VMware vSphere Automation SDKs Programming Guide

Update 2
Modified on 21 APR 2021
VMware vSphere 7.0
VMware ESXi 7.0
vCenter Server 7.0

vmware®

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

https://docs.vmware.com/

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

Contents

About vSphere Automation SDKs Programming Guide

VMware vSphere Automation SDKs Programming Guide provides information about how to use the VMware vSphere® Automation™ SDK to automate different vSphere management tasks.

At VMware, we value inclusion. To foster this principle within our customer, partner, and internal community, we have updated this guide to remove instances of non-inclusive language.

Intended Audience

This manual is intended for anyone who wants to develop applications for accessing and using vSphere features such as virtual machine management, tagging, content libraries, managing the life cycle of clusters with the vSphere Lifecycle Manager, vSphere with Tanzu, and so on. The information is written for developers who have understanding of the targeted vSphere features and some experience with Java and Python programming languages, or the RESTful programming conventions.

Updated Information

This *VMware vSphere Automation SDKs Programming Guide* is updated with each release of the product or when necessary.

This table provides the update history of the *VMware vSphere Automation SDKs Programming Guide*.

Revision	Description
7 OCT 2021	■ Added information about VM deployment in vSphere with Tanzu. See Virtual Machines in vSphere with Tanzu. ■ Added information about self-service namespace templates. See Self-Service Namespace Management.
21 APR 2021	■ Added Virtual Machine Guest Operations. ■ Added Chapter 12 Configuring and Managing vSphere Native Key Provider.
23 MAR 2021	Initial release.

Introduction to the vSphere Automation SDKs

1

The vSphere Automation SDKs bundle client libraries for accessing new vSphere features such as using content libraries and existing features such as virtual machine configuration and management, vSphere tags and attributes, and so on.

The vSphere Automation SDKs contain sample applications and API reference documentation for the vSphere services that are accessible via the vSphere Automation API endpoint. The vSphere Automation SDKs also provide sample code that demonstrates how you can establish a secure connection with the vSphere Automation API endpoint and access the available vSphere services for working with vSphere objects.

vSphere Automation SDKs support two programming languages for developing client applications for managing components in your virtual environment by accessing the vSphere Automation API services.

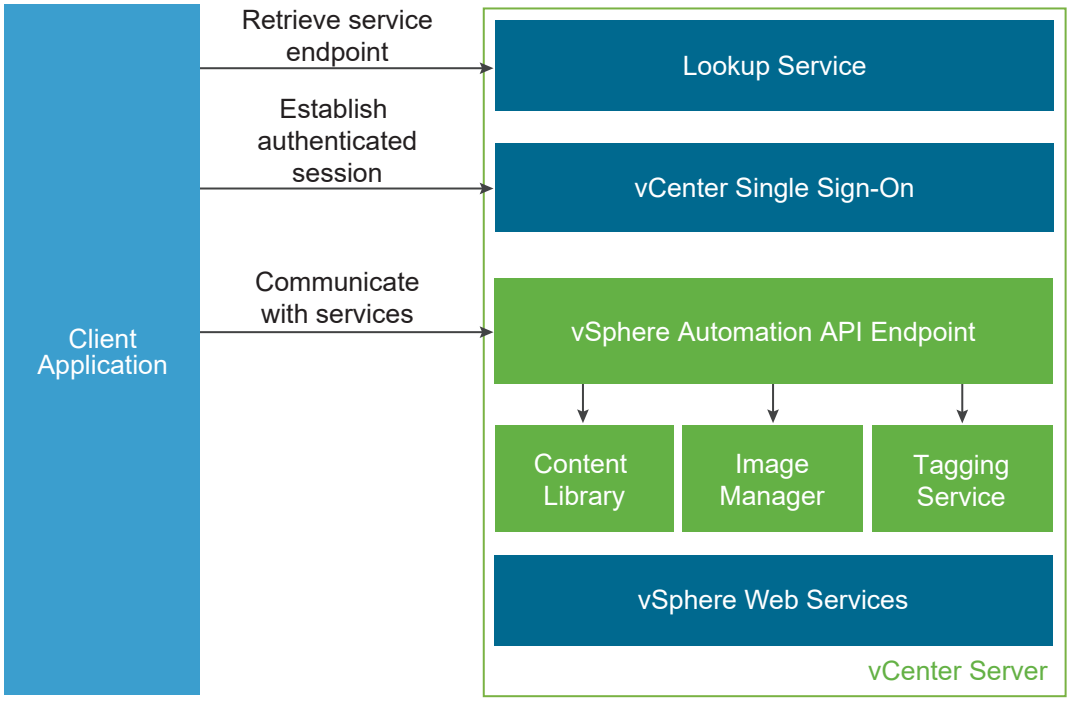
This chapter includes the following topics:

vSphere Automation SDK Overview

The vSphere Automation API provides a unified programming interface to vSphere Automation services that you can use through the SDKs provided for two programming languages. The vSphere Automation API provides a service-oriented architecture for accessing resources in the virtual environment by issuing requests to the vSphere Automation API endpoint.

Client applications use the vSphere Automation API to communicate with infrastructure and management vCenter Server services.

Figure 1-1. Communication Model Between a Client Application and the vSphere Automation API



Client applications use the Lookup Service to retrieve the vCenter Single Sign-On endpoint, the vSphere Automation API endpoint, and the endpoints of services that are exposed through the VMware vSphere® API. To access vSphere Automation services such as those for managing content libraries and vSphere tags, client applications send requests to the vSphere Automation API endpoint. By using the vCenter Single Sign-On service, client applications can either establish an authenticated vSphere Automation session, or authenticate individual requests to the vSphere Automation API endpoint.

Client applications can access services that are exposed through the vSphere Web Services API by using the VMware vSphere® Management SDK.

SDK Developer Setup

To start developing client applications that use the vSphere Automation API, you must download the desired SDK from the VMware GitHub Repository or from the respective vSphere Automation SDK landing page at https://developer.vmware.com and set up your development environment. You can find instructions for setting up your development environment in the README files contained in each vSphere Automation SDK.

API Explorer

The API Explorer is a VMware utility similar to the Managed Object Browser (MOB). Access the API Explorer at `https://<vcenter_ip_address_or_fqdn>/apiexplorer` or from the **API Explorer** tab of the **Developer Center** pane in vSphere Client. Use the utility to connect to the vSphere Automation API endpoint, browse through the available vSphere REST APIs, and make calls to your vCenter Server instance.

SDK Samples

The vSphere Automation SDKs provide code samples that demonstrate how the vSphere Automation API work and how they interoperate with the vSphere Web Services API. The code snippets included in this documentation are based on the samples in the vSphere Automation SDKs.

Supported Programming Languages

The vSphere Automation SDKs provide bindings for two different programming languages that let you build client applications on your preferred programming language.

- VMware vSphere Automation SDK for Java
- VMware vSphere Automation SDK for Python

Components of the vSphere Automation Virtualization Layer

2

At the core of vSphere Automation is vSphere, which provides the virtualization layer of the software-defined data center. You can use vSphere deployment options for vCenter Server and ESXi hosts to build virtual environments of different scales.

This chapter includes the following topics:

Components and Services of the vSphere Environment

Starting with vSphere 7.0, the installation and setup of vSphere is simplified to the deployment and upgrade of vCenter Server. vCenter Server is a preconfigured virtual machine optimized for running the vCenter Server service and the vCenter Server components. vCenter Server service acts as a central administrator for ESXi hosts.

Authentication Services Installed with vCenter Server

The vCenter Server group of authentication services includes vCenter Single Sign-On, License Service, Lookup Service, and VMware Certificate Authority. The services installed with the vCenter Server appliance are common to the entire virtual environment. A vCenter Server can be connected to one or more vCenter Server instances. In a deployment that consists of more than one vCenter Server, the data of each service is replicated across all vCenter Server instances.

In the client applications that use the vSphere Automation API, you use the vCenter Single Sign-On and the Lookup Service on the vCenter Server to provide a range of functionality.

Authentication and Session Management

You use the vCenter Single Sign-On service to establish an authenticated session with the vSphere Automation API endpoint. You send credentials to the vCenter Single Sign-On service and receive a SAML token that you use to retrieve a session ID from the vSphere Automation API endpoint. Alternatively, you can access the vSphere Automation APIs in a sessionless manner. You must simply include the SAML token in every request that you issue to the vSphere Automation API endpoint.

Service Discovery

You use the Lookup Service to retrieve the endpoint URL for the vCenter Single Sign-On service on the vCenter Server, the location of the vCenter Server instances, and the vSphere Automation API endpoint.

Components Installed with vCenter Server

vCenter Server is a central administration point for ESXi hosts. The group of components installed when you install vCenter Server include the vCenter Server service, vSphere Client, VMware vSphere® Auto Deploy™, VMware vSphere® ESXi™ Dump Collector, VMware vSphere® Syslog Collector, and vSphere Lifecycle Manager service.

You can use the vSphere Automation API endpoint to access the following services running on vCenter Server.

Content Library

You can use content libraries to share virtual machines, vApps, and other files, such as ISO, OVA, and text files, across the software-defined data center. You can create, share, and subscribe to content libraries on the same vCenter Server instance or on a remote instance. Sharing content libraries promotes consistency, compliance, efficiency, and automation in deploying workloads at scale.

You can also create OVF and VM templates from virtual machines and vApps in hosts, resource pools, and clusters. You can then use the OVF and VM templates to deploy new virtual machines and vApps.

Starting with vSphere 7.0, you can edit the contents of a VM template. You can check out the library item that contains the VM template. After editing the VM template, check in the library item to save the changes to the virtual machine.

Virtual Machine

You can use the vSphere Automation APIs to create, configure, and manage the life cycle of virtual machines in your environment.

Starting with vSphere 7.0, you can also clone, create an instant clone, migrate, register, and unregister a virtual machine.

vSphere Lifecycle Manager

Starting with vSphere 7.0, the life cycle of ESXi hosts and clusters can be managed through the VMware vSphere® Lifecycle Manager™ feature. Based on the current state of the hosts in a cluster, you can easily create a desired software specification by using the contents of a software depot. Then you validate the desired software specification and you apply the specification on all hosts in the cluster.

vSphere Tags

With vSphere tags you can attach metadata to vSphere objects, and as a result, make it easier to filter and sort these objects. You can use the vSphere Automation APIs to automate the management of vSphere tags.

vSphere with Tanzu

Starting with vSphere 7.0, you can enable vSphere with Tanzu on an existing vSphere cluster in your environment. Create and configure namespaces on the Supervisor Clusters to run Kubernetes workloads in dedicated resource pools.

Retrieving Service Endpoints

3

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

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 3-1. Service Registration Filter Parameters

Filter Types	Value	Description
`LookupServiceRegistrationServiceType`	`product= "com.vmware.cis"`	vSphere Automation namespace.
	`type="cs.vapi"`	Identifies the vSphere Automation service.

Table 3-1. Service Registration Filter Parameters (continued)

Filter Types	Value	Description
`LookupServiceRegistrationEndpointType`	`type="com.vmware.vapi.endpoint"`	Specifies the endpoint path for the service.
	`protocol="vapi.json.https.public"`	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 3-2. 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`

Table 3-2. Input Data for URL Retrieval for the Lookup Service Registration Filter (continued)

Service	Input Data	Value
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.

This example uses the steps that are described in the Connect to the Lookup Service and Retrieve the Service Registration Object procedure.

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 LsPorType.getLsPort();

((BindingProvider)lsProvider).getRequestContext().put(BindingProvider.ENDPOINT_ADDRESS_PROPERT
Y, lookupServiceUrl);

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

This example uses the steps that are described in the Connect to the Lookup Service and Retrieve the Service Registration Object procedure.

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 Service 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 `LookupServiceHelper.java` sample file.

This example uses the steps that are described in the Retrieve Service Endpoints on vCenter Server Instances procedure.

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

```
...

/**
 * Define filter criterion for retrieving a service endpoint.
 * Omit the nodeID parameter to retrieve the endpoints hosted
 * on all vCenter Server instances in the environment.
 */

  List<LookupServiceRegistrationInfo> lookupServiceUrls(String prod,
                                                        String svcType,
```

```
                                                        String proto,
                                                        String epType,
                                                        String nodeID){

    LookupServiceRegistrationServiceType filterServiceType =
                                                       new
LookupServiceRegistrationServiceType();
    filterServiceType.setProduct(prod);
    filterServiceType.setType(svcType);

    LookupServiceRegistrationEndpointType filterEndpointType =
                                                       new
LookupServiceRegistrationEndpointType();
    filterEndpointType.setProtocol(proto);
    filterEndpointType.setType(epType);

    LookupServiceRegistrationFilter filterCriteria = new LookupServiceRegistrationFilter();
    filterCriteria.setServiceType(filterServiceType);
    filterCriteria.setEndpointType(filterEndpointType);
    filterCriteria.setNode(nodeID);

    return lsPort.list(serviceRegistration, filterCriteria);
  }

...
```

Python Example of Retrieving a Service 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.

This example uses the steps that are described in the Retrieve Service Endpoints on vCenter Server Instances procedure.

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
    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.

This example uses the steps that are described in the Retrieve a vCenter Server ID by Using the Lookup Service procedure.

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 serviceAtttr :
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.

This example uses the steps that are described in the Retrieve a vCenter Server ID by Using the Lookup Service procedure.

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.

This example uses the steps that are described in the Retrieve a vSphere Automation API Endpoint on a vCenter Server Instance procedure.

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.

This example uses the steps that are described in the Retrieve a vSphere Automation API Endpoint on a vCenter Server Instance procedure.

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

4

To perform operations on services in the vSphere environment, you must create an authenticated connection to the services that you want to use. With the vSphere Automation SDKs you can authenticate and access vSphere Automation services.

Client applications can choose from two supported authentication patterns for accessing services in the virtual environment.

For better security, client applications can request a security token to authenticate connections with the vSphere Automation services.

To invoke operations on services, client applications must create a security context. The security context represents the client authentication. You can achieve authentication by using one of the following mechanisms.

Password-Based Authentication

To authenticate with user name and password, you connect to the vSphere Automation endpoint with vCenter Single Sign-On user credentials and obtain a session identifier (ID). The user account credentials are validated by the vSphere Automation endpoint, and must be present in the vCenter Single Sign-On identity store. The session ID is valid only for the service endpoint that you want to access and that issues the session ID.

Token-Based Authentication

Client applications can authenticate by using the vCenter Single Sign-On component on vCenter Server. vCenter Single Sign-On includes the Security Token Service (STS) that issues security tokens. The token must comply with the Security Assertion Markup Language (SAML) specification, which defines an XML-based encoding for communicating authentication data.

vCenter Single Sign-On supports two types of security tokens: bearer token and Holder-of-Key (HoK) token. To acquire a SAML token, client applications must issue a token request to vCenter Single Sign-On.

Client applications can present a SAML token to the vSphere Automation endpoint in exchange for a session identifier with which they can perform a series of authenticated operations.

To retrieve a session ID for the vSphere Web Services endpoint, you provide the SAML token to the vSphere Web services endpoint. For more information about creating an authenticated session to access the vSphere Web Services, see the *vSphere Web Services SDK Programming Guide* documentation.

This chapter includes the following topics:

Create a vSphere Automation Session with User Credentials

With the vSphere Automation SDKs , you can create authenticated sessions by using only user credentials.

You connect to the vSphere Automation endpoint by using a user name and password known to the vCenter Single Sign-On service. The vSphere Automation uses your credentials to authenticate with the vCenter Single Sign-On Service and obtain a SAML token.

Prerequisites

- Retrieve the vSphere Automation endpoint URL from the Lookup Service.
- Verify that you have valid user credentials for the vCenter Single Sign-On identity store.

Procedure

1. Create a connection stub by specifying the vSphere Automation endpoint URL and the message protocol to be used for the connection.
2. Create a stub configuration instance and set the specific security context to be used.

   The security context object uses the user name and password that are used for authenticating to the vCenter Single Sign-On service.
3. Create a `Session` stub that uses the stub configuration instance.
4. Call the create operation on the `Session` stub to create an authenticated session to the vSphere Automation endpoint.

   The operation returns a session identifier.
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

You can use the authenticated session to access vSphere Automation services. For more information about creating stubs to the vSphere Automation services, see Chapter 5 Accessing vSphere Automation Services.

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

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

This example uses the steps that are described in the Create a vSphere Automation Session with User Credentials procedure

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 vCloud Suite Session with Username and Password

This example is based on code in the `connection_workflow.py` sample file. This file is located in the following vSphere Automation SDK for Python directory: `client/samples/src/com/vmware/vcloud/suite/sample/workflow`.

```
from vars import (my_vapi_hostname,
                  my_sso_username,
                  my_sso_password,
                  my_stub_config)

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
from requests.packages.urllib3 import disable_warnings
from requests.packages.urllib3.exceptions import InsecureRequestWarning
disable_warnings(InsecureRequestWarning)

# 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)
```

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 3 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.

Java Example of Retrieving a SAML Token

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

This example uses the steps that are described in the Retrieve a SAML Token procedure.

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

```
...

LookupServiceHelper lookupServiceHelper = new LookupServiceHelper(
            this.lookupServiceUrl);

System.out.println("\nStep 2: Discover the Single Sign-On service URL"
                   + " from lookup service.");
```

```
String ssoUrl = lookupServiceHelper.findSsoUrl();

System.out.println("\nStep 3: Connect to the Single Sign-On URL and "
                   + "retrieve the SAML bearer token.");
SamlToken samlBearerToken = SsoHelper.getSamlBearerToken(ssoUrl,
    username,
    password);
```

Python Example of Retrieving a SAML Token

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

This example uses the steps that are described in the Retrieve a SAML Token procedure.

This example uses the following global variables.

- *my_vapi_hostname*
- *my_sso_username*
- *my_sso_password*

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

```
from vsphere.samples.common import sso

# Use the SsoAuthenticator utility class to retrieve
# a bearer SAML token from the vCenter Single Sign-On service.
sso_url = 'https://' + my_vapi_hostname + ':7444/ims/STSService'
authenticator = sso.SsoAuthenticator(sso_url)
saml_token = authenticator.get_bearer_saml_assertion(my_sso_username,
                                                     my_sso_password,
                                                     delegatable=True)
```

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.

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

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

This example uses the steps that are described in the Create a vSphere Automation Session with a SAML Token

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 SAML security context using SAML bearer token
SecurityContext samlSecurityContext =
        SecurityContextFactory.createSamlSecurityContext(
            samlBearerToken, null);

// Create a stub configuration with SAML security context
StubConfiguration stubConfig =
```

```
        new StubConfiguration(samlSecurityContext);

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

// Log in 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 a SAML Token

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

This example uses the steps that are described in the Create a vSphere Automation Session with a SAML Token

This example uses the following global variables.

- *my_vapi_hostname*
- *my_stub_config*
- *saml_token*

The example assumes that you previously obtained a vSphere Automation API URL from the Lookup Service, and a SAML token from the vCenter Single Sign-On Service.

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

# Add SAML token security context to the connector.
saml_token_context = create_saml_bearer_security_context(saml_token)
connector.set_security_context(saml_token_context)

# Create a stub configuration by using the SAML token security context.
my_stub_config = StubConfigurationFactory.new_std_configuration(connector)

# Create a Session stub with SAML token 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)
```

Create a Web Services Session

To develop a complex workflow, you might need to send requests to vSphere Web Services running in your virtual environment. To achieve this, you access the vSphere Web Services API by using the Web Services endpoint.

The vSphere Web Services API also supports session-based access. To establish an authenticated session, you can send the SAML token retrieved from the vCenter Single Sign-On service to a vSphere Web Service. In return, you receive a session identifier that you can use to access the service. For more information about accessing Web Services and additional examples, see the *vSphere Web Services SDK Programming Guide* documentation.

The vSphere Automation SDK for Python supports a simplified way of creating connections to the Web Services API by using the `pyVim` library.

Prerequisites

- Retrieve the vSphere Web Services endpoint URL from the Lookup Service.
- Obtain a SAML token from the vCenter Single Sign-On service.

Procedure

1. Connect to the vSphere Web Services endpoint.
2. Send the SAML token to a specific Web service to create an authenticated session.

3 Add the retrieved session ID to the service content object.

The Service Content object gives you access to several server-side managed objects that represent vSphere services and components.

Java Example of Creating a vSphere Web Services Session

This example is based on the code in the `VapiAuthenticationHelper.java` and `VimUtil.java` samples.

This example uses the steps that are described in the Create a Web Services Session procedure.

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

```
... // Log in to the vSphere Web Services endpoint and retrieve a session
identifier. SamlTokenElement tokenElement = ssoConnection.getSamlBearerTokenElement(); String
sessionId = LoginByTokenSample.LoginUsingSAMLToken(tokenElement, vimUrl, null, null); //
Use the VimPortType and VimService objects from // the vSphere Web Services API for
accessing Web Services and // retrieve the request context. VimService vimService =
new VimService(); vimPortType vimPort = vimService.getVimPort(); // Add the retrieved
session ID to the request context. Map<String, Object> ctxt = ((BindingProvider)
vimPort).getRequestContext(); ctxt.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY, vimUrl);
ctxt.put(BindingProvider.SESSION_MAINTAIN_PROPERTY, true); Map<String, List<String>> headers
= (Map<String, List<String>>) ctxt.getMessageContext.HTTP_REQUEST_HEADERS); if (headers
== null) { headers = new HashMap<String, List<String>>(); } headers.put("Cookie",
Arrays.asList(vcSessionId); ctxt.put(MessageContext.HTTP_REQUEST_HEADERS, headers); // Use
the session ID context when retrieving the ServiceContent object. // The ServiceContent
object gives you access to a number of // server-side managed objects that represent vSphere
services and components. // For more information about the vSphere Web Services, // see the
vSphere Web Services SDK Programming Guide documentation.
ServiceContent serviceContent = VimUtil.getServiceContent(vimPort);
```

Python Example of Creating a Web Services Session

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

This example uses the steps that are described in the Create a Web Services Session procedure.

This example uses the following global variables.

- *my_ws_url*
- *my_sso_username*
- *my_sso_password*

The *my_ws_url* variable represents the URL of the vCenter Server Web Services API endpoint. You can retrieve the endpoint URL from the Lookup Service.

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

```
...

# Extract the hostname from the endpoint URL.
url_scheme, url_host, url_path, url_params, url_query, url_fragment = \
      urlparse(my_ws_url)
pattern = '(?P<host>[^:/ ]+).?(?P<port>[0-9]*)'
match = re.search(pattern, url_host)
host_name = match.group('host')

# Invoke the SmartConnect() method by supplying
# the host name, user name, and password.
service_instance_stub = SmartConnect(host=host_name,
                                     user=my_sso_username,
                                     pwd=my_sso_password)

# Retrieve the service content.
service_content = service_instance_stub.RetrieveContent()
```

Accessing vSphere Automation Services

5

vSphere Automation SDK provides mechanisms for creating remote stubs to give clients access to vSphere Automation services.

The sequence of tasks you must follow to create a remote stub starts with creating a `ProtocolFactory`. You use the protocol factory object to create a `ProtocolConnection`. Connection objects provide the basis for creating stub interfaces tovSphere Automation services.

When you establish a connection to the vSphere Automation endpoint, you can create a `StubFactory` object and a `StubConfiguration` object. With these objects, you can create the remote stub for the vSphere Automation service that you want to access.

The complete connection sequence also includes SSL truststore support and a temporary `StubConfiguration` that you use for SAML token authentication and session creation.

SSL Handshake

The vSphere Automation endpoint (`https://host/api`) is an SSL-enabled service that requires client authentication during login. The SSL connection relies on certificate verification supported by the Java security architecture. The Java security architecture defines truststores for SSL connections. A truststore contains vCenter Single Sign-On credentials. You use a truststore to verify credentials from a vCenter Server instance.

The vSphere Automation SDK for Java includes an SSL utility sample code that supports the creation of a truststore for the HTTP connection, `com.vmware.vcloud.suite.samples.common.SslUtil`.

Note The vSphere Automation SDK for Java SSL utility creates an instance of the Java security certificate class `X509TrustManager`. This instance declares an override client-side method, `checkServerTrusted`, that accepts all HTTPS certificates. This method is suitable only for development environments. For a production environment, do not use the `X509TrustManager` override methods. Instead, set up a truststore for use by the default `X509TrustManager` implementation.

For greater security, use an external utility to create a certificate store:

```
keytool -import -noprompt -trustcacerts \
-alias <alias name> \
-file <certificate file> \
-keystore <truststore filename> \
-storepass <truststore password>
```

This chapter includes the following topics:

Access a vSphere Automation Service

To access a vSphere Automation service, you must have a valid session connection. The sequence for accessing a vSphere Automation service includes creating a protocol connection object and using it to create the service stub.

Prerequisites

Establish a connection to the vSphere Automation endpoint URL. For more information about the authentication mechanisms that you can use, see Chapter 4 Authentication Mechanisms.

Procedure

1. Create a protocol factory object.
2. Create a protocol connection object to access an API provider.

   The vSphere Automation API clients use `ApiProvider` instances to invoke operations on services running in the virtual environment. To invoke an operation, you must specify the target service and operation, input parameters, and execution context.
3. Create a `StubFactory` object by using the `ApiProvider` instance.
4. Create a `StubConfiguration` instance and set the security context to be used for the service stub.

5 Create the stub for the vSphere Automation service interface by calling the create method of the `StubFactory` instance. Pass the service class and the `StubConfiguration` instance as arguments.

Java Example of Accessing a vSphere Automation Service

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

This example shows the steps for creating an authenticated session to the vSphere Automation endpoint and creating the service stub for the Content Library API provider.

This example uses the steps described in the Access a vSphere Automation Service procedure.

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

```
...

// Log in by using username/password
      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 by using the stub configuration.
Session session =
        this.stubFactory.createStub(Session.class, stubConfig);

// Log in 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);

// Create service stubs for the Content Library service.
Library libraryService = stubFactory.createStub(Library.class, stubConfig);
```

