u administrator@vsphere.local:<password> 'https://<server>:443/api/vcenter/lcm/update/pending'
```

Example: Get Update Details

This example retrieves details about a specific vCenter Server update. The update is identified by the ID provided in the URL. The update can be either a minor, in-place, update or a major, migration-based, upgrade.

```
curl -X GET -u administrator@vsphere.local:<password> 'https://<server>:443/api/vcenter/lcm/update/pending/7.0.0.20000'
```

Example: Run Update Pre-Checks

This example performs source update pre-checks and identifies whether the provided update ID corresponds to a minor update or a major upgrade. In case of a minor update, the operation invokes the appropriate update API. In case of a major upgrade, the operation downloads and installs the `requirements.rpm` file for the upgrade. The pre-check can be a long-running task, which you can monitor by using the task ID that the operation returns.

```
curl -X POST -u administrator@vsphere.local:<password> 'https://<server>:443/api/vcenter/lcm/update/pending/7.0.0.50000/precheck-report?vmw-task=true' -k
```

Example: Monitor the Pre-Check Task

This example tracks the status, progress, and retrieves the final result of the pre-check operation.

```
curl -X GET -u administrator@vsphere.local:<password> 'https://<server>:5480/rest/cis/tasks/3f77223f-1edb-4992-9c48-2d75d7b3b91d:com.vmware.vcenter.lcm.update.precheck_report' -k
```

Example: Get Report

This example retrieves information about the location of the actual report for downloading.

```
curl -X GET -u administrator@vsphere.local:<password> 'https://<server>:443/api/vcenter/lcm/reports/abcd123_report.csv'
```

List Available Products and Manage Associated Products

You can automate the management of VMware products associated with vCenter Server by using the API.

This procedure includes the operations that you can use to manage the product catalog and associated products.

Prerequisites

- Verify that you have an active authenticated session with vCenter Server.

Procedure

- 1 Create a stub configuration.
- 2 Retrieve the product catalog.
- 3 Retrieve a list of VMware products associated with vCenter Server.
- 4 Associate a VMware product with vCenter Server.
- 5 Update a VMware product associated with vCenter Server.
- 6 Delete a product from the list of VMware products associated with vCenter Server.

Python Example of Listing Available Products and Managing Associated Products

This example shows how you can retrieve the product catalog, list the associated products, add, update, and delete products associations. The example is based on the `discovery_sample.py` sample.

Note For a complete and up-to-date version of the sample code, see the `vsphere-automation-sdk-python` VMware repository at GitHub.

```

...
    # Create a stub configuration
    stub_config = StubConfigurationFactory.new_std_configuration(connector)
    self.product_client = ProductCatalog(stub_config)
    self.associated_products_client = AssociatedProducts(stub_config)

def run(self):
    # Product catalog
    product_catalog = self.product_client.list()
    print("Product catalog list: \n", product_catalog)

    # Associated products
    associated_products = self.associated_products_client.list()
    print("Associated products list : \n", associated_products)

    # Add product
    spec = {'product_name': 'VMware Identity Manager', 'version': '3.3', 'deployments': '3'}
    add_associated_product = self.associated_products_client.create(spec)
    print('Added new product. \n', add_associated_product)

    associated_products = self.associated_products_client.list()
    print("Associated products after adding the product: \n", associated_products)

    # Update product
    update_spec = {'deployments': '9'}
    update_associated_product = self.associated_products_client.update(add_associated_product,
update_spec)
    associated_products = self.associated_products_client.list()
    print("Associated products after updating the product: \n", associated_products)

    # Delete product
    delete_associated_product = self.associated_products_client.delete(add_associated_product)
    associated_products = self.associated_products_client.list()
    print("Associated products after deleting the product: \n{0}", associated_products)
...

```

List Available Updates

You can retrieve a list of available vCenter Server updates, details about the updates, and pre-check information by using the API.

Prerequisites

- Verify that you have an active authenticated session with vCenter Server.

Procedure

- 1 Create a stub configuration.
- 2 Retrieve a list of available vCenter Server updates.

If there are available updates, you can retrieve details about the updates.

- 3 Retrieve pre-check information.

Python Example of Listing Available Updates

This example shows how you can retrieve a list of available vCenter Server updates, details about the updates, and pre-check information. The example is based on the `update_sample.py` sample.

Note For a complete and up-to-date version of the sample code, see the `vsphere-automation-sdk-python` VMware repository at GitHub.