```
// Invoke an operation of the Content Library service.
List<String> listContentLibraries = libraryService.list();
```

ESXi Hosts

6

Use the vSphere Automation APIs to run general operations on the ESXi hosts in your vSphere environment.

You can retrieve information about the hosts, create a standalone host, disconnect, and reconnect an ESXi host to a vCenter Server system. In vSphere 7.0, you can also use the ESXi Token service to create a JWT token for a host and use it to authenticate to services running on that particular host.

This chapter includes the following topics:

Retrieving Information About ESXi Hosts

You retrieve information about the ESXi hosts running in a vCenter Server instance by listing only the ESXi hosts that you are interested in.

To filter the ESXi hosts on a vCenter Server instance and get only the ones you want, call the `list` function and pass your criteria with a `HostTypes.FilterSpec` instance. Use the `HostTypes.FilterSpec` instance to combine several filter criteria by including one or more of the following properties:

- The name or unique identifier of the host.
- Clusters, data centers, or folders that contain the host.
- Connection state of the host which can be one of the following: `CONNECTED`, `DISCONNECTED`, or `NOT_RESPONDING`.
- Power state of the host which can be one of the following: `POWERED_OFF`, `POWERED_ON`, or `STANDBY`.

The function returns a list of `com.vmware.vcenter.HostTypes.Summary` objects that contain information about up to 2500 hosts that match all specified criteria. You can use the list to retrieve information about the returned ESXi hosts.

Adding a Single ESXi Host to vCenter Server

You can use the vSphere Automation APIs to add a standalone host to a vCenter Server instance.

Add a single ESXi host to a vCenter Server instance by calling the `create` function and passing a `com.vmware.vcenter.HostTypes.CreateSpec` instance as parameter. When you create the host specification, make sure that you set the IP address or the DNS resolvable host name and the administrator credentials.

Disconnecting and Reconnecting ESXi Hosts

You can use the vSphere Automation APIs to connect ESXi hosts to a vCenter Server instance and make the hosts managed. You can temporarily disconnect a managed host from a vCenter Server instance and reconnect the host, for example, to refresh the ESX agents on the host.

When you add a host to a vCenter Server instance, the host is connected to vCenter Server and becomes a managed host. To disconnect a managed host from a vCenter Server instance, call the `disconnect` function and pass the host identifier as a parameter. The managed host and its associated virtual machines remain in the inventory but vCenter Server temporarily stops managing and monitoring them.

To reconnect a managed host to a vCenter Server instance, call the `connect` function and pass the host identifier as a parameter. As a result, the connection status of the host changes, and vCenter Server resumes managing the host and its associated virtual machines.

If you want to delete a host and all its associated virtual machines from the inventory, you can remove the host from the vCenter Server instance. To delete a disconnected host from a vCenter Server instance, call the `delete` function and pass the host identifier as a parameter.

Accessing ESXi Host Services

The vSphere Automation APIs include interfaces for managing access to ESXi host services. You can access the services running on an ESXi host for a limited period without the need to enter user credentials multiple times.

You can use the vSphere Automation APIs to create and manage user profiles on ESXi hosts. Authenticated users can access services running on the host according to the permissions associated with their profile. To create a user profile, call the `create` function of the `ClientProfiles` interface and pass as a parameter a `ClientProfilesTypes.CreateSpec` instance. Use the `ClientProfilesTypes.CreateSpec` instance to specify the access rights that the users have for managing ESXi resources and services.

When a user enters credentials from a client application, the ESXi Token service uses the Linux pluggable authentication module (PAM) mechanism to verify the user credentials. If the user is authenticated, the Token service returns a token.

In vSphere 7.0, the implementation of the Token service accepts a user name and password as user credentials and returns a JSON Web Token (JWT). The JWT is compliant with RFC 7519, and consists of three parts.

Table 6-1. JWT Structure

Element	Description
Header	Contains the algorithm that is used for generating the token signature and the token type. The header is usually Base64 encoded.
Payload	Contains the user data represented in the JSON format and Base64 encoded. The payload stores information about the user name, expiration time of the token, and so on.
Signature	Contains data for verifying the token.

Note The token format and technology might change in future releases, and might be backward incompatible.

Store the received token in the client application and use the token to authenticate to the ESXi services that you want to access.

Access ESXi Host Services with a Token

Starting with vSphere 7.0, you can access services and resources running on an ESXi host by using a token issued in exchange for user credentials. Use the token to authenticate for a limited time and avoid resending user credentials for each API call.

You send valid user credentials to the `Token` service and receive a token which you can use to authenticate to the ESXi services.

Prerequisites

Obtain the password of the **root** user for the host.

Procedure

1. Create a connection stub with the host by specifying the host name and the message protocol to use for the connection.
2. Create a stub configuration instance and set the security context to use for authentication.

   The `SecurityContext` object can use either a user name and password or a SAML token to authenticate to the host by sending the information to the PAM service.
3. Create a `Session` stub that uses the stub configuration.
4. Call the `create` function of the `Token` service and pass the `SecurityContext` object to create an authenticated session to the host.

   The operation returns a `TokenInfo` object that contains the access token.

What to do next

You can add the received token to the security context and update the session security context. You can then use the authenticated session to access services running on the host for a limited period.

Managing the Life Cycle of Hosts and Clusters

7

Starting with vSphere 7.0, you can manage the life cycle of ESXi hosts collectively by using the vSphere Lifecycle Manager feature through the vSphere Automation APIs.

You can automate the life cycle management of a cluster by performing the following operations:

- Retrieve information about the current state of the host or cluster.
- Create a desired state that includes a specific version of the ESXi host. You can also add some compatible partner software and firmware components and add-ons.
- Validate the desired state to detect any discrepancies between the desired state and the host hardware.
- Check the compliance of a cluster against the desired state and determine whether some additional steps must be taken to ensure the success of the cluster remediation.
- Apply the desired state on a cluster.

You can use the vSphere Lifecycle Manager to manage the life cycle of hosts that meet the following requirements:

- Hosts must be of version 7.0 and later.
- Hosts must be stateful.
- Hosts must include only components that belong to integrated solutions, such as VMware vSAN™ and VMware vSphere® High Availability.

This chapter includes the following topics:

vSphere Lifecycle Manager Terms

You must understand the basic terminology that is used within this chapter to be able to use the vSphere Lifecycle Manager functionality efficiently.

vSphere Lifecycle Manager Terminology

Term	Definition
Upgrade, update, and patch	You can upgrade to another major version of the software running on an ESXi host, and install patches and updates that include smaller changes, bug fixes, or other small improvements.
Depot	A depot is a well-defined folder structure that is used for distributing payloads and their metadata. Depots are consumed by different products and features such as the vSphere Lifecycle Manager and ESXCLI. The vSphere Lifecycle Manager works with three types of depots: online, offline, and UMDS. See Software Depots.
Component	A component is the smallest unit that the vSphere Lifecycle Manager uses during the installation and update processes. Software vendors use components to encapsulate a group of payloads that can be managed together.
Base image	A base image is a collection of components that shape the bootable ESXi used for the installation or upgrade process. Base images are currently distributed only by VMware and support x86 servers. VMware provides new versions of the base image for each upgrade, update, and patch release of the ESXi. Base images are hosted at the VMware online depot that is available by default to the vSphere Lifecycle Manager. Furthermore, you can download a different base image version, in the form of an offline ZIP bundle, from https://my.vmware.com/web/vmware/downloads.
Add-on	An add-on is a collection of components that different OEMs provide on top of a base image. Vendors use an add-on to group some customizations for a family of servers. Unlike base images, add-ons are not complete and are not sufficient to boot an ESXi. Each add-on must have a unique name and version. An add-on allows vendors to add, remove, or update components that are part of the VMware base image, if there are no unresolved dependencies and conflicts between the components.
Solution	A solution contains one or more components, and provides information about its constraints and compatibility with the different ESXi versions. For example, from the perspective of the vSphere Lifecycle Manager solutions are VMware NSX-T, VMware NSX-V, VMware vSphere® High Availability, vSAN.
Desired state	A desired state of a cluster is represented with a software specification. The desired state defines a set of components that a user wants to install on a single ESXi host or on a cluster of hosts.
OEMs	Original Equipment Manufacturers. VMware partners enrolled in the VMware Partner Connect application, such as Dell, Inc., HP Inc., Lenovo Group Ltd., and so on.

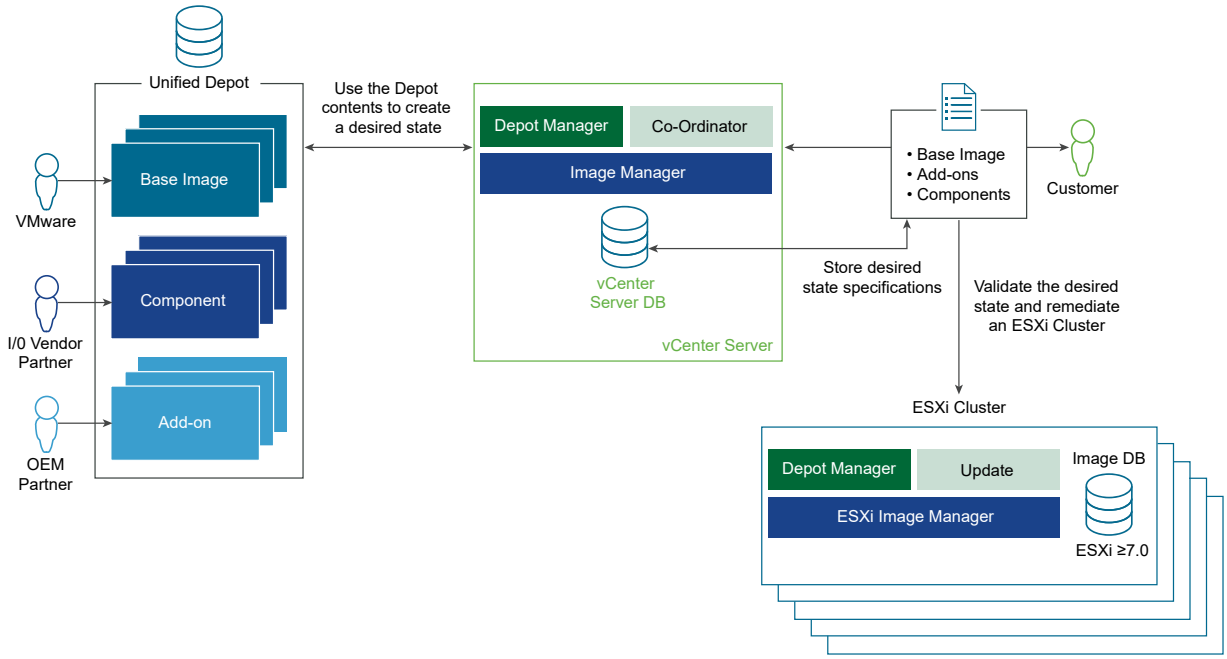
Term	Definition
IOVP	I/O Vendor Partner. Qualified VMware partners providing certified I/O device drivers for network and storage host bus adapters.
Third-party software providers	Providers of I/O filters, device drivers, CIM modules, and so on, that are not part of VMware partner programs.

vSphere Lifecycle Manager Overview

The vSphere Lifecycle Manager feature provides means for managing the life cycle of hosts on a cluster level. The functionality of this feature is achieved through several major services that are running on both vCenter Server and ESXi.

After you install vSphere 7.0, you can access the feature through several major services available on both the vCenter Server and on each ESXi host.

Figure 7-1. vSphere Lifecycle Manager System Architecture



Depot Manager

For each upgrade and patch release, VMware, OEMs, and other third-party companies make the software updates of their products available to the customers. Software updates are distributed to different locations and in different formats depending on the way they are accessed, downloaded, and used. The Depot Manager allows these different software resources to be presented to the vSphere customers in a unified format and as a result, makes them easier to use.

Depending on your environment and specific use case, the Depot Manager gives you access to the software updates within three different types of depots: online, offline, and UMDS. See Software Depots.

The Depot Manager that runs on the vCenter Server instance achieves the following goals:

- Represents the contents of all depots in a unified way.
- Caches the payloads and their metadata locally prior to their use.
- Enables depot overrides on a cluster level for Remote Office/Branch Office (ROBO) cluster environment, or Edge computing environment.

The Depot Manager that runs on hosts, serves as a proxy to the vCenter Server Depot Manager. If ROBO cluster or Edge computing environments are enabled, the Depot Manager that runs on the ESXi hosts serves as a proxy to the nearby vSphere Lifecycle Manager compatible depot.

Image Manager

The Image Manager allows you to create a desired state that you can apply on a cluster or a standalone host. The specification describes all components, add-ons, and the base image that you can use to update or upgrade the hosts in your environment. The Image Manager supports validation of the desired state and the detecting drifts from the desired state.

Coordinator and ESXi Updater Managers

The Coordinator Manager runs on the vCenter Server instance and makes sure that the desired state is applied to all hosts in the cluster. This module also runs pre-checks to evaluate how each host in the cluster is affected by the remediation and whether you must take some additional steps to ensure the success of the procedure. The Coordinator Manager also allows you to query the status of the remediation operation for the cluster and for each host part of the cluster.

The ESXi Updater Manager takes care of the actual remediation happening on each host.

Options for Managing the ESXi Life Cycle

Based on your needs and environment setup, you can choose from several methods for managing the life cycle of the ESXi hosts. The vSphere Lifecycle Manager provides means for updating all hosts in a cluster with a desired software state.

Methods and Tools for ESXi Life Cycle Management

Today, VMware provides the following methods for managing the life cycle of ESXi hosts:

- Interactive installation and upgrade. Use this method for smaller deployments of less than five hosts. You install and boot the ESXi by using the ESXi installer ISO image.
- Scripted installation and upgrade. Use this method for installing or upgrading multiple hosts with similar configuration settings. You create an installation or upgrade script that contains the ESXi installation options. Then you boot the ESXi installer and run the script.
- VMware vSphere® Auto Deploy™. Use this method for provisioning hundreds of physical hosts with the ESXi software. vSphere Auto Deploy works with image profiles and host profiles to provision the hosts. An image profile is considered to be the bootable ESXi image provided by VMware or partners.

By default, vSphere Auto Deploy does not store the ESXi state on the host itself. Each time the host boots, the vSphere Auto Deploy provisions the host with the image profile. After the initial installation of the ESXi host, you can set up a host profile that causes the host to store the ESXi image and data on the local or a remote disk, or a USB drive. This process is similar to a scripted installation.

You can use the VMware vSphere® ESXi™ Image Builder CLI to examine the public VMware software depot and create image profiles with a customized set of updates, patches, and drivers.

A host profile defines some host configuration setup such as networking and storage configuration. To achieve some consistency with the hosts configurations in your environment, you can create a host profile for a single host and then apply the configuration to the other hosts in your environment.

- VMware vSphere® Update Manager™. Use this product to automate the patching, upgrading, and updating of the ESXi hosts in your environment. You can use the vSphere Update Manager through the vSphere Client to update your hosts up to version 7.0. See *vSphere Update Manager Installation and Administration Guide*. Starting with vSphere 7.0, to manage the life cycle of the hosts in your environment, you can use the vSphere Lifecycle Manager through the vSphere Client. See *Managing Host and Cluster Lifecycle*. This chapter of the vSphere Automation SDKs Programming Guide discusses how you can access and use the functionality provided by the vSphere Lifecycle Manager through the APIs.

vSphere Lifecycle Manager Features

Starting with vSphere 7.0, you can use the vSphere Automation APIs to manage the life cycle of hosts collectively by using the vSphere Lifecycle Manager. You can access and use the following vSphere Lifecycle Manager functionality:

- Depot management. You can add, remove, explore the contents of different types of depots. See Software Depots. The content of the depots is provided by VMware and VMware partners. Partners can use the ESXi Packaging Kit (EPK) to assemble a custom bootable ESXi image. The custom image can then be shared to other third-party customers and used through the Depot Manager. For more information about how to create custom ESXi images, see *ESXi Packaging Kit (EPK) Development Guide*.
- Desired software state. You can create, edit, and delete a desired software state for a cluster on which the vSphere Lifecycle Manager is enabled. A desired software state must contain at least a single ESXi image provided by VMware. You can also set an add-on provided by OEMs, and one or more components by different software vendors. Furthermore, during the process of creating the desired software state, you can check the validity of the specification and compare the current state of the hosts in the cluster with the desired software state.
- Cluster remediation. You can apply the desired state on each of the hosts in a cluster which current state is different from the desired specification. Applying a desired state on a cluster level has the following prerequisites:
  - The cluster must have the vSphere Lifecycle Manager enabled.

- All hosts in the cluster must store their data on a local or remote disk, or on a USB drive.
- All hosts in the cluster must be of version 7.0 or higher.
- All hosts must contain only components that the vSphere Lifecycle Manager can recognize and maintain. If a host contains some old content that the vSphere Lifecycle Manager does not recognize, the content is removed from the host during remediation.

Software Depots

A software depot represents a well-defined file structure used for storing and hosting the ESXi software updates, patches, and upgrades that VMware, partners, and third-party vendors provide. You can use the vSphere Automation APIs to manage the life cycle of the hosts in your environment by applying software updates hosted on different depots.

Software depots are managed by the Depot Manager which is part of the vSphere Lifecycle Manager. Software depots contain the actual payloads and the metadata of the software updates. Depending on the way you access the software updates, the Depot Manager recognizes three types of software depots: online, offline, and UMDS.

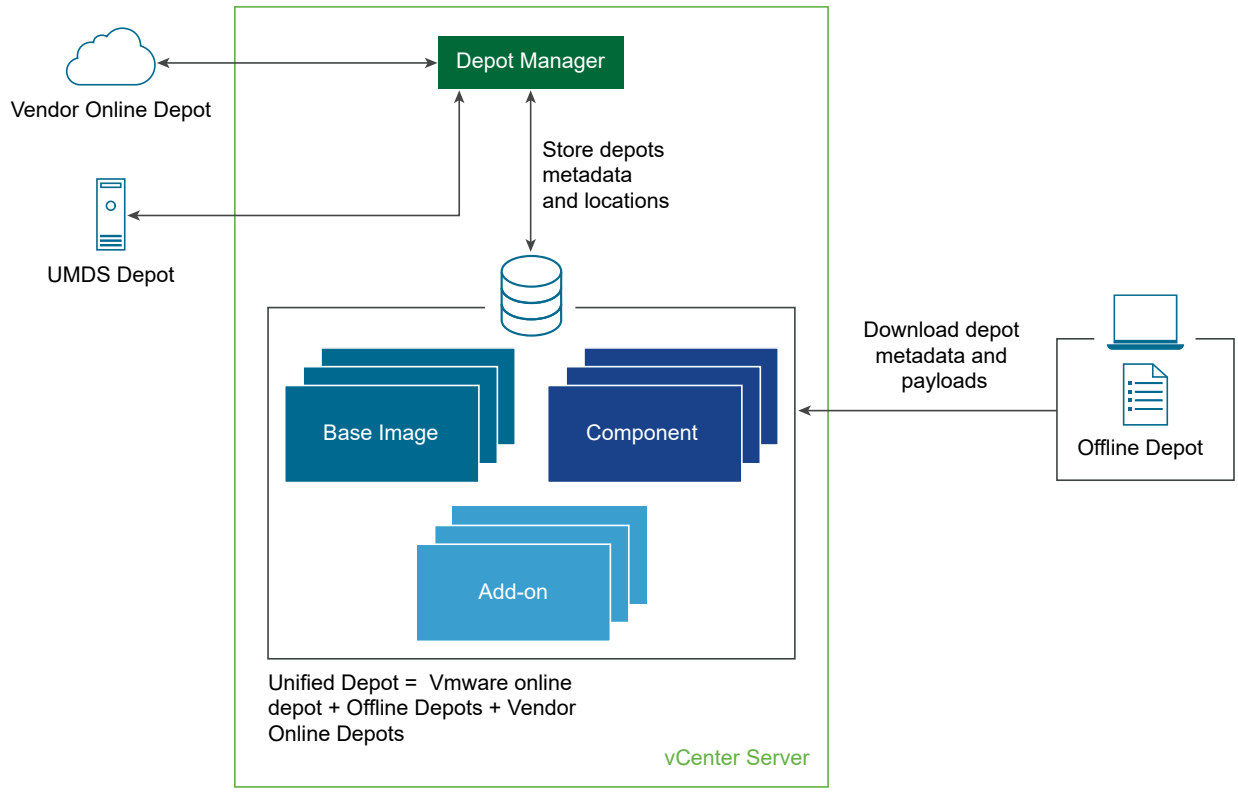
The following section of the documentation explains the concept of a software depot in terms of the vSphere Lifecycle Manager feature. You can also find common use cases available through the APIs for working with the different types of depots and their content.

Types of Software Depots

The Depot Manager works with the software updates provided by three different types of software depots: online, offline, and UMDS.

Regardless of the different way in which you access each type of software depot, all depots have the same structure. The same depot structure allows content from different vendors to be uploaded to one depot. By default, you can access the content of the VMware online depot at `https://hostupdate.vmware.com/software/VUM/PRODUCTION/...`. Furthermore, partners and third-party customers can use the ESXi Packaging Kit to build and distribute software updates in the form of offline or online depots. You can access their content by adding the online vendor depot to the vSphere Lifecycle Manager or by downloading the content of the offline depot to the vCenter Server instance.

Figure 7-2. Types of Software Depots



Online Depot

VMware and partners upload software updates to the VMware online depot at `https://hostupdate.vmware.com/software/VUM/PRODUCTION/...` or to a custom online depot. Software updates can be patches of the ESXi base image, different versions of the partner add-ons, the IOVP drivers certified by VMware, and the VMware Tools™ updates. Online depots are accessible through a URL. By default, you can see the base images, add-ons, and components provided within the VMware online depot at the following locations:

- `https://hostupdate.vmware.com/software/VUM/PRODUCTION/addon-main/vmw-depot-index.xml`
- `https://hostupdate.vmware.com/software/VUM/PRODUCTION/main/vmw-depot-index.xml`
- `https://hostupdate.vmware.com/software/VUM/PRODUCTION/iovp-main/vmw-depot-index.xml`
- `https://hostupdate.vmware.com/software/VUM/PRODUCTION/vmtools-main/vmw-depot-index.xml`

When you deploy the vCenter Server, the vSphere Lifecycle Manager is configured to access the VMware online depot, by default. You can use the vSphere Automation APIs to add a custom online depot to be managed by the Depot Manager. The metadata of the newly added online depot is not synchronized immediately. To synchronize the metadata, you can run a synchronization operation or wait for the scheduled synchronization to take place.

The Depot Manager stores in the vCenter Server database only the metadata of the software updates and the location of the added online depots. You can create a schedule to synchronize the software updates metadata stored in the vCenter Server with the metadata available in the accessible depots. The payloads of the software updates are downloaded only during the cluster remediation process.

To add, remove, list, and retrieve information about the online depots, you can use the `com.vmware.esx.settings.depots.Online` interface. See Working with Online Depots.

Offline Depot

The offline depot is also called an offline bundle and is distributed as a downloadable ZIP file. Offline depots contain both the metadata and the payloads of the software update. Partners and third-party customers can use the ESXi Packaging Kit to build and distribute offline bundles. You can download offline bundles delivered by VMware from `https://my.vmware.com/web/vmware/downloads`. When you add an offline depot to the vSphere Lifecycle Manager, the software updates are downloaded to the vCenter Server database.

To manage offline depots, you can use the `com.vmware.esx.settings.depots.Offline` interface. See Working with Offline Depots.

UMDS Depot

In case the vCenter Server instance is in an air-gapped environment and has no access to any wire or wireless network, you can use a UMDS depot. The VMware vSphere® Update Manager™ Download Service

(UMDS) is available as a `VMware-UMDS-7.0.0.-build_number.tar.gz` file within the ISO image of the vCenter Server 7.0 installer. Install the UMDS on a machine that has Internet access and is different from the machine on which the vSphere Lifecycle Manager is running. For further information about how to install and configure the UMDS module, see the *vSphere Update Manager Installation and Administration Guide*.

You can set up a synchronization schedule for downloading specific software updates from online vendor depots to the UMDS depot. Then use these updates to create desired software state for the clusters in your environment.

To manage UMDS depots through the vSphere Automation APIs, you can use the `com.vmware.esx.settings.depots.Umds` interface.

Working with Online Depots

You can use the vSphere Automation APIs to add online depots to the list of currently configured online software depots.

Use online depots to add new content over time to the management scope of the Depot Manager. The Depot Manager periodically updates the software depots metadata stored on the vCenter Server instance. In case new software updates are uploaded to the online depots, the Depot Manager makes sure that the metadata stored on the vCenter Server database is updated accordingly.

To add an online depot to the Depot Manager, you must first create the online depot specification by using the `com.vmware.esx.settings.depots.OnlineTypes.CreateSpec` class. To specify the URI to the `vendor-index.xml` file of the online depot, use the `setLocation(location)` method of the `OnlineTypes.CreateSpec` class. Optionally, you can add a description and enable the depot. By default, when you add an online depot to the Depot Manager, the depot is enabled and its metadata is synchronized following the defined schedule. If you want to synchronize the added online depot immediately, call the `sync_Task()` method of the `com.vmware.esx.settings.Depots` interface. When you complete the depot specification, call the `create(spec)` method of the `com.vmware.esx.settings.depots.Online` interface to add the depot.

You can edit the depot description and disable the depot by creating an `com.vmware.esx.settings.depots.OnlineTypes.UpdateSpec` object and pass it to the `update( depot, update_spec)` method of the `com.vmware.esx.settings.depots.Online` interface.

You can remove an online depot from the list of currently configured depots by using the `delete(depotID)` method of the `com.vmware.esx.settings.depots.Online` interface. The invocation of this method does not remove the already downloaded metadata and payloads from the deleted depot. You cannot delete the default VMware online depot, you can only disable it.

To retrieve a list of currently configured online depots, call the `list()` method of the `Online` interface. You can also retrieve information about a currently configured online depot by using the `get(depotID)` method of the `Online` interface.

Working with UMDS Depots

In an air-gapped vCenter Server environment, you can use the vSphere Automation APIs to add a UMDS depot to the depots managed by the Depot Manager.

After you install and configure the Update Manager Download Service (UMDS) on a physical machine with Internet access, you can add the UMDS depot to the Depot Manager. Only one UMDS depot can be added at a time to the Depot Manager. When you add a UMDS depot, its content is not immediately synchronized. To synchronize the content of the UMDS depots, you must call the `sync_Task()` method of the `com.vmware.esx.settings.Depots` interface or wait for the scheduled synchronization to take place.

To add a UMDS depot, call the `set(set_spec)` method of the `com.vmware.esx.settings.depots.Umds` interface and pass a `com.vmware.esx.settings.depots.UmdsTypes.SetSpec` object as an argument. The UMDS specification must contain the URI location to the `index.xml` file of the depot. Optionally, you can set a description and indicate whether the depot must be enabled. By enabling the UMDS depot, you instruct the Depot Manager to synchronize only the content that is available on that depot.

You can always edit the initial UMDS depot settings, by calling the `update(update_spec)` method of the `Umds` interface and passing an `com.vmware.esx.settings.depots.UmdsTypes.UpdateSpec` object.

To retrieve information about the currently configured UMDS depot, use the `get()` method of the `Umds` interface. You can remove a currently configured UMDS depot by calling the `delete()` method of the `Umds` interface.

Synchronizing Software Depots

The VMware online depot, the vendor online depots, and the UMDS depot must be synchronized regularly if you want to have the most recent software updates delivered by VMware, partners, and other third-party vendors. Use the vSphere Automation APIs to create a synchronization schedule or to synchronize the added depot immediately.

The Depot Manager does not synchronize immediately the metadata of the newly added online and UMDS depots. If you want to force the synchronization and not wait for the scheduled synchronization to take place, call the `sync_Task()` method of the `com.vmware.esx.settings.Depots` interface. You can also define a custom schedule to sync the metadata from the currently configured online or UMDS depots.

To create a custom schedule for checking for new software updates, you must first define the schedule parameters by using the `com.vmware.esx.settings.depots.SyncScheduleTypes.Schedule` class. Then you can add the schedule to the schedule specification by using the `setSchedule(schedule)` method of the `com.vmware.esx.settings.depots.SyncScheduleTypes.Spec` class. Optionally, you can use the schedule specification to add an email to which notifications will be sent and define whether updates will be downloaded automatically. To apply the custom schedule, call the `set(spec)` method of the `com.vmware.esx.settings.depots.SyncSchedule` interface.

The default schedule is set to update the metadata daily at a random time. To reset the schedule to the default settings, call the `set(spec)` of the `SyncSchedule` interface and pass `null` as an argument.

Working with Offline Depots

An offline depot is a ZIP file that contains the metadata and payloads of software updates and follows the same structure as the online depot. Use the vSphere Automation APIs to import the content of an offline depot to the vCenter Server database.

To add an offline depot to the depots managed by the Depot Manager, you must first create an offline depot specification by using the `com.vmware.esx.settings.depots.OfflineTypes.CreateSpec` class. When defined the offline depot parameters, call the `create_Task(create_spec)` method of the `com.vmware.esx.settings.depots.Offline` interface. Depending on the location of the offline depot, when you create the offline depot specification, you must provide either the URI location or the file ID returned by the Jetty Web server embedded in the vSphere Lifecycle Manager. You set the type of the source from which the offline depot is downloaded by using the `setSourceType(sourceType)` method of the `OfflineTypes.CreateSpec` class.

Pull Depot Content from a URI

To indicate that the offline depot resides on a URI location, call the `setLocation(java.net.URI location)` method of the `OfflineTypes.CreateSpec` class. You can pass as an argument the depot location in one of the following URI schemes: `http`, `https`, or `file`. If you provide an HTTPS location to the offline depot, make sure you also provide a certificate trusted by the VMware Certificate Authority (VMCA) or a custom certificate from the VMware Endpoint Certificate Store (VECS). See *vSphere Authentication*.

Push Depot Content to the Depot Manager

To push the content of an offline depot to the Depot Manager, you must first upload the ZIP file to the Jetty Web server at the `https://<vcenter_FQDN>:9087/vum-fileupload` URL. The server returns a file identifier that you can pass as an argument to the `setFileId(fileId)` method of the `OfflineTypes.CreateSpec` class.

Managing Depot Overrides

In case you have a smaller Remote Office/ Branch Office (ROBO) cluster environment, or Edge computing environment, your clusters have no, or limited access to the Internet and limited connection to a vCenter Server instance. In such an environment, you can use the Depot Manager to fetch the metadata and payloads of a desired software state from a local to the cluster depot.

To remediate a ROBO cluster, you must have access to a local software depot that hosts the components of the desired state. You can use the Depot Manager to either export the whole vSphere Lifecycle Manager depot to an offline bundle, or export only the content of the desired state required for remediating the ROBO cluster.

To export the desired state image from the vSphere Lifecycle Manager depot, call the `export(cluster,export_spec)` method of the `com.vmware.esx.settings.clusters.Software` interface. This method returns the URI of the offline bundle that is hosted on the vSphere Lifecycle Manager Jetty Web server. To move the content of the offline bundle to the ROBO location, you must physically copy the ZIP file, unarchive, and mount its content to an HTTP server inside the ROBO environment.

The vSphere Automation APIs offer the `com.vmware.esx.settings.clusters.DepotOverrides` interface to redirect a ROBO cluster to download software updates from a local repository and not from the vSphere Lifecycle Manager depot on the vCenter Server instance. To add a depot override location to a ROBO cluster, call the `add(cluster,override_depot)` method of the `DepotOverrides` interface. Specify the URI location of the local depot with the `setLocation(location)` method of the `com.vmware.esx.settings.clusters.DepotOverridesTypes.Depot` class. During the ROBO cluster remediation, the Depot Manager instructs the hosts in the ROBO cluster to download the software updates from the configured local depots within the ROBO cluster.

Inspecting Depot Contents

You can use the vSphere Automation APIs to inspect the contents of the already synchronized and imported depots. You can list the available base images, add-ons, and components, or retrieve some detailed information about a specific software update.

To retrieve a list of the base images available on a vCenter Server instance, call the `list(filter_spec)` method of the `com.vmware.esx.settings.depot_content.BaseImages` interface. To narrow the list of returned base images and retrieve only items matching to specific criteria, you must pass a `com.vmware.esx.settings.depot_content.BaseImagesTypes.FilterSpec` instance as an argument to the list method. Use the retrieved list to get some information about each base image, including their display name and version, release date, and their category. You can also get some detailed information about a single base image by using the `get(version)` method of the `com.vmware.esx.settings.depot_content.base_images.Versions` interface. The information includes a list of the components present in this base image.

To retrieve a list of all currently available add-ons in the vSphere Lifecycle Manager depot, call the `list(filter_spec)` method of the `com.vmware.esx.settings.depot_content.AddOns` interface and pass `null` as an argument. You can filter the available add-ons by using some specific criteria such as the add-on vendor, name, versions, or minimum version. You can retrieve some detailed information about a single add-on version, by calling the `get(name,version)` method of the `com.vmware.esx.settings.depot_content.add_ons.Versions` interface. The information includes the list of components part of the add-on, and the list of components that were removed by this add-on version.

To retrieve a list of all components currently available in the vSphere Lifecycle Manager depot, you can call the `list(filter_spec)` method of the `com.vmware.esx.settings.depot_content.Components` interface and pass `null` as an argument. To retrieve a list of components that matches some specific criteria, define your preferences with a `com.vmware.esx.settings.depot_content.ComponentsTypes.FilterSpec` instance and pass it as an argument to the list method. Use the retrieved list to get some information regarding each component.

Enabling a Cluster to Use a Software Specification

If you want to use the vSphere Lifecycle Manager to manage the life cycle of clusters in your environment, you have two options for enabling this feature. You can enable the vSphere Lifecycle Manager when you create the cluster. You can also turn an already created cluster into one managed by the vSphere Lifecycle Manager.

Creating a Cluster with Enabled vSphere Lifecycle Manager

Currently, you can use only the vSphere Web Services APIs to create a cluster in your virtual environment. Starting with vSphere 7.0, you can specify the desired state during the cluster creation.

You must use the vSphere Web Services APIs to create a cluster. Call the `Folder.CreateClusterEx` method and pass as arguments the name for the new cluster and a `ClusterConfigSpecEx` data object. In the data object, among other properties, you can specify the desired state for the cluster. The `desiredSoftwareSpec` property in the `ComputeResourceConfigSpec` data object contains the desired software specification for the cluster. This property is available for applications using the vSphere Web Services APIs 7.0 and later. You can create a `DesiredSoftwareSpec` data object and specify the base image that must be applied on the cluster with the `baseImageSpec` property. Optionally, you can specify a vendor add-on to be added to the software specification with the `vendorAddOnSpec` property.

Enabling an Existing Cluster to Use vSphere Lifecycle Manager

If you want to manage the life cycle of a cluster by using a single software specification, you must first enable the vSphere Lifecycle Manager on that cluster. You can use the vSphere Automation APIs to enable a cluster to use the vSphere Lifecycle Manager feature.

Before you enable the vSphere Lifecycle Manager on a cluster, you can check whether the cluster meets all prerequisites. The vSphere Lifecycle Manager can be enabled for a cluster only if the following requirements are met:

- All hosts in the cluster are of version 7.0 or later.
- All hosts in the cluster are stateful.
- All hosts in the cluster include only components that belong to integrated solutions, such as VMware vSAN™ and VMware vSphere® High Availability.
- None of the hosts in the cluster are in the process of active remediation through the VMware vSphere® Update Manager™.
- The cluster has a desired state already created for it.

If you want to run a preliminary check about whether all hosts in the cluster meet these requirements, call the `check_Task(cluster_ID,check_spec)` method of the `com.vmware.esx.settings.clusters.enablement.Software` interface. Pass as arguments the cluster ID, and optionally, a `com.vmware.esx.settings.clusters.enablement.SoftwareTypes.CheckSpec` instance.

The cluster ID represents the unique identifier for a cluster resource. You can retrieve the cluster ID by calling the `getCluster()` method of the `com.vmware.vcenter.ClusterTypes.Summary` instance. `Summary` instances are returned when you filter the clusters available on a vCenter Server instance. To retrieve commonly used information about clusters including their IDs, call the `list(filter_spec)` method of the `com.vmware.vcenter.Cluster` interface. Pass as argument a `com.vmware.vcenter.ClusterTypes.FilterSpec` instance if you want to retrieve information only about clusters that match specific criteria.

You can pass a `CkeckSpec` instance to specify which checks can be skipped during the cluster preliminary check. Though you can skip some checks with this operation, the Image Manager runs all checks during the enablement operation. If you leave the check specification empty, all checks are run for each host in the cluster. You can select among the following checks to be skipped when running a pre-check operation:

- `SoftwareTypes.CheckType.SOFTWARE`. Checks whether there are any orphaned vSphere Installation Bundles (VIBs) and any software that cannot co-exist with the vSphere Lifecycle Manager.
- `SoftwareTypes.CheckType.VERSION`. Checks whether all hosts in the cluster are of version greater than a predefined one.
- `SoftwareTypes.CheckType.STATELESSNESS`. Checks whether there are any stateless hosts in the cluster. The vSphere Lifecycle Manager can be enabled only if the cluster does not contain stateless hosts.
- `SoftwareTypes.CheckType.VUM_REMEDIATION`. Checks whether any of the hosts in the cluster are currently remediated through the VMware vSphere® Update Manager™.
- `SoftwareTypes.CheckType.SOFTWARE_SPECIFICATION_EXISTENCE`. Checks whether there is a software specification already associated with this cluster. In case this check reports that the cluster does not have a software specification, you must first create a draft software specification for this cluster and then commit the draft.

To enable a cluster to be managed with the vSphere Lifecycle Manager, call the `enable_Task(cluster_ID,enable_spec)` method of the `com.vmware.esx.settings.clusters.enablement.Software` interface. Pass as arguments the cluster ID and optionally, a `com.vmware.esx.settings.clusters.enablement.SoftwareTypes.EnableSpec` instance. To specify checks that you want to be skipped during the enablement process, pass the `EnableSpec` instance. Currently, you can only skip the `SoftwareTypes.CheckType.SOFTWARE` check.

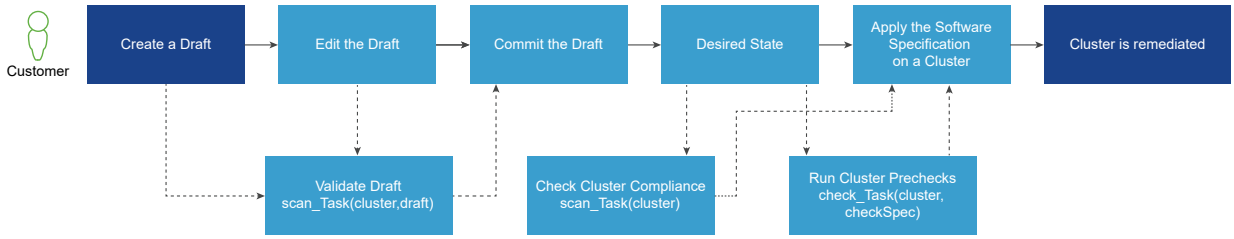
You can also get information about which clusters in your environment are managed with a single software specification. Call the `get(cluster_ID)` method and pass the cluster ID as an argument.

Working with Draft Software Specifications

You create a draft software specification to describe the components of the desired state that you want to apply on a cluster.

The draft software specification is the working copy of the desired software state. Only one user at a time is allowed to edit a single draft for a cluster. Before saving the changes to the edited draft version, you can validate the content of the draft.

Figure 7-3. Drafts Workflow



A typical workflow for working with draft software specifications starts with creating a draft software specification for a specific cluster. If the cluster already has a software specification defined, this action takes the latest committed draft and you can edit its contents according to your needs. If the cluster has no software specification created yet, this method creates an empty draft. The only mandatory item for a draft is a base image of a specific version.

After you complete adding components, an add-on, and a base image, you can save your changes by committing the draft. This operation results in setting the committed draft as the current desired state of the cluster. Before committing the draft software specification, you can validate the contents of the draft or check whether all hosts in the cluster are compliant with the draft.

If the commit operation is successful, the draft becomes the desired state for the cluster. You can now export the software specification and use it, for example, in a ROBO cluster scenario. You can also validate the compliance of the hosts in the cluster against the desired state and then apply the software specification, if feasible.

Creating a Draft Software Specification

To describe the components of a desired state for a cluster, create a draft software specification and save the desired state when ready.

To edit an existing desired state or to create an empty draft software specification, call the `create(cluster_ID)` method of the `com.vmware.esx.settings.clusters.software.Drafts` interface. Pass the cluster ID as an argument to the method. The method returns a draft ID which you can use to add a base image, an add-on, or some components to draft.

Editing a Draft Software Specification

After you have created a draft software specification, use the vSphere Automation APIs to edit its items.

To set a base image to a draft software specification, call the `set(cluster_ID, draft_ID,base_image_spec)` method of the `com.vmware.esx.settings.clusters.software.drafts.software.BaseImage` interface and pass as arguments the cluster and draft IDs, and the base image specification. If the draft contains a

base image, this method overwrites the existing image. The base image specification contains the version of the bootable ESXi that must be included in the desired state. To retrieve details about the base image that is currently present in a draft, call the `get(cluster_ID,draft_ID)` method of the `BaseImage` interface. Pass as arguments the cluster and draft IDs. Use the returned `com.vmware.esx.settings.BaseImageInfo` object to query the version, display name and version, and the release date of the ESXi.

To add an OEM add-on to a draft software specification, call the `set(cluster_ID,draft_ID,addon_spec)` method of the `com.vmware.esx.settings.clusters.software.drafts.software.AddOn` interface. Pass as arguments to this method the cluster and draft IDs, and the add-on specification. If you want to remove an add-on from a draft, call the `delete(cluster_ID,draft_ID)` method of the `AddOn` interface and pass as arguments the cluster and draft IDs.

You can add a component, change the version, or delete an existing component from a draft software specification. To change the version of a component included in a draft, call the `set(cluster_ID,draft_ID,component_ID,version)` method of the `com.vmware.esx.settings.clusters.software.drafts.software.Components` interface. This method adds the component specified with the `component_ID` and `version` arguments to the draft, if it is missing. To remove a component from a draft, call the `delete(cluster_ID,draft_ID,component_ID)` method. You can change multiple components in a draft by calling the `update(cluster_ID,draft_ID,update_spec)` method of the `Components` interface. To specify the components you want to remove, add, or update for a draft, pass a `com.vmware.esx.settings.clusters.software.drafts.software.ComponentsTypes.UpdateSpec` instance as an argument to this method. You can also use the `Components` interface to retrieve information about all components present in a draft, or a single component.

Validating the Draft Software Specification

Before saving a draft and turning it into a desired state for a cluster, you can check whether the specification is complete and valid. You can also check whether the draft specification drifts in any way from the current state of the cluster.

The `Drafts` interface offers two methods for validating your draft. To check whether the draft is complete, there are no conflicts between the draft components, or unresolved dependencies, call the `validate_Task(cluster_ID,draft_ID)` method. Pass as arguments to this method the cluster and draft IDs. You validate whether there were any other drafts committed for this cluster, which will make the current commit operation invalid. The method also validates whether all components defined in the software specification are available in the depot metadata. This method does not run compliance checks against the cluster. To check whether all hosts in the cluster are compliant with the draft software specification, call the `scan_Task(cluster_ID,draft_ID)` method of the `Drafts` interface. Pass as arguments to this method the cluster and draft IDs. This method results in running a comparison between the draft specification and the current state of each host in the cluster.

Committing the Draft Software Specification

When you commit a draft software specification, it becomes the desired state for the cluster.

To save the draft that you created for a cluster, call the `commit_Task(cluster_ID,draft_ID,commit_spec)` method of the `Drafts` interface. The Image Manager component runs a validation check before the draft gets saved to the database. This method returns a commit ID. You can use the ID to retrieve information about a specific commit such as the author of the commit, the time when the draft was committed, and so on.

Working with Desired Software States

When you commit a draft software specification, you make the committed draft the desired state for that cluster. If all hosts in the cluster are compliant with the desired state, you can remediate the cluster.

You can use the methods provided with the `com.vmware.esx.settings.clusters.Software` interface to manage a desired state for a cluster. Before you apply a desired state on a cluster, you can run pre-checks to ensure that all hosts in the cluster are in a good state to be remediated. The pre-checks verify whether any of the hosts in the cluster must be rebooted or is in maintenance mode. You can also check the compliance of the cluster against the desired state. See Checking the Compliance of the Cluster Against the Desired State.

You can export a software specification created for a cluster by using one of the following formats:

- An offline bundle in a ZIP file format.
- An ISO image.
- A JSON file.

Use the vSphere Lifecycle Manager APIs to import a software specification as a draft and then edit it. You have several options for running the import operation depending on the location and format of the desired software state.

Exporting and Importing a Desired State

Use the vSphere Automation APIs to export the desired software state of a cluster. Then you can import the desired state to a different cluster in the same or a different vCenter Server instance.

Exporting a Desired State

To export a desired state of a cluster, use the `export(cluster_ID,export_spec)` method of the `com.vmware.esx.settings.clusters.Software` interface. This method does not export any information about the solutions available on the cluster since the constraints set by these solutions might not be applicable for another cluster. Pass as parameters the cluster ID and an `com.vmware.esx.settings.clusters.SoftwareTypes.ExportSpec` instance. You can export a desired software state by using one of the following options:

- Export an ISO image. Call the `setExportIsoImage(exportIsoImage)` method of the `ExportSpec` instance and pass `true` as an argument. Use the exported ISO image for performing clean installs and for bootstrapping purposes. You can upload the ISO file into the Jetty Web server on the target vCenter Server instance but you cannot use ISO files to manage the life cycle of clusters through the vSphere Lifecycle Manager feature.
- Export an offline bundle in a ZIP file format. Call the `getExportOfflineBundle()` method of the `ExportSpec` instance. You can use the exported offline bundle to create a depot and add its components to the resources managed by the Depot Manager module.
- Export a JSON file holding the desired state specification. Call the `setExportSoftwareSpec(exportSoftwareSpec)` method of the `ExportSpec` instance and pass `true` as argument. You can then reuse the JSON file to apply the desired state that it contains to another cluster in the same or in a different vCenter Server instance. Note that the JSON file holds only the description of the desired state. You must check whether all components described in the JSON file are available in the depot for the target cluster. See Importing a Desired State Specification for information about how you can use a desired state specification for another cluster.

Importing a Desired State Specification

To import a desired state of a cluster and assign it to another cluster in the same or different vCenter Server instance, use the `importSoftwareSpec(cluster_ID,import_spec)` method of the `com.vmware.esx.settings.clusters.software.Drafts` interface. Pass as parameters the cluster ID and a `com.vmware.esx.settings.clusters.software.DraftsTypes.ImportSpec` instance. Use the `ImportSpec` instance to describe the download source and the source type of the imported software specification. Depending on the location and source type of the exported desired state, you can choose from the following import options:

- Import a file from the vCenter Server or your local file system. Call the `setFileId(fileId)` method of the `ImportSpec` instance. Pass as argument the file ID of the software specification which was previously uploaded on the Jetty Web server running on the vCenter Server at `https://<vcenter_FQDN>:9087/vum-fileupload` URL. You can also use this option to import a specification file that resides on your local file system. Make sure you set the source type of the import specification to `SourceType.PUSH` through the `setSourceType(sourceType)` method of the `ImportSpec` instance.

- Import a file that resides on a URI location. Call the `setLocation(location)` method of the `ImportSpec` instance. Pass as argument the URI location of the software specification file. The software specification can be pulled from a URI location with one of the following schemes: `file`, `http`, or `https`. You can use this import mechanism only if you set the source type to `SourceType.PULL`.
- Import a desired state as a JSON string. Call the `setSoftwareSpec(softwareSpec)` method of the `ImportSpec` instance. Pass as argument the JSON string representing the software specification you want to import. Use this mechanism only if you set the source type to `SourceType.JSON_STRING`.

Checking the Compliance of the Cluster Against the Desired State

Before applying a desired state on a cluster, you can scan all hosts in the cluster against the desired state and check the cluster compliance against the desired state.

To check the compliance of all hosts in a cluster, call the `scan_Task(cluster_ID)` method of the `com.vmware.esx.settings.clusters.Software` interface. Pass as an argument the cluster ID. This method compares the desired state against the current state of each host in the cluster and as a result calculates the cluster compliance.

You can retrieve the cluster compliance status by calling the `get(cluster_ID)` method of the `com.vmware.esx.settings.clusters.software.Compliance` interface and passing as argument the cluster ID. As a result you receive a `com.vmware.esx.settings.ClusterCompliance` object. You can use the `ClusterCompliance` instance to retrieve the following information:

- The overall cluster compliance status regarding the target version of the components described within the cluster desired state.
- The impact of applying the desired state on the cluster in case the cluster is non-compliant.
- A list of all compliant hosts in the cluster.
- A list of the incompatible hosts in the cluster.
- A list of the non-compliant hosts in the cluster.
- A list of the unavailable hosts which cannot be checked for compliance against the desired state.
- The ID of the committed draft for which the compliance check is performed. If the `getCommit()` method of the `ClusterCompliance` instance returns `null`, the compliance check is run against a draft software specification.
- The compliance status of each host in the cluster.
- The notifications returned by the compliance check operation.
- The time that the compliance check takes.

A cluster or a host can have one of the following compliance statuses regarding the target versions:

- `ComplianceStatus.COMPLIANT`. The target versions of the components described in the desired state of the cluster are the same as the versions of the components currently present on the hosts in the cluster.
- `ComplianceStatus.NON_COMPLIANT`. The desired state of the cluster describes components with higher versions than the versions of the components currently present on the hosts in the cluster. Non-compliant clusters are those clusters which have orphaned VIBs, or components on the hosts that are not present in the desired state specification.
- `ComplianceStatus.INCOMPATIBLE`. One or more hosts in the cluster have components with higher versions than the components described in the desired state specification.
- `ComplianceStatus.UNAVAILABLE`. The current state of one or more hosts in the cluster cannot be retrieved and as a result the compliance check cannot be performed.

You can check the compliance impact of applying the desired state on a non-compliant cluster by calling the `getImpact()` method of the `ClusterCompliance` instance. A `com.vmware.esx.settings.ComplianceImpact` object is returned. Use the object to retrieve information about the steps you must take to remediate the cluster successfully. You might need to reboot a host or put a host into maintenance mode to remediate the cluster successfully.

Hardware Compatibility Data

The hardware compatibility data contains information about the compatibility between ESXi hosts and ESXi versions.

You can use the `com.vmware.esx.hcl.CompatibilityData` interface to retrieve information about the compatibility data or to update the local compatibility data on a vCenter Server instance. To retrieve information about the compatibility data stored on your vCenter Server instance, call the `get()` method. To update the local compatibility data with the latest version available from the official VMware source, call the `update_Task()` method.

Checking the Hardware Compatibility of an ESXi Host

You can query the hardware compatibility for a host before upgrading to a new ESXi version. You can also download the information generated by the hardware compatibility report.

To use interfaces form the `com.vmware.esx.hcl.hosts` package, you must verify that you have accepted to participate in the CEIP and there is available compatibility data.

You can use the `com.vmware.esx.hcl.hosts.CompatibilityReleases` interface to list available releases for generating a compatibility report for a specific ESXi host. To list the locally available ESXi releases for the host that can be used to generate a compatibility report, call the `list(host_ID)` method. The list includes only major and update releases. Patch releases are not listed.

You can use the `com.vmware.esx.hcl.hosts.CompatibilityReport` interface to generate a hardware compatibility report for an ESXi host against a specific ESXi release. To return the last generated hardware compatibility report for a specific host, call the `get(host_ID)` method. To generate a hardware compatibility report for a specific host against specific ESXi release, call the `create_Task(host_ID,spec)` method.

You can use the `com.vmware.esx.hcl.Reports` interface to download information generated by the hardware compatibility report. To retrieve the URI location for downloading a compatibility report, call the `get(report_ID)` method.

Configuring Remediation Settings

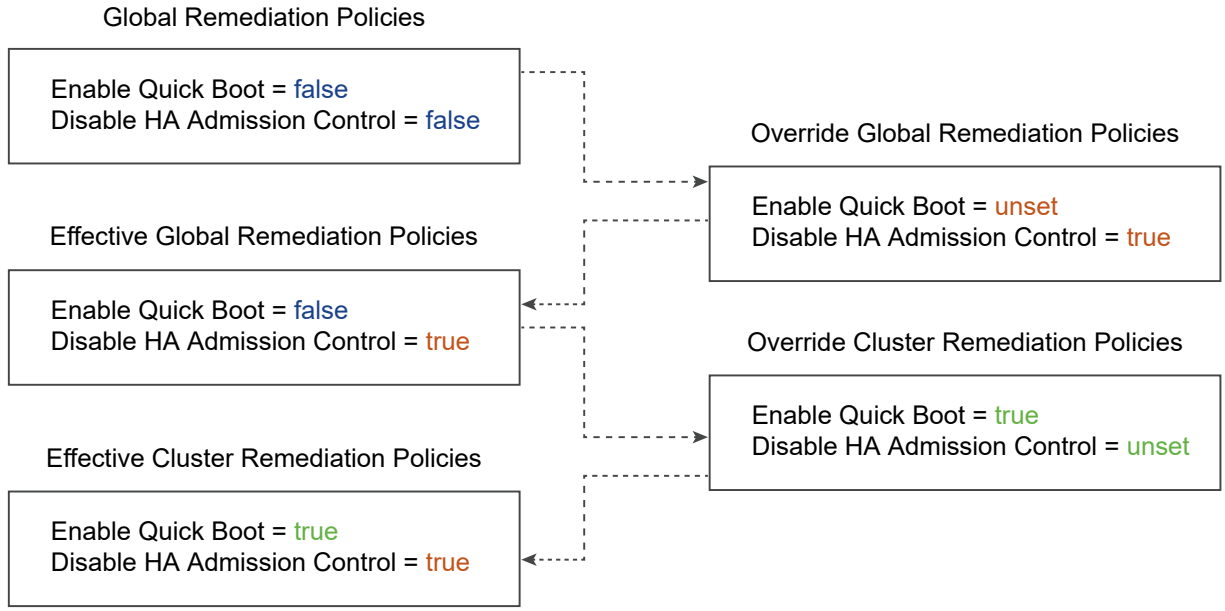
You can control the behavior of the ESXi hosts and virtual machines during the remediation process. You can create a global remediation policy that applies to all clusters in a vCenter Server instance. You can also set a remediation policy to a specific cluster.

When you run cluster compliance checks, the Coordinator module runs a series of checks on each host to determine their state and whether some additional actions must be taken to ensure the success of the remediation operation. In case one or more hosts in the cluster are evaluated as non-compliant, additional checks are run on those hosts to evaluate whether they must be rebooted or put into maintenance mode. Currently, VMware provides a set of behavior controls (remediation policies) regarding the virtual machines and the hosts in a cluster. This set of remediation policies might change with the next vSphere release.

How Remediation Policies Overrides Work

The vSphere Lifecycle Manager provides a default global policy configuration that must be applied on each cluster during remediation. Through the vSphere Automation APIs, you can change the global policies and create some cluster-specific policies. Before remediating a cluster, you can use the APIs to determine the effective global and cluster-specific remediation policies. The following graphic describes how the mechanism of policy overrides works.

Figure 7-4. How Remediation Policies Work



All clusters in a vCenter Server instance inherit the default or the overridden global policy settings unless the global policy is explicitly overridden on a cluster level.

Editing Global or Cluster-Specific Remediation Policies

To view the currently set global remediation policy, call the `get(cluster_ID)` method of the `com.vmware.esx.settings.defaults.clusters.policies.Apply` interface. You receive a `com.vmware.esx.settings.defaults.clusters.policies.ApplyTypes.ConfiguredPolicySpec` instance that contains the configuration settings of the global remediation policy. To edit a global remediation policy, call the `set(policy_spec)` method of the `com.vmware.esx.settings.defaults.clusters.policies.Apply` interface. Pass as an argument a `com.vmware.esx.settings.defaults.clusters.policies.ApplyTypes.ConfiguredPolicySpec` instance and define new values to the global policy settings. To view the effective global remediation policy settings for a cluster, call the `get()` method of the `com.vmware.esx.settings.defaults.clusters.policies.apply.Effective` interface. The method returns an `EffectivePolicySpec` instance that contains the effective global policies applicable for all clusters in your vCenter Server environment.

To view the cluster-specific remediation policies, call the `get(cluster_ID)` method of the `com.vmware.esx.settings.clusters.policies.Apply` interface. The method returns a `com.vmware.esx.settings.clusters.policies.ApplyTypes.ConfiguredPolicySpec` instance that contains the cluster-specific policies to be applied during remediation. To change the cluster-specific policy, call the `set(cluster_ID,policy_spec)` method of the `Apply` interface. Pass as argument a `com.vmware.esx.settings.clusters.policies.ApplyTypes.ConfiguredPolicySpec`

instance and describe the cluster-specific remediation policies. To view the effective cluster-specific policies, call the `get(cluster_ID)` method of the `com.vmware.esx.settings.clusters.policies.apply.Effective` interface. The method returns an `EffectivePolicySpec` instance that describes the effective cluster-specific policies.

Remediation Policy Options

Use the `com.vmware.esx.settings.defaults.clusters.policies.ApplyTypes.ConfiguredPolicySpec` and `com.vmware.esx.settings.clusters.policies.ApplyTypes.ConfiguredPolicySpec` classes to describe a global or cluster-specific remediation policy. For the vSphere 7.0 release, VMware provides the following methods to configure a global or cluster-specific policy.

Method	Description
`setDisableDpm(disableDpm)`	Disable the VMware Distributed Power Management (DPM) feature for all clusters or for a specific cluster. DPM monitors the resource consumption of the virtual machines in a cluster. If the total available resource capacity of the hosts in a cluster is exceeded, DPM powers off (or recommends powering off) one or more hosts after migrating their virtual machines. When resources are considered underutilized and capacity is needed, DPM powers on (or recommends powering on) hosts. Virtual machines are migrated back to these hosts. During the cluster remediation, the vSphere Lifecycle Manager cannot wake up and remediate hosts that are automatically put into a stand-by mode by DPM. These hosts stay non-compliant when DPM turns them on. The vSphere Distributed Resource Scheduler (DRS) is unable to migrate virtual machines to the hosts which are not remediated with the desired state for the cluster. To disable DPM during the cluster remediation, call the `setDisableDpm(disableDpm)` method of the `ConfiguredPolicySpec` instance and pass as argument `true`. By default, the vSphere Lifecycle Manager temporarily disables DPM and turns on the hosts to complete the remediation. DPM is enabled again when the cluster remediation finishes.
`setDisableHac(disableHac)`	Disable the vSphere HA admission control. vSphere HA uses admission control to ensure that a cluster has sufficient resources to guarantee the virtual machines recovery when a host fails. If vSphere HA admission control is enabled during remediation, putting a cluster into maintenance mode fails because vMotion cannot migrate virtual machines within the cluster for capacity reasons. To allow the vSphere Lifecycle Manager to temporary disable vSphere HA admission control, call the `setDisableHac(disableHac)` method of the `ConfiguredPolicySpec` instance and pass as argument `true`. By default, the vSphere HA admission control is enabled because DRS should be able to detect issues with the admission control and disable it to allow the remediation to complete.
`setEvacuateOfflineVms(evacuateOfflineVms)`	Migrate the suspended and powered off virtual machines from the hosts that must enter maintenance mode to other hosts in the cluster. To enable this remediation policy, call the `setEvacuateOfflineVms(evacuateOfflineVms)` method of the `ConfiguredPolicySpec` instance and pass as argument `true`. By default, this setting is disabled in the global remediation policy.

Method	Description
`setFailureAction(failureAction)`	Specify what actions the vSphere Lifecycle Manager must take if a host fails to enter maintenance mode during the remediation. To configure this policy on a global or cluster-specific level, call the `setFailureAction(failureAction)` method of the `ConfiguredPolicySpec` instance. Pass as argument an `ApplyTypes.FailureAction` instance. You can set the number of times that the vSphere Lifecycle Manager tries to put a host into maintenance mode and the delay between the tries. When the threshold is reached and the host failed to enter maintenance mode, the cluster remediation fails. By default, the vSphere Lifecycle Manager tries to put a host into maintenance mode three times with a five minute delay between each try before the cluster remediation fails.
`setPreRemediationPowerAction (preRemediationPowerAction)`	Specify how the power state of the virtual machines must change before the host enters maintenance mode. If DRS is not enabled on a cluster or the automation level of a DRS cluster is not set to fully automated, the Coordinator module fails to remediate the cluster if the remediation requires a reboot or maintenance mode. You can set a policy that powers off or suspends the virtual machines on hosts that must be rebooted or must enter maintenance mode during remediation. The DRS takes care of changing the power state of the virtual machines when the host enters and exits maintenance mode. To set a policy for the power state of the virtual machines during remediation, call the `setPreRemediationPowerAction (preRemediationPowerAction)` method of the `ConfiguredPolicySpec` instance. You can pass as argument one of the following values: ■ `PreRemediationPowerAction.DO_NOT_CHANGE_VMS_POWER_STATE` - ■ `PreRemediationPowerAction.POWER_OFF_VMS` ■ `PreRemediationPowerAction.SUSPEND_VMS` Pass as argument a `PreRemediationPowerAction` instance and define whether the power state of the virtual machines must remain unchanged, or they must be powered off, or suspended. By default, the Coordinator must leave the power state of the virtual machines unchanged.
`setEnableQuickBoot(enableQuickBoot)`	Reduce the reboot time of an ESXi host by skipping all the hardware initialization processes and restarting only the hypervisor. This policy is applicable only if the host platform supports the Quick Boot feature. To enable the Quick Boot feature on the hosts during remediation, call the `setEnableQuickBoot(enableQuickBoot)` method of the `ConfiguredPolicySpec` instance and pass as argument `true`. By default, this policy is disabled.

Remediating an ESXi Cluster

You can use the vSphere Automation APIs to initiate the remediation of a cluster through the vSphere Lifecycle Manager.

To remediate a cluster with a desired state, call the `apply_Task(cluster_ID,apply_spec)` method of the `com.vmware.esx.settings.clusters.Software` interface. Pass as argument a `SoftwareTypes.ApplySpec` instance and specify whether the VMware End User License Agreement (EULA) is accepted. You can also set the minimum commit ID of the draft software specification that must be used for remediating the cluster. Upon successful completion of the remediation task, all hosts in the cluster will have the same software state.

Virtual Machine Configuration and Management

8

A virtual machine is a software computer that, like a physical computer, runs an operating system and applications. The virtual machine consists of a set of specification and configuration files and is backed by the physical resources of a host. Each virtual machine encapsulates a complete computing environment and runs independently of the underlying hardware.

Starting with vSphere 6.5, you can create virtual machines, configure virtual machine settings, and perform power operations on the virtual machines through the vSphere Automation APIs.

Starting with vSphere 7.0, you can use the vSphere Automation APIs to perform various virtual machine management operations. For example, you can deploy virtual machines by using several approaches, clone an existing virtual machine, create an instant clone of a running virtual machine. You can also install VMware Tools which enables you to manage the life cycle and customize the networking and identity settings of the guest operating system installed on the virtual machine.

This chapter includes the following topics:

Creating Virtual Machines

You can use the vSphere Automation APIs to create virtual machines depending on your needs and infrastructure setup.

You can create a basic virtual machine and then configure it according to your needs. You can also create a more comprehensive virtual machine and then edit its settings. To create a virtual machine, you must specify the datastore, resource pool, folder, or host where the virtual machine is placed. Later, you can customize the virtual machine by specifying the boot options, number of CPUs, the guest OS, and virtual NIC. See Creating a Virtual Machine Without a Clone or Template and Configuring Virtual Machines.

You can use a turned off virtual machine to create a VM template from which to deploy other virtual machines. See Create a VM Template in a Content Library from a Virtual Machine. You can also mark a virtual machine as template by calling the `VirtualMachine.MarkAsTemplate` method from the vSphere Web Services APIs. See *vSphere Web Services SDK Programming Guide*.

You can capture a virtual machine or a vApp in an OVF template and store the template in a content library. Then you can use the OVF template to deploy a virtual machine or a vApp in your environment. See Creating Virtual Machines and vApps from Templates in a Content Library.

You can use the vSphere Automation APIs to create a duplicate of an existing virtual machine and customize specific attributes of the clone. Starting with vSphere 7.0, you can also create an instant clone of a running virtual machine that continues running from the current state of the source virtual machine.

Creating a Virtual Machine Without a Clone or Template

You can create a virtual machine by using the `VM.create` method. The method takes as parameter a `CreateSpec` instance that describes the details of the virtual machine.

When you create a virtual machine without a template or clone, you can configure the virtual hardware, including processors, hard disc, memory. To create a virtual machine, you must specify the virtual machine attributes by using the `CreateSpec` class. For example, you can specify a name, boot options, networking, and memory for the new virtual machine.

All attributes are optional except the virtual machine placement information that you must provide by using the `PlacementSpec` class. Use the virtual machine placement specification to set the datastore, cluster, folder, host, or resource pool of the created virtual machine. You must make sure that all these vSphere objects are located in the same data center in a vCenter Server instance.

For more information, refer to the *API Reference* documentation inside the SDK.

Java Example of Creating a Basic Virtual Machine

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

This example uses the information provided in Creating a Virtual Machine Without a Clone or Template.

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

```
...

    private void createBasicVM(
        VMTypes.PlacementSpec vmPlacementSpec, String standardNetworkBacking) {
        // Create the scsi disk as a boot disk
        DiskTypes.CreateSpec bootDiskCreateSpec =
                new DiskTypes.CreateSpec.Builder().setType(
                    DiskTypes.HostBusAdapterType.SCSI)
                    .setScsi(new ScsiAddressSpec.Builder(0l).setUnit(0l)
```

```
                        .build())
                    .setNewVmdk(new DiskTypes.VmdkCreateSpec())
                    .build();

        // Create a data disk
        DiskTypes.CreateSpec dataDiskCreateSpec =
                new DiskTypes.CreateSpec.Builder().setNewVmdk(
                    new DiskTypes.VmdkCreateSpec()).build();
        List<DiskTypes.CreateSpec> disks = Arrays.asList(bootDiskCreateSpec,
            dataDiskCreateSpec);

        // Create a nic with standard network backing
        EthernetTypes.BackingSpec nicBackingSpec =
                new EthernetTypes.BackingSpec.Builder(
                    BackingType.STANDARD_PORTGROUP).setNetwork(
                        standardNetworkBacking).build();
        EthernetTypes.CreateSpec nicCreateSpec =
                new EthernetTypes.CreateSpec.Builder().setStartConnected(true)
                    .setBacking(nicBackingSpec)
                    .build();
        List<EthernetTypes.CreateSpec> nics = Collections.singletonList(
            nicCreateSpec);

        // Specify the boot order
        List<DeviceTypes.EntryCreateSpec> bootDevices = Arrays.asList(
            new DeviceTypes.EntryCreateSpec.Builder(DeviceTypes.Type.ETHERNET)
                .build(),
            new DeviceTypes.EntryCreateSpec.Builder(DeviceTypes.Type.DISK)
                .build());
      VMTypes.CreateSpec vmCreateSpec = new VMTypes.CreateSpec.Builder(
            this.vmGuestOS).setName(BASIC_VM_NAME)
                .setBootDevices(bootDevices)
                .setPlacement(vmPlacementSpec)
                .setNics(nics)
                .setDisks(disks)
                .build();
        System.out.println("\n\n#### Example: Creating Basic VM with spec:\n"
                           + vmCreateSpec);
        this.basicVMId = vmService.create(vmCreateSpec);

...
```

Python Example of Creating a Basic Virtual Machine

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

This example uses the information provided in Creating a Virtual Machine Without a Clone or Template.

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

```
...
```

```
def create_basic_vm(stub_config, placement_spec, standard_network):
    """
    Create a basic VM.

    Using the provided PlacementSpec, create a VM with a selected Guest OS
    and provided name.

    Create a VM with the following configuration:
    * Create 2 disks and specify one of them on scsi0:0 since it's the boot disk
    * Specify 1 ethernet adapter using a Standard Portgroup backing
    * Setup for PXE install by selecting network as first boot device

    Use guest and system provided defaults for most configuration settings.
    """
    guest_os = testbed.config['VM_GUESTOS']

    boot_disk = Disk.CreateSpec(type=Disk.HostBusAdapterType.SCSI,
                                scsi=ScsiAddressSpec(bus=0, unit=0),
                                new_vmdk=Disk.VmdkCreateSpec())
    data_disk = Disk.CreateSpec(new_vmdk=Disk.VmdkCreateSpec())

    nic = Ethernet.CreateSpec(
        start_connected=True,
        backing=Ethernet.BackingSpec(
            type=Ethernet.BackingType.STANDARD_PORTGROUP,
            network=standard_network))

    boot_device_order = [BootDevice.EntryCreateSpec(BootDevice.Type.ETHERNET),
                         BootDevice.EntryCreateSpec(BootDevice.Type.DISK)]

    vm_create_spec = VM.CreateSpec(name=vm_name,
                                   guest_os=guest_os,
                                   placement=placement_spec,
                                   disks=[boot_disk, data_disk],
                                   nics=[nic],
                                   boot_devices=boot_device_order)
    print('\n# Example: create_basic_vm: Creating a VM using spec\n-----')
    print(pp(vm_create_spec))
    print('-----')

    vm_svc = VM(stub_config)
    vm = vm_svc.create(vm_create_spec)

    print("create_basic_vm: Created VM '{}' ({})".format(vm_name, vm))

    vm_info = vm_svc.get(vm)
    print('vm.get({}) -> {}'.format(vm, pp(vm_info)))

    return vm

...
```

Configuring Virtual Machines

You configure a virtual machine in the process of creation by using `com.vmware.vcenter.CteareSpec`. You can later view and edit virtual machine settings by adding or changing the type of the storage controllers, configure the virtual disks, boot options, CPU and memory information, or networks. Virtual machine settings can be configured when cloning, registering and relocating an existing virtual machine.

Name and Location

You specify the display name and the location of the virtual machine by using the `CreateSpec` and `PlacementSpec` classes.

When you create your virtual machine, use the `setName` method of the `CreateSpec` class to pass as argument the display name of the virtual machine.

You must create also a `PlacementSpec` instance that describes the location of the virtual machine in regards to the resources of a given vCenter Server instance. Use the `setPlacement(PlacementSpec placement)` method of the `CreateSpec` class to set the placement information for the virtual machine. You can set one or all of the following vSphere resources: datastore, cluster, folder, host, and resource pool.

Java Example of Configuring the Name and Placement of a Virtual Machine

This example is based on the code in the `CreateDefaultVM.java` and `PlacementHelper.java` sample files.

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

```
...

    private static final String DEFAULT_VM_NAME = "Sample-Default-VM";
    private VM vmService;
    private GuestOS vmGuestOS = GuestOS.WINDOWS_9_64;
    private String defaultVMId;

              ...

    public VMTypes.PlacementSpec getPlacementSpecForCluster(
            StubFactory stubFactory, StubConfiguration sessionStubConfig,
            String datacenterName, String clusterName,
            String vmFolderName, String datastoreName) {

            String clusterId =
                    ClusterHelper.getCluster(stubFactory,
                        sessionStubConfig,
                        datacenterName,
                        clusterName);
            System.out.println("Selecting cluster " + clusterName + "(id="
                               + clusterId + ")");
```

```
            String vmFolderId =
                    FolderHelper.getFolder(stubFactory,
                        sessionStubConfig,
                        datacenterName,
                        vmFolderName);
            System.out.println("Selecting folder " + vmFolderName + "id=("
                               + vmFolderId + ")");

            String datastoreId =
                    DatastoreHelper.getDatastore(stubFactory,
                        sessionStubConfig,
                        datacenterName,
                        datastoreName);
            System.out.println("Selecting datastore " + datastoreName + "(id="
                               + datastoreId + ")");

            /*
             *  Create the virtual machine placement spec with the datastore, resource pool,
             *  cluster and vm folder
             */
            VMTypes.PlacementSpec vmPlacementSpec = new VMTypes.PlacementSpec();
            vmPlacementSpec.setDatastore(datastoreId);
            vmPlacementSpec.setCluster(clusterId);
            vmPlacementSpec.setFolder(vmFolderId);

            return vmPlacementSpec;
        }

    private void createDefaultVM() {
        VMTypes.PlacementSpec vmPlacementSpec =
                     this.getPlacementSpecForCluster(
                     this.vapiAuthHelper.getStubFactory(),
                     this.sessionStubConfig,
                     this.datacenterName,
                     this.clusterName,
                     this.vmFolderName,
                     this.datastoreName);


        VMTypes.CreateSpec vmCreateSpec =
                new VMTypes.CreateSpec.Builder(this.vmGuestOS)
                    .setName(DEFAULT_VM_NAME)
                    .setPlacement(vmPlacementSpec)
                    .build();

                ...
    }
```

Python Example of Configuring the Placement of a Virtual Machine

This example is based on the code in the `vm_placement_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_placement_spec_for_resource_pool(stub_config,
                                         datacenter_name,
                                         vm_folder_name,
                                         datastore_name):
    """
    Returns a VM placement spec for a resourcepool. Ensures that the
    vm folder and datastore are all in the same datacenter which is specified.
    """
    resource_pool = resource_pool_helper.get_resource_pool(stub_config,
                                                           datacenter_name)

    folder = folder_helper.get_folder(stub_config,
                                      datacenter_name,
                                      vm_folder_name)

    datastore = datastore_helper.get_datastore(stub_config,
                                               datacenter_name,
                                               datastore_name)

    # Create the vm placement spec with the datastore, resource pool and vm
    # folder
    placement_spec = VM.PlacementSpec(folder=folder,
                                      resource_pool=resource_pool,
                                      datastore=datastore)

    print("get_placement_spec_for_resource_pool: Result is '{}'".
          format(placement_spec))
    return placement_spec
```

Hardware Version

The hardware version of a virtual machine reflects the virtual hardware features supported by a virtual machine. These features depend on the physical hardware available on the ESXi host on which the virtual machine is running.

Virtual hardware features include the BIOS and Extensible Firmware Interface (EFI), the maximum number of CPUs, the maximum memory configuration, and other hardware characteristics.

When you create a virtual machine, the default hardware version of the virtual machine is the most recent version available on the host where the virtual machine is created. For information about the latest VMware products and virtual hardware versions, see Virtual machine hardware versions (1003746).

To set a different than the default hardware version, call the `setHardwareVersion(hardwareVersion)` function of the `com.vmware.vcenter.VMTypes.CreateSpec` class. Use the `HardwareTypes.Version` class to define a valid hardware version for a virtual machine. For information about the hardware features available for the virtual hardware versions, see Hardware features available with virtual machine compatibility settings (2051652).

You can set a lower virtual hardware version of a virtual machine than the highest supported by the ESXi host on which the virtual machine is running. Setting a lower hardware version can provide flexibility and is useful in the following cases:

- To help you standardize testing and deployment in your environment.
- In case you do not need the hardware features of the latest hardware version of the host.
- To maintain compatibility with hosts with a lower hardware version.

Boot Options

You can configure the boot options of a virtual machine by using the `setBoot(CreateSpec boot)` method of the `CreateSpec` class.

The method takes as argument the `BootTypes.CreateSpec` class. You can select one of the following settings when booting the virtual machine:

- Delay - Indicates a delay in milliseconds before starting the firmware boot process when the virtual machine is powered on.
- Retry - Indicates whether the virtual machine automatically retries to boot after a failure.
- Retry delay - Indicates a delay in milliseconds before retrying the boot process after a failure.
- Enter setup mode - If set to `true`, indicates that the firmware boot process automatically enters BIOS setup mode the next time the virtual machine boots. The virtual machine resets this flag to `false` once it enters setup mode.
- EFI legacy boot - If set to `true`, indicates that the EFI legacy boot mode is used.

Java Example of Configuring the Boot Options of a Virtual Machine

This example is based on the code in the `BootConfiguration.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.

```
...

    private String vmName;
    private String vmId;
    private BootTypes.Info originalBootInfo;
    private Boot bootService;

                    ...
```

```
           this.bootService = vapiAuthHelper.getStubFactory().createStub(Boot.class,
                      this.sessionStubConfig);

        System.out.println("\n\n#### Setup: Get the virtual machine id");
        this.vmId = VmHelper.getVM(vapiAuthHelper.getStubFactory(),
              sessionStubConfig,
              vmName);
         // Print the current boot configuration
         System.out.println("\n\n#### Print the original Boot Info");
         BootTypes.Info bootInfo = this.bootService.get(this.vmId);
         System.out.println(bootInfo);

         // Save the current boot info to verify that we have cleaned up properly
         this.originalBootInfo = bootInfo;

         System.out.println(
             "\n\n#### Example: Update firmware to EFI for boot configuration.");
         BootTypes.UpdateSpec bootUpdateSpec = new BootTypes.UpdateSpec.Builder()
             .setType(BootTypes.Type.EFI)
             .build();
         this.bootService.update(this.vmId, bootUpdateSpec);
         System.out.println(bootUpdateSpec);
         bootInfo = this.bootService.get(this.vmId);
         System.out.println(bootInfo);

         System.out.println(
             "\n\n#### Example: Update boot firmware to tell it to enter setup"
             + " mode on next boot.");
         bootUpdateSpec = new BootTypes.UpdateSpec.Builder()
             .setEnterSetupMode(true)
             .build();
         this.bootService.update(this.vmId, bootUpdateSpec);
         System.out.println(bootUpdateSpec);
         bootInfo = this.bootService.get(this.vmId);
         System.out.println(bootInfo);

         System.out.println(
             "\n\n#### Example: Update firmware to introduce a delay in boot "
             + "process and automatically reboot after a failure to boot, "
             + "retry delay = 30000 ms.");
         bootUpdateSpec = new BootTypes.UpdateSpec.Builder()
             .setDelay(10000l)
             .setRetry(true)
             .setRetryDelay(30000l)
             .build();
         this.bootService.update(this.vmId, bootUpdateSpec);
         bootInfo = this.bootService.get(this.vmId);
         System.out.println(bootInfo);

...
```

Python Example of Configuring the Boot Options

The following example is based on the code of the `boot.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.

```
...

"""
Demonstrates how to configure the settings used when booting a virtual machine.

Sample Prerequisites:
The sample needs an existing VM.
"""

vm = None
vm_name = None
stub_config = None
boot_svc = None
cleardata = False
orig_boot_info = None

...

def run():
    global vm
    vm = get_vm(stub_config, vm_name)
    if not vm:
        exit('Sample requires an existing vm with name ({}). '
             'Please create the vm first.'.format(vm_name))
    print("Using VM '{}' ({}) for Boot Sample".format(vm_name, vm))

    # Create Boot stub used for making requests
    global boot_svc
    boot_svc = Boot(stub_config)

    print('\n# Example: Get current Boot configuration')
    boot_info = boot_svc.get(vm)
    print('vm.hardware.Boot.get({}) -> {}'.format(vm, pp(boot_info)))

    # Save current Boot info to verify that we have cleaned up properly
    global orig_boot_info
    orig_boot_info = boot_info

    print('\n# Example: Update firmware to EFI for Boot configuration')
    update_spec = Boot.UpdateSpec(type=Boot.Type.EFI)
    print('vm.hardware.Boot.update({}, {})'.format(vm, update_spec))
    boot_svc.update(vm, update_spec)
    boot_info = boot_svc.get(vm)
    print('vm.hardware.Boot.get({}) -> {}'.format(vm, pp(boot_info)))

    print('\n# Example: Update boot firmware to tell it to enter setup mode on '
          'next boot')
```

```
    update_spec = Boot.UpdateSpec(enter_setup_mode=True)
    print('vm.hardware.Boot.update({}, {})'.format(vm, update_spec))
    boot_svc.update(vm, update_spec)
    boot_info = boot_svc.get(vm)
    print('vm.hardware.Boot.get({}) -> {}'.format(vm, pp(boot_info)))

    print('\n# Example: Update boot firmware to introduce a delay in boot'
          ' process and to reboot')
    print('# automatically after a failure to boot. '
          '(delay=10000 ms, retry=True,')
    print('# retry_delay=30000 ms')
    update_spec = Boot.UpdateSpec(delay=10000,
                                  retry=True,
                                  retry_delay=30000)
    print('vm.hardware.Boot.update({}, {})'.format(vm, update_spec))
    boot_svc.update(vm, update_spec)
    boot_info = boot_svc.get(vm)
    print('vm.hardware.Boot.get({}) -> {}'.format(vm, pp(boot_info)))

...
```

Guest Operating System

The guest operating system that you specify affects the supported devices and available number of virtual CPUs.

When you create a virtual machine, you specify the guest operating system by using the `setGuestOS(GuestOS guestOS)` method of the `VMTypes.CreateSpec` class. The `GuestOS` class defines the valid guest OS types that you can choose from for configuring a virtual machine.

After the create operation finishes successfully, you can install the guest operating system on the mew virtual machine in the same way as you install it on a physical machine. For further information on installing a guest operating system, refer to the *Guest Operating System Installation Guide* at http://partnerweb.vmware.com/GOSIG/home.html and the *vSphere Virtual Machine Administration* guide.

Starting with vSphere 7.0, you can use the vSphere Automation APIs to install the VMware Tool on the quest operating system and perform some guest OS customizations. See Installing VMware Tools.

CPU and Memory

The `CreateSpec` class allows you to specify the CPU and memory configuration of a virtual machine.

To change the CPU and memory configuration settings, use the `CpuTypes.UpdateSpec` and `MemoryTypes.UpdateSpec` classes.

CPU Configuration

You can set the number of CPU cores in the virtual machine by using the `setCount` method of the `CpuTypes.UpdateSpec` class. The supported range of CPU cores depends on the guest operating system and virtual hardware version of the virtual machine. If you set `CpuTypes.Info.getHotAddEnabled()` and `CpuTypes.Info.getHotRemoveEnabled()` to `true`, you allow virtual processors to be added or removed from the virtual machine at runtime.

Memory Configuration

You can set the memory size of a virtual machine by using the `setSizeMiB` method of the `MemoryTypes.UpdateSpec` class. The supported range of memory sizes depends on the configured guest operating system and virtual hardware version of the virtual machine. If you set `MemoryTypes.UpdateSpec.setHotAddEnabled()` to `true` while the virtual machine is not powered on, you enable adding memory while the virtual machine is running.

Java Example of Configuring the CPU and Memory of a Virtual Machine

This example is based on the code in the `CpuConfiguration.java` and `MemoryConfiguration.java` sample files.

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

```
...

    private String vmName;
    private String vmId;
    private Memory memoryService;
    private Cpu cpuService;
...

        this.memoryService = vapiAuthHelper.getStubFactory().createStub(Memory.class,
                    this.sessionStubConfig);

        this.vmId = VmHelper.getVM(vapiAuthHelper.getStubFactory(), sessionStubConfig,
vmName);

        // Update the memory size of the virtual machine
        MemoryTypes.UpdateSpec memoryUpdateSpec = new
MemoryTypes.UpdateSpec.Builder().setSizeMiB(8 * 1024l).build();
        memoryService.update(this.vmId, memoryUpdateSpec);
        memoryInfo = memoryService.get(this.vmId);

        // Enable adding memory while the virtual machine is running
        memoryUpdateSpec = new
MemoryTypes.UpdateSpec.Builder().setHotAddEnabled(true).build();
        memoryService.update(this.vmId, memoryUpdateSpec);
             ...

        this.cpuService = vapiAuthHelper.getStubFactory().createStub(Cpu.class,
            this.sessionStubConfig);
```

```
          // Get the current CPU information
        CpuTypes.Info cpuInfo = cpuService.get(this.vmId);

         // Update the number of CPU cores
        CpuTypes.UpdateSpec cpuUpdateSpec = new CpuTypes.UpdateSpec.Builder()
            .setCount(2l).build();
        cpuService.update(this.vmId, cpuUpdateSpec);
        cpuInfo = cpuService.get(this.vmId);

         // Update the number of cores per socket in the virtual machine and
         // allow CPU cores to be added to the virtual machine while it is running
        cpuUpdateSpec = new
CpuTypes.UpdateSpec.Builder().setCoresPerSocket(2l).setHotAddEnabled(true).build();
        cpuService.update(this.vmId, cpuUpdateSpec);
...
```

Python Example of Configuring the CPU and Memory of a Virtual Machine

These examples are based on the code in the `cpu.py` and `memory.py` sample files.

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

The following example shows how you can update the CPU configuration of a virtual machine.

```
...

vm = None
vm_name = None
stub_config = None
cpu_svc = None
cleardata = False
orig_cpu_info = None

...
        server, username, password, cleardata, skip_verification, vm_name = \
            parse_cli_args_vm(testbed.config['VM_NAME_DEFAULT'])
        stub_config = vapiconnect.connect(server,
                                          username,
                                          password,
                                          skip_verification)
...
def run():
    global vm
    vm = get_vm(stub_config, vm_name)
    if not vm:
        exit('Sample requires an existing vm with name ({}). '
             'Please create the vm first.'.format(vm_name))
    print("Using VM '{}' ({}) for Cpu Sample".format(vm_name, vm))

    # Create CPU stub used for making requests
    global cpu_svc
    cpu_svc = Cpu(stub_config)
```

```
    # Get the current CPU configuration
    cpu_info = cpu_svc.get(vm)
    print('vm.hardware.Cpu.get({}) -> {}'.format(vm, pp(cpu_info)))

    # Save current CPU info to verify that we have cleaned up properly
    global orig_cpu_info
    orig_cpu_info = cpu_info

    # Update the number of CPU cores of the virtual machine
    update_spec = Cpu.UpdateSpec(count=2)
    print('vm.hardware.Cpu.update({}, {})'.format(vm, update_spec))
    cpu_svc.update(vm, update_spec)

    # Get the new CPU configuration
    cpu_info = cpu_svc.get(vm)
    print('vm.hardware.Cpu.get({}) -> {}'.format(vm, pp(cpu_info)))

    # Update the number of cores per socket and
    # enable adding CPUs while the virtual machine is running
    update_spec = Cpu.UpdateSpec(cores_per_socket=2, hot_add_enabled=True)
    print('vm.hardware.Cpu.update({}, {})'.format(vm, update_spec))
    cpu_svc.update(vm, update_spec)
...
```

The following example demonstrates how you can add memory to a running virtual machine.

```
...

vm = None
vm_name = None
stub_config = None
memory_svc = None
cleardata = False
orig_memory_info = None

...
    server, username, password, cleardata, skip_verification, vm_name = \
            parse_cli_args_vm(testbed.config['VM_NAME_DEFAULT'])
    stub_config = vapiconnect.connect(server,
                                      username,
                                      password,
                                      skip_verification)

...

    global vm
    vm = get_vm(stub_config, vm_name)
    if not vm:
        exit('Sample requires an existing vm with name ({}). '
             'Please create the vm first.'.format(vm_name))
    print("Using VM '{}' ({}) for Memory Sample".format(vm_name, vm))

    # Create Memory stub used for making requests
    global memory_svc
```

```
    memory_svc = Memory(stub_config)

    # Get the current Memory configuration
    memory_info = memory_svc.get(vm)
    print('vm.hardware.Memory.get({}) -> {}'.format(vm, pp(memory_info)))

    # Update the memory size of the virtual machine
    update_spec = Memory.UpdateSpec(size_mib=8 * 1024)
    print('vm.hardware.Memory.update({}, {})'.format(vm, update_spec))
    memory_svc.update(vm, update_spec)

    # Get the new Memory configuration
    memory_info = memory_svc.get(vm)
    print('vm.hardware.Memory.get({}) -> {}'.format(vm, pp(memory_info)))

    # Enable adding memory while the virtual machine is running
    update_spec = Memory.UpdateSpec(hot_add_enabled=True)
    print('vm.hardware.Memory.update({}, {})'.format(vm, update_spec))
    memory_svc.update(vm, update_spec)

...
```

Networks

You configure network settings so that a virtual machine can communicate with the host and with other virtual machines. When you configure a virtual machine, you can add network adapters (NICs) and specify the adapter type.