```
...
    # Create a stub configuration
    stub_config = StubConfigurationFactory.new_std_configuration(connector)
    self.pending_client = Pending(stub_config)
    self.precheck_client = PrecheckReport(stub_config)

def run(self):
    # List updates
    available_updates = self.pending_client.list()
    print("vCenter Server available updates - ", available_updates)
    if available_updates.updates:
        target_version = available_updates.updates[0].version
        update_details = self.pending_client.get(target_version)
        print("vCenter Server available update details - ", update_details)

        # Get pre-check result
        precheck_result = self.precheck_client.create_task(target_version)
        print("Pre-upgrade checks task id started with:
\n{0}").format(precheck_result.get_task_id())
    ...
```

Retrieve a Report

You can retrieve a report generated by the interoperability and pre-check operations by using the API.

Prerequisites

- Verify that you have an active authenticated session with vCenter Server.

Procedure

- 1 Create a stub configuration.
- 2 Retrieve the report details.

Python Example of Retrieving a Report

This example shows how you can retrieve a report. The example is based on the `lcm_sample.py` sample.

Note For a complete and up-to-date version of the sample code, see the `vsphere-automation-sdk-python` VMware repository at GitHub.

```
...
    # Create a stub configuration
    stub_config = StubConfigurationFactory.new_std_configuration(connector)
    self.report_client = Reports(stub_config)

    def run(self):
        # Retrieve report
        report_details = self.report_client.retrieve('com.vmware.vcenter.lcm.report')
        print("Report details - ", report_details)
...

```

Updating vCenter Server

9

vCenter Server provides interfaces to perform software updates.

Before applying updates, you must make sure that your environment is prepared for the vCenter Server software update process.

This chapter includes the following topics:

- [Applying vCenter Server Software Updates](#)

Applying vCenter Server Software Updates

You can automate the installation of vCenter Server software updates to ensure that your system is stable and protected. Software updates can include security fixes, performance optimizations, and new features.

Security patches usually address vulnerabilities in third-party components and do not affect the vCenter Server functionality. vCenter Server bug fixes can introduce changes to the functionality without affecting the data format or database schema of the system.

Each update contains metadata with information about the updated content, for example, whether high-priority OS updates are included. The update metadata includes a list of components to be updated, the release date of the update, a list of fixed issues, time and disk space requirements, and information whether a reboot is required. The metadata can also contain a new vCenter Server version number, including a build number. In addition to the metadata, an update can contain optional components such as update scripts, new versions of vCenter Server software components, and new versions of OS components.

vCenter Server can obtain software updates from either a URL or an ISO image. The URL can either point to the VMware Web repository or to a custom repository in which you upload the updates in ZIP format. To perform an update by using an ISO image, attach the image to the CD/DVD drive of the vCenter Server instance.

There are multiple phases of the update process. For details, see [vCenter Server Software Update Workflow](#).

If you want to prevent issues related to the possibility of update installation failures, you should create a backup or take a snapshot of your vCenter Server instance before you start the update process. A backup can also be useful when an update is successfully installed. For example, you might decide to revert to the previous version if you encounter any undesired system behavior related to functional changes in the new software version.

Table 9-1. User Operations

Operation	Description
Get state information	You can retrieve information about the update state.
Check for update	You can check whether a new update is available.
Get update information	You can retrieve information about the available updates.
Get update requirements	You can retrieve information about the update requirements.
Stage	You can initiate the download of the update. Note The check phase must have completed successfully before you can stage the update.
Get staging status	You can retrieve information about the status of the stage operation. Note You must provide the task ID value that you received as a response when you initiated the stage operation.
Install	You can initiate the installation of the update. Note The update must be staged before you can install it.
Get installation status	You can retrieve information about the status of the install operation. Note You must provide the task ID value that you received as a response when you initiated the install operation.
Stage and install	You can initiate the download of the update and the installation starts when the download completes.

You can run software update operations by using the API or sending an HTTP request.

Note When you send the requests, you must use an authentication.

If you send requests to port 5480, you must authenticate with vCenter Server **root** credentials. If you send requests to the vCenter Server reverse proxy port, you must authenticate with vCenter Single Sign-On credentials.

The following HTTP requests show the syntax that you can use to perform the available user operations.