You can add virtual Ethernet adapters to a virtual machine by using the `VMTypes.CreateSpec.setNics` method. Pass as argument a `List` of `EthernetTypes.CreateSpec` objects that provide the configuration information of the created virtual Ethernet adapters. You can set the MAC address type to `EthernetTypes.MacAddressType.MANUAL`, `EthernetTypes.MacAddressType.GENERATED`, or `EthernetTypes.MacAddressType.ASSIGNED`. Select `MANUAL` to specify the MAC address explicitly.

You can specify also the physical resources that back a virtual Ethernet adapter by using the `EthernetTypes.BackingSpec.setType` method. The method takes as argument one of the following types: `EthernetTypes.BackingType.STANDARD_PORTGROUP`, `HOST_DEVICE`, `DISTRIBUTED_PORTGROUP`, or `OPAQUE_NETWORK`.

Java Example of Configuring the Virtual Machine Network

This example is based on the code in the `EthernetConfiguration.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.

```
...

    private String vmName;
    private String datacenterName;
```

```
    private String stdPortgroupName;
    private String distPortgroupName;
    private String vmId;
    private List<String> createdNics = new ArrayList<String>();
    private Power powerService;
    private Ethernet ethernetService;

...
        this.ethernetService = vapiAuthHelper.getStubFactory().createStub(
            Ethernet.class, this.sessionStubConfig);

        // Get the virtual machine ID
        this.vmId = VmHelper.getVM(vapiAuthHelper.getStubFactory(),
            sessionStubConfig,
            vmName);

        // List all Ethernet adapters of the virtual machine
        List<EthernetTypes.Summary> nicSummaries = this.ethernetService.list(
            this.vmId);
        System.out.println("\n\n#### List of all Ethernet NICS on the VM:\n"
                           + nicSummaries);

        // Get info for each Ethernet adapter on the VM
        System.out.println("\n\n####Print info for each Ethernet NIC on the"
                                   + " vm.");
        for (EthernetTypes.Summary ethSummary : nicSummaries) {
            EthernetTypes.Info ethInfo = this.ethernetService.get(vmId,
                ethSummary.getNic());
            System.out.println(ethInfo);
        }

        // Create Ethernet NIC by using STANDARD_PORTGROUP with default settings
        String stdNetworkId = NetworkHelper.getStandardNetworkBacking(
            this.vapiAuthHelper.getStubFactory(), sessionStubConfig,
            this.datacenterName, this.stdPortgroupName);
        EthernetTypes.CreateSpec nicCreateSpec =
                new EthernetTypes.CreateSpec.Builder().setBacking(
                    new EthernetTypes.BackingSpec.Builder(
                        EthernetTypes.BackingType.STANDARD_PORTGROUP)
                            .setNetwork(stdNetworkId).build()).build();
        String nicId = this.ethernetService.create(this.vmId, nicCreateSpec);
        this.createdNics.add(nicId);
        EthernetTypes.Info nicInfo = this.ethernetService.get(this.vmId, nicId);

       // Update the Ethernet NIC with a different backing
        EthernetTypes.UpdateSpec nicUpdateSpec = new
EthernetTypes.UpdateSpec.Builder().setBacking(
                    new
EthernetTypes.BackingSpec.Builder(EthernetTypes.BackingType.STANDARD_PORTGROUP)
                            .setNetwork(stdNetworkId).build()).build();
        this.ethernetService.update(this.vmId, lastNicId, nicUpdateSpec);
        nicInfo = this.ethernetService.get(this.vmId, lastNicId);

        // Update the Ethernet NIC configuration
        nicUpdateSpec = new EthernetTypes.UpdateSpec.Builder()
```

```
            .setAllowGuestControl(false)
            .setStartConnected(false)
            .setWakeOnLanEnabled(false)
            .build();
        this.ethernetService.update(this.vmId, lastNicId, nicUpdateSpec);
        nicInfo = this.ethernetService.get(this.vmId, lastNicId);

        // Powering on the VM to connect the virtual Ethernet adapter to its backing
        this.powerService.start(this.vmId);
        nicInfo = this.ethernetService.get(this.vmId, lastNicId);

        // Connect Ethernet NIC after powering on the VM
        this.ethernetService.connect(this.vmId, lastNicId);
        nicInfo = this.ethernetService.get(this.vmId, lastNicId);

        // Disconnect Ethernet NIC after powering on VM
        this.ethernetService.disconnect(this.vmId, lastNicId);
        nicInfo = this.ethernetService.get(this.vmId, lastNicId);
...
```

Python Example of Configuring the Virtual Machine Network

This example is based on the code in the `ethernet.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.

```
...

vm = None
vm_name = None
stub_config = None
ethernet_svc = None
cleardata = False
nics_to_delete = []
orig_nic_summaries = None
...

        server, username, password, cleardata, skip_verification, vm_name = \
            parse_cli_args_vm(testbed.config['VM_NAME_DEFAULT'])
        stub_config = vapiconnect.connect(server,
                                          username,
                                          password,
                                          skip_verification)

    global vm
    vm = get_vm(stub_config, vm_name)
    if not vm:
        exit('Sample requires an existing vm with name ({}). '
             'Please create the vm first.'.format(vm_name))
    print("Using VM '{}' ({}) for Disk Sample".format(vm_name, vm))

    # Get standard portgroup to use as backing for sample
    standard_network = network_helper.get_standard_network_backing(
```

```
        stub_config,
        testbed.config['STDPORTGROUP_NAME'],
        testbed.config['VM_DATACENTER_NAME'])

    # Create Ethernet stub used for making requests
    global ethernet_svc
    ethernet_svc = Ethernet(stub_config)
    vm_power_svc = Power(stub_config)
    nic_summaries = ethernet_svc.list(vm=vm)

    # Save current list of Ethernet adapters to verify that we have cleaned
    # up properly
    global orig_nic_summaries
    orig_nic_summaries = nic_summaries

    global nics_to_delete

    # Create Ethernet Nic using STANDARD_PORTGROUP with the default settings
    nic_create_spec = Ethernet.CreateSpec(
        backing=Ethernet.BackingSpec(
            type=Ethernet.BackingType.STANDARD_PORTGROUP,
            network=standard_network))
    nic = ethernet_svc.create(vm, nic_create_spec)
    nics_to_delete.append(nic)
    nic_info = ethernet_svc.get(vm, nic)

    # Create Ethernet Nic by using STANDARD_PORTGROUP
    nic_create_spec = Ethernet.CreateSpec(
        start_connected=True,
        allow_guest_control=True,
        mac_type=Ethernet.MacAddressType.MANUAL,
        mac_address='01:23:45:67:89:10',
        wake_on_lan_enabled=True,
        backing=Ethernet.BackingSpec(
            type=Ethernet.BackingType.STANDARD_PORTGROUP,
            network=standard_network))
    nic = ethernet_svc.create(vm, nic_create_spec)
    nics_to_delete.append(nic)
    nic_info = ethernet_svc.get(vm, nic)

    # Update the Ethernet NIC with a different backing
    nic_update_spec = Ethernet.UpdateSpec(
        backing=Ethernet.BackingSpec(
            type=Ethernet.BackingType.STANDARD_PORTGROUP,
            network=standard_network))
    ethernet_svc.update(vm, nic, nic_update_spec)
    nic_info = ethernet_svc.get(vm, nic)

    # Update the Ethernet NIC configuration
    nic_update_spec = Ethernet.UpdateSpec(
        wake_on_lan_enabled=False,
        mac_type=Ethernet.MacAddressType.GENERATED,
        start_connected=False,
        allow_guest_control=False)
    ethernet_svc.update(vm, nic, nic_update_spec)
```

```
    nic_info = ethernet_svc.get(vm, nic)

    # Powering on the VM to connect the virtual Ethernet adapter to its backing
    vm_power_svc.start(vm)
    nic_info = ethernet_svc.get(vm, nic)

    # Connect the Ethernet NIC after powering on the VM
    ethernet_svc.connect(vm, nic)

    # Disconnect the Ethernet NIC while the VM is powered on
    ethernet_svc.disconnect(vm, nic)
...
```

Managing Virtual Machines

Virtual machines can be configured like physical computers. You can change the guest operating system settings after installing VMware Tools. You can add and remove virtual machines from the vCenter Server inventory. You can also move virtual machines from one host or storage location to another.

Filtering Virtual Machines

You can retrieve commonly used information about virtual machines that match specific criteria. You can retrieve information for up to 4000 virtual machines in a single vCenter Server instance.

You can retrieve a list of virtual machines in a single vCenter Server instance by filtering the results based on a specific requirement. For example, you can use as filter criteria the power state of the virtual machines, or the host, cluster, data center, folder, or resource pool that must contain the virtual machines. In case you specify multiple filter criteria, only virtual machines that match all filter criteria are returned.

To retrieve a list of the virtual machines that match your specific criteria, call the `list` methods of the `VM` service. The method takes as parameter the `VMTypes.FilterSpec` instance that you can use to provide your filter criteria.

Java Example of Filtering Virtual Machines

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

The following code example shows how you can retrieve the VM ID of a virtual machine with a specific name.

This example uses the information provided in Filtering Virtual Machines.

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

```
...

    public static String getVM(
```

```
        StubFactory stubFactory, StubConfiguration sessionStubConfig,
        String vmName) {
        VM vmService = stubFactory.createStub(VM.class, sessionStubConfig);

        // Get summary information about the virtual machine
        VMTypes.FilterSpec vmFilterSpec = new VMTypes.FilterSpec.Builder()
            .setNames(Collections.singleton(vmName)).build();
        List<VMTypes.Summary> vmList = vmService.list(vmFilterSpec);
        assert vmList.size() > 0 && vmList.get(0).getName().equals(
            vmName) : "VM with name " + vmName + " not found";

        return vmList.get(0).getVm();
    }
...
```

Python Example of Filtering Virtual Machines

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

This example uses the information provided in Filtering Virtual Machines.

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_vms(stub_config, vm_names):
    """Return identifiers of a list of vms"""
    vm_svc = VM(stub_config)
    vms = vm_svc.list(VM.FilterSpec(names=vm_names))

    if len(vms) == 0:
        print('No vm found')
        return None

    print("Found VMs '{}' ({})".format(vm_names, vms))
    return vms
```

Installing VMware Tools

VMware Tools is a set of drivers and utilities that you install on the guest operating system of a virtual machine to enhance the performance of the guest OS. VMware Tools also improve the management of the virtual machine. For each guest OS, VMware provides a specific binary-compatible version of VMware Tools.

Before you install VMware tools, you must install and boot the guest operating system.

To mount and unmount the VMware Tools installer CD as a CD-ROM for the guest operating system, use the methods of the `com.vmware.vcenter.vm.tools.Installer` interface.

To mount the VMware Tools, call the `connect(vm_ID)` method of the `Installer` interface. On Windows guest operating systems with activated Autorun feature, this method automatically initiates the installation of the VMware Tools and requires a user input to complete. On other guest operating systems, this method only mounts the VMware Tools disk image on the virtual CD/DVD drive and the user is required to do some guest OS-specific actions. For example, for some Linux distributions, the user is required to extract the contents of the VMware Tools installation archive and run the installer.

To unmount the VMware Tools installer CD image from the virtual CD/DVD drive, call the `disconnect(vm_ID)` method of the `Installer` interface.

To monitor the status of the VMware Tools installation, you can first retrieve the properties of the VMware Tools by calling the `get(vm_VM)` method of the `com.vmware.vcenter.vm.Tools` interface. The method returns a `ToolsTypes.Info` object which you can use to call the `getVersionStatus()` and `getRunState()` methods.

To upgrade the VMware Tools, call the `upgrade(vm_ID, commandLineOptions)` method of the `Tools` interface. The method takes as arguments the ID of the virtual machine on which the VMware Tools is installed and running. Use the `commandLineOptions` argument to specify the command-line options that you want to pass to the installer to modify the tools installation procedure. You can monitor the upgrade operation in the same way you monitor the installation operation.

To update the properties of the VMware Tools, call the `update(vm_ID, update_spec)` method of the `Tools` interface. Pass as argument a `ToolsTypes.UpdateSpec` object and define the tools upgrade policy settings for the virtual machine by calling the `setUpgradePolicy(upgradePolicy)` method.

Performing Virtual Machine Power Operations

You can start, stop, reboot, and suspend virtual machines by using the methods of the `Power` class.

A virtual machine can have one of the following power states:

- `PowerTypes.State.POWERED_ON` - Indicates that the virtual machine is running. If a guest operating system is not currently installed, you can perform the guest OS installation in the same way as for a physical machine.
- `PowerTypes.State.POWERED_OFF` - Indicates that the virtual machine is not running. You can still update the software on the physical disk of the virtual machine, which is impossible for physical machines.
- `PowerTypes.State.SUSPENDED` - Indicates that the virtual machine is paused and can be resumed. This state is the same as when a physical machine is in standby or hibernate state.

To perform a power operation on a virtual machine, you can use one of the methods of the `Power` class. Before you call one of the methods to change the power state of a virtual machine, you must first check the current state of the virtual machine by using the `Power.get` method. Pass as argument the virtual machine identifier.

Following is a list if the power operations:

- `Power.start` - Powers on a powered off or suspended virtual machine. The method takes as argument the virtual machine identifier.
- `Power.stop` - Powers off a powered on or suspended virtual machine. The method takes as argument the virtual machine identifier.
- `Power.suspend` - Pauses all virtual machine activity for a powered on virtual machine. The method takes as argument the virtual machine identifier.
- `Power.reset` - Shuts down and restarts the guest operating system without powering off the virtual machine. Although this method functions as a `stop` method that is followed by a `start` method, the two operations are atomic with respect to other clients, meaning that other power operations cannot be performed until the `reset` method completes.

Java Example of Powering On a Virtual Machine

This example is based on the code in the `EthernetConfiguration.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.

```
...

    private String vmName;
    private String vmId;
    private Power powerService;

...
      this.powerService = vapiAuthHelper.getStubFactory().createStub(
            Power.class, this.sessionStubConfig);
      this.vmId = VmHelper.getVM(vapiAuthHelper.getStubFactory(),
            sessionStubConfig,
            vmName);
      this.powerService.start(this.vmId);

...
```

Python Example of Powering On a Virtual Machine

This example is based on the code in the `ethernet.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.

```
...

   vm = None
   vm_name = None
   stub_config = None
```

```
    server, username, password, cleardata, skip_verification, vm_name = \
             parse_cli_args_vm(testbed.config['VM_NAME_DEFAULT'])
     stub_config = vapiconnect.connect(server, username, password, skip_verification)

     # Get the virtual machine ID
     vm = get_vm(stub_config, vm_name)

     # Create the Power stub  for running power operations on virtual machines
     vm_power_svc = Power(stub_config)
     vm_power_svc.start(vm)
```

Registering and Unregistering Virtual Machines

When you create a virtual machine, it becomes part of the vCenter Server inventory and is registered to the host and vCenter Server. If you remove a virtual machine from the vCenter Server inventory, it becomes unusable. Virtual machine files remain in the same datastore but you cannot power on the virtual machine when it is not registered in the inventory.

You can temporarily remove a virtual machine from vCenter Server by unregistering it. Virtual machine files are not deleted from the datastore. Though all high-level information about the virtual machine such as statistics, resource pool association, permissions, and alarms, is removed from the host and the vCenter Server instance. To remove a virtual machine from the inventory, call the `unregister(vm_ID)` method of the `VM` service and pass as argument the ID of the virtual machine.

To restore a virtual machine to the vCenter Server inventory, and make it usable again, call the `register(register_spec)` method of the `VM` service. You pass as argument a `VMTypes.RegisterSpec` instance that contains information about the current location of the virtual machine files on the datastore. You can also define the location within the vCenter Server inventory, for example, the cluster, folder, or the host, where you want to register the virtual machine. After registration, the virtual machine takes its resources (CPU, memory, and so on) from the resource pool or host to which it is registered.

If you no longer need a virtual machine and you want to free up datastore space, you can permanently delete a virtual machine from the inventory. Call the `delete(vm_ID)` method of the `VM` service and pass as argument the ID of the virtual machine. Upon a successful completion of the operation, the virtual machine files are removed from the datastore, including the configuration file and the virtual disk files.

Virtual Machine Guest Operations

The vSphere Automation APIs enable you to run operations on the guest operating system such as running scripts, performing power operations, and customizing the network and identity settings.

Upload and Run a Script on a Guest Operating System

You can create a directory, create files, copy a script, and run the script on a guest operating system.

Before you perform operations on the guest operating system, you must prepare the environment. You must create the `Process.CreateSpec` for initiating processes in the guest and create the `Transfer.CreateSpec` for the file transfer to or from the guest. The content of the optional `FileAttributeCreateSpec` associated with the `Transfer.CreateSpec` establishes the direction of the transfer and controls guest operating system specific file attributes. You must set up equivalent `_download` and `_upload` functions for URL management of the file transfer to or from the guest. You can also create an argument parser for standard inputs, such as server, user name, and password, and you can add custom input arguments.

Procedure

1. Find the virtual machine on which the guest operating system runs, verify that VMware Tools is running, and provide credentials.
2. Create a temporary directory from which to run the script and capture any output.
3. Create temporary files for `stdout` and `stderr`.
4. (Optional) Copy the script that you want to run.
5. Start the script and capture `stdout` and `stderr` in the temporary files that you created earlier.
6. Create a loop to handle processes that run longer.
7. Create a download URL and copy the results.
8. Clean up the temporary files and directories on the guest operating system.

Python Example of Uploading and Running a Script on a Guest Operating System

This example is based on the code in the `guest_ops.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.

```
...

    # Create the Process.CreateSpec for initiating processes in the guest
    def _process_create_spec(self, path, args=None, dir=None, env={}):
        return Processes.CreateSpec(path=path,
                                    arguments=args,
                                    working_directory=dir,
                                    environment_variables=env)

    # Create the Transfer.CreateSpec for the file transfer to/from the guest
    def _create_transfer_spec(self,
                              path,
                              attributes=None):
        return Transfers.CreateSpec(attributes=attributes,
                                    path=path)

    # Create a FileAttributeCreateSpec for a generic (non-OS specific) guest
```

```
    def _fileAttributeCreateSpec_Plain(self,
                                       size,
                                       overwrite=None,
                                       last_modified=None,
                                       last_accessed=None):
        return Transfers.FileCreationAttributes(size,
                                                overwrite=overwrite,
                                                last_modified=last_modified,
                                                last_accessed=last_accessed)

    # Create a FileAttributeCreateSpec for a linux (Posix) guest
    def _fileAttributeCreateSpec_Linux(self,
                                       size,
                                       overwrite=None,
                                       last_modified=None,
                                       last_accessed=None,
                                       owner_id=None,
                                       group_id=None,
                                       permissions=None):
        posix = Transfers.PosixFileAttributesCreateSpec(owner_id=owner_id,
                                                        group_id=group_id,
                                                        permissions=permissions)
        return Transfers.FileCreationAttributes(size,
                                                overwrite=overwrite,
                                                last_modified=last_modified,
                                                last_accessed=last_accessed,
                                                posix=posix)

    def _download(self,
                  url,
                  expectedLen=None):
        urloptions = urlparse(url)
        # Skip server cert verification.
        # This is not recommended in production code.
        conn = httpclient.HTTPSConnection(urloptions.netloc,
                                          context=ssl._create_unverified_context())

        conn.request("GET", urloptions.path + "?" + urloptions.query)
        res = conn.getresponse()
        if res.status != 200:
            print("GET request failed with errorcode : %s" % res.status)
            raise HTTPError(res.status, res.reason)
        body = res.read().decode()

        return body

    def _upload(self, url, body):
        urloptions = urlparse(url)
        conn = httpclient.HTTPSConnection(urloptions.netloc,
                                          context=ssl._create_unverified_context())

        headers = {"Content-Length": len(body)}
        # Skip server cert verification.
        # This is not recommended in production code.
        conn.request("PUT", urloptions.path + "?" + urloptions.query,
```

```
                    body,
                    headers)
        res = conn.getresponse()
        if res.status != 200:
            print("PUT request failed with errorcode : %s" % res.status)
            raise HTTPError(res.status, res.reason)

        return res

    def __init__(self):
        # Create argument parser for standard inputs:
        # server, username, password, cleanup and skipverification
        parser = sample_cli.build_arg_parser()

        # Add your custom input arguments
        parser.add_argument('--vm_name',
                            action='store',
                            help='Name of the testing vm')
        parser.add_argument('--root_user',
                            action='store',
                            help='Administrator account user name')
        parser.add_argument('--root_passwd',
                            action='store',
                            help='Administrator account password')

        args = sample_util.process_cli_args(parser.parse_args())
        self.vm_name = args.vm_name
        self.root_user = args.root_user
        self.root_passwd = args.root_passwd

        self.cleardata = args.cleardata

        # Skip server cert verification if needed.
        # This is not recommended in production code.
        session = get_unverified_session() if args.skipverification else None

        # Connect to vSphere client
        self.client = create_vsphere_client(server=args.server,
                                            username=args.username,
                                            password=args.password,
                                            session=session)

    def run(self):
        # Using vAPI to find VM.
        filter_spec = VM.FilterSpec(names=set([self.vm_name]))
        vms = self.client.vcenter.VM.list(filter_spec)
        if len(vms) != 1:
            raise Exception('Could not locate the required VM with name ' +
                            self.vm_name + '. Please create the vm first.')
        if vms[0].power_state != 'POWERED_ON':
            raise Exception('VM is not powered on: ' + vms[0].power_state)
        vm_id = vms[0].vm

        # Check that vmtools svc (non-interactive user) is running.
        info = self.client.vcenter.vm.guest.Operations.get(vm_id)
```

```
        if info.guest_operations_ready is not True:
            raise Exception('VMware Tools/open-vm-tools is not running as required.')

        # Establish the user credentials that will be needed for all Guest Ops APIs.
        creds = Credentials(interactive_session=False,
                            user_name=self.root_user,
                            password=self.root_passwd,
                            type=Credentials.Type.USERNAME_PASSWORD)

        # Step 2 - Create a temporary directory from which to run the command and capture any
output
        tempDir = self.client.vcenter.vm.guest.filesystem.Directories.create_temporary(
                    vm_id, creds, '', '', parent_path=None)

        # Step 3 - Create temproary files to reveive stdout and stderr as needed.
        stdout = self.client.vcenter.vm.guest.filesystem.Files.create_temporary(
                   vm_id, creds, '', '.stdout', parent_path=tempDir)
        stderr = self.client.vcenter.vm.guest.filesystem.Files.create_temporary(
                   vm_id, creds, '', '.stderr', parent_path=tempDir)

        # Step 4 - (Optional) Copy the script to be run.
        scriptPath = self.client.vcenter.vm.guest.filesystem.Files.create_temporary(
                       vm_id, creds, '', '.sh', tempDir)

        # Create script contents and transfer to the guest.
        baseFN = os.path.basename(scriptPath)
        script = ('#! /bin/bash\n'
                  '#    ' +
                  baseFN + '\n'
                  '\n'
                  'sleep 5    # Adding a little length to the process.\n'
                  'ps -ef\n'
                  'echo\n'
                  'rpm -qa | sort\n'
                  '\n')
        print(script)
        attr = self._fileAttributeCreateSpec_Linux(size=len(script),
                                                   overwrite=True,
                                                   permissions='0755')
        spec = self._create_transfer_spec(path=scriptPath,
                                          attributes=attr)
        toURL = self.client.vcenter.vm.guest.filesystem.Transfers.create(vm_id,
                                                                         creds,
                                                                         spec)
        res = self._upload(toURL, script)

        # Check that the uploaded file size is correct.
        info = self.client.vcenter.vm.guest.filesystem.Files.get(vm_id,
                                                                 creds,
                                                                 scriptPath)
        if info.size != len(script):
            raise Exception('Uploaded file size not as epected.')

        # Step 5 - Start the program on the guest, capturing stdout and stderr in the
separate temp files obtained earlier.
```

```
        options = (" > " + stdout + " 2> " + stderr)

        spec = self._process_create_spec(scriptPath,
                                         args=options,
                                         dir=tempDir)
        pid = self.client.vcenter.vm.guest.Processes.create(vm_id, creds, spec)
        print('process created with pid: %s\n' % pid)

        # Step 6 - Need a loop to wait for the process to finish to handle longer running
processes.
        while True:
            time.sleep(1.0)
            try:
                # List the single process for pid.
                result = self.client.vcenter.vm.guest.Processes.get(vm_id,
                                                                    creds,
                                                                    pid)
                if result.exit_code is not None:
                    print('Command: ' + result.command)
                    print('Exit code: %s\n' % result.exit_code)
                    break
                if result.finished is None:
                    print('Process with pid %s is still running.' % pid)
                    continue
            except Exception as e:
                raise e

        # Step 7 Copy out the results (stdout).
        spec = self._create_transfer_spec(path=stdout)
        # Create the download URL
        fromURL = self.client.vcenter.vm.guest.filesystem.Transfers.create(vm_id,
                                                                           creds,
                                                                           spec)
        body = self._download(fromURL, expectedLen=info.size)
        print("-----------  stdout  ------------------")
        print(body)
        print("----------------------------------------")

        # Optionally the contents of "stderr" could be downloaded.

        # And finally, clean up the temporary files and directories on the
        # Linux guest.  Deleting the temporary diretory and its contents.
        self.client.vcenter.vm.guest.filesystem.Directories.delete(vm_id,
                                                                   creds,
                                                                   tempDir,
                                                                   recursive=True)

...
```

Content Library Service

9

The Content Library service provides means for managing content libraries in the context of a single or multiple vCenter Server instances deployed in your virtual environment. You can use the vSphere Automation APIs to access the service and use the provided functionality.

Administrators can use content libraries to share VM and vApp templates, and other types of files, such as ISO images, text files, and so on, across the vCenter Server instances in their virtual environment. Sharing templates across your virtual environment promotes consistency, compliance, efficiency, and automation in deploying workloads at scale.

- Content Library Overview

  A content library instance represents a container for a set of library items. A content library item instance represents the logical object stored in the content library, which might be one or more usable files.

- Querying Content Libraries

  You can create queries to find libraries that match your criteria. You can also retrieve a list of all libraries or only the libraries of a specific type.

- Content Libraries

  The Content Library API provides services that allow you to create and manage content libraries programmatically. You can create a local library and publish it for the entire virtual environment. You can also subscribe to use the contents of a local library and enable automatic synchronization to ensure that you have the latest content.

- Library Items

  A library item groups multiple files within one logical unit. You can perform various tasks with the items in a content library.

- Content Library Support for OVF and OVA Packages

  You can use the objects and methods provided by the Content Library API to manage OVF and OVA packages.

- Creating Virtual Machines and vApps from Templates in a Content Library

  You can create VM and OVF templates from virtual machines and vApps in your inventory. You can then deploy virtual machines and vApps from the templates that are stored in a content library.

Content Library Overview

A content library instance represents a container for a set of library items. A content library item instance represents the logical object stored in the content library, which might be one or more usable files.

You create and manage the content of a content library on a single vCenter Server instance, but you can distribute the content to other vCenter Server instances. Depending on your needs, you can maintain two types of content libraries: local and subscribed. You can shape the contents of a library item and then combine several library items in a local content library. Furthermore, you can publish the library to make its content available to other users.

- Content Library Types

  You can create two types of libraries, local and subscribed.

- Content Library Items

  Library items are VM templates, vApp templates, or other VMware objects that can be contained in a content library. VMs and vApps have several files, such as log files, disk files, memory files, and snapshot files that are part of a single library item. You can create library items in a specific local library or remove items from a local library. You can also upload files to an item in a local library so that the libraries subscribed to it can download the files to their NFS or SMB server, or datastore.

- Content Library Storage

  When you create a local library, you can store its contents on a datastore managed by the vCenter Server instance or on a remote file system.

Content Library Types

You can create two types of libraries, local and subscribed.

Local library.

You can create a local library as the source for content you want to save or share. Create the local library on a single vCenter Server instance. You can add items to a local library or remove them. You can publish a local library and as a result this content library service endpoint can be accessed by other vCenter Server instances in your virtual environment. When you publish a library, you can configure a password for authentication.

Subscribed library.

You can create a subscribed library and populate its content by synchronizing to a local library. A subscribed library contains copies of the local library files or just the metadata of the library items. The local library can be located on the same vCenter Server instance as the subscribed library, or the subscribed library can reference a local library on a different vCenter Server instance. You cannot add library items to a subscribed library. You can only add items to the source library. After synchronization, both libraries will contain the same items.

Content Library Items

Library items are VM templates, vApp templates, or other VMware objects that can be contained in a content library. VMs and vApps have several files, such as log files, disk files, memory files, and snapshot files that are part of a single library item. You can create library items in a specific local library or remove items from a local library. You can also upload files to an item in a local library so that the libraries subscribed to it can download the files to their NFS or SMB server, or datastore.

For information about the tasks that you can perform by using the content library service, see Content Libraries.

Content Library Storage

When you create a local library, you can store its contents on a datastore managed by the vCenter Server instance or on a remote file system.

Depending on the type of storage that you have, you can use Virtual Machine File System (VMFS) or Network File System (NFS) for storing content on a datastore.

For storing content on a remote file system, you can enter the path to the NFS storage that is mounted on the Linux file system of the vCenter Server instance. For example, you can use the following URI formats: `nfs://<server>/<path>?version=4` and `nfs://<server>/<path>`. If you have a vCenter Server instance that runs on a Windows machine, you can specify the Server Massage Block (SMB) URI to the Windows shared folders that store the library content. For example, you can use the following URI format: `smb://<unc-server>/<path>`.

Java Example of Storing Library Content on a Datastore

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

For more information about storing the contents of a local library, see Content Library Storage.

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

```
...

// Create a StorageBacking instance for storing the library content on a datastore.

     StorageBacking libraryBacking = new StorageBacking();
     libraryBacking.setType(Type.DATASTORE);

/*
 * Pass the value of the datastore ManagedObjectReference.
 * See the vSphere Web Services SDK Programming Guide
 * and the vSphere Web Services SDK samples. In addition, the  vSphere
                        Automation SDK for Java provides
 * the VimUtil utility class in the vmware.samples.vim.helpers package.
 * You can use the utility to retrieve the ManagedObjectReference
 * of the datastore entity.
```

```
 */
     libraryBacking.setDatastoreId("datastore-123");

// Create a LibraryModel that represents a local library backed on a datastore.

      LibraryModel libraryModel = new LibraryModel();
      libraryModel.setName("AcmeLibrary");
      libraryModel.setDescription("Local library backed by VC datastore");
      libraryModel.setType(LibraryType.LOCAL);
      libraryModel.setStorageBackings(Collections.singletonList(libraryBacking));
```

Python Example of Storing Library Content on a Datastore

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

For more information about storing the contents of a local library, see Content Library Storage.

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 StorageBacking instance of datastore type.
library_backing = library_client.StorageBacking()
library_backing.type = library_client.StorageBacking.Type.DATASTORE

# Pass the value of the datastore managed object reference.
# The
                        vSphere Automation SDK for Python contains
# the GetDatastoreByName class, which sample resource is located in the
# /client/samples/src/com/vmware/vcloud/suite/sample/vim/helpers/ directory.
# You can use the utility to retrieve the managed object reference of the datastore entity.
library_backing.datastore_id = 'datastore-123'

# Create a LibraryModel that represents a local library backed on a datastore.
library_model = content_client.LibraryModel()
library_model.name = 'AcmeLibrary'
library_model.storage_backings = [library_backing]
```

Querying Content Libraries

You can create queries to find libraries that match your criteria. You can also retrieve a list of all libraries or only the libraries of a specific type.

- Listing All Content Libraries

  You can retrieve a list of all content library IDs in your virtual environment, regardless of their type, by using the `Library` service.

- Listing Content Libraries of a Specific Type

  You can use the vSphere Automation API to retrieve content libraries of a specific type. For example, you can list only the local libraries in your virtual environment.

- Listing Content Libraries by Using Specific Search Criteria

  You can filter the list of content libraries and retrieve only the libraries that match your specific criteria. For example, you might want to publish all local libraries with a specific name.

Listing All Content Libraries

You can retrieve a list of all content library IDs in your virtual environment, regardless of their type, by using the `Library` service.

You can use the `list` function to retrieve all local and subscribed libraries in your system.

Java Example of Retrieving a List of All Content Libraries

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

This example uses the steps that are described in Listing All Content Libraries.

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

```
...

// Access the Library Service.
   Library libraryService = vapiAuthHelper.getStubFactory().createStub(Library.class,
sessionStubConfig);

// List all content libraries.
   List<String> allLibraries = libraryService.list();
   System.out.println("List of all library identifiers: /n");
   for (String cl : allLibraries) {
         System.out.println(cl);
   }
```

Python Example of Retrieving a List of All Content Libraries

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

This example uses the steps that are described in Listing All Content Libraries.

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

```
...

library_stub = content_client.Library(my_stub_config)
libraries = library_stub.list()
print('List of all library identifiers:')
for library_id in library_ids :
  library = library_stub.get(library_id)
  print('Library ID {}: {}'.format(library_id, library.name))
```

Listing Content Libraries of a Specific Type

You can use the vSphere Automation API to retrieve content libraries of a specific type. For example, you can list only the local libraries in your virtual environment.

If you want to retrieve only a list of the local libraries, you must retrieve the `LocalLibrary` service and use the `list` function on the `LocalLibrary` service. To list only subscribed libraries, you must retrieve the `SubscribedLibrary` service and call the `list` function on the `SubscribedLibrary` service.

Listing Content Libraries by Using Specific Search Criteria

You can filter the list of content libraries and retrieve only the libraries that match your specific criteria. For example, you might want to publish all local libraries with a specific name.

To a filter with specific search criteria, call the `find (find_spec)` function of the `Library` service. Pass as argument a `LibraryTypes.FindSpec` instance that contains your search criteria. Upon a successful completion of the call, you receive a list of all content libraries that match your search criteria.

Java Example of Retrieving a List of All Local Libraries with a Specific Name

This example retrieves a list of all local libraries with the name **`AcmeLibrary`** that exist in your virtual environment.

Note For related code samples, see the `vsphere-automation-sdk-java` VMware repository at GitHub.

```
...

// Create a FindSpec instance to set your search criteria.
   FindSpec findSpec = new FindSpec();

// Filter the local content libraries by using a library name.
   findSpec.setName("AcmeLibrary");
   findSpec.setType(LibraryType.LOCAL);
   List<String> ids = libraryService.find(findSpec);
```

Python Example of Retrieving a List of All Local Libraries with a Specific Name

This example retrieves a list of all local libraries with the name **`AcmeLibrary`** that exist in your virtual environment.

Note For related code samples, see the `vsphere-automation-sdk-python` VMware repository at GitHub.

```
...

# Create a FindSpec object to specify your search criteria.
find_spec = content_client.Library.FindSpec()
```

```
find_spec.name = 'AcmeLibrary'
find_spec.type = content_client.LibraryModel.LibraryType.LOCAL

# Invoke the find() function by using the FindSpec instance.
library_stub = content_client.Library(my_stub_config)
library_ids = library_stub.find(find_spec)
```

Content Libraries

The Content Library API provides services that allow you to create and manage content libraries programmatically. You can create a local library and publish it for the entire virtual environment. You can also subscribe to use the contents of a local library and enable automatic synchronization to ensure that you have the latest content.

- Create a Local Content Library

  You can create a local content library programmatically by using the vSphere Automation API. The API allows you to populate the content library with VM and vApp templates. You can use these templates to deploy virtual machines or vApps in your virtual environment.

- Publish an Existing Content Library

  To make the library content available for other vCenter Server instances across the vSphere Automation environment, you must publish the library. Depending on your workflow, select a method for publishing the local library. You can publish a local library that already exists in your vSphere Automation environment.

- Publish a Library at the Time of Creation

  You can publish a local library at the time of creation to enable other libraries to subscribe and use the library content.

- Subscribe to a Content Library

  You can subscribe to local content libraries. When you subscribe to a library, you must specify the backing storage for the library content. If the library requires basic authentication, you must also provide the correct user name and password.

- Synchronize a Subscribed Content Library

  When you subscribe to a published library, you can configure the settings for downloading and updating the library content.

- Editing the Settings of a Content Library

  You can update the settings of content library types in your virtual environment by using the vSphere Automation API.

- Removing the Content of a Subscribed Library

  You can free storage space in your virtual environment by removing the subscribed library content that you no longer need.

- Delete a Content Library

  When you no longer need a content library, you can invoke the `delete` method on either the `LocalLibrary` or the `SubscribedLibrary` service depending on the library type.

Create a Local Content Library

You can create a local content library programmatically by using the vSphere Automation API. The API allows you to populate the content library with VM and vApp templates. You can use these templates to deploy virtual machines or vApps in your virtual environment.

Procedure

1. Create a `StorageBacking` instance and define the storage location.
2. Create a `LibraryModel` instance and set the properties of the new local library.
3. Access the `LocalLibrary` object which is part of the vSphere Automation API service interfaces.
4. Call the create function on the `LocalLibrary` object and pass the `LibraryModel` as a parameter.

Java Example of Creating a Local Library

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

This example uses the steps that are described in the Create a Local Content Library procedure.

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

```
...

// Create a StorageBacking instance to back the library content on the local file system.
     StorageBacking libraryBacking = new StorageBacking();
     libraryBacking.setType(Type.OTHER);
     libraryBacking.setStorageUri(URI.create("file:///tmp"));
     libraryModel.setStorageBackings(Collections.singletonList(libraryBacking));

// Create a LibraryModel that represents a local library.
     LibraryModel libraryModel = new LibraryModel();
     libraryModel.setType(LibraryModel.LibraryType.LOCAL);
     libraryModel.setName("AcmeLibrary");

// Access the LocalLibrary service by using the  endpoint.
     LocalLibrary localLibraryService =
this.vapiAuthHelper.getStubFactory().createStub(LocalLibrary.class, sessionStubconfig);

// Call the create method of the LocalLibrary service passing as an
// argument the LibraryModel instance.
     String libraryId = localLibraryService.create(UUID.randomUUID().toString(), libraryMod
```

Python Example of Creating a Local Content Library

This example creates a local library with name `AcmeLibrary` , which is stored on the local file system where vCenter Server runs.

This example uses the steps that are described in the Create a Local Content Library procedure.

Note For related code samples, see the `vsphere-automation-sdk-python` VMware repository at GitHub.

```
...

# 1 - Create a storage backing instance on a local file system.
library_backing = library_client.StorageBacking()
library_backing.type = library_client.StorageBacking.Type.OTHER
library_backing.storage_uri = 'file:///tmp'

# 2 - Create a Library model to specify properties of the new library.
library_model = content_client.LibraryModel()
library_model.type = content_client.LibraryModel.LibraryType.LOCAL
library_model.name = 'AcmeLibrary'
library_model.storage_backings = [library_backing]

# 3 - Call the create() method, passing the library model as a parameter.
idem_token = str(uuid.uuid4())
local_library_stub = content_client.LocalLibrary(my_stub_config)
library_id = local_library_stub.create(create_spec=library_model,
                                       client_token=idem_token)
```

Publish an Existing Content Library

To make the library content available for other vCenter Server instances across the vSphere Automation environment, you must publish the library. Depending on your workflow, select a method for publishing the local library. You can publish a local library that already exists in your vSphere Automation environment.

Procedure

1. Retrieve a reference to the `LocalLibrary` service.
2. Retrieve an existing local library by using the library ID.
3. Create a `PublishInfo` instance to define how the library is published.
4. Specify the authentication method to be used by a subscribed library to authenticate to the local library. You can enable either basic authentication or no authentication. Basic authentication requires a user name and password.

5 (Optional) If you publish the library with basic authentication, you must specify a user name and password for the `PublishInfo` instance, which must be used for authentication.

Important Use the predefined user name **`vcsp`** or leave the user name undefined. You must set only a password to protect the library.

6 Set the local library to published.

7 Use the retrieved local library to configure it with the `PublishInfo` instance.

8 Update the properties of the `LibraryModel` object returned for the local library.

Java Example of Publishing an Existing Content Library

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

This example uses the steps that are described in the Publish an Existing Content Library procedure.

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

```
...

// Access the LocalLibrary service.
      LocalLibrary localLibraryService =
this.vapiAuthHelper.getStubFactory().createStub(LocalLibrary.class, sessionStubconfig);

// Retrieve an existing local library.
      LibraryModel libraryModel = localLibraryService.get(libraryId);
      PublishInfo publishInfo = new PublishInfo();

// Configure how the local library is published by using BASIC authentication.
      publishInfo.setUserName("vcsp");
      publishInfo.setPassword("password".toCharArray());
      publishInfo.setAuthenticationMethod(AuthenticationMethod.BASIC);

// Set the local library to published and update the library instance.
      publishInfo.setPublished(true);
      libraryModel.setPublishInfo(publishInfo);
      localLibraryService.update(libraryModel.getId(), libraryModel);
```

Python Example of Publishing an Existing Content Library

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

This example uses the steps that are described in the Publish an Existing Content Library procedure.

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

```
...

# Retrieve an existing local library.
local_library_stub = content_client.LocalLibrary(my_stub_config)
local_library = local_library_stub.get(my_library_id)

# Specify how the local library is published, using BASIC authentication.
publish_info = library_client.PublishInfo()
publish_info.user_name = 'vcsp' # Can omit this value; if specified, it must be 'vcsp'.
publish_info.password = 'password'
publish_info.authentication_method = library_client.PublishInfo.AuthenticationMethod.BASIC
publish_info.published = True

# Update the LibraryModel object retieved in step 1
# and configure it with the PublishInfo object.
local_library.publish_info = publish_info

# Use the LibraryModel object to update the library instance.
local_library_stub.update(library_id=my_library_id,
update_spec=local_library)
```

Publish a Library at the Time of Creation

You can publish a local library at the time of creation to enable other libraries to subscribe and use the library content.

Procedure

1. Retrieve the `LocalLibrary` service.
2. Create a `PublishInfo` instance to define how the library is published.
3. Specify the authentication method to be used by a subscribed library to authenticate to the local library.

   You can enable either basic authentication or no authentication on the library. Basic authentication requires a user name and password.
4. (Optional) If you publish the library with basic authentication, you must specify a user name and password for the `PublishInfo` instance, which must be used for authentication.

   Important Use the predefined user name **`vcsp`** or leave the user name undefined. You must set only a password to protect the library.
5. Create a `LibraryModel` instance and configure the instance.

6 Set the library type to local and use the configured `PublishInfo` instance to set the library to published.

7 Define where the content of the local library is stored by using the `StorageBacking` class.

8 Create a published local library.

Subscribe to a Content Library

You can subscribe to local content libraries. When you subscribe to a library, you must specify the backing storage for the library content. If the library requires basic authentication, you must also provide the correct user name and password.

Note If you subscribe to libraries created with basic authentication on a vCenter Server instance, make sure that you pass **`vcsp`** as an argument for the user name.

Prerequisites

Required privileges:

- **Content library.Create subscribed library** on the vCenter Server instance where you want to create the library.
- **Datastore.Allocate space** on the destination datastore.

Procedure

1 Create a `StorageBacking` instance to define the location where the content of the subscribed library is stored.

2 Create a `SubscriptionInfo` instance to define the subscription behavior of the library.

   a Provide the mechanism to be used by the subscribed library to authenticate to the published library.

   You can select between no authentication and basic authentication depending on the settings of the published library you subscribe to. If the library is published with basic authentication, you must set the basic authentication in the `SubscriptionInfo` instance. To match the credentials of the published library, set the user name and the password of the `SubscriptionInfo` instance.

   b Provide the URL to the endpoint where the metadata of the published library is stored.

   c Define the synchronization mechanism of the subscribed library.

   You can select between two synchronization modes: automatic and on demand. If you enable automatic synchronization for a subscribed library, both the content and the metadata are synchronized with the published library. To save storage space, you can synchronize the subscribed library on demand and update only the metadata for the published library content.

   d Set the thumbprint that is used for validating the certificate of the published library.

3 Create a `LibraryModel` instance and set the library type to subscribed (`LibraryModel.LibraryType.SUBSCRIBED`).

4 Access the `SubscribedLibrary` service and create the subscribed library instance.

Java Example of Subscribing to a Published Library

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

This example uses the steps that are described in the Subscribe to a Content Library procedure.

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

```
...

// Create a StorageBacking instance to store
// the contents of the subscribed library on the local file system.
       StorageBacking libraryBacking = new StorageBacking();
       libraryBacking.setType(StorageBacking.Type.OTHER);
       libraryBacking.setStorageUri(URI.create("/mnt/nfs/cls-root"));

// Create a new SubscriptionInfo object to define the subscription
// behavior of the library.
       SubscriptionInfo subscriptionInfo = new SubscriptionInfo();
       subscriptionInfo.setAuthenticationMethod
         (com.vmware.content.library.SubscriptionInfo.AuthenticationMethod.BASIC);
       subscriptionInfo.setUserName("libraryUser");
       subscriptionInfo.setPassword("password".toCharArray());
       subscriptionInfo.setSubscriptionUrl(URI.create("https://www.acmecompary.com/
library_inventory/lib.json"));

// Specify that the content of the subscribed library will be downloaded immediately.
       subscriptionInfo.setAutomaticSyncEnabled(true);

// Set an SHA-1 hash of the SSL certificate of the remote endpoint.

subscriptionInfo.setSslThumbprint("9B:00:3F:C4:4E:B1:F3:F9:0D:70:47:48:E7:0B:D1:A7:0E:DE:60:A5
");

// Create a new LibraryModel object for the subscribed library.
       LibraryModel libraryModel = new LibraryModel();
       libraryModel.setType(LibraryModel.LibraryType.SUBSCRIBED);
       libraryModel.setName("SubscrLibrary");

// Attach the storage backing and the subscription info to the library model.
       libraryModel.setStorageBackings(Collections.singletonList(libraryBacking));
       libraryModel.setSubscriptionInfo(subscriptionInfo);

// Create the new subscribed library.
       String clientToken = UUID.randomUUID().toString();
       SubscribedLibrary subscribedLibService =
```

```
this.vapiAuthHelper.getStubFactory().createStub(SubscribedLibrary.class, sessionStubconfig);
       String subscribedLibId = subscribedLibService.create(clientToken, libraryModel);
```

Python Example of Subscribing to a Published Library

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

This example uses the steps that are described in the Subscribe to a Content Library procedure.

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 StorageBacking instance on a local file system.
library_backing = library_client.StorageBacking()
library_backing.type = library_client.StorageBacking.Type.OTHER
library_backing.storage_uri = '/mnt/nfs/cls-root'

# Create a new SubscriptionInfo object to describe the subscription behavior.
subscription_info = library_client.SubscriptionInfo()
subscription_info.authentication_method =
library_client.SubscriptionInfo.AuthenticationMethod.BASIC
subscription_info.user_name = 'libraryUser'
subscription_info.password = 'password'
subscription_info.subscription_url = 'https://www.acmecompary.com/library_inventory/lib.json'
subscription_info.automatic_sync_enabled = True
subscription_info.ssl_thumbprint
= '9B:00:3F:C4:4E:B1:F3:F9:0D:70:47:48:E7:0B:D1:A7:0E:DE:60:A5'

# Create a new LibraryModel object for the subscribed library.
library_model = content_client.LibraryModel()
library_model.type = content_client.LibraryModel.LibraryType.SUBSCRIBED
library_model.name = 'subscrLibrary'

# Attach the storage backing and the subscription info to the library model.
library_model.storage_backings = [library_backing]
library_model.subscription_info = subscription_info

# Create the new library instance.
idem_token = str(uuid.uuid4())
local_library_stub = content_client.LocalLibrary(my_stub_config)
library_id = local_library_stub.create(create_spec=library_model,client_token=idem_token)
```

Synchronize a Subscribed Content Library

When you subscribe to a published library, you can configure the settings for downloading and updating the library content.

- You can enable automatic synchronization of the subscribed library and download a copy of the content of the local library immediately.

- You can save storage space and download only the metadata for the items that are part of the local library.

To ensure that your subscribed library contains the most recent published library content, you can force a synchronization task.

Procedure

1 Access the `SubscribedLibrary` vSphere Automation service.

2 Retrieve the subscribed library ID from the `SubscribedLibrary` service.

3 Force the synchronization of the subscribed library.

Results

The synchronization operation depends on the update settings of the subscribed library. If the subscribed library is configured to update only on demand, only the metadata of the library items will be synchronized.

Editing the Settings of a Content Library

You can update the settings of content library types in your virtual environment by using the vSphere Automation API.

Table 9-1. Options for Updating a Content Library

Content Library Types	Option

Local content library	You can change the settings of a local library before calling the update function on the `LocalLibrary` object:
	■ Before a library is published, you can edit the following settings:
	  ■ The name of a local library that is retrieved by using the `LocalLibrary` object
	  ■ The human-readable description of a local library retrieved by using the `LocalLibrary` object
	■ After a library is published, you must first retrieve the `PublishInfo` instance of the published library you want. You can use the instance to configure the following settings.
	  ■ Unpublish the local library.
	  ■ Change the authentication method of the library.
	  ■ Change the password that must be used for authentication.

Subscribed content library	You can edit the settings of a subscribed library if you retrieve the `SubscriptionInfo` instance associated with it. To apply the changes, you must update the library by using the `SubscribedLibrary` object. You can configure the following settings: ■ The authentication method required by the local library ■ The user name and password of the subscribed library for authentication to the local library ■ The method for synchronizing the metadata and the content of the subscribed library ■ The thumbprint used for validating the SSL certificate of the local library

Removing the Content of a Subscribed Library

You can free storage space in your virtual environment by removing the subscribed library content that you no longer need.

You can create a subscribed library with the option to download the library content on demand. As a result, only the metadata for the library items is stored in the associated with the subscribed library storage. When you want to deploy a virtual machine from a VM temple in the subscribed library, you must synchronize the subscribed library to download the entire published library content. When you no longer need the VM template, you can call the `evict` function on the `SubscribedLibrary` service. You must provide the subscribed library ID to this function. As a result, the subscribed library content that is cached on the backing storage is deleted.

If the subscribed library is not configured to synchronize on demand, an exception is thrown. In this case the subscribed library always attempts to have the most recent published library content.

Delete a Content Library

When you no longer need a content library, you can invoke the `delete` method on either the `LocalLibrary` or the `SubscribedLibrary` service depending on the library type.

Procedure

1. Access the `SubscribedLibrary` or the `LocalLibrary` service by using the vSphere Automation Endpoint.
2. Retrieve the library ID you want to delete.
3. Call the `delete` function on the library service and pass the library ID as argument.

   All library items cashed on the storage backing are removed asynchronously. If this operation fails, you must manually remove the content of the library.

Library Items

A library item groups multiple files within one logical unit. You can perform various tasks with the items in a content library.

You can upload files to a library item in a local library and update existing items. You can download the content of a library item from a subscribed library and use the item, for example, to deploy a virtual machine. You can remove the content of a library item from a subscribed library to free storage space and keep only the metadata of the library item. When you no longer need local library items, you can delete them and they are removed from the subscribed library when a synchronization task is completed.

You can create a library item from a specific item type, for example **`.ovf`** and VM template. The Content Library service must support the library item type to handle the item correctly. If no support is provided for a specified type, the Content Library service handles the library item in the default way, without adding metadata to the library item or guiding the upload process. For information about the supported VM template types, see the *vSphere Virtual Machine Administration* documentation.

- Create an Empty Library Item

  You can create as many library items as needed and associate them with a local content library.
- Querying Library Items

  You can perform numerous query operations on library items.
- Edit the Settings of a Library Item

  You can edit the name, description, and type of a library item.
- Upload a File from a Local System to a Library Item

  You can upload different types of files from a local system to a library item that you want to use in the vSphere Automation environment.

- Upload a File from a URL to a Library Item

  You can upload different types of files from a local system to a library item that you want to use in the vSphere Automationenvironment.
- Download Files to a Local System from a Library Item

  You might want to download files to a local system from a library item and then make changes to the files before you use them.
- Synchronizing a Library Item in a Subscribed Content Library

  The items in a subscribed library have features that are distinct from the items in a local library. Synchronizing the content and the metadata of an item in a subscribed library depends on the synchronization mechanism of the subscribed library.
- Removing the Content of a Library Item

  You can remove the content from a library item to free space on your storage.
- Deleting a Library Item

  You can remove a library item from a local library when you no longer need it.

Create an Empty Library Item

You can create as many library items as needed and associate them with a local content library.

Procedure

1. Access the `Item` service by using the vSphere Automation endpoint.
2. Instantiate the `ItemModel` class.
3. Define the settings of the new library item.
4. Associate the library item with an existing local library.
5. Invoke the `create` function on the `Item` object to pass the library item specification and the unique client token.

What to do next

Upload content to the new library item. See Upload a File from a Local System to a Library Item and Upload a File from a URL to a Library Item.

Java Example of Creating a Library Item

This example shows how to create an empty library item that stores an ISO image file.

This example uses the steps that are described in the Create an Empty Library Item procedure.

Note For related code samples, see the `vsphere-automation-sdk-java` VMware repository at GitHub.

```
...
```

```
// Create an instance of the ItemModel class and specify the item settings.
ItemModel libItemSpec = new ItemModel();
libItemSpec.setName("ESXi ISO image");
libItemSpec.setDescription("ISO image with the latest security patches for ESXi");
libItemSpec.setType("iso");

// Associate the item with an existing content library.
libItemSpec.setLibraryId("<content_library_ID>");

// Create the new Item instance, using the specified model.
Item libItemService = this.vapiAuthHelper.getStubFactory().createStub(Item.class,
sessionStubconfig);
String itemID = UUID.randomUUID().toString();
String newItem = libItemService.create(itemID, libItemSpec);
```

Python Example of Creating a Library Item

This example shows how to create an empty library item that stores an ISO image file.

This example uses the steps that are described in the Create an Empty Library Item procedure.

Note For related code samples, see the `vsphere-automation-sdk-python` VMware repository at GitHub.

```
...

# 1 - Create an instance of the ItemModel class and specify the item settings.
item_model = library_client.ItemModel()
item_model.name = 'ESXi ISO image'
item_model.description = 'ISO image with the latest security patches for ESXi'
item_model.type = 'iso'

# 2 - Associate the new item with an existing library.
item_model.library_id = my_library_id

# 3 - Create the new instance of the Item class, using the specified model.
idem_token = str(uuid.uuid4())
item_stub = library_client.Item(my_stub_config)
item_id = item_stub.create(create_spec=item_model, client_token=idem_token)
```

Querying Library Items

You can perform numerous query operations on library items.

You can retrieve a list of all items in a library, retrieve a library item that has a specific type or name, and find a library item that is not cached on the disk. You can then update the library item content from the subscribed library.

List Library Items

You can use the `list` method of the `Item` object to retrieve a list of all items in a particular library.

Prerequisites

Verify that you have access to the `Item` service.

Procedure

1. Retrieve the ID of the content library whose items you want to list.
2. List the items of the specific library.
3. Retrieve a list of the files that belong to a library item.

Example

You can see an example query operation in the code example for Edit the Settings of a Library Item. The beginning of the example lists the items of a published library and prints a list with the names and size of each file in the listed items.

List Library Items That Match Specific Criteria

You can filter the items contained in a library and retrieve only the items matching specific criteria. For example, you might want to remove or update only specific items in a library.

Prerequisites

Verify that you have access to the `Item` service.

Procedure

1. Create an instance in the `FindSpec` class.
2. Specify the filter properties by using the `FindSpec` instance.
3. List the items matching the specified filter.

Results

A list of items matching the filter criteria is created as a result.

Edit the Settings of a Library Item

You can edit the name, description, and type of a library item.

Prerequisites

Verify that you have access to the `Item` service.

Procedure

1. Retrieve the item that you want to update.
2. Create an `ItemModel` instance.
3. Change the human-readable name and description of the library item.
4. Update the library item with the configured item model.

Java Example of Changing the Settings for a Library Item

This example shows how to find an item by using the item name and then how to change the name and description of the retrieved item.

This example uses the steps that are described in the Edit the Settings of a Library Item procedure.

Note For related code samples, see the `vsphere-automation-sdk-java` VMware repository at GitHub.

```
...

// List the items in a published library
     Item libItemService = this.vapiAuthHelper.getStubFactory().createStub(Item.class,
sessionStubconfig);
     List<String> itemIds = libItemService.list(libraryId.getId());
     for (String itemId : itemIds) {
         ItemModel singleItem = libItemService.get(itemId);

// List the files uploaded to each library item and print their names and size
         com.vmware.content.library.item.File itemFilesService =
this.vapiAuthHelper.getStubFactory().createStub(com.vmware.content.library.item.File.class,
sessionStubconfig);
         List<com.vmware.content.library.item.FileTypes.Info> fileInfos =
                itemFilesService.list(itemId);
             for (com.vmware.content.library.item.FileTypes.Info singleFile : fileInfos) {
                 System.out.println("Library item with name "+ singleFile.getName() + " has
size
                                         " + singleFile.getSize());
             }

// Change the name and description of the library item with the specified name
               if (singleItem.getName().equals("simpleVmTemplate")) {
                    ItemModel libItemUpdated = new ItemModel();
                    libItemUpdated.setName("newItemName");
                    libItemUpdated.setDescription("Description of the newItemName");
                    libItemService.update(singleItem.getId(), libItemUpdated);
                }
        }
```

Python Example of Changing the Settings for a Library Item

This example shows how to find an item by using the item name and then how to change the name and description of the retrieved item.

This example uses the steps that are described in the Edit the Settings of a Library Item procedure.

Note For related code samples, see the `vsphere-automation-sdk-python` VMware repository at GitHub.

```
...

# 1 - List the items in a published library.
```

```
item_stub = library_client.Item(my_stub_config)
item_ids = item_stub.list(my_library_id)

# 2 - List the files uploaded to each library item and print their names and sizes.
file_stub = item_client.File(my_stub_config)
for item_id in item_ids :
  item = item_stub.get(item_id)
  file_infos = file_stub.list(item_id)
  for file_info in file_infos :
    print('Library item {} has file {} with size {}'.format(item.name, file_info.name,
file_info.size))

# 3 - For a library item with a specified name,
#     create an ItemModel to change the name and description of the library item.
if item.name == 'simpleVmTemplate' :
  print('Library item {} with description {}'.format(item.name, item.description))
  item_model = library_client.ItemModel()
  item_model.name = 'newItemName'
  item_model.description = 'Description of the newItemName'
  item_stub.update(library_item_id=item_id,
                   update_spec=item_model)
  print('has been changed to:')
  print('library item {} with description {}'.format(item_model.name, item_model.description))
```

Upload a File from a Local System to a Library Item

You can upload different types of files from a local system to a library item that you want to use in the vSphere Automation environment.

Prerequisites

- Create an empty library item. See Create an Empty Library Item.
- Verify that you have access to the `UpdateSession` and `File` services.

Procedure

1. Create an `UpdateSessionModel` instance to track the changes that you make to the library item.
2. Create an update session by using the `UpdateSession` service.
3. Create an `AddSpec` instance to describe the upload method and other properties of the file to be uploaded.
4. Create the request for changing the item by using the `File` service.
5. Upload the file that is on the local system.
6. Complete and delete the update session to apply the changes to the library item.

Java Example of Uploading Files to a Library Item from a Local System

This example shows how to upload an ISO image file from a local system to a library item.

This example uses the steps that are described in the Upload a File from a Local System to a Library Item procedure.

Note For related code samples, see the `vsphere-automation-sdk-java` VMware repository at GitHub.

```
...

// Access the com.vmware.content.library.item.updatesession.File.
// and the UpdateSession services by using the vSphere Automation Endpoint.
    File uploadFileService = this.vapiAuthHelper.getStubFactory().createStub(File.class,
sessionStubconfig);
    UpdateSession uploadService=
this.vapiAuthHelper.getStubFactory().createStub(UpdateSession.class, sessionStubconfig);

// Create an UpdateSessionModel instance to track the changes you make to the item.
    UpdateSessionModel updateSessionModel = new UpdateSessionModel();
    updateSessionModel.setLibraryItemId(newItem);

// Create a new update session.
    String clientToken = UUID.randomUUID().toString();
    String sessionId = uploadService.create(clientToken, updateSessionModel);

// Create an instance of the HttpClient class which is part of the
// com.vmware.vcloud.suite.samples.common package.
    try {
       HttpClient httpClient = new HttpClient(true);

// Create a new AddSpec instance to describe the properties of the file to be uploaded.
       FileTypes.AddSpec fileSpec = new FileTypes.AddSpec();
       fileSpec.setName("ESXi patch");
       fileSpec.setSourceType(FileTypes.SourceType.PUSH);

// Link the ISO file specification to the update session.
       FileTypes.Info fileInfo = uploadFileService.add(sessionId, fileSpec);

// Use the HTTP library to upload the file to the library item.
       URI uploadUri = fileInfo.getUploadEndpoint().getUri();
       java.io.File file = new java.io.File("/updates/esxi/esxi_patch.iso");
       String transferUrl = uploadUri.toURL().toString();
       httpClient.upload(file, transferUrl);

// Mark the upload session as completed.
       uploadService.complete(sessionId);
    } finally {
         uploadService.delete(sessionId);
    }
```

Python Example of Uploading Files to a Library Item from a Local System

This example shows how to upload an ISO image file from the local system to a library item.

This example uses the steps that are described in the Upload a File from a Local System to a Library Item procedure.

Note For related code samples, see the `vsphere-automation-sdk-python` VMware repository at GitHub.

```
...

# 1 - Create an instance of the ItemModel class and specify the item settings.
item_model = library_client.ItemModel()
item_model.name = 'ESXi patches'
item_model.description = 'ESXi security patches'
item_model.type = 'iso'
item_model.library_id = my_library_id
idem_token = str(uuid.uuid4())
item_stub = library_client.Item(my_stub_config)
item_id = item_stub.create(create_spec=item_model,
client_token=idem_token)

# 2 - Create an UpdateSessionModel instance to track the changes you make to the item.
update_session_model = item_client.UpdateSessionModel()
update_session_model.library_item_id = item_id

# 3 - Create an update session from the model.
idem_token = str(uuid.uuid4())
update_session_stub = update_session_client.UpdateSession(my_stub_config)
session_id = update_session_stub.create(create_spec=update_session_model
client_token=idem_token)

try :
  # 4 - Create a new AddSpec instance to describe the properties of the file to be uploaded.
  file_spec = update_session_client.AddSpec()
  file_spec.name = 'ESXi patch'
  file_spec.source_type = update_session_client.File.SourceType.PUSH

  # 5 - Link the ISO file spec to the update session.
  update_file_stub = update_session_stub.File(my_stub_config)
  file_info = update_file_stub.File.add(update_session_id=session_id,
                                        file_spec=file_spec)
  # 6 - Use HTTP library to upload the file to the library item.
  upload_uri = file_info.upload_endpoint.uri
  file_name = "/updates/esxi/esxi_patch.iso"
  host = urlparse.urlsplit(upload_uri)
  connection = httplib.HTTPConnection(host.netloc)
  with open(file_name, "rb") as f :
  connection.request("PUT", upload_uri, f)

  # 7 - Commit the updates.
  library_item_service.UpdateSession.complete(session_id)

  finally :
  # 8 - Delete the session.
  library_item_service.UpdateSession.delete(session_id)
```

Upload a File from a URL to a Library Item

You can upload different types of files from a local system to a library item that you want to use in the vSphere Automationenvironment.

Prerequisites

- Create an empty library item. See Create an Empty Library Item.
- Verify that you have access to the `UpdateSession` and `File` services.

Procedure

1. Create an `UpdateSessionModel` instance to track the changes that you make to the library item.
2. Create an update session by using the `UpdateSession` service.
3. Create a file specification to describe the upload method and other properties of the file to be uploaded.
4. Specify the location of the file to be uploaded by creating a `TransferEndpoint` instance.
5. Add the file source endpoint to the file specification.
6. Create a request for changing the item by using the configured file specification.
7. Complete the update session to apply the changes to the library item.

Java Example of Uploading a File from a URL to a Library Item

This example shows how to upload a file from a URL to a library item. The example is based on the code in the `ItemUploadHelper.java` sample file.

This example uses the steps that are described in the Upload a File from a URL to a Library Item procedure.

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

```
...

// Create a new library item. See Create an Empty Library
     Item.

...

// Access the com.vmware.content.library.item.updatesession.File
// and the UpdateSession services by using the vSphere Automation Endpoint.
   File uploadFileService = this.vapiAuthHelper.getStubFactory().createStub(File.class,
sessionStubConfig);
   UpdateSession uploadService =
this.vapiAuthHelper.getStubFactory().createStub(UpdateSession.class, sessionStubConfig);

// Create an UpdateSessionModel instance to track the changes you make to the item.
        UpdateSessionModel updateSessionModel = new UpdateSessionModel();
```

```
        updateSessionModel.setLibraryItemId(newItem);

// Create a new update session.
        String clientToken = UUID.randomUUID().toString();
        String sessionId = uploadService.create(clientToken, updateSessionModel);

// Create a new AddSpec instance to describe the properties of the file to be uploaded.
         FileTypes.AddSpec fileSpec = new AddSpec();
         fileSpec.setName("ESXi patch");
         fileSpec.setSourceType(SourceType.PULL);

// Specify the location from which the file is uploaded to the library item.
         TransferEndpoint endpoint = new TransferEndpoint();
         endpoint.setUri(URI.create("http://www.acme.com/patches_ESXi65/ESXi_patch.iso"));
         fileSpec.setSourceEndpoint(endpoint);
         uploadFileService.add(sessionId, fileSpec);

// Mark the session as completed.
         uploadService.complete(sessionId);
```

Python Example of Uploading a File from a URL to a Library Item

This example shows how to upload a file from a URL to a library item. The example is based on the code in the `cls_api_helper.py` sample file.

This example uses the steps that are described in the Upload a File from a URL to a Library Item procedure.

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 a new library item to hold the uploaded file.
item_model = library_client.ItemModel()
item_model.name = 'ESXi patches'
item_model.description = 'ESXi security patches'
item_model.type = 'iso'
item_model.library_id = my_library_id
idem_token = str(uuid.uuid4())
item_stub = library_client.Item(my_stub_config)
item_id = item_stub.create(create_spec=item_model, client_token=idem_token)

# 2 - Create an UpdateSessionModel instance to track the changes you make to the item.
update_session_model = item_client.UpdateSessionModel()
update_session_model.library_item_id = item_id

# 3 - Create an update session from the model.
idem_token = str(uuid.uuid4())
update_session_stub = update_session_client.UpdateSession(my_stub_config)
session_id = update_session_stub.create(create_spec=update_session_model,
client_token=idem_token)
```

```
try :
  # 4 - Create a new AddSpec instance to describe the properties of the file to be uploaded.
  file_spec = update_session_client.AddSpec()
  file_spec.name = 'ESXi patch'
  file_spec.source_type = update_session_client.File.SourceType.PULL

  # 5 - Specify the location from which the file is to be uploadod to the library item.
  endpoint = item_client.TransferEndpoint()
  endpoint.uri = 'http://www.example.com/patches_ESXi65/ESXi_patch.iso'
  file_spec.source_endpoint = endpoint

  # 6 - Link the file specification to the update session.
  update_file_stub = update_session_client.File(my_stub_config)
  update_file_stub.File.add(update_session_id=session_id, file_spec=file_spec)

  # 7 - Mark session as completed, to initiate the asynchronous transfer.
  update_session_stub.complete(session_id)
```

Download Files to a Local System from a Library Item

You might want to download files to a local system from a library item and then make changes to the files before you use them.

Procedure

1. Create a download session model to specify the item, which contains the file that you want to download.
2. Access the `File` service and retrieve the file that you want to export to your system within the new download session.
3. Prepare the files that you want to download and wait until the files are in the prepared state.
4. Retrieve the download endpoint URI of the files.
5. Download the files with an HTTP GET request.
6. Delete the download session after all files are downloaded.

Java Example of Downloading Files from a Library Item to Your Local System

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

This example uses the steps that are described in the Download Files to a Local System from a Library Item procedure.

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

```
...

// Access the DownloadSession service.
     DownloadSession downloadSessionService =
vapiAuthHelper.getStubFactory().createStub(DownloadSession.class, sessionStubConfig);
```

```
// Create a new download session model.
     DownloadSessionModel downloadSessionModel = new DownloadSessionModel();
     downloadSessionModel.setLibraryItemId(libItem.getId());
     String downloadSessionId = downloadSessionService.create(UUID.randomUUID().toString(),
downloadSessionModel);

// Access the File service and retrieve the files you want to export.
     File downloadFileService = vapiAuthHelper.getStubFactory().createStub(File.class,
sessionStubConfig);
     List<FileTypes.Info> downloadFileInfos = downloadFileService.list(downloadSessionId);
     for (FileTypes.Info downloadFileInfo : downloadFileInfos) {

// Make sure all files are in the prepared state before you precede with the downloading
operation.
         downloadFileService.prepare(downloadSessionId, downloadFileInfo.getName(),
EndpointType.HTTPS);
         long timeOut = 360;
         Long endTime = System.currentTimeMillis() + timeOut * 1000;
         try {
              Thread.sleep(5000);
         } catch (InterruptedException e) {
                   System.out.println(e);
         }
         FileTypes.PrepareStatus expectedStatus =
com.vmware.content.library.item.downloadsession.File.PrepareStatus.PREPARED;
         downloadFileInfo = downloadFileService.get(downloadSessionId,
downloadFileInfo.getName());
         FileTypes.PrepareStatus currentStatus = downloadFileInfo.getStatus();
         if (currentStatus == expectedStatus) {

// When the files are prepared, you can retrieve the download information for each file.
            downloadFileInfo = downloadFileService.get(downloadSessionId,
downloadFileInfo.getName());
            try {
                 URI downloadUri = downloadFileInfo.getDownloadEndpoint().getUri();
                 String downloadUrl = downloadUri.toURL().toString();

// Run an HTTP GET request and pass the download endpoints of the files.
                 HttpClient httpClient = new HttpClient(true);
                 InputStream inputStream = httpClient.downloadFile(downloadUrl);
                 String fileNameDownload = downloadFileInfo.getName();
                 File tmpDir = new java.io.File("tmp");
                 tmpDir.mkdir();
                 String fullPath = tmpDir.getAbsolutePath() +
                       System.getProperty("file.separator") + fileNameDownload;

// Copy the files to the directory on your machine.
                 Files.copy(inputStream,
Paths.get(fullPath),StandardCopyOption.REPLACE_EXISTING);
            } catch (MalformedURLException e) {
               System.out.println("Failed to download due to IOException!" + e);
               throw new RuntimeException("Failed to download due to IOException!", e);
            } catch (IOException e) {
               System.out.println("IO exception during download" + e);
```

```
                throw new RuntimeException("Failed to download due to IOException!", e);

// Delete the download session after all files are downloaded.
            } finally {
                 downloadFileService.delete(downloadSessionId);
            }
        } else {
            while (endTime > System.currentTimeMillis()) {
                downloadFileInfo = downloadFileService.get(downloadSessionId,
                             downloadFileInfo.getName());
                currentStatus = downloadFileInfo.getStatus();
                if (currentStatus == expectedStatus) {
                    return;
                } else if (currentStatus ==
com.vmware.content.library.item.downloadsession.File.PrepareStatus.ERROR) {
                   System.out.println("DownloadSession Info : " +
downloadSessionService.get(downloadSessionId));
                  throw new RuntimeException("Error while waiting for download file status to
be PREPARED...");
              }
          }
      }
   }
```

Python Example of Downloading Files from a Library Item to Your Local System

This example uses the code in the `cls_api_helper.py` sample file.

This example uses the steps that are described in the Download Files to a Local System from a Library Item procedure.

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 a new download session model.
download_session_model = item_client.DownloadSessionModel()
download_session_model.library_item_id = my_library_item_id
download_session_stub = item_client.DownloadSession(my_stub_config)
idem_token = str(uuid.uuid4())
download_session_id = download_session_stub.create(create_spec=download_session_model,
client_token=idem_token)

# 2 - Access the File service and retrieve the files you want to export.
download_session_file_stub = download_session_client.File(my_stub_config)
file_infos = download_session_file_stub.list(download_session_id)
for file_info in file_infos :
  download_session_file_stub.prepare(download_session_id, file_info.name)

# 3 - Wait until the file is in the prepared state before downloading.
  download_info = download_session_file_stub.get(download_session_id, file_info.name)
  while (DownloadSessionFile.PrepareStatus.PREPARED != download_info.status) :
  time.sleep(30)
```

```
# 4 - Download the file with an HTTP GET request.
  response = urllib2.urlopen(download_info.download_endpoint.uri)
  file_path = os.path.join(my_directory, file_info.name)
  with open(file_path, 'wb') as f :
  f.write(response.read())

# 5 - Delete the download session after all files are downloaded.
download_session_stub.delete(download_session_id)
```

Synchronizing a Library Item in a Subscribed Content Library

The items in a subscribed library have features that are distinct from the items in a local library. Synchronizing the content and the metadata of an item in a subscribed library depends on the synchronization mechanism of the subscribed library.

Table 9-2. Options for Synchronizing a Library Item

Synchronization Type of the Subscribed Library	Description
Synchronized on demand	If the subscribed library is synchronized on demand, you can use the `sync` method on the `SubscribedItem` service and pass as arguments the library item ID and `true`. When you perform the task, both the item metadata and the content are synchronized. To synchronize only the metadata of the item, pass the library ID and `false` as arguments to the method.
Not synchronized on demand	If the subscribed library is not synchronized on demand, you can use the `sync` method on the `SubscribedItem` service and pass as argument the item ID. In this case, the content of the item is always synchronized and the Boolean value is ignored when the call is run.
Synchronized automatically	If the subscribed library is synchronized automatically, you can also use the `sync` method to force the synchronization of an item. Method execution depends on whether the subscribed library is synchronized on demand.

Removing the Content of a Library Item

You can remove the content from a library item to free space on your storage.

If you create a subscribed library with the option to synchronize library content on demand, only the metadata for the library items is stored. When you want to use the items in the library, you must force synchronization on the items to download their content. When you no longer need the files in an item, you can remove the cached content of the library item and free storage space. To achieve this task use the `evict` function of the `SubscribedItem` object.

Deleting a Library Item

You can remove a library item from a local library when you no longer need it.

To remove a library item from a library, you can call the `delete` method on the `Item` object and pass the library item ID as an argument. The item content is asynchronously removed from the storage.

You cannot remove items from a subscribed library. If you remove an item from a local library, the item is removed from the subscribed library when you perform a synchronization task on the subscribed library item.

Content Library Support for OVF and OVA Packages

You can use the objects and methods provided by the Content Library API to manage OVF and OVA packages.

Open Virtualization Format (OVF) is an industry standard that describes metadata about a virtual machine image in an XML format. An OVF package includes an XML descriptor file and optionally disk images, resource files (such as ISO files), manifest files, and certificate files.

An OVA package is a single file that contains all OVF package files in an archived form. After you upload an OVA package to a content library, the OVA file is converted to the standard OVF package.

When you try to upload signed content to a content library, you might receive preview warnings. Signed content can be either OVF or OVA packages that contain manifest and certificate files. If you do not respond to the preview warnings, the upload fails. To complete an upload operation successfully, you must ignore any preview warnings by using the `WarningBehavior` class.

With the vSphere Automation API, you can use the OVF package in a content library to deploy virtual machines and vApps on hosts, resource pools, and clusters. You can also use the API to create OVF packages in content libraries from vApps and virtual machines on hosts, resource pools, and clusters.

When you create library items to store OVF packages, you must set the item type to `ovf`. To comply with the specific standards of the OVF packages, the vSphere Automation API provides the `LibraryItem` class.

Working with OVF and OVA Packages in a Content Library

You can upload an OVF or OVA package to a library item by using the `UpdateSession` interface. You can also download an OVF and OVA packages from a content library to your local file system.

In case you want to upload an OVF package, the location of the content determines whether you can pull the content from a URL or push the content directly to a content library. For information about uploading content to library items, see Upload a File from a Local System to a Library Item and Upload a File from a URL to a Library Item.

To download the files that are included in an OVF or OVA package to your local file system, use the `DownloadSession` interface. For more information, see Download Files to a Local System from a Library Item .

Upload an OVF or an OVA Package from a URL to a Library Item

You can upload an OVF or an OVA package from a Web server to a library item.

Note If you try to upload a signed OVF package and it returns preview warnings, you must ignore the preview warnings to complete the upload.

Prerequisites

- Create a new local content library or retrieve the desired existing content library.
- Required privileges: **Content library.Add library** item and **Content library.Update files** on the library.

Procedure

1. Create an empty library item.
2. Create an update session object.
3. Create an `AddSpec` object to describe the properties and the upload location of the OVF descriptor file or of the OVA package file.
4. Link the `AddSpec` object to the update session.

   All files that are included in the OVF package are automatically uploaded.
5. Complete the asynchronous transfer.

Python Example of Uploading an OVF Package from a URL to a Library Item

This example is based on the `ovf_import_export.py` sample file.

This example uses the steps that are described in the Upload an OVF or an OVA Package from a URL to a Library Item procedure.

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 a empty library item to describe the virtual machine.
item_model = library_client.ItemModel()
item_model.name = "ubuntu-vm"
item_model.description = "ubuntu 7.0"
item_model.library_id = my_library_id
item_model.type = "ovf"
client_token = str(uuid.uuid4())
item_stub = library_client.Item(my_stub_config)
item_id = item_stub.create(create_spec=item_model,
client_token=client_token)

# 2 - Create an update session.
update_session_model = item_client.UpdateSessionModel()
update_session_model.library_item_id = item_id
```

```
client_token = str(uuid.uuid4())
update_session_stub = update_session_client.UpdateSession(my_stub_config)
session_id = update_session_stub.create(create_spec=update_session_model,
client_token=client_token)

# 3 - Create a file specification for the OVF envelope file.
file_spec = update_session_client.AddSpec()
file_spec.name = "ubuntu.ovf"
file_spec.source_type = File.SourceType.PULL
endpoint = item_client.TransferEndpoint()
endpoint.uri = "http://www.example.com/images/ubuntu.ovf"
file_spec.source_endpoint = endpoint

# 4 - Link the file specification to the update session.
update_file_stub = update_session_client.File(my_stub_config)
update_file_stub.File.add(update_session_id=session_id,
file_spec=file_spec)

# 5 - Initiate the asynchronous transfer.
update_session_stub.complete(session_id)
```

Upload an OVF or OVA Package from a Local File System to a Library Item

You can upload an OVF or OVA package from a local file system. This procedure describes how to use the `AddSpec` object after you have created a library item and initiated an update session.