```
https://<server>:5480/rest/appliance/update
```

```
https://<server>:5480/rest/appliance/update/pending?source_type=DEFAULT
```

```
https://<server>:5480/rest/appliance/update/pending/<target_version>
```

```
https://<server>:5480/rest/appliance/update/pending/<target_version>/requirements
```

```
https://<server>:5480/rest/appliance/update/pending/<target_version>?action=stage
```

```
https://<server>:5480/rest/appliance/task/<update_stage_task_id>
```

```
https://<server>:5480/rest/appliance/update/pending/<target_version>?action=install
```

```
https://<server>:5480/rest/appliance/task/<update_install_task_id>
```

```
https://<server>:5480/rest/appliance/update/pending/<target_version>?action=stage-and-install
```

```
https://<server>/rest/appliance/update
```

```
https://<server>/rest/appliance/update/pending?source_type=DEFAULT
```

```
https://<server>/rest/appliance/update/pending/<target_version>
```

```
https://<server>/rest/appliance/update/pending/<target_version>/requirements
```

```
https://<server>/rest/appliance/update/pending/<target_version>?action=stage
```

```
https://<server>/rest/appliance/task/<update_stage_task_id>
```

```
https://<server>/rest/appliance/update/pending/<target_version>?action=install
```

```
https://<server>/rest/appliance/task/<update_install_task_id>
```

```
https://<server>/rest/appliance/update/pending/<target_version>?action=stage-and-install
```

In addition to sending HTTP requests, you can also run cURL commands to perform update operations. See [cURL Examples of Performing vCenter Server Software Update Operations](#).

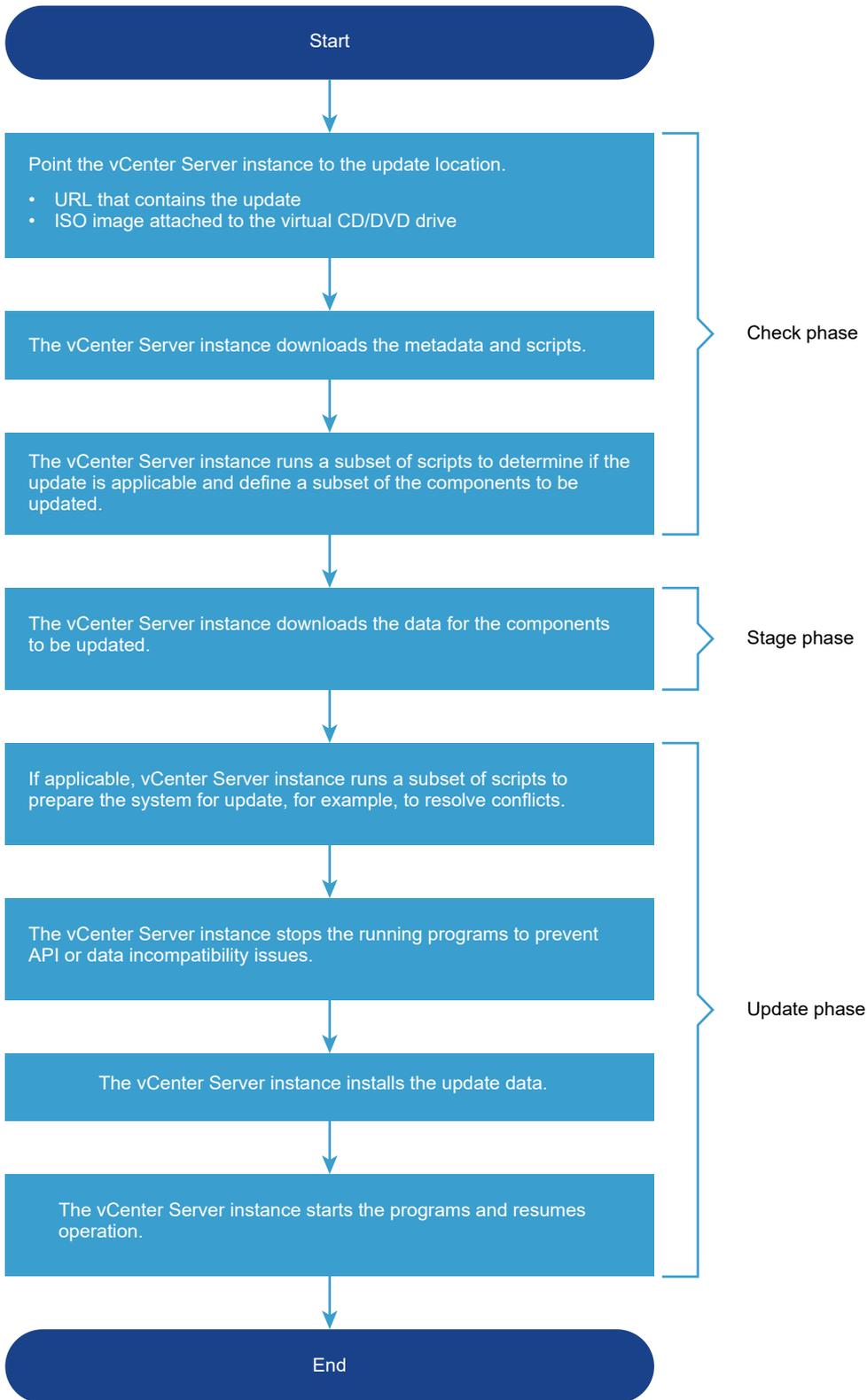
vCenter Server Software Update Workflow

The vCenter Server software update process consists of three major phases. In the first phase, the vCenter Server instance performs various checks, in the second phase it stages the update, and applies the update in the final phase.

To initiate the update process, you must choose whether the vCenter Server instance should obtain software updates from a URL or an ISO image. If you use an ISO image to update the vCenter Server instance, the image must remain attached to the CD/DVD drive of the instance during the stage and install operations.

The workflow in [Figure 9-1. Update Process Workflow](#) describes the standard steps of the update process.

Figure 9-1. Update Process Workflow



You can automate checks for new updates and staging of updates by using an update policy. For example, you can set an update policy to make the vCenter Server instance perform automatic checks for new updates at midnight every day. If there are new updates available, the vCenter Server instance can stage them automatically. Using an update policy reduces the waiting time by automating the first two phases and giving you the option to initiate only the update phase manually.

cURL Examples of Performing vCenter Server Software Update Operations

The following cURL command examples show the syntax for performing update operations such as checking for, staging, and installing updates, as well as retrieving information about update status, and setting update policies.

Example: Check for an Update

This example queries a custom URL for a new update.