Note If you try to upload a signed OVF package and it returns preview warnings, you must ignore the preview warnings to complete the upload.

Prerequisites

- Create a new local content library or retrieve the desired existing content library.
- Required privileges: **Content library.Add library item** and **Content library.Update files** on the library.

Procedure

1. Create a library item.
2. Create an update session.
3. Create an `AddSpec` object to describe the properties and the upload locations of the OVF descriptor file or of the OVA package file.
4. Link the `AddSpec` object to the update session.
5. (Optional) Create an `AddSpec` object for each VMDK file included in the OVF package.
6. Add all `AddSpec` objects to the update session.

   If you upload an OVF package and it has a VMDK file included, you must repeat steps 5 and 6. If you are uploading a signed OVF package, steps 5 and 6 must also be repeated for the manifest and certificate files included in the OVF package.

7 Initiate the upload operation.

8 Complete the update session.

9 Delete the session.

Python Example of Uploading an OVF Package to a Library Item from Your Local File System

This example is based on the `ovf_import_export.py` sample file.

This example uses the steps that are described in the Upload an OVF or OVA Package from a Local File System to a Library Item procedure.

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 an empty library item to describe the VM/VApp.
item_model = library_client.ItemModel()
item_model.name = "ubuntu-vm"
item_model.description = "ubuntu 7.0"
item_model.library_id = my_library_id
item_model.type = "ovf"
client_token = str(uuid.uuid4())
item_stub = library_client.Item(my_stub_config)
item_id = item_stub.create(create_spec=item_model, client_token=client_token)

# 2 - Create an update session.
update_session_model = item_client.UpdateSessionModel()
update_session_model.library_item_id = item_id
client_token = str(uuid.uuid4())
update_session_stub = update_session_client.UpdateSession(my_stub_config)
session_id = update_session_stub.create(create_spec=update_session_model,
client_token=client_token)

try :
  # 3 - Create a file spec for the OVF envelope file.
  file_spec = update_session_client.AddSpec()
  file_spec.name = "ubuntu.ovf"
  file_spec.source_type = update_session_client.File.SourceType.PUSH

  # 4 - Link the OVF file spec to the update session.
  update_file_stub = update_session_client.File(my_stub_config)
  file_info = update_file_stub.File.add(update_session_id=session_id, file_spec=file_spec)
  upload_uri = file_info.upload_endpoint.uri

  # 5 - Use HTTP library to push the file, out of band.
  file_name = "/medias/vms/ubuntu.ovf"
  host = urlparse.urlsplit(upload_uri)
  connection = httplib.HTTPConnection(host.netloc)
  with open(file_name, "rb") as f :
    connection.request("PUT", upload_uri, f)
```

```
  # 6 - Create a file spec for the VMDK file.
  file_spec = update_session_client.AddSpec()
  file_spec.name = "ubuntu_disk.vmdk"
  file_spec.source_type = File.SourceType.PUSH

  # 7 - Add the VMDK file spec to the update session.
  file_info = update_file_stub.File.add(update_session_id=session_id, file_spec=file_spec)
  upload_uri = file_info.upload_endpoint().uri

  # 8 - Use HTTP library to push the file.
  file_name = "/medias/storage/ubuntu_disk.vmdk"
  host = urlparse.urlsplit(upload_uri)
  connection = httplib.HTTPConnection(host.netloc)
  with open(file_name, "rb") as f :
    connection.request("PUT", upload_uri, f)

  # 9 - Commit the updates.
  update_session_stub.complete(session_id)

finally :
  # 10 - Delete the session.
  update_session_stub.delete(session_id)
```

Python Example of Uploading an OVA Package to a Library Item

This example is based on the `signed_ova_import.py` sample file.

This example uses the steps that are described in the Upload an OVF or OVA Package from a Local File System to a Library Item procedure.

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

```
...

# 1 - Specify the OVA filename and location.
SIGNED_OVA_FILENAME = 'nostalgia-signed.ova'
SIGNED_OVA_RELATIVE_DIR = '../resources/signedOvaWithCertWarning'

# 2 - Create a new library item in the content library for uploading the files.
self.lib_item_id = self.helper.create_library_item(library_id=self.local_lib_id,
                                                   item_name=self.lib_item_name,
                                                   item_description='Sample template from ova
file',
                                                   item_type='ovf')

# 3 - Set a pointer to the OVA file you want to upload.
ova_file_map = self.helper.get_ova_file_map(self.SIGNED_OVA_RELATIVE_DIR,
                                            local_filename=self.SIGNED_OVA_FILENAME)

# 4 - Create a new update session for uploading the files.
session_id = self.client.upload_service.create(
    create_spec=UpdateSessionModel(library_item_id=self.lib_item_id),
    client_token=generate_random_uuid())
```

```
self.helper.upload_files_in_session(ova_file_map, session_id)

# 5 - Wait for terminal preview state and obtain preview warnings.
self.wait_for_terminal_preview_state(session_id, AVAILABLE)
session = self.client.upload_service.get(session_id)
preview_info = session.preview_info

# 6 - Ignore preview warnings on session, if any.
ignore_warning_behaviors = []
for warning_type in preview_warning_types:
    warning_behavior = WarningBehavior(type=warning_type, ignored=True)
    ignore_warning_behaviors.append(warning_behavior)
self.client.upload_service.update(session_id, update_spec=UpdateSessionModel(
    warning_behavior=ignore_warning_behaviors))

# 7 - Complete the update session.
self.client.upload_service.complete(session_id)

# 8 - Delete the session.
self.client.upload_service.delete(session_id)
```

Creating Virtual Machines and vApps from Templates in a Content Library

You can create VM and OVF templates from virtual machines and vApps in your inventory. You can then deploy virtual machines and vApps from the templates that are stored in a content library.

Create a VM Template in a Content Library from a Virtual Machine

By using the vSphere Automation API or HTTP requests, you can create a VM template in a content library from an existing virtual machine in your vCenter Server inventory.

When you call the `create` function of the `com.vmware.vcenter.vm_template.LibraryItems` service, a VM template is created as a library item in your local content library. If the operation is successful, the `LibraryItems` service returns the ID of the newly created library item.

To create a library item that contains a VM template, you can use the `ctreate` function of the `LibraryItems` interface, or the `POST https://<vCenter_Server_IP>/api/vcenter/vm-template/library-items` HTTP request. You can review the information about a VM template by using the `get` function of the `LibraryItems` interface or the `GET https://<vCenter_Server_IP>/api/vcenter/vm-template/library-items/<VM_Template_Item_ID>` HTTP request. For information about how to create a VM template by using the vSphere Client, see the *vSphere Virtual Machine Administration* documentation.

For information about the available and mandatory parameters, see the *API Reference* documentation.

Prerequisites

- Verify that you have administrative privileges on your vCenter Server instance.
- Verify that you created a vSphere Automation session to your vCenter Server.
- Verify that you created a local library by using the vSphere Client or the vSphere Automation APIs.

Procedure

1 Get the ID of your ESXi host on which you want to store the VM template.

  You can use the `list` function of the `com.vmware.vcenter.Host` interface or the `GET https://<vCenter_Server_IP>/api/vcenter/host` HTTP request.

2 Get the ID of the datastore on which you want to store the VM template files.

  You can use the `list` function of the `com.vmware.vcenter_client.Datastore` interface or the `GET https://<vCenter_Server_IP>/api/vcenter/datastore` HTTP request.

3 Get the ID of the virtual machine that you want to save as a VM template.

  You can use the `list` method of the `com.vmware.vcenter_client.VM` interface or the `GET https://<vCenter_Server_IP>/api/vcenter/vm` HTTP request.

4 Get the ID of your local library.

  You can get the list of the local libraries in your vCenter Server and review the information about each library by using the `list` and `get(library_id)` functions of the `com.vmware.content_client.LocalLibrary` or the following HTTP requests: `GET https://<vCenter_Server_IP>/api/com/vmware/content/local-library` and `GET https://<vCenter_Server_IP>/api/com/vmware/content/library/id:<library_id>`.

5 Create a library item specification for the VM template.

  You can use the `com.vmware.vcenter.vm_template.LibraryItems.CreateSpec` class or the body of the `POST https://<vCenter_Server_IP>/api/vcenter/vm-template/library-items` HTTP request.

  a Specify the local library, source virtual machine, and the name of the library item by using the `library`, `source_vm`, and `name` parameters. You must use the IDs of the local library and source virtual machine.

  b Specify the placement information for your VM template.

    You can use the `LibraryItems.CreatePlacementSpec` class or the `placement` parameters in the body of the HTTP request. To specify the host, resource pool, cluster, and folder, you must use their IDs.

c Specify the datastore on which you want to store the log, configuration, and disk files of your VM template.

To specify the storage backing for the VM template, you can use the `com.vmware.vcenter.vm_template.LibraryItems.CreateSpecVmHomeStorage` and `com.vmware.vcenter.vm_template.LibraryItems.CreateSpecDiskStorage` classes or the `vm_home_storage` and `disk_storage` parameters in the body of the HTTP request. You must use the ID of the datastore.

d Include the placement and storage specifications in the library item specification.

6 Create a library item for storing the VM template.

You can use the `create(spec)` function of the `com.vmware.vcenter.vm_template.LibraryItems` interface or send the `POST https://<vCenter_Server_IP>/api/vcenter/vm-template/library-items` request.

Results

If the operation is successful, the `LibraryItems` service returns the ID of the library item that contains the VM template. For information about the available responses, see the *API Reference* documentation.

What to do next

- Review the information stored in the library item by using the `get(VM_template_item_ID)` function of the `com.vmware.vcenter.vm_template.LibraryItems` interface or the `GET https://<vCenter_Server_IP>/api/vcenter/vm-template/library-items/<library_item_id>` HTTP request. If you did not save the ID of the library item holding the VM template, you can check the UUID by using the vSphere Client. The URN ends with the ID of the library item and has the following format: `urn:vapi:com.vmware.content.library.Item:<VMTemplateItemID>`.

Python Example of Creating a VM Template in a Content Library from a Virtual Machine

This example shows how you can create a VM template from a virtual machine and add the template to a content library by using the vSphere Automation API. The example is based on the `create_vm_template.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.

```
class CreateVmTemplate(SampleBase):
    ...
    def _setup(self):
        # Required arguments
        self.vm_name = self.args.vmname
        self.datacenter_name = self.args.datacentername
        self.resource_pool_name = self.args.resourcepoolname
```

```
        self.datastore_name = self.args.datastorename

        # Optional arguments
        self.item_name = (self.args.itemname if self.args.itemname
                          else rand('vmtx-item-'))

        self.servicemanager = self.get_service_manager()
        self.client = ClsApiClient(self.servicemanager)
        self.helper = ClsApiHelper(self.client, self.skip_verification)

        session = get_unverified_session() if self.skip_verification else None
        self.vsphere_client = create_vsphere_client(server=self.server,
                                                    username=self.username,
                                                    password=self.password,
                                                    session=session)

    def _execute(self):
        # Get the identifiers for the virtual machine and resource pool
        vm_id = get_vm(self.vsphere_client, self.vm_name)
        assert vm_id
        resource_pool_id = get_resource_pool(self.vsphere_client,
                                             self.datacenter_name,
                                             self.resource_pool_name)
        assert resource_pool_id

        # Create a local content library
        storage_backings = self.helper.create_storage_backings(self.servicemanager,
                                                               self.datastore_name)
        self.library_id = self.helper.create_local_library(storage_backings,
                                                           self.library_name)

        # Build the library item create specification
        create_spec = VmtxLibraryItem.CreateSpec()
        create_spec.source_vm = vm_id
        create_spec.library = self.library_id
        create_spec.name = self.item_name
        create_spec.placement =
VmtxLibraryItem.CreatePlacementSpec(resource_pool=resource_pool_id)

        # Create a new library item from the source virtual machine
        self.item_id = self.client.vmtx_service.create(create_spec)
    ...
```

Create an OVF Template in a Content Library from a Virtual Machine or vApp

You can create library items from existing virtual machines or vApp. Use those library items later to deploy virtual machines and vApps on hosts and clusters in your vCenter Server environment.

Procedure

1 Create a `com.vmware.vcenter.ovf.LibraryItemTypes.DeployableIdentity` instance to specify the source virtual machine or vApp to be captured in an OVF template.

2 Create a `com.vmware.vcenter.ovf.LibraryItemTypes.CreateTarget` instance to identify the content library where the OVF template is stored.

3 Create a `com.vmware.vcenter.ovf.LibraryItemTypes.CreateSpec` instance to specify the properties of the OVF template.

4 Initiate a synchronous create operation by invoking the `create` function of the `com.vmware.vcenter.ovf.LibraryItem` service.

5 Verify the results of the `create` operation.

Java Example of Creating an OVF Template in a Content Library from a Virtual Machine

This example shows how to capture a virtual machine in an OVF template and store the file in a new library item in a specified library.

This example uses the steps that are described in the Create an OVF Template in a Content Library from a Virtual Machine or vApp procedure.

Note For related code samples, see the `vsphere-automation-sdk-java` VMware repository at GitHub.

```
...

// Specify the resource to be captured.
LibraryItemTypes.DeployableIdentity deployableIdentity = new
LibraryItemTypes.DeployableIdentity();
deployableIdentity.setType("VirtualMachine");
deployableIdentity.setId("vm-32");

// Create a target spec to identify a library to hold the new item.
LibraryItemTypes.CreateTarget createTarget = new LibraryItemTypes.CreateTarget();
createTarget.setLibraryId(myLibraryId);

// Specify OVF properties.
LibraryItemTypes.CreateSpec createSpec = new LibraryItemTypes.CreateSpec();
createSpec.setName("snap-32");
createSpec.setDescription("Snapshot of VM-32");

// Initiate synchronous capture operation.
LibraryItem itemStub = myStubFactory.createStub(LibraryItem.class, myStubConfiguration);
String clientToken = UUID.randomUUID().toString();
LibraryItemTypes.CreateResult result = itemStub.create(clientToken, deployableIdentity,
createTarget, createSpec);

// Verify capture status.
System.out.printf("Resource Type=%s (ID=%s) status:",
                  deployableIdentity.getType(),
                  deployableIdentity.getId());
  if (result.getSucceeded() == true) {
     System.out.println("Resource captured.");
```

```
}else {
   System.out.println("Capture failed.");
}

if (result.getError() != null) {

  for (OvfError error : result.getError().getErrors()) {
      System.out.printf("Error: %s", error.getMessage().toString());
  }

  for (OvfWarning warning : result.getError().getWarnings()) {
      System.out.printf("Warning: %s", warning.getMessage().toString());
  }

  for (OvfInfo info : result.getError().getInformation()) {
      List<LocalizableMessage> messages = info.getMessage();
  }

  for (LocalizableMessage message : messages) {
    System.out.printf("Message: %s", message.toString());
  }
}
```

Python Example of Creating an OVF Template in a Content Library from a Virtual Machine

This example shows how to capture a virtual machine in an OVF template and store the files in a new library item in a specified library.

This example uses the steps that are described in the Create an OVF Template in a Content Library from a Virtual Machine or vApp procedure.

Note For related code samples, see the `vsphere-automation-sdk-python` VMware repository at GitHub.

```
...

# Specify the resource to be captured.
deployable_identity = ovf_client.LibraryItem.DeployableIdentity();
deployable_identity.type = "VirtualMachine"
deployable_identity.id = "vm-32"

# Create a target spec to identify a library to hold the new item.
create_target = ovf_client.LibraryItem.CreateTarget()
create_target.library_id = my_library_id

# Specify OVF properties.
create_spec = ovf_client.LibraryItem.CreateSpec()
create_spec.name = "snap-32"
create_spec.description = "Snapshot of VM-32"

# Initiate synchronous capture operation.
lib_item_stub = ovf_client.LibraryItem(my_stub_config)
```

```
client_token = str(uuid.uuid4())
result = lib_item_stub.create(source=deployable_identity,
                              target=create_target,
                              create_spec=create_spec,
                              client_token=client_token)

# Verify capture status.
print("Resource Type={} (ID={}) status:".format(deployable_identity.type,
deployable_identity.id))

if result.succeeded == True :
  print("Resource captured.”)
else :
  print("Capture failed.")
if result.error is not None :
  for error in result.error.errors :
    print("Error {}".format(error.message))
  if len(result.error.warnings) > 0 :
    print("Warnings:")
    for warning in result.error.warnings :
      print("{}".format(warning.message))
  if len(result.error.information) > 0 :
    print("Messages:")
    for info in result.error.information :
      for message in info.messages :
        print("{}".format(message))
```

Deploy a Virtual Machine from a VM Template in a Content Library

By using the vSphere Automation APIs, you can deploy a virtual machine from a VM template stored in a content library.

To deploy a virtual machine from a VM template in a content library, call the `deploy` function of the `com.vmware.vcenter.vm_template.LibraryItems` interface or use the `POST https://<vCenter_Server_IP>/api/vcenter/vm-template/library-items/<VM_Template_Item_ID>?action=deploy` HTTP request. You can specify the power state and customize the guest operation system prior to the virtual machine deployment.

For information about the available and mandatory parameters, see the *API Reference* documentation.

Prerequisites

- Verify that you have administrative privileges on your vCenter Server instance.
- Verify that you created a vSphere Automation session to your vCenter Server instance.

Procedure

1 Review the information stored in the VM template library item.

  You can use the `get(VM_template_item_ID)` function of the `com.vmware.vcenter.vm_template.LibraryItems` interface and pass the ID of your VM

template item or the `GET https://<vCenter_Server_IP>/api/vcenter/vm-template/library-items/<VM_Template_Item_ID>` HTTP request. If you did not save the ID of your item, you can select the UUID of your VM template item by using the vSphere Client. The URN ends with the ID of the item and has the following format: `urn:vapi:com.vmware.content.library.Item:<VMTemplateItemID>`.

2 Get the ID of the host on which you want to deploy the virtual machine.

You can use the `list` function of the `com.vmware.vcenter.Host` interface or the `GET https://<vCenter_Server_IP>/api/vcenter/host` HTTP request.

3 Get the ID of the resource pool to which you want to add your virtual machine.

You can use the `list` function of the `com.vmware.vcenter.ResourcePool` interface or the `https://<vCenter_Server_IP>/api/vcenter/resource-pool` HTTP request.

4 Get the ID of the `VIRTUAL_MACHINE` folder to which you want to add your virtual machine.

You can use the `list` function of the `com.vmware.vcenter.Folder` interface or the `GET https://<vCenter_Server_IP>/api/vcenter/folder` HTTP request.

5 Get the ID of the datastore on which you want to store log, configuration, and disk files of the virtual machine.

You can use the `list` function of the `com.vmware.vcenter.Datastore` interface or the `GET https://<vCenter_Server_IP>/api/vcenter/datastore` HTTP request.

6 Create a deployment specification.

You can use the `com.vmware.vcenter.vm_template.LibraryItems.DeploySpec` class or the body of the `POST https://<vCenter_Server_IP>/api/vcenter/vm-template/library-items/<VM_Template_Item_ID>?action=deploy` HTTP request.

a Specify a name and description of the virtual machine that you want to deploy.

b Specify the place in your inventory on which you want to deploy the virtual machine such as an ESXi host, resource pool, and VM folder.

You can use the `com.vmware.vcenter.vm_template.LibraryItems.DeployPlacementSpec` class or the `placement` parameter in the body of the request. You must use the IDs of your inventory objects.

c Specify the datastore on which you want to store the log, configuration, and disk files of the virtual machine. You must use the ID of the datastore.

You can use the `DeploySpecVmHomeStorage` and `DeploySpecDiskStorage` classes or the `vm_home_storage`, and `disk_storage` parameters in the body of the request.

d (Optional) Specify the guest operating system and hardware customization specifications that you want to apply to the virtual machine during the deployment process. Add this information to the deployment specification.

You can use the `GuestCustomizationSpec` and `HardwareCustomizationSpec` classes or the `guest_customization` and `hardware_customization` parameters in the body of the request. You can get a list of the guest operating system customization specifications that are available in your vCenter Server by using the `list` function of the `com.vmware.vcenter.guest.CustomizationSpecs` interface or the `GET https://<vCenter_Server_IP>/api/vcenter/guest/customization-specs` HTTP request.

e Include the placement and storage specifications in the deployment specification.

7 Deploy a virtual machine from your VM template.

You can use the `deploy(template_library_item, spec)` function of the `com.vmware.vcenter.vm_template.LibraryItems` interface by passing the library item ID where the VM template is stores and the deployment specification. You can also send the `POST https://<vCenter_Server_IP>/api/vcenter/vm-template/library-items/<VM_Template_Item_ID>?action=deploy` HTTP request.

Results

If the operation is successful, the ID of the deployed virtual machine is returned. For information about the possible exceptions, see the *API Reference* documentation.

Python Example of Deploying a VM from a VM Template Library Item

This example shows how you can deploy a VM from a VM Template library item by using the API. The example is based on the `deploy_vm_template.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.

```
class DeployVmTemplate(SampleBase):
    ...
    def _setup(self):
        # Required arguments
        self.item_name = self.args.itemname
        self.datacenter_name = self.args.datacentername
        self.folder_name = self.args.foldername
        self.resource_pool_name = self.args.resourcepoolname
        self.datastore_name = self.args.datastorename

        # Optional arguments
        self.vm_name = self.args.vmname if self.args.vmname else rand('vm-')

        self.servicemanager = self.get_service_manager()
        self.client = ClsApiClient(self.servicemanager)
        self.helper = ClsApiHelper(self.client, self.skip_verification)
```

```
        session = get_unverified_session() if self.skip_verification else None
        self.vsphere_client = create_vsphere_client(server=self.server,
                                                    username=self.username,
                                                    password=self.password,
                                                    session=session)

    def _execute(self):
        # Get the identifiers of the resources used for deployment
        item_id = self.helper.get_item_id_by_name(self.item_name)
        assert item_id
        folder_id = get_folder(self.vsphere_client,
                               self.datacenter_name,
                               self.folder_name)
        assert folder_id
        resource_pool_id = get_resource_pool(self.vsphere_client,
                                             self.datacenter_name,
                                             self.resource_pool_name)
        assert resource_pool_id
        datastore_id = get_datastore_id(self.servicemanager,
                                        self.datastore_name)
        assert datastore_id

        # Build the deployment specification
        placement_spec = VmtxLibraryItem.DeployPlacementSpec(
            folder=folder_id,
            resource_pool=resource_pool_id)
        vm_home_storage_spec = VmtxLibraryItem.DeploySpecVmHomeStorage(
            datastore=datastore_id)
        disk_storage_spec = VmtxLibraryItem.DeploySpecDiskStorage(
            datastore=datastore_id)
        deploy_spec = VmtxLibraryItem.DeploySpec(
            name=self.vm_name,
            placement=placement_spec,
            vm_home_storage=vm_home_storage_spec,
            disk_storage=disk_storage_spec)

        # Deploy a virtual machine from the VM template item
        self.vm_id = self.client.vmtx_service.deploy(item_id, deploy_spec)
        self.vm = get_obj_by_moId(self.servicemanager.content,
                                  [vim.VirtualMachine], self.vm_id)
    ...
```

Deploy a Virtual Machine or vApp from an OVF Template in a Content Library

You can use the `com.vmware.vcenter.ovf.LibraryItem` service to deploy a virtual machine or vApp on a host, cluster, or resource pool from a library item.

Procedure

1 Create a `com.vmware.vcenter.ovf.LibraryItemTypes.DeploymentTarget` instance to specify the deployment location of the virtual machine or vApp.

2 Instantiate the `com.vmware.vcenter.ovf.LibraryItemTypes.ResourcePoolDeploymentSpec` class to define all necessary parameters for the deployment operation.

   For example, you can assign a name for the deployed virtual machine or vApp, and accept the End User License Agreements (EULAs) to complete the deployment successfully.

3 (Optional) Retrieve information from the descriptor file of the OVF template and use the information during the OVF template deployment.

4 Call the `deploy` method on the `LibraryItem` service.

5 Verify the outcome of the deployment operation.

Java Example of Deploying a Virtual Machine from a Library Item in a Resource Pool

This example shows how to deploy a virtual machine from a local library item in a resource pool. You can also see how to verify the results of the deployment operation.

This example uses the steps that are described in the Deploy a Virtual Machine or vApp from an OVF Template in a Content Library procedure.

Note For related code samples, see the `vsphere-automation-sdk-java` VMware repository at GitHub.

```
...

// Create a virtual machine deployment specification to accept any network resource.
ResourcePoolDeploymentSpec deploymentSpec = new ResourcePoolDeploymentSpec();
String vmName = "MyVirtualMachine";
deploymentSpec.setName(vmName);
deploymentSpec.setAcceptAllEULA(true);


// Create a deployment target specification to accept any resource pool.
String clusterName = "myCluster";
ManagedObjectReference clusterMoRef = VimUtil.getCluster(this.vimAuthHelper.getVimPort(),
this.vimAuthHelper.getServiceContent(), clusterName);
DeploymentTarget deploymentTarget = new DeploymentTarget();
deploymentTarget.setResourcePoolId(clusterMoRef.getValue());


// Retrieve the library items OVF information and use it for populating the
// deployment spec instance.
LibraryItem libItemStub = stubFactory.createStub(LibraryItem.class, myStubConfiguration);
OvfSummary ovfSummary = libItemStub.filter(libItemId, deploymentTarget);
deploymentSpec.setAnnotation(ovfSummary.getAnnotation());
String clientToken = UUID.randomUUID().toString();
DeploymentResult result = libItemStub.deploy(clientToken,libItemId,
                                                        deploymentTarget,
                                                        deploymentSpec);

// Verify the status of the resource deployment.
System.out.printf("Resource Type=%s (ID=%s) status: ",
```

```
                    result.getResourceId().getType(),
                    result.getResourceId().getId());

  if (result.getSucceeded() == true) {
      System.out.println("Resource instantiated.");

  } else {
      System.out.println("Instantiation failed.");
  }
```

Python Example of Deploying a Virtual Machine from a Library Item on a Resource Pool

This example is based on the `deploy_ovf_template.py` sample file.

This example uses the steps that are described in the Deploy a Virtual Machine or vApp from an OVF Template in a Content Library procedure.

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 VM deployment specification to accept any network resource.
deployment_spec = ovf_client.LibraryItem.ResourcePoolDeploymentSpec()
deployment_spec.accept_all_eula = True

# Create deployment target spec to accept any resource pool.
target_spec = ovf_client.LibraryItem.DeploymentTarget()

# Initiate synchronous deployment operation.
item_stub = ovf_client.LibraryItem(my_stub_config)
result = item_stub.deploy(my_library_item_id,
                          target_spec,
                          deployment_spec,
                          client_token=str(uuid.uuid4())

# Verify deployment status.
print("Resource Type={} (ID={}) status:".format(result.resource_id.type,
result.resource_id.id))
if result.succeeded == True :
  print("Resource instantiated.”)
else :
  print("Instantiation failed.")
if result.error is not None :
  for error in result.error.errors :
    print("Error {}".format(error.message))
  if len(result.error.warnings) > 0 :
    print("Warnings:")
    for warning in result.error.warnings :
      print("{}".format(warning.message))
  if len(result.error.information) > 0 :
    print("Messages:")
```

```
    for info in result.error.information :
      for message in info.messages :
        print("{}".format(message))
```

vSphere Tag Service

10

The vSphere Automation Tag Service supports the definition of tags that you can associate with vSphere objects or vSphere Automation resources.

Starting with vSphere 6.5, the vSphere Automation APIs provide programmatic access to creating and managing vSphere tags in your vSphere inventory.

For example, if you want to tag your virtual machines by guest operating system type, you can create a category called **`operating system`**. You can specify that it applies to virtual machines only and that only a single tag can be applied to a virtual machines at any time. This category can include the following tags: **`Windows`**, **`Linux`**, and **`Mac OS`**.

This chapter includes the following topics:

Creating vSphere Tags

You create a vSphere tag to add metadata to objects in the vSphere inventory. Tags are grouped in categories and each tag must have at least one category related to it. After you create the tag, you can associate the tag with a vSphere object.

Tags and categories can span multiple vCenter Server instances.

- If multiple on-premises vCenter Server instances are configured to use Enhanced Linked Mode, tags and tag categories are replicated across all these vCenter Server instances.
- When you use Hybrid Linked Mode, tags and tag categories are maintained across your linked domain. That means the on-premises SDDC and the VMware Cloud on AWS SDDC share tags and tag attributes. For more information about Hybrid Linked Mode, see "Hybrid Linked Mode" in the *VMware Cloud on AWS Product Documentation*.

Creating a Tag Category

You create tags in the context of a tag category. You must create a category before you can add tags within that category.

A tag category has the following properties:

- name
- description
- cardinality, or how many tags it can contain
- the types of elements to which the tags can be assigned

You can associate tags with both vSphere API managed objects and VMware vSphere Automation API resources.

Java Example of Creating a Tag Category

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

This example is based on the information that is provided in Creating a Tag Category.

The category `create()` function returns an identifier that you use when you create a tag for that category. The empty set for the associable types indicates that any object type can be associated with a tag in this category.

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

```
...

Category categoryStub = myStubFactory.createStub(Category.class,
                                                  myStubConfiguration);

// Set up a tag category create spec.
CategoryTypes.CreateSpec createSpec = new CategoryTypes.CreateSpec();
createSpec.setName("favorites");
createSpec.setDescription("My favorite virtual machines.");
createSpec.setCardinality(CategoryModel.Cardinality.MULTIPLE);
Set<String> associableTypes = new HashSet<String>();
createSpec.setAssociableTypes(associableTypes);

String newCategoryId = categoryStub.create(createSpec);
```

Python Example of Creating a Tag Category

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

This example is based on the information that is provided in Creating a Tag Category.

The category `create()` function returns an identifier that you use when you create a tag for that category. The empty set for the associable types indicates that any object type can be associated with a tag in this category.

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

```
...

category_stub = tagging_client.Category(my_stub_config)

# Set up a tag category create spec.
tc_create_spec = category_stub.CreateSpec(name = 'favorites',
                                          description = 'My favorite virtual machines',
                                          cardinality = CategoryModel.Cardinality.MULTIPLE,
                                          associable_types = set())

# Create the tag category.
fav_category_id = category_stub.create(create_spec)
```

Creating a Tag

After you create a tag category, you can create tags within that category

A tag has the following properties:

- name
- description
- category ID

Java Example of Creating a Tag

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

This example creates a tag specification and then uses it to create the tag. The tag specification references the category identifier that was returned from the category create operation. Use the returned tag identifier for subsequent operations on the tag.

This example is based on the information that is provided in Creating a Tag.

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

```
...

Tag tagStub = myStubFactory.createStub(Tag.class,
                                       myStubConfiguration);

// Set up a tag create spec.
TagTypes.CreateSpec spec = new TagTypes.CreateSpec();
spec.setName("red");
```

```
spec.setDescription("My favorite color");
spec.setCategoryId(newCategoryId);

String tagId = tagStub.create(spec);
```

Python Example of Creating a Tag

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

This example creates a tag specification and then uses it to create the tag. The tag specification references the category identifier that was returned from the category create operation. Use the returned tag identifier for subsequent operations on the tag.

This example is based on the information that is provided in Creating a Tag.

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

```
...

# Set up a tag create spec.
tag_create_spec = tag_stub.CreateSpec(name='red',
                                      description='My favorite color',
                                      category_id=fav_category_id)

# Create the tag.
tag_stub = tagging_client.Tag(my_stub_config)
tag_id = tag_stub.create(create_spec)
```

Creating Tag Associations

After you create a tag category and create a tag within the category, you can associate the tag with a vSphere managed object or a vSphere Automation resource. An association is a simple link that contains no data of its own. You can enumerate objects that are attached to a tag or tags that are attached to an object.

Tag associations are local to a vCenter Server instance. When you request a list of tag associations from a vCenter Server system, it enumerates only the associations that it has stored.

When you associate a tag with an object, the object's type must match one of the associable types specified for the category to which the tag belongs.

Assign the Tag to a Content Library

After you create a tag, you can assign the tag to a vSphere Automation resource.

Procedure

1 Construct a dynamic object identifier for the library.

   The dynamic identifier includes the type and ID of the object.

2 Attach the tag to the content library.

Java Example of Assigning a Tag to a Content Library

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

This example uses the steps that are described in the Assign the Tag to a Content Library procedure.

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 - Create a dynamic type ID for the content library.
DynamicID libraryDynamicId = new DynamicID(Library.RESOURCE_TYPE,
                                           myLibrary.getId());

// 2- Attach the tag to the ClusterComputeResource managed object.
TagAssociation tagAssociationStub = myStubFactory.createStub(TagAssociation.class,
                                                             myStubConfig);
tagAssociationStub.attach(myLibrary.getId(),
                          libraryDynamicId);
```

Python Example of Assigning a Tag to a Content Library

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

This example uses the steps that are described in the Assign the Tag to a Content Library procedure.

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 a dynamic type ID for the content library.
library_dynamic_id = DynamicID(type=Library.RESOURCE_TYPE,
                               id=my_library.id)

# 2- Attach the tag to the ClusterComputeResource managed object.
tag_association_stub = tagging_client.TagAssociationStub(my_stub_config)
tag_association_stub.attach(tag_id,
                            library_dynamic_id)
```

Assign a Tag to a Cluster

After you create a tag, you can assign the tag to a vSphere managed object. Tags make the inventory objects in your virtual environment more sortable and searchable.

This procedure describes the steps for applying tag a to a cluster object in your inventory.

Prerequisites

Obtain the managed object identifier for the specified cluster.

To get the managed object identifier of the `ClusterComputeResource`, you must access vCenter Server by using the vSphere Web Services API. For more information about how to access Web Services, see Create a Web Services Session.

Procedure

1. Construct a dynamic object identifier for the cluster.

   The dynamic identifier includes the type and ID of the managed object reference.
2. Attach the tag to the cluster.

Java Example of Assigning a Tag to a Cluster

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

This example is based on the information that is provided in Assign a Tag to a Cluster.

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 the MOID of the ClusterComputeResource from its name.
ManagedObjectReference clusterMoRef = VimUtil.getCluster(vimPort,
                                                          serviceContent,
                                                          myClusterName);

// 2 - Create a dynamic type ID for the cluster.
DynamicID clusterDynamicId = DynamicID(clusterMoRef.getType(),
                                       clusterMoRef.getValue());

// 3 - Attach the tag to the ClusterComputeResource managed object.
TagAssociation tagAssociationStub = myStubFactory.createStub(TagAssociation.class,
                                                             myStubConfig);
tagAssociationStub.attach(tagId,
                          clusterDynamicId);
```

Python Example of Assigning a Tag to a Cluster

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

This example is based on the information that is provided in Assign a Tag to a Cluster.

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

```
...
```

```
# 1 - Determine the MOID of the ClusterComputeResource from its name.
cluster_object = get_obj(service_content,
                         [vim.ClusterComputeResource],
                         my_cluster_name)
cluster_moid = cluster_obj._GetMoId()

# 2 - Create a dynamic type ID for the cluster.
dynamic_id = DynamicID(type='ClusterComputeResource', id=cluster_moid)

# 3 - Attach the tag to the ClusterComputeResource managed object.
tag_association_stub = tagging_client.TagAssociation(my_stub_config)
tag_association_stub.attach(tag_id=tag_id,
                            object_id=dynamic_id)
```

Updating a Tag

To update a tag, you must create an update spec for the tag. In the update spec, you set values for the fields to be changed, and omit values for the other fields. When you do an update operation using the update spec, only the fields that contain values are changed.

For example, you might use a timestamp in a tag description to identify a resource's last reconfiguration. After reconfiguring the resource, you update the tag description to contain the current time.

Java Example of Updating a Tag Description

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

This example adds timestamp in a tag description to identify when a resource was last reconfigured. The tag description is updated with the timestamp after the resources is reconfigured.

This example is based on the information that is provided in Updating a Tag .

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

```
...

String newDateTime = Dateformat.getDateInstance().format(new Date());
String newDescription = String.format("Server tag updated at (%s).", newDateTime);

TagTypes.UpdateSpec updateSpec = new TagTypes.UpdateSpec();
updateSpec.setDescription(newDescription);
tagStub.update(myTagId, updateSpec);
```

Python Example of Updating a Tag Description

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

This example adds timestamp in a tag description to identify when a resource was last reconfigured. The tag description is updated with the timestamp after the resources is reconfigured.

This example is based on the information that is provided in Updating a Tag .

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

```
...

tag_stub = tagging_client.Tag(my_stub_config

# 1 - Format the current time.
date_time = time.strftime('%d/%m/%Y %H:%M:%S')
description = 'Server tag updated at ' + date_time

# 2 - Set up a tag update spec.
tag_update_spec = tag_stub.UpdateSpec()
tag_update_spec.description = description

# 3 - Apply the update spec to change the tag description.
tag_stub.update(tag_id, tag_update_spec)
```

Managing Certificates

11

You can use the vSphere Automation API to manage the life cycle of certificates.

The API provides operations for managing TLS (`MACHINE_SSL_CERT`) certificates, trusted root certificate chains, and VMware Certificate Authority (VMCA) root certificates. The `tls_csr` interface provides an operation for generating a certificate signing request (CSR). The `tls` interface provides operations for retrieving, renewing, or replacing the TLS certificate. The `trusted_root_chains` interface provides operations for creating, retrieving, or deleting trusted root certificate chains. The `vmca_root` interface provides an operation for replacing the VMCA root certificate. The TLS certificate and trusted root chain certificates are maintained in the VMware Endpoint Certificate Store (VECS) and provide the means for services inside vCenter Server to communicate in a secure manner.

Table 11-1. User Operations

Operation	Description
Generate a CSR	You can generate a CSR by providing a valid specification. If the operation is successful, you receive a CSR in PEM format. You can use the CSR only to replace the TLS certificate, because the private key is stored in the VECS.
Get TLS certificate	You can retrieve the TLS certificate which contains information such as serial number, issuer, validity, thumbprint, and so on.
Renew TLS certificate	You can renew the validity of the TLS certificate for a specified period. The duration should be less than or equal to 730 days. If you do not specify the duration, the default value of 730 days is applied.
Replace TLS certificate with a custom signed certificate	You can replace the TLS certificate with a third-party or custom Certificate Authority (CA) signed certificate.
Replace TLS certificate with a VMCA-signed certificate	You can replace the TLS certificate with a VMCA-signed certificate.
Create a trusted root certificate chain	You can publish a trusted root certificate chain to vCenter Server by providing a valid specification. If the operation is successful, you receive a unique identifier of the last certificate present in the root chain.
List trusted root certificates	You can retrieve the identifiers of all trusted root certificates that are published to vCenter Server.
Get trusted root certificate information	You can retrieve the PEM certificate by providing the identifier of the certificate. The certificate identifier can be retrieved by using the List trusted root certificates operation.

Table 11-1. User Operations (continued)

Operation	Description
Delete a trusted root certificate	You can delete a specific certificate by providing the identifier. The certificate identifier can be retrieved by using the List trusted root certificates operation.
Replace the VMCA root certificate	You can reset the VMCA root certificate by generating a new one. When you reset the VMCA root certificate, the TLS and solution user certificates are automatically regenerated by using the new VMCA certificate.

This chapter includes the following topics:

HTTP Requests for Certificate Management

You can use HTTP requests to generate a CSR, retrieve, renew, or replace certificates, retrieve, create, or delete trusted root certificate chains, and replace the VMCA root certificate.

HTTP Requests

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

Note Before you send requests, you must authenticate with administrator credentials.

- Generate a CSR

```
POST https://<server>/api/vcenter/certificate-management/vcenter/tls-csr
```

- Get TLS certificate

```
GET https://<server>/api/vcenter/certificate-management/vcenter/tls
```

- Renew TLS certificate

```
POST https://<server>/api/vcenter/certificate-management/vcenter/tls?action=renew
```

- Replace TLS certificate with a custom signed certificate

```
PUT https://<server>/api/vcenter/certificate-management/vcenter/tls
```

- Replace TLS certificate with a VMCA-signed certificate

```
POST https://<server>/api/vcenter/certificate-management/vcenter/tls?action=replace-vmca-signed
```

- Create a trusted root certificate chain

```
POST https://<server>/api/vcenter/certificate-management/vcenter/trusted-root-chains
```

- List trusted root certificates

```
GET https://<server>/api/vcenter/certificate-management/vcenter/trusted-root-chains
```

- Get trusted root certificate information

```
GET https://<server>/api/vcenter/certificate-management/vcenter/trusted-root-chains/{chain}
```

- Delete a trusted root certificate

```
DELETE https://<server>/api/vcenter/certificate-management/vcenter/trusted-root-chains/
{chain}
```

- Replace the VMCA root certificate

```
POST https://<server>/api/vcenter/certificate-management/vcenter/vmca-root
```

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

HTTP Status Codes

Table 11-2. HTTP Status Codes lists the status codes that you can receive when you send HTTP requests.

Table 11-2. HTTP Status Codes

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	The operation is unsuccessful.	■ Generate a CSR ■ Get TLS certificate ■ Renew TLS certificate ■ Replace TLS certificate with a custom signed certificate ■ Replace TLS certificate with a VMCA-signed certificate ■ Create a trusted root certificate chain ■ Replace the VMCA root certificate

Table 11-2. HTTP Status Codes (continued)

HTTP Status Code	Description	Operations that Return the Status Code
403	There is an authorization issue.	■ Create a trusted root certificate chain ■ List trusted root certificates ■ Get trusted root certificate information ■ Delete a trusted root certificate
404	The object you are trying to perform an operation on is missing.	■ Get TLS certificate ■ Replace TLS certificate with a custom signed certificate ■ Get trusted root certificate information ■ Delete a trusted root certificate

cURL Examples of Certificate Management Operations

The following cURL command examples show the syntax for operations that you can use to manage TLS and trusted root certificates.

Prerequisites

- Verify that the certificate management service is running on your vCenter Server instance.
- Verify that you have the session ID that is required to invoke the API operations. You can obtain the session ID by running the following command.

```
curl -u 'administrator@vsphere.local:<password>' -X POST -k https://<server>:443/api/com/vmware/cis/session
```

Example: Renew the TLS Certificate

This example renews the existing TLS certificate issued by the VMware Certificate Authority (VMCA).

Note The duration of the renewed certificate is explicitly set to 730 days in the input spec, which is the default and maximum value. If you do not specify the duration in the input spec, the default value of 730 days is applied.

```
curl --insecure -H 'Content-Type:application/json' --request POST --data-ascii
'{"duration":"730"}' --url https://<server>/api/vcenter/certificate-management/vcenter/tls/?
action=renew --header 'vmware-api-session-id:8ab92796a606801c233a2189a1e8f823'
```

Example: Generate a CSR

This example generates a CSR and private key on the vCenter Server instance. The private key remains on the machine.

You can perform this operation as part of a use case scenario in which you want to replace the VMCA-issued TLS certificate with a TLS certificate issued by a custom Certificate Authority (CA). You must use the CSR and obtain a certificate from the external CA to replace the existing certificate. For details on the replacement operation, see Replace the TLS Certificate with a Generated Certificate.

```
curl --insecure -H 'Content-Type:application/json' --request POST --data-ascii '{"spec":
{"key_size": "2048","common_name":"sc-rdops-vm05-
dhcp-154-50.eng.vmware.com","country":"US","locality":"PA","state_or_province":"CA","organizat
ion":"VMware","organization_unit":"SSO",email_address":"abc@xyz.com","subject_alt_name":
["local.vmware.com,abc.eng.com,192.168.1.1"]} }' --url https://<server>/api/vcenter/
certificate-management/vcenter/tls-csr --header 'vmware-api-session-
id:4916bc4a8d37d3742277d0e26ac28faa'
```

Example: Replace the TLS Certificate with a Generated Certificate

This example replaces the existing TLS certificate with another certificate obtained from a CSR that you generated. You must provide the obtained certificate in PEM format in the input spec.

Note You must generate a CSR before you can replace a certificate.

You can perform this operation as part of a use case scenario in which you want to replace the VMCA-issued TLS certificate with a TLS certificate issued by a custom Certificate Authority (CA). You must use the CSR and obtain a certificate from the external CA to replace the existing certificate. For details on the CSR generation operation, see Generate a CSR.

```
curl -i --insecure -H 'Content-Type:application/json' --request PUT --data-ascii '{"spec":
{"cert":"-----BEGIN CERTIFICATE-----
\nMIIEJTCCAw2gAwIBAgIJAM5BdOvJGi+MMA0GCSqGSIb3DQEBCwUAMIGnMQswCQYD\nVQQDDAJDQTEXMBUGCgmSJomT8i
xkARkWB3ZzcGhlcmUxFTATBgoJkiaJk/
IsZAEZ\nFgVsb2NhbDELMAkGA1UEBhMCVVMxEzARBgNVBAgMCkNhbGlmb3JuaWExKTAnBgNV\nBAoMIHNjMS0xMC03OC0x
MDYtMTY4LmVuZy52bXdhcmUuY29tMRswGQYDVQQLDBJW\nTXdhcmUgRW5naW5lZXJpbmcwHhcNMTkwOTEyMDkyNDIwWhcN
MjAwNzA4MDkyNDIw\nWjBsMQswCQYDVQQGEwJJTjEMMAoGA1UECAwDQmdsMQwwCgYDVQQHDANOR0wxDDAK\nBgNVBAoMA3
ZtdzEzMDEGA1UEAwwqc2MyLXJkb3BzLXZtMDctZGhjcC0yNDUtMjA0\nLmVuZy52bXdhcmUuY29tMIIBIjANBgkqhkiG9w
0BAQEFAAOCAQ8AMIIBCgKCAQEA\n3o1uY1FRL0fJX9k4GnPhp5hIFvbHcYTU+WgPDDtboskcJwUSybOxLu6s2gRHjDH4\n
x0VQQ2U9DtlIds62jJErOqhstSmip8SQmhrVa1eN9ORwFeFEjHrFuAAdhKQirWj7\nu93kFv3vyoEp6vf0ZrTVvK9P4MZ3
xO8ZWed6EiU6ju+eNJvEd1lJ+3l0InvORFp0\nH/
V7LvfwA1G0rwbCzKQ+VWWZsO4cLMAoXqReXN9E2q2CtpGPXUCA7SLBXasrQxda\nELPXSDn+Dnnql319GLGkiJDa8k1K6R
qZ6knu1dwGvBNw5P6LWhsLqRz44RSr27Zw\npvarlVuVnab/
5b6DfgHgiQIDAQABo4GNMIGKMBUGA1UdEQQOMAyCBHNjMi2HBAoB\nAQEwHwYDVR0jBBgwFoAUphwxwKuWlxqZgFdHYJLb
yRrprB8wUAYIKwYBBQUHAQEE\nRDBCMEAGCCsGAQUFBzAChjRodHRwczovL3NjMS0xMC03OC0xMDYtMTY4LmVuZy52\nbX
dhcmUuY29tL2FmZC92ZWNzL2NhMA0GCSqGSIb3DQEBCwUAA4IBAQAyydRgWRBf\n8hVkC89yE912kRqh9sQyN2VtnjEQ0e
l+HB9FAYlhlYgW4mFK+f50NliyiKsGiPT6\nvL/
5Txub3CyLmMuzBgr2r8DnSiOntN9OJdF+FuFmGN6KvK9RvNpJwhtFjjVnDc45\nGYUyAhNpXvLec+DyAJDdqBtTDy9Vqyp
PBHGhPoMNDjnHI+Zj7svS+duunGD+A9y6\n9+HJKyK+TnhlCDcms/kmwvUWjBt56p6OmPXGpXz8aUNe/
byL59gqbgPBQoV1ASnu\nvJm5sXiehzwdYglnCIdbCebL7tdJRh8Qsv1mQ7gfuOrjFtfVfSAbIjUPRH5o4LHa\nOvCeaa6
p+dsw\n-----END CERTIFICATE-----"}}' --url https://<server>/api/vcenter/certificate-
management/vcenter/tls/ --header 'vmware-api-session-id:4916bc4a8d37d3742277d0e26ac28faa'
```

Example: Replace the TLS Certificate by Providing a Generated Certificate and the Root Certificate

This example replaces the existing TLS certificate with another certificate obtained from a CSR that you generated. The generated CSR is signed by a third-party CA. The private key is already present on the system, so the custom certificate and the root certificate are provided as input.

```
curl -i --insecure -H 'Content-Type:application/json' --request PUT --data-ascii '{"spec":
{"cert":"-----BEGIN CERTIFICATE-----
\nMIIEVzCCAz+gAwIBAgIJAO3rzi2tfQt2MA0GCSqGSIb3DQEBCwUAMIGwMQswCQYD\nVQQDDAJDQTEXMBUGCgmSJomT8i
xkARkWB3ZzcGhlcmUxFTATBgoJkiaJk/
IsZAEZ\nFgVsb2NhbDELMAkGA1UEBhMCVVMxEzARBgNVBAgMCkNhbGlmb3JuaWExMjAwBgNV\nBAoMKXNjLXJkb3BzLXZt
MTItZGhjcC0xMTktMTIwLmVuZy52bXdhcmUuY29tMRsw\nGQYDVQQLDBJWTXdhcmUgRW5naW5lZXJpbmcwHhcNMjAwMjA2
MTQzOTE4WhcNMjIw\nMjA2MDIzOTE4WjBBMTIwMAYDVQQDDClzYy1yZG9wcy12bTEyLWRoY3AtMTE5LTEy\nMC5lbmcudm
13YXJlLmNvbTELMAkGA1UEBhMCVVMwggEiMA0GCSqGSIb3DQEBAQUA\nA4IBDwAwggEKAoIBAQCX/IAUNuRK8E0VPISX/
899ded41orLLb4qMcn8nM8UlulB\nIQ6c5dEGH90jz//
sOvABskiQAOu5zoS0N42LMP5nfOflqksaMuxkjBoA+nn9pfQe\n3nticFDBXoufz9ADZPhIwnR4mWFngGURm2GZPxOZDgY
vFlgmbDA8cMAVl/7Clvye\n4Moz9gD+vCwd/
5ZOAx7qs7Xhd95h7Sh79MSquTQufwHgxOlhnWlSFwqzzeztjAhd\nLJQnV0vtd3wPiOL32rWXWuyVlKYN/
xKliuy3W73xKxyXpW6GDomY9ZIBbq1LbHMe\nrxsRWJ4XCtLDvz/
j2dG7pkAMqJtXwyHDar32scmRAgMBAAGjgeEwgd4wCwYDVR0P\nBAQDAgOoMDQGA1UdEQQtMCuCKXNjLXJkb3BzLXZtMTI
tZGhjcC0xMTktMTIwLmVu\nZy52bXdhcmUuY29tMB0GA1UdDgQWBBRVjbTf4YMZ3jO6KiDl6aVIXlMZXjAfBgNV\nHSMEG
DAWgBRyC5hEctkyVLsj4gcSaTVllBxukjBZBggrBgEFBQcBAQRNMEswSQYI\nKwYBBQUHMAKGPWh0dHBzOi8vc2MtcmRvc
HMtdm0xMi1kaGNwLTExOS0xMjAuZW5n\nLnZtd2FyZS5jb20vYWZkL3ZlY3MvY2EwDQYJKoZIhvcNAQELBQADggEBAEnny
7ss\n7V6jOYHd/
jwzTSM9wKqUyIJE2gZqPs7gXyajt4czRQLczh9bgb9CP5GsO+cQ48GE\nRfEPJXFwTapY+8LHiKQTkMVRmfLUWwtPt3gQH
WuaDrFJ9H08VB1+s4ULDmFJvWgD\nJd96dp4r1H0psE0mi9yKdHqQ1cRKsNuBpTxJhLmCbl+bxbO6VTdoGX72s4CzBDuv\
nxP3eg3OFy46DobgMA7IidFjFv7AMtKZb8sDn8xIQnHQ7+43o31hJICwAQ/VzZ50d\n6Mod6tKwUf/
szYFxEmxdQ1QzQy5HFOu/trPDZmmKJ+IxQYpe6OimQAoYv7+bkMnv\nC+7wgKSF07J7RJ4=\n-----END
CERTIFICATE-----","key":null,"root_cert":"-----BEGIN CERTIFICATE-----
\nMIIESzCCAzOgAwIBAgIJAPinMDDbh9UYMA0GCSqGSIb3DQEBCwUAMIGwMQswCQYD\nVQQDDAJDQTEXMBUGCgmSJomT8i
xkARkWB3ZzcGhlcmUxFTATBgoJkiaJk/
IsZAEZ\nFgVsb2NhbDELMAkGA1UEBhMCVVMxEzARBgNVBAgMCkNhbGlmb3JuaWExMjAwBgNV\nBAoMKXNjLXJkb3BzLXZt
MTItZGhjcC0xMTktMTIwLmVuZy52bXdhcmUuY29tMRsw\nGQYDVQQLDBJWTXdhcmUgRW5naW5lZXJpbmcwHhcNMjAwMjAz
MTQzOTE4WhcNMzAw\nMTMxMTQzOTE4WjCBsDELMAkGA1UEAwwCQ0ExFzAVBgoJkiaJk/
IsZAEZFgd2c3Bo\nZXJlMRUwEwYKCZImiZPyLGQBGRYFbG9jYWwxCzAJBgNVBAYTAlVTMRMwEQYDVQQI\nDApDYWxpZm9y
bmlhMTIwMAYDVQQKDClzYy1yZG9wcy12bTEyLWRoY3AtMTE5LTEy\nMC5lbmcudm13YXJlLmNvbTEbMBkGA1UECwwSVk13
YXJlIEVuZ2luZWVyaW5nMIIB\nIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAvPCxji6AMd7lv5IzXbmpzik1\nyj
7GeGvtXqkfg1WaO200/tUcvgKrYUJln8/
EiypkUIJCesaIztPAimAhK6blZDPu\nh1hlKRAcVp5rNf12Wa1Bd6z+puJ4tA8I39bED4yUY+bfaVyo86pFh4DuObEtDJo
K\nOnhek59P2ZOM6bQs0AV0jOhL3bvjsjEMQ4ocMzhDDybUJ2FRmp68YwFi2bCKKuLV\nCgP3t+X3DXTdjiZSUICO7mZs/
VRgOSw3tt3DvBFiclSH/oIhN+7mUOs0gKoD6a11\nY3i9sHUyULaTe+3/
lg4DRF+vv2UZ90l6QOcZsIMt3hoqkCG3ZPhkv0NnxOkcyQID\nAQABo2YwZDAdBgNVHQ4EFgQUcguYRHLZMlS7I+IHEmk1
ZZQcbpIwHwYDVR0RBBgw\nFoEOZW1haWxAYWNtZS5jb22HBH8AAAEwDgYDVR0PAQH/BAQDAgEGMBIGA1UdEwEB\n/
wQIMAYBAf8CAQAwDQYJKoZIhvcNAQELBQADggEBABYgtMn7+S/4LL754it6nDlo\nToey6r1qXCw/
KRrfBDk52aoF+VMtSadNp6EAk2AxmX9GtU+ucsqApcjxiPRIooOK\nbly8+/zLVFpbO83M2jGYT9Te/
R8sncbPzNCr0KmtSpsPn0BBGGgZitGRBwvYpeeJ\njOalWy2zZHE8eqsP8qSh/
Qug5hwBue68YqoyzwYQbiffeIakUl0bG1rXia9+ZIdI\nk+MCU7686vIXd9GUev2gyRptlbjkrn/
Wx8KOKkSQx7VM+ahinyx09U1YsU3OR+H8\nrnLzdGnPbwt7igLaAP1uaXLsOxKfCLTpa0N7Z6bmp5E4LhI/
uNO8V5xOPRD1jvY=\n-----END CERTIFICATE-----"} }' --url https://<server>/api/vcenter/
certificate-management/vcenter/tls/ --header 'vmware-api-session-
id:6c12e3392f1aec9c99556d23c9fe4cc0'
```

Example: Replace the TLS Certificate by Providing a Generated Certificate and Private Key

This example replaces the existing TLS certificate with another certificate obtained from a CSR that you generated externally. The third-party root certificate is already present in the trusted root store, so the custom certificate and the private key are provided as input.

```
curl -i --insecure -H 'Content-Type:application/json' --request PUT --data-ascii '{"spec":
{"cert":"-----BEGIN CERTIFICATE-----
\nMIIEVzCCAz+gAwIBAgIJAO3rzi2tfQt2MA0GCSqGSIb3DQEBCwUAMIGwMQswCQYD\nVQQDDAJDQTEXMBUGCgmSJomT8i
xkARkWB3ZzcGhlcmUxFTATBgoJkiaJk/
IsZAEZ\nFgVsb2NhbDELMAkGA1UEBhMCVVMxEzARBgNVBAgMCkNhbGlmb3JuaWExMjAwBgNV\nBAoMKXNjLXJkb3BzLXZt
MTItZGhjcC0xMTktMTIwLmVuZy52bXdhcmUuY29tMRsw\nGQYDVQQLDBJWTXdhcmUgRW5naW5lZXJpbmcwHhcNMjAwMjA2
MTQzOTE4WhcNMjIw\nMjA2MDIzOTE4WjBBMTIwMAYDVQQDDClzYy1yZG9wcy12bTEyLWRoY3AtMTE5LTEy\nMC5lbmcudm
13YXJlLmNvbTELMAkGA1UEBhMCVVMwggEiMA0GCSqGSIb3DQEBAQUA\nA4IBDwAwggEKAoIBAQCX/IAUNuRK8E0VPISX/
899ded41orLLb4qMcn8nM8UlulB\nIQ6c5dEGH90jz//
sOvABskiQAOu5zoS0N42LMP5nfOflqksaMuxkjBoA+nn9pfQe\n3nticFDBXoufz9ADZPhIwnR4mWFngGURm2GZPxOZDgY
vFlgmbDA8cMAVl/7Clvye\n4Moz9gD+vCwd/
5ZOAx7qs7Xhd95h7Sh79MSquTQufwHgxOlhnWlSFwqzzeztjAhd\nLJQnV0vtd3wPiOL32rWXWuyVlKYN/
xKliuy3W73xKxyXpW6GDomY9ZIBbq1LbHMe\nrxsRWJ4XCtLDvz/
j2dG7pkAMqJtXwyHDar32scmRAgMBAAGjgeEwgd4wCwYDVR0P\nBAQDAgOoMDQGA1UdEQQtMCuCKXNjLXJkb3BzLXZtMTI
tZGhjcC0xMTktMTIwLmVu\nZy52bXdhcmUuY29tMB0GA1UdDgQWBBRVjbTf4YMZ3jO6KiDl6aVIXlMZXjAfBgNV\nHSMEG
DAWgBRyC5hEctkyVLsj4gcSaTVllBxukjBZBggrBgEFBQcBAQRNMEswSQYI\nKwYBBQUHMAKGPWh0dHBzOi8vc2MtcmRvc
HMtdm0xMi1kaGNwLTExOS0xMjAuZW5n\nLnZtd2FyZS5jb20vYWZkL3ZlY3MvY2EwDQYJKoZIhvcNAQELBQADggEBAEnny
7ss\n7V6jOYHd/
jwzTSM9wKqUyIJE2gZqPs7gXyajt4czRQLczh9bgb9CP5GsO+cQ48GE\nRfEPJXFwTapY+8LHiKQTkMVRmfLUWwtPt3gQH
WuaDrFJ9H08VB1+s4ULDmFJvWgD\nJd96dp4r1H0psE0mi9yKdHqQ1cRKsNuBpTxJhLmCbl+bxbO6VTdoGX72s4CzBDuv\
nxP3eg3OFy46DobgMA7IidFjFv7AMtKZb8sDn8xIQnHQ7+43o31hJICwAQ/VzZ50d\n6Mod6tKwUf/
szYFxEmxdQ1QzQy5HFOu/trPDZmmKJ+IxQYpe6OimQAoYv7+bkMnv\nC+7wgKSF07J7RJ4=\n-----END
CERTIFICATE-----","key":"-----BEGIN PRIVATE KEY-----
\nMIIEvQIBADANBgkqhkiG9w0BAQEFAASCBKcwggSjAgEAAoIBAQCX/IAUNuRK8E0V\nPISX/
899ded41orLLb4qMcn8nM8UlulBIQ6c5dEGH90jz//
sOvABskiQAOu5zoS0\nN42LMP5nfOflqksaMuxkjBoA+nn9pfQe3nticFDBXoufz9ADZPhIwnR4mWFngGUR\nm2GZPxOZD
gYvFlgmbDA8cMAVl/7Clvye4Moz9gD+vCwd/
5ZOAx7qs7Xhd95h7Sh7\n9MSquTQufwHgxOlhnWlSFwqzzeztjAhdLJQnV0vtd3wPiOL32rWXWuyVlKYN/
xKl\niuy3W73xKxyXpW6GDomY9ZIBbq1LbHMerxsRWJ4XCtLDvz/
j2dG7pkAMqJtXwyHD\nar32scmRAgMBAAECggEAIUnKe6LyKX+sQmR43hnHRkFWq17CE3I3XmJ9TPjDWk0b\nVYp5+t9TA
rZj5v4dnY3jF1wHDJNHDfEvm5E3TS8z6VKwL9s2i8xvfi45W9GAeo+o\ngkJUX+EdrtyYVC5d7wBkaPs+K1Pcw6CZglu36
qqoDjPGIvYP5Ip5niNLzu71e4Tj\n8fdWIgGLja05Z1CeWQnUVVqMHzrcyreCdGSxmiqzwY5ReuoruDrDtQrRnacZQ31t\
nVKPEEm9J3tTgaNjqIJ1SptzN0cAdr4HngbhFr+ypRufaNNv+UcVm073McC1ZXns1\nhrUhjpF7e7EtF8zI3SyMdrMXG2i
trKks5jtNdybKqQKBgQDHrjVb21xyBX3ircxC\nujHY+x0mUeKcbkLdJn2luDgXOxyepJljrGrDBvbUop5bekvCTSuHpST
Xnasa9Aj9\n42QI+GvJ2j34Wr2H86IaBJeIBtMcgbbPQa1VM7wSHaPyK1azukfqGRbDWmjmsRu9\nQTQWnMT4q5uxO7E/
URK7ZVMyWwKBgQDC2pE/c9JH5e6nPUy6qxtDT5g/
IU+hYA83\n0wmCMz0zIe6Dvl6yW99vC3toCBzurgWncGYx4vrXFBMEunR2lY+xE4MCXqJ8e6eS\n+hvFCIueCtqsbZGKFW
hvVjJTqPoJbEPC2odcRyO/m2mWBUPMFF9KRpaN/
u1vRpKf\npfQ2UPIfgwKBgCodzXVddEc8b0vjeoTMfv0UIhbWCKUUAJhiqPfOPk5wEl5Hu26x\njCcsjd1Vm28SAW6vFjJ
bG8U6pT9fH5JlZtEZjKqblJyQjw9gzN0OAOQZBGgr1+Ip\nKSUib1Wm2NCYYYYxEUAIIKv9osidrIsjeJgYYtdMYt9kBsO
jFv9dpMmnAoGALo6g\nw+N9q/
Yxhr4r1jJKClQ7UWI5L5rPKAyBqh1qEwyZe9sBr2YqRdMdgmbl2tWzviUq\nFeNhAgDm5mtSpn7n3WyHEgrgkhPNn12pQv
PewD1hsG9hpfelg2y6C6lFTPkUW7tx\nOkg5L3AH03OglmOLvSiOkpEF1F+tav3xEwVWRNcCgYEAhlb1eXHMpasnSNxp5G
Gn\nBtwrMkucP62AjNvp7gciDDOQKfrwQol2p3XHDr7WMrUSVoXZ9QhP/
4usujIdB4xw\naZlv2bHHbkXnbfQrR1QEExhk+MfCKjuZ1q90mwE66iOXy69yLUaZUYDxG0SX97Gt\nWlQZwaQLpbmJXnX
```

```
WNCRQMg8=\n-----END PRIVATE KEY-----""} }' --url https://<server>/api/vcenter/certificate-
management/vcenter/tls/ --header 'vmware-api-session-id:4a0942a047adf40e1f64d46d48283f88'
```

Example: Replace the TLS Certificate by Providing a Generated Certificate, Private Key, and CA Certificate

This example replaces the existing TLS certificate with another certificate obtained from a CSR that you generated externally. The third-party root certificate is not present in the trusted root store, so the custom certificate, its private key, and the CA certificate are provided as input.

```
curl -i --insecure -H 'Content-Type:application/json' --request PUT --data-ascii '{"spec":
{"cert":"-----BEGIN CERTIFICATE-----
\nMIIEVzCCAz+gAwIBAgIJAO3rzi2tfQt2MA0GCSqGSIb3DQEBCwUAMIGwMQswCQYD\nVQQDDAJDQTEXMBUGCgmSJomT8i
xkARkWB3ZzcGhlcmUxFTATBgoJkiaJk/
IsZAEZ\nFgVsb2NhbDELMAkGA1UEBhMCVVMxEzARBgNVBAgMCkNhbGlmb3JuaWExMjAwBgNV\nBAoMKXNjLXJkb3BzLXZt
MTItZGhjcC0xMTktMTIwLmVuZy52bXdhcmUuY29tMRsw\nGQYDVQQLDBJWTXdhcmUgRW5naW5lZXJpbmcwHhcNMjAwMjA2
MTQzOTE4WhcNMjIw\nMjA2MDIzOTE4WjBBMTIwMAYDVQQDDClzYy1yZG9wcy12bTEyLWRoY3AtMTE5LTEy\nMC5lbmcudm
13YXJlLmNvbTELMAkGA1UEBhMCVVMwggEiMA0GCSqGSIb3DQEBAQUA\nA4IBDwAwggEKAoIBAQCX/IAUNuRK8E0VPISX/
899ded41orLLb4qMcn8nM8UlulB\nIQ6c5dEGH90jz//
sOvABskiQAOu5zoS0N42LMP5nfOflqksaMuxkjBoA+nn9pfQe\n3nticFDBXoufz9ADZPhIwnR4mWFngGURm2GZPxOZDgY
vFlgmbDA8cMAVl/7Clvye\n4Moz9gD+vCwd/
5ZOAx7qs7Xhd95h7Sh79MSquTQufwHgxOlhnWlSFwqzzeztjAhd\nLJQnV0vtd3wPiOL32rWXWuyVlKYN/
xKliuy3W73xKxyXpW6GDomY9ZIBbq1LbHMe\nrxsRWJ4XCtLDvz/
j2dG7pkAMqJtXwyHDar32scmRAgMBAAGjgeEwgd4wCwYDVR0P\nBAQDAgOoMDQGA1UdEQQtMCuCKXNjLXJkb3BzLXZtMTI
tZGhjcC0xMTktMTIwLmVu\nZy52bXdhcmUuY29tMB0GA1UdDgQWBBRVjbTf4YMZ3jO6KiDl6aVIXlMZXjAfBgNV\nHSMEG
DAWgBRyC5hEctkyVLsj4gcSaTVllBxukjBZBggrBgEFBQcBAQRNMEswSQYI\nKwYBBQUHMAKGPWh0dHBzOi8vc2MtcmRvc
HMtdm0xMi1kaGNwLTExOS0xMjAuZW5n\nLnZtd2FyZS5jb20vYWZkL3ZlY3MvY2EwDQYJKoZIhvcNAQELBQADggEBAEnny
7ss\n7V6jOYHd/
jwzTSM9wKqUyIJE2gZqPs7gXyajt4czRQLczh9bgb9CP5GsO+cQ48GE\nRfEPJXFwTapY+8LHiKQTkMVRmfLUWwtPt3gQH
WuaDrFJ9H08VB1+s4ULDmFJvWgD\nJd96dp4r1H0psE0mi9yKdHqQ1cRKsNuBpTxJhLmCbl+bxbO6VTdoGX72s4CzBDuv\
nxP3eg3OFy46DobgMA7IidFjFv7AMtKZb8sDn8xIQnHQ7+43o31hJICwAQ/VzZ50d\n6Mod6tKwUf/
szYFxEmxdQ1QzQy5HFOu/trPDZmmKJ+IxQYpe6OimQAoYv7+bkMnv\nC+7wgKSF07J7RJ4=\n-----END
CERTIFICATE-----","key":"-----BEGIN PRIVATE KEY-----
\nMIIEvQIBADANBgkqhkiG9w0BAQEFAASCBKcwggSjAgEAAoIBAQCX/IAUNuRK8E0V\nPISX/
899ded41orLLb4qMcn8nM8UlulBIQ6c5dEGH90jz//
sOvABskiQAOu5zoS0\nN42LMP5nfOflqksaMuxkjBoA+nn9pfQe3nticFDBXoufz9ADZPhIwnR4mWFngGUR\nm2GZPxOZD
gYvFlgmbDA8cMAVl/7Clvye4Moz9gD+vCwd/
5ZOAx7qs7Xhd95h7Sh7\n9MSquTQufwHgxOlhnWlSFwqzzeztjAhdLJQnV0vtd3wPiOL32rWXWuyVlKYN/
xKl\niuy3W73xKxyXpW6GDomY9ZIBbq1LbHMerxsRWJ4XCtLDvz/
j2dG7pkAMqJtXwyHD\nar32scmRAgMBAAECggEAIUnKe6LyKX+sQmR43hnHRkFWq17CE3I3XmJ9TPjDWk0b\nVYp5+t9TA
rZj5v4dnY3jF1wHDJNHDfEvm5E3TS8z6VKwL9s2i8xvfi45W9GAeo+o\ngkJUX+EdrtyYVC5d7wBkaPs+K1Pcw6CZglu36
qqoDjPGIvYP5Ip5niNLzu71e4Tj\n8fdWIgGLja05Z1CeWQnUVVqMHzrcyreCdGSxmiqzwY5ReuoruDrDtQrRnacZQ31t\
nVKPEEm9J3tTgaNjqIJ1SptzN0cAdr4HngbhFr+ypRufaNNv+UcVm073McC1ZXns1\nhrUhjpF7e7EtF8zI3SyMdrMXG2i
trKks5jtNdybKqQKBgQDHrjVb21xyBX3ircxC\nujHY+x0mUeKcbkLdJn2luDgXOxyepJljrGrDBvbUop5bekvCTSuHpST
Xnasa9Aj9\n42QI+GvJ2j34Wr2H86IaBJeIBtMcgbbPQa1VM7wSHaPyK1azukfqGRbDWmjmsRu9\nQTQWnMT4q5uxO7E/
URK7ZVMyWwKBgQDC2pE/c9JH5e6nPUy6qxtDT5g/
IU+hYA83\n0wmCMz0zIe6Dvl6yW99vC3toCBzurgWncGYx4vrXFBMEunR2lY+xE4MCXqJ8e6eS\n+hvFCIueCtqsbZGKFW
hvVjJTqPoJbEPC2odcRyO/m2mWBUPMFF9KRpaN/
u1vRpKf\npfQ2UPIfgwKBgCodzXVddEc8b0vjeoTMfv0UIhbWCKUUAJhiqPfOPk5wEl5Hu26x\njCcsjd1Vm28SAW6vFjJ
bG8U6pT9fH5JlZtEZjKqblJyQjw9gzN0OAOQZBGgr1+Ip\nKSUib1Wm2NCYYYYxEUAIIKv9osidrIsjeJgYYtdMYt9kBsO
jFv9dpMmnAoGALo6g\nw+N9q/
Yxhr4r1jJKClQ7UWI5L5rPKAyBqh1qEwyZe9sBr2YqRdMdgmbl2tWzviUq\nFeNhAgDm5mtSpn7n3WyHEgrgkhPNn12pQv
PewD1hsG9hpfelg2y6C6lFTPkUW7tx\nOkg5L3AH03OglmOLvSiOkpEF1F+tav3xEwVWRNcCgYEAhlb1eXHMpasnSNxp5G
Gn\nBtwrMkucP62AjNvp7gciDDOQKfrwQol2p3XHDr7WMrUSVoXZ9QhP/
```

```
4usujIdB4xw\naZlv2bHHbkXnbfQrR1QEExhk+MfCKjuZ1q90mwE66iOXy69yLUaZUYDxG0SX97Gt\nWlQZwaQLpbmJXnX
WNCRQMg8=\n-----END PRIVATE KEY-----","root_cert":"-----BEGIN CERTIFICATE-----
\nMIIESzCCAzOgAwIBAgIJAPinMDDbh9UYMA0GCSqGSIb3DQEBCwUAMIGwMQswCQYD\nVQQDDAJDQTEXMBUGCgmSJomT8i
xkARkWB3ZzcGhlcmUxFTATBgoJkiaJk/
IsZAEZ\nFgVsb2NhbDELMAkGA1UEBhMCVVMxEzARBgNVBAgMCkNhbGlmb3JuaWExMjAwBgNV\nBAoMKXNjLXJkb3BzLXZt
MTItZGhjcC0xMTktMTIwLmVuZy52bXdhcmUuY29tMRsw\nGQYDVQQLDBJWTXdhcmUgRW5naW5lZXJpbmcwHhcNMjAwMjAz
MTQzOTE4WhcNMzAw\nMTMxMTQzOTE4WjCBsDELMAkGA1UEAwwCQ0ExFzAVBgoJkiaJk/
IsZAEZFgd2c3Bo\nZXJlMRUwEwYKCZImiZPyLGQBGRYFbG9jYWwxCzAJBgNVBAYTAlVTMRMwEQYDVQQI\nDApDYWxpZm9y
bmlhMTIwMAYDVQQKDClzYy1yZG9wcy12bTEyLWRoY3AtMTE5LTEy\nMC5lbmcudm13YXJlLmNvbTEbMBkGA1UECwwSVk13
YXJlIEVuZ2luZWVyaW5nMIIB\nIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAvPCxji6AMd7lv5IzXbmpzik1\nyj
7GeGvtXqkfg1WaO200/tUcvgKrYUJln8/
EiypkUIJCesaIztPAimAhK6blZDPu\nh1hlKRAcVp5rNf12Wa1Bd6z+puJ4tA8I39bED4yUY+bfaVyo86pFh4DuObEtDJo
K\nOnhek59P2ZOM6bQs0AV0jOhL3bvjsjEMQ4ocMzhDDybUJ2FRmp68YwFi2bCKKuLV\nCgP3t+X3DXTdjiZSUICO7mZs/
VRgOSw3tt3DvBFiclSH/oIhN+7mUOs0gKoD6a11\nY3i9sHUyULaTe+3/
lg4DRF+vv2UZ90l6QOcZsIMt3hoqkCG3ZPhkv0NnxOkcyQID\nAQABo2YwZDAdBgNVHQ4EFgQUcguYRHLZMlS7I+IHEmk1
ZZQcbpIwHwYDVR0RBBgw\nFoEOZW1haWxAYWNtZS5jb22HBH8AAAEwDgYDVR0PAQH/BAQDAgEGMBIGA1UdEwEB\n/
wQIMAYBAf8CAQAwDQYJKoZIhvcNAQELBQADggEBABYgtMn7+S/4LL754it6nDlo\nToey6r1qXCw/
KRrfBDk52aoF+VMtSadNp6EAk2AxmX9GtU+ucsqApcjxiPRIooOK\nbly8+/zLVFpbO83M2jGYT9Te/
R8sncbPzNCr0KmtSpsPn0BBGGgZitGRBwvYpeeJ\njOalWy2zZHE8eqsP8qSh/
Qug5hwBue68YqoyzwYQbiffeIakUl0bG1rXia9+ZIdI\nk+MCU7686vIXd9GUev2gyRptlbjkrn/
Wx8KOKkSQx7VM+ahinyx09U1YsU3OR+H8\nrnLzdGnPbwt7igLaAP1uaXLsOxKfCLTpa0N7Z6bmp5E4LhI/
uNO8V5xOPRD1jvY=\n-----END CERTIFICATE-----"} }' --url https://<server>/api/vcenter/
certificate-management/vcenter/tls/ --header 'vmware-api-session-
id:4a0942a047adf40e1f64d46d48283f88'
```

Example: Intermediate Root Certificates in a CA Certificate Chain

This example illustrates the use case where the CA certificate chain consists of one or more intermediate root certificates and how to order these certificates in the chain. The following list provides an example certificate chain configuration.