```
curl -X GET -k -u root:<root_password> "https://<server>:5480/rest/appliance/update/pending?source_type=LOCAL_AND_ONLINE&url=https://<custom_url>"
```

Example: Stage an Update

This example initiates the staging of the update.

```
curl -X POST -k -u root:<root_password> -H "Content-Type: application/json" -d '{"version": "<target_version>"}' https://<server>:5480/rest/appliance/update/pending/<target_version>?action=stage
```

Example: Install an Update

This example initiates the installation of the update.

```
curl -X POST -k -u root:<root_password> -H "Content-Type: application/json" -d '{"version": "<target_version>", "user_data": [{"key": "vmdir.password", "value": "<ssso_password>"}]}' https://<server>/rest/appliance/update/pending/<target_version>?action=install
```

Example: Stage and Install an Update

This example downloads the update and installs it when the download completes.

```
curl -X POST -k -u root:<root_password> -H "Content-Type: application/json" -d '{"version": "<target_version>", "user_data": [{"key": "vmdir.password", "value": "<ssso_password>"}]}' https://<server>/rest/appliance/update/pending/<target_version>?action=stage-and-install
```

Example: Retrieve Update Status

This example retrieves information about the update state.

```
curl -X GET -k -u root:<root_password> https://<server>:5480/rest/appliance/update
```

Example: Set an Update Policy

This example sets an update policy to check a custom URL for new updates at specific times every Friday and Saturday, and if a new update is available, it is staged automatically.

```
curl -X PUT -k -u root:<root_password> -H "Content-Type: application/json" -d '{"policy": {"auto_stage": true, "check_schedule": [{"day": "FRIDAY", "hour": 23, "minute": 30}, {"day": "SATURDAY", "hour": 12, "minute": 30}], "custom_URL": "https://123.com"}}'
```