```
CertificateA (Signed by CertificateB)
CertificateB (Signed by CertificateC)
CertificateC (Signed by RootCertificate)
RootCertificate
```

The following command replaces a certificate (*CertificateToReplace*) that is signed by the first certificate from the list (*CertificateA*).

```
curl -i --insecure -H 'Content-Type:application/json' --request PUT --data-ascii '{"spec":
{"cert":"<CertificateToReplace>","key":"<PrivateKey>","root_cert":"<CertificateA>\n<Certificat
eB>\n<CertificateC>\n<RootCertificate>" }' --url https://<server>/api/vcenter/certificate-
management/vcenter/tls/ --header 'vmware-api-session-id:4a0942a047adf40e1f64d46d48283f88'
```

Example: Replace the TLS Certificate with a VMCA-Signed Certificate

This example replaces the existing TLS certificate with a VMCA-signed certificate.

```
curl --insecure -H 'Content-Type:application/json' --request POST --data-ascii '{"spec":
{"organization": "VMware","organization_unit": "SSO","locality": "PA","state_or_province":
"CA","country": "US",email_address":"abc@xyz.com","subject_alt_name":
["local.vmware.com,abc.eng.com,192.168.1.1"]}}' --url https://<server>/api/vcenter/
```

```
certificate-management/vcenter/tls?action=replace-vmca-signed --header 'vmware-api-session-
id:819f96e088f3358b4f588cb3932df171'
```

Example: Create and Add Trusted Root Certificates

This example creates two trusted root certificate chains and publishes them to vCenter Server.

```
curl --insecure -H 'Content-Type:application/json' --request POST --data-ascii '{"spec":
{"cert_chain":{"cert_chain":["-----BEGIN CERTIFICATE-----
\nMIIDwjCCAqqgAwIBAgIJAI1OflMjc0LfMA0GCSqGSIb3DQEBCwUAMHYxCzAJBgNV\nBAYTAklOMQswCQYDVQQIDAJLQT
EMMAoGA1UEBwwDQkxSMQ8wDQYDVQQKDAZWTXdh\ncmUxDDAKBgNVBAsMA1NTTzEMMAoGA1UEAwwDQ0ExMR8wHQYJKoZIhv
cNAQkBFhBz\naWduMUB2bXdhcmUuY29tMB4XDTE5MDEwMjA2MTIyMloXDTI4MTIzMDA2MTIyMlow\ndjELMAkGA1UEBhMC
SU4xCzAJBgNVBAgMAktBMQwwCgYDVQQHDANCTFIxDzANBgNV\nBAoMBlZNd2FyZTEMMAoGA1UECwwDU1NPMQwwCgYDVQQD
DANDQTExHzAdBgkqhkiG\n9w0BCQEWEHNpZ24xQHZtd2FyZS5jb20wggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAw\nggEKAo
IBAQDHuDDoAyGj6FGLZOIxMEK7oO2LhbfGbIbBiXTR5WWSkTsmsxy0Vge5\nhbVEkGW2OjgIxvmqBC/
nVeH1b4gTJAZFmJ6lrh6Ri8HC5cyIePVJkz/
PR08SbKmy\nmagd02N6ZqBgMEr3eQ2NTtqUOutvphRT5f+fyGKL5uPjOrhNn6v8GDrIF4wUY6aV\nWYDG6Mcay/
cv814PZoTIJa0juIEfJXzOO0gxzAY6Jwi6k3DmLkps7zFErRbWUwYR\niaa46LKRHRlX71h0gsWfx7TNdCvQ8emiPXsYsq
UkOy9+MSfr3CsQcPzNy8qDbImt\ngK6z2T4vvV7r5Iir5srD7yyWm5rKmtFDAgMBAAGjUzBRMB0GA1UdDgQWBBSv6kwh\n
VWkFQ/se4wRz3PayMJTjgzAfBgNVHSMEGDAWgBSv6kwhVWkFQ/se4wRz3PayMJTj\ngzAPBgNVHRMBAf8EBTADAQH/
MA0GCSqGSIb3DQEBCwUAA4IBAQC2yEXM2fTCYRvh\noD40MrDLK/g+mKSixvsXtebTga47fHi8LxnT6KXGc44ZMT/
HTSzwk2alYG8EXHK1\nFZeNNFnhYmS24DLgrCq+9p/yThotbfWe6vaUZ87jgbAP9HRAsq/9HYW3s0lUBD4i\ne/
FZrBGRjgdtXVQ0tm5N6TVRQq2IwVPQ3niv36KLFu9MmAMhlIIZ3y8sX4Bha13q\nmhOCM74/
qw4d88kGgq9lnebpwhmmXl5IOScZX39gJpsgpWQ4a1lhOTWWLT5NYu3z\nxiS9Jc1hr0PWtKE5eWSVu6mMmEx9Tqov/
KKMRBCP/pp4aHyn0NlWFtHl7MtWrGC7\nohzPCShe\n-----END CERTIFICATE-----","-----BEGIN
CERTIFICATE-----
\nMIID5jCCAs6gAwIBAgIBCDANBgkqhkiG9w0BAQsFADB2MQswCQYDVQQGEwJJTjEL\nMAkGA1UECAwCS0ExDDAKBgNVBA
cMA0JMUjEPMA0GA1UECgwGVk13YXJlMQwwCgYD\nVQQLDANTU08xDDAKBgNVBAMMA0NBMTEfMB0GCSqGSIb3DQEJARYQc2
lnbjFAdm13\nYXJlLmNvbTAeFw0xOTAxMDIwNjE3MDZaFw0yNDA2MjQwNjE3MDZaMGgxCzAJBgNV\nBAYTAklOMQswCQYD
VQQIDAJLQTEPMA0GA1UECgwGVk13YXJlMQwwCgYDVQQLDANT\nU08xDDAKBgNVBAMMA0NBMjEfMB0GCSqGSIb3DQEJARYQ
c2lnbjAdm13YXJlLmNv\nbTCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBAL3s5ycFPQmgffQmZKaE\nM/
0ymZgh/Kz3txTmWpAiEPGpGdrulDfwubDEbOXfHtsWfcvj48iDa6Nn4g5bNrej\naMoBEIKd0WeV9fwnL/
i2wYFiKKhLYiWaHDm5BT79YVaBLEMK6BL/
9wc2FoUI2vEf\nQyVSuDuKWSrwx3gB2IFC2q7BpzT3kgq1HmWKVA52nFpMgbe1zlRy9sV08bBTybMO\nzm/
Z0c4+a5Y0P1fO6ThiCF+92s0jMow0Bm96qN3nQm6lMgbcY+5um7RgOuBY4iSF\nKTblVDMS/rZAQkPwcP/
E8AxcywRazx46awCfe3NAasiVBuI/iADc63SmYs+z+0cS\n8qECAwEAAaOBjDCBiTAMBgNVHRMEBTADAQH/
MAsGA1UdDwQEAwIBBjAsBglghkgB\nhvhCAQ0EHxYdT3BlblNTTCBHZW5lcmF0ZWQgQ2VydGlmaWNhdGUwHQYDVR0OBBYE
\nFDexYpQDPTkuYf9M47ILnGOg5Fh/MB8GA1UdIwQYMBaAFK/
qTCFVaQVD+x7jBHPc\n9rIwlOODMA0GCSqGSIb3DQEBCwUAA4IBAQAMKy6fM7ldYf/IlMSR/
0zH4gTauR8Z\nERXkRD65SXa9YgOkp/U59mhlGsfxeAze47jXjD7GNTNpLogYFQkXP9yrIpyYKjRP\n0I8zo8faY/
9hEJn2pHZTaYKgZICw0rlfCwGF/
so1cxnkocoIsmA56lMPT5xcmyFc\nkvwEBgTb8WgXUTnR0MA20puGI8aaXsAHOwQYM8nexvrfSbJADYJtcG73YqjswNYk\
niloSd/uslyhmvb1HVyix794SxAIEybs177ijKOxdicq3XogaeGhOIymvDcCv/
55J\n5FgJY341cCZmESPyC1GkuX52OSoZartB1jhSd5cKKlaLobFbTTajs9oa\n-----END
CERTIFICATE-----"]}}}' --url https://<server>/api/vcenter/certificate-management/vcenter/
trusted-root-chains/ --header 'vmware-api-session-id:e594038d4c1023afe86b2c14b0b741f0'
```

Example: List the Trusted Root Certificates

This example lists the IDs of all trusted root certificates that are published to vCenter Server.

```
curl --insecure --request GET --url https://<server>/api/vcenter/
certificate-management/vcenter/trusted-root-chains/ --header 'vmware-api-session-id:
e594038d4c1023afe86b2c14b0b741f0'
```

Example: Get Trusted Root Certificate Information

This example retrieves information about a trusted root certificate with ID AFEA4C2155690543FB1EE30473DCF6B23094E383.

```
curl --insecure --request GET --url https://<server>/api/vcenter/certificate-management/
vcenter/trusted-root-chains/AFEA4C2155690543FB1EE30473DCF6B23094E383 --header 'vmware-api-
session-id: e594038d4c1023afe86b2c14b0b741f0'
```

Example: Delete a Trusted Root Certificate

This example deletes the trusted root certificate with ID AFEA4C2155690543FB1EE30473DCF6B23094E383.

```
curl --insecure --request DELETE --url https://<server>/api/vcenter/certificate-management/
vcenter/trusted-root-chains/AFEA4C2155690543FB1EE30473DCF6B23094E383 --header 'vmware-api-
session-id: e594038d4c1023afe86b2c14b0b741f0'
```

Example: Replace the Root Certificate

This example resets the VMCA root certificate by generating a new one. When you reset the VMCA root certificate, the TLS and solution user certificates are automatically regenerated by using the new VMCA certificate.

```
curl --insecure -H ‘Content-Type:application/json’ --request POST --url https://
<server>/api/vcenter/certificate-management/vcenter/vmca-root/ --header ‘vmware-api-session-
id:1a0e5a003faa646e7fe7bf19f4baff96’
```

Configuring and Managing vSphere Native Key Provider

12

Using a VMware vSphere® Native Key Provider™ in your vSphere environment requires some preparation. After you configure vSphere Native Key Provider, you can use all features which previously would have required an external KMS, including creating virtual Trusted Platform Modules (vTPMs) on your virtual machines.

After your environment is set up for vSphere Native Key Provider, you can use the vSphere Client and API to create vTPMs. If you purchase the vSphere Enterprise+ edition, you can also encrypt virtual machines and virtual disks, and encrypt existing virtual machines and disks. For more information on the vSphere Native Key Provider functionality, see *vSphere Security*.

This chapter includes the following topics:

vSphere Native Key Provider Operations

You can use the vSphere Automation API to perform vSphere Native Key Provider operations.

You can retrieve, add, configure, or remove providers. You can also export and import provider configuration.

Table 12-1. User Operations

Operation	Description
List host providers	You can list all available providers on an ESXi host.
Get host provider	You can retrieve details about a specific provider on an ESXi host.
List providers	You can list all providers.
Get provider	You can retrieve details about a specific provider.
Create provider	You can add a new provider to your environment.
Update provider	You can update the configuration of an existing provider.
Delete provider	You can remove a provider.

Table 12-1. User Operations (continued)

Operation	Description
Export configuration	You can export the configuration of a provider.
Import configuration	You can import a provider by using a previously exported configuration.

HTTP Requests for vSphere Native Key Provider Operations

You can use HTTP requests to retrieve, add, configure, or remove providers. You can also export and import provider configuration.

HTTP Requests

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

Note Before you send requests, you must authenticate with administrator credentials or as a user with appropriate cryptographic operations privileges. For information about the required privileges, see *vSphere Security*.

- List host providers

```
GET https://<server>/api/vcenter/crypto-manager/hosts/<host>/kms/providers
```

- Get host provider

```
GET https://<server>/api/vcenter/crypto-manager/hosts/<host>/kms/providers/<provider>
```

- List providers

```
GET https://<server>/api/vcenter/crypto-manager/kms/providers
```

- Get provider

```
GET https://<server>/api/vcenter/crypto-manager/kms/providers/<provider>
```

- Create provider

```
POST https://<server>/api/vcenter/crypto-manager/kms/providers
```

- Update provider

```
PATCH https://<server>/api/vcenter/crypto-manager/kms/providers/<provider>
```

- Delete provider

```
DELETE https://<server>/api/vcenter/crypto-manager/kms/providers/<provider>
```

- Export configuration

```
POST https://<server>/api/vcenter/crypto-manager/kms/providers?action=export
```

- Import configuration

```
POST https://<server>/api/vcenter/crypto-manager/kms/providers?action=import
```

For information about the content and syntax of the HTTP query parameters, path parameters, and request body parameters, see the *API Reference* documentation.

vSphere Trust Authority

13

You can use the vSphere Automation API to perform vSphere Trust Authority operations.

vSphere Trust Authority is a foundational technology that enhances workload security. vSphere Trust Authority establishes a greater level of trust in your organization by associating an ESXi host's hardware root of trust to the workload itself. For details about vSphere Trust Authority, see the *vSphere Security* documentation.

The procedures in this chapter are based on the REST API. For details, see the *vSphere Automation REST API Reference*.

This chapter includes the following topics:

Configure a vSphere Trust Authority Cluster

You can use HTTP requests to perform vSphere Trust Authority Cluster management operations.

You can retrieve details about vSphere Trust Authority Clusters, update the state of a cluster, and check the result of the update operation. Some operations require you to specify parameters in the body of the HTTP request according to your vSphere Trust Authority environment. For details about the syntax of each HTTP request body, see the *API Reference* documentation.

Prerequisites

- Verify that you have access to a working vSphere Trust Authority environment.
- Verify that you have Trusted Infrastructure administrative privileges.

Procedure

1 Retrieve a list of clusters for a vCenter Server instance that are configured as Trust Authority Clusters.

```
GET https://<server>/api/vcenter/trusted-infrastructure/trust-authority-clusters
```

You receive the list in the response body.

2 Update the state of a cluster.

```
PATCH https://<server>/api/vcenter/trusted-infrastructure/trust-authority-clusters/
<cluster>?vmw-task=true
```

You receive the task ID in the response body. You can use the task ID to check the status of the task by running the following HTTP request.

```
GET https://<server>/api/cis/tasks/<task_ID>
```

3 Check the result of the last update operation for the same cluster.

```
GET https://<server>/api/vcenter/trusted-infrastructure/trust-authority-clusters/<cluster>
```

You receive the ID and current state of the cluster in the response body.

Configure Key Providers

You can use HTTP requests to perform Key Provider management operations.

You can retrieve, add, update, remove, and retrieve details about Key Providers. Some operations require you to specify parameters in the body of the HTTP request according to your vSphere Trust Authority environment. For details about the syntax of each HTTP request body, see the *API Reference* documentation.

Prerequisites

- Verify that you have access to a working vSphere Trust Authority environment.
- Verify that you have Trusted Infrastructure administrative privileges.

Procedure

1 Retrieve a list of Key Providers to see which Key Providers the cluster is using.

```
GET https://<server>/api/vcenter/trusted-infrastructure/trust-authority-clusters/
<cluster>/kms/providers?vmw-task=true
```

You receive the task ID in the response body. You can use the task ID to check the status of the task by running the following HTTP request.

```
GET https://<server>/api/cis/tasks/<task_ID>
```

2 Add a new Key Provider which all hosts in the cluster can use.

```
POST https://<server>/api/vcenter/trusted-infrastructure/trust-authority-clusters/
<cluster>/kms/providers?vmw-task=true
```

You receive the task ID in the response body. You can use the task ID to check the status of the task by running the following HTTP request.

```
GET https://<server>/api/cis/tasks/<task_ID>
```

3 Retrieve information about a Key Provider to verify the configuration.

```
GET https://<server>/api/vcenter/trusted-infrastructure/trust-authority-clusters/
<cluster>/kms/providers/<provider>?vmw-task=true
```

You receive the task ID in the response body. You can use the task ID to check the status of the task by running the following HTTP request.

```
GET https://<server>/api/cis/tasks/<task_ID>
```

4 Update an existing Key Provider to modify the connection details and primary key for it.

```
PATCH https://<server>/api/vcenter/trusted-infrastructure/trust-authority-clusters/
<cluster>/kms/providers/<provider>?vmw-task=true
```

You receive the task ID in the response body. You can use the task ID to check the status of the task by running the following HTTP request.

```
GET https://<server>/api/cis/tasks/<task_ID>
```

5 Remove a Key Provider.

```
DELETE https://<server>/api/vcenter/trusted-infrastructure/trust-authority-clusters/
<cluster>/kms/providers/<provider>?vmw-task=true
```

You receive the task ID in the response body. You can use the task ID to check the status of the task by running the following HTTP request.

```
GET https://<server>/api/cis/tasks/<task_ID>
```

If the operation completes successfully, the cluster can no longer use that Key Provider.

Establish Trust Between Key Provider and Key Server

You can use HTTP requests to perform trust management operations.

You can list and update server certificates, retrieve, generate, and update client certificates, generate a CSR, and set the key server credential. Some operations require you to specify parameters in the body of the HTTP request according to your vSphere Trust Authority environment. For details about the syntax of each HTTP request body, see the *API Reference* documentation.

Prerequisites

- Verify that you have access to a working vSphere Trust Authority environment.
- Verify that you have Trusted Infrastructure administrative privileges.

Procedure

1. List the remote server certificates on the configured key servers to verify the trusted key servers.

   ```
   GET https://<server>/api/vcenter/
   trusted-infrastructure/trust-authority-clusters/<cluster>/kms/providers/<provider>/peer-
   certs/current?server_names=<value-1>&server_names=<value-2>&trusted=<true>&vmw-task=true
   ```

   You receive the task ID in the response body. You can use the task ID to check the status of the task by running the following HTTP request.

   ```
   GET https://<server>/api/cis/tasks/<task_ID>
   ```

2. Retrieve the list of trusted server certificates.

   ```
   GET https://<server>/api/vcenter/trusted-infrastructure/trust-authority-clusters/
   <cluster>/kms/providers/<provider>/peer-certs/trusted?vmw-task=true
   ```

   You receive the task ID in the response body. You can use the task ID to check the status of the task by running the following HTTP request.

   ```
   GET https://<server>/api/cis/tasks/<task_ID>
   ```

3. Update the trusted server certificates.

   Note This operation overwrites the existing list of trusted certificates.

   ```
   POST https://<server>/api/vcenter/trusted-infrastructure/trust-authority-clusters/
   <cluster>/kms/providers/<provider>/peer-certs/trusted?vmw-task=true
   ```

   You receive the task ID in the response body. You can use the task ID to check the status of the task by running the following HTTP request.

   ```
   GET https://<server>/api/cis/tasks/<task_ID>
   ```

4 Retrieve the existing client certificate.

```
GET https://<server>/api/vcenter/trusted-infrastructure/trust-authority-clusters/
<cluster>/kms/providers/<provider>/client-certificate?vmw-task=true
```

You receive the task ID in the response body. You can use the task ID to check the status of the task by running the following HTTP request.

```
GET https://<server>/api/cis/tasks/<task_ID>
```

If the operation is successful, you receive the client certificate in PEM format.

5 Generate a new self-signed client certificate, used to establish a secure connection to the key server.

Note This operation overwrites the existing client certificate.

```
POST https://<server>/api/vcenter/trusted-infrastructure/trust-authority-clusters/
<cluster>/kms/providers/<provider>/client-certificate?vmw-task=true
```

You receive the task ID in the response body. You can use the task ID to check the status of the task by running the following HTTP request.

```
GET https://<server>/api/cis/tasks/<task_ID>
```

If the operation is successful, you can provide the newly generated self-signed client certificate to the key server to establish trust with the Key Provider.

6 Update the client certificate to specify what Key Provider should use to authenticate with the key server.

Note If a client certificate already exists, this operation overwrites it.

```
PATCH https://<server>/api/vcenter/trusted-infrastructure/trust-authority-clusters/
<cluster>/kms/providers/<provider>/client-certificate?vmw-task=true
```

You receive the task ID in the response body. You can use the task ID to check the status of the task by running the following HTTP request.

```
GET https://<server>/api/cis/tasks/<task_ID>
```

7 Generate a certificate signing request (CSR) for the client certificate.

Note If a CSR already exists, this operation overwrites it.

```
POST https://<server>/api/vcenter/trusted-infrastructure/trust-authority-clusters/
<cluster>/kms/providers/<provider>/client-certificate/csr?vmw-task=true
```

You receive the task ID in the response body. You can use the task ID to check the status of the task by running the following HTTP request.

```
GET https://<server>/api/cis/tasks/<task_ID>
```

If the operation is successful, you receive the client CSR in PEM format and the host ID which issued it. The generated CSR can later be signed by a third party. The signed CSR should be replicated and set on each host.

8 Set the key server credential for key servers that require a password.

```
PUT https://<server>/api/vcenter/trusted-infrastructure/trust-authority-clusters/
<cluster>/kms/providers/<provider>/credential?vmw-task=true

"secret string"
```

You receive the task ID in the response body. You can use the task ID to check the status of the task by running the following HTTP request.

```
GET https://<server>/api/cis/tasks/<task_ID>
```

Configure Trusted TPMs of Attested ESXi Hosts on a Cluster Level

You can use HTTP requests to manage remote attestation configuration for TPM trust.

You can add, list, remove, and retrieve details about TPM CA certificates and TPM endorsement keys. You can also set and retrieve TPM 2.0 attestation settings. Some operations require you to specify parameters in the body of the HTTP request according to your vSphere Trust Authority environment. For details about the syntax of each HTTP request body, see the *API Reference* documentation.

Prerequisites

- Verify that you have access to a working vSphere Trust Authority environment.
- Verify that you have Trusted Infrastructure administrative privileges.

Procedure

1 Add a new TPM CA certificate to a Trusted Cluster to specify a trusted platform OEM.

```
POST https://<server>/api/vcenter/trusted-infrastructure/trust-authority-clusters/
<cluster>/attestation/tpm2/ca-certificates?vmw-task=true
```

You receive the task ID in the response body. You can use the task ID to check the status of the task by running the following HTTP request.

```
GET https://<server>/api/cis/tasks/<task_ID>
```

2 Retrieve a list of configured TPM CA certificates on a Trusted Cluster to identify the trusted platform OEMs.

```
GET https://<server>/api/vcenter/trusted-infrastructure/trust-authority-clusters/<cluster>/
attestation/tpm2/ca-certificates?vmw-task=true
```

You receive the task ID in the response body. You can use the task ID to check the status of the task by running the following HTTP request.

```
GET https://<server>/api/cis/tasks/<task_ID>
```

3 Remove a TPM CA certificate from a Trusted Cluster because a platform OEM is no longer trusted.

```
DELETE https://<server>/api/vcenter/trusted-infrastructure/trust-authority-clusters/
<cluster>/attestation/tpm2/ca-certificates/<name>?vmw-task=true
```

You receive the task ID in the response body. You can use the task ID to check the status of the task by running the following HTTP request.

```
GET https://<server>/api/cis/tasks/<task_ID>
```

4 Retrieve details about a specific TPM CA certificate on a Trusted Cluster to get more information about the trusted platform OEM.

```
GET https://<server>//api/vcenter/trusted-infrastructure/trust-authority-clusters/
<cluster>/attestation/tpm2/ca-certificates/<name>?vmw-task=true
```

You receive the task ID in the response body. You can use the task ID to check the status of the task by running the following HTTP request.

```
GET https://<server>/api/cis/tasks/<task_ID>
```

5 Add a new TPM endorsement key to a Trusted Cluster to specify a trusted ESXi host.

```
POST https://<server>/api/vcenter/trusted-infrastructure/trust-authority-clusters/
<cluster>/attestation/tpm2/endorsement-keys?vmw-task=true
```

You receive the task ID in the response body. You can use the task ID to check the status of the task by running the following HTTP request.

```
GET https://<server>/api/cis/tasks/<task_ID>
```

6 Retrieve a list of configured TPM endorsement keys in a Trusted Cluster to identify the trusted ESXi hosts.

```
GET https://<server>/api/vcenter/trusted-infrastructure/trust-authority-clusters/<cluster>/
attestation/tpm2/endorsement-keys?vmw-task=true
```

You receive the task ID in the response body. You can use the task ID to check the status of the task by running the following HTTP request.

```
GET https://<server>/api/cis/tasks/<task_ID>
```

7 Remove a TPM endorsement key from a Trusted Cluster because an ESXi host is no longer trusted.

```
DELETE https://<server>/api/vcenter/trusted-infrastructure/trust-authority-clusters/
<cluster>/attestation/tpm2/endorsement-keys/<name>?vmw-task=true
```

You receive the task ID in the response body. You can use the task ID to check the status of the task by running the following HTTP request.

```
GET https://<server>/api/cis/tasks/<task_ID>
```

8 Retrieve details about a specific TPM endorsement key on a Trusted Cluster to get more information about the trusted ESXi host.

```
GET https://<server>/api/vcenter/trusted-infrastructure/trust-authority-clusters/<cluster>/
attestation/tpm2/endorsement-keys/<name>?vmw-task=true
```

You receive the task ID in the response body. You can use the task ID to check the status of the task by running the following HTTP request.

```
GET https://<server>/api/cis/tasks/<task_ID>
```

9 Set the TPM 2.0 attestation settings by specifying that TPM endorsement keys on a Trusted Cluster do not need to be signed because the trusted OEM does not sign endorsement keys.

```
PATCH https://<server>/api/vcenter/trusted-infrastructure/trust-authority-clusters/
<cluster>/attestation/tpm2/settings?vmw-task=true

{
    "require_endorsement_keys" : false,
    "require_certificate_validation" : true
}
```

You receive the task ID in the response body. You can use the task ID to check the status of the task by running the following HTTP request.

```
GET https://<server>/api/cis/tasks/<task_ID>
```

10 Determine the TPM 2.0 attestation settings in a Trusted Cluster.

```
GET https://<server>/api/vcenter/trusted-infrastructure/trust-authority-clusters/<cluster>/
attestation/tpm2/settings?vmw-task=true
```

You receive the task ID in the response body. You can use the task ID to check the status of the task by running the following HTTP request.

```
GET https://<server>/api/cis/tasks/<task_ID>
```

Configure Trusted ESXi Builds on a Cluster Level

You can use HTTP requests to manage trusted instances of ESXi software on a cluster level.

You can import, list, remove, and retrieve details about ESXi base images. Some operations require you to specify parameters in the body of the HTTP request according to your vSphere Trust Authority environment. For details about the syntax of each HTTP request body, see the *API Reference* documentation.

Prerequisites

- Verify that you have access to a working vSphere Trust Authority environment.
- Verify that you have Trusted Infrastructure administrative privileges.

Procedure

1 Import ESXi metadata as a new trusted base image to each host in a vSphere Trust Authority Cluster.

```
POST https://<server>/api/vcenter/trusted-infrastructure/trust-authority-clusters/
<cluster>/attestation/os/esx/base-images?action=import-from-imgdb&vmw-task=true

"YmluYXJ5"
```

You receive the task ID in the response body. You can use the task ID to check the status of the task by running the following HTTP request.

```
GET https://<server>/api/cis/tasks/<task_ID>
```

2 Retrieve a list of trusted ESXi base images in a vSphere Trust Authority Cluster.

```
GET https://<server>/api/vcenter/trusted-infrastructure/trust-authority-clusters/<cluster>/
attestation/os/esx/base-images?
version=<value-1>&version=<value-2>&display_name=<value-1>&display_name=<value-2>&health=<v
alue-1>&health=<value-2>&vmw-task=true
```

You receive the task ID in the response body. You can use the task ID to check the status of the task by running the following HTTP request.

```
GET https://<server>/api/cis/tasks/<task_ID>
```

3 Remove an ESXi base image that should no longer be trusted from a vSphere Trust Authority Cluster.

```
DELETE https://<server>/api/vcenter/trusted-infrastructure/trust-authority-clusters/
<cluster>/attestation/os/esx/base-images/<version>?vmw-task=true
```

You receive the task ID in the response body. You can use the task ID to check the status of the task by running the following HTTP request.

```
GET https://<server>/api/cis/tasks/<task_ID>
```

4 Retrieve details about a trusted ESXi base image version in a vSphere Trust Authority Cluster.

```
GET https://<server>/api/vcenter/trusted-infrastructure/trust-authority-clusters/<cluster>/
attestation/os/esx/base-images/<version>?vmw-task=true
```

You receive the task ID in the response body. You can use the task ID to check the status of the task by running the following HTTP request.

```
GET https://<server>/api/cis/tasks/<task_ID>
```

Retrieve vSphere Trust Authority Components Information

You can use HTTP requests to retrieve information about Attestation Service and Key Provider Service instances running on hosts.

You can use the retrieved information to connect to the hosts running the vSphere Trust Authority components. Some operations require you to specify parameters in the body of the HTTP request according to your vSphere Trust Authority environment. For details about the syntax of each HTTP request body, see the *API Reference* documentation.

Prerequisites

- Verify that you have access to a working vSphere Trust Authority environment.
- Verify that you have Trusted Infrastructure administrative privileges.

Procedure

1 Retrieve detailed information, including the certificates, about the Attestation Service instance running on a Trust Authority Host.

```
GET https://<server>/api/vcenter/trusted-infrastructure/trust-authority-hosts/<host>/
attestation/
```

You receive the details in the response body. You can use the retrieved information to import the Attestation Service instance into a Workload vCenter Server.

2 List Trust Authority Hosts running an Attestation Service instance by using filters.

```
POST https://<server>/api/vcenter/trusted-infrastructure/trust-authority-hosts/attestation?
projection=<value>
```

You receive the results that match your criteria in the response body. You can use the retrieved information to review the Attestation Service instances.

3 Retrieve detailed information, including the certificates, about the Key Provider Service instance running on a Trust Authority Host.

```
GET https://<server>/api/vcenter/trusted-infrastructure/trust-authority-hosts/<host>/kms/
```

You receive the details in the response body. You can use the retrieved information to import the Key Provider Service instance into a Workload vCenter Server.

4 List Trust Authority Hosts running a Key Provider Service instance by using filters.

```
POST https://<server>/api/vcenter/trusted-infrastructure/trust-authority-hosts/attestation?
projection=<value>kms?projection=<value>&action=query
```

You receive the results that match your criteria in the response body. You can use the retrieved information to review the Key Provider Service instances.

Configure vSphere Trust Authority Components

You can use HTTP requests to perform Key Provider Service and Attestation Service management operations.

You can register, list, remove, and retrieve details about Key Provider Service and Attestation Service instances. Some operations require you to specify parameters in the body of the HTTP request according to your vSphere Trust Authority environment. For details about the syntax of each HTTP request body, see the *API Reference* documentation.

Prerequisites

- Verify that you have access to a working vSphere Trust Authority environment.
- Verify that you have Trusted Infrastructure administrative privileges.

Procedure

1 Register a Key Provider Service instance in a Workload vCenter Server.

```
POST https://<server>/api/vcenter/trusted-infrastructure/kms/services
```

The Key Provider Service instance is propagated to all Workload ESXi hosts that the Workload vCenter Server manages.

2 Register an Attestation Service instance in a Workload vCenter Server.

```
POST https://<server>/api/vcenter/trusted-infrastructure/attestation/services
```

The Attestation Service instance is propagated to all Workload ESXi hosts that the Workload vCenter Server manages.

3 List Key Provider Service instances registered in a Workload vCenter Server by using filters.

```
POST https://<server>/api/vcenter/trusted-infrastructure/kms/services?action=query
```

You receive the results that match your criteria in the response body. You can use the filtered list to retrieve the health status of the Key Provider Service instances.

4 List Attestation Service instances registered in a Workload vCenter Server by using filters.

```
POST https://<server>/api/vcenter/trusted-infrastructure/attestation/services?action=query
```

You receive the results that match your criteria in the response body. You can use the filtered list to retrieve the health status of the Attestation Service instances.

5 Remove a registered Key Provider Service instance.

```
DELETE https://<server>/api/vcenter/trusted-infrastructure/kms/services/<service>
```

The Workload ESXi hosts can no longer retrieve keys by using that Key Provider Service instance.

6 Remove a registered Attestation Service instance.

```
DELETE https://<server>/api/vcenter/trusted-infrastructure/attestation/services/<service>
```

The Workload ESXi hosts can no longer attest that their configuration is secure by using that Attestation Service instance.

7 Retrieve detailed information, including the certificates, for a registered Key Provider Service instance.

```
GET https://<server>/api/vcenter/trusted-infrastructure/kms/services/<service>
```

You receive the details in the response body. You can use the retrieved information to verify the Key Provider Service instance.

8 Retrieve detailed information, including the certificates, for a registered Attestation Service instance.

```
GET https://<server>/api/vcenter/trusted-infrastructure/attestation/services/<service>
```

You receive the details in the response body. You can use the retrieved information to verify the Attestation Service instance.

Configure vSphere Trust Authority Components for Trusted Clusters

You can use HTTP requests to manage Key Provider Service and Attestation Service instances that a Trusted Cluster is configured to use.

You can configure, list, remove, and retrieve details about Key Provider Service and Attestation Service instances. Some operations require you to specify parameters in the body of the HTTP request according to your vSphere Trust Authority environment. For details about the syntax of each HTTP request body, see the *API Reference* documentation.

Prerequisites

- Verify that you have access to a working vSphere Trust Authority environment.
- Verify that you have Trusted Infrastructure administrative privileges.

Procedure

1. Configure a cluster in a Workload vCenter Server to use a registered Key Provider Service instance.

   ```
   POST https://<server>/api/vcenter/trusted-infrastructure/trusted-clusters/<cluster>/kms/
   services?vmw-task=true
   ```

   You receive the task ID in the response body. You can use the task ID to check the status of the task by running the following HTTP request.

   ```
   GET https://<server>/api/cis/tasks/<task_ID>
   ```

   If the operation is successful, the Key Provider Service instance is propagated to all Trusted ESXi hosts in the cluster.

2. Configure a cluster in a Workload vCenter Server to use a registered Attestation Service instance.

   ```
   POST https://<server>/api/vcenter/trusted-infrastructure/trusted-clusters/<cluster>/
   attestation/services?vmw-task=true
   ```

   You receive the task ID in the response body. You can use the task ID to check the status of the task by running the following HTTP request.

   ```
   GET https://<server>/api/cis/tasks/<task_ID>
   ```

   If the operation is successful, the Attestation Service instance is propagated to all Trusted ESXi hosts in the cluster.

3. List Key Provider Service instances used by a cluster by using filters.

   ```
   POST https://<server>/api/vcenter/trusted-infrastructure/trusted-clusters/<cluster>/kms/
   services?action=query
   ```

You receive the results that match your criteria in the response body. You can use the filtered list to retrieve the health status of the Key Provider Service instances.

4 List Attestation Service instances used by a cluster by using filters.

```
POST https://<server>/api/vcenter/trusted-infrastructure/trusted-clusters/<cluster>/
attestation/services?action=query
```

You receive the results that match your criteria in the response body. You can use the filtered list to retrieve the health status of the Attestation Service instances.

5 Remove a Key Provider Service instance from the configuration of a Trusted Cluster.

```
DELETE https://<server>/api/vcenter/trusted-infrastructure/trusted-clusters/<cluster>/kms/
services/<service>?vmw-task=true
```

You receive the task ID in the response body. You can use the task ID to check the status of the task by running the following HTTP request.

```
GET https://<server>/api/cis/tasks/<task_ID>
```

If the operation is successful, the Trusted ESXi hosts can no longer retrieve keys by using that Key Provider Service instance.

6 Remove a registered Attestation Service instance from the configuration of a Trusted Cluster.

```
DELETE https://<server>/api/vcenter/trusted-infrastructure/trusted-clusters/<cluster>/
attestation/services/<service>?vmw-task=true
```

You receive the task ID in the response body. You can use the task ID to check the status of the task by running the following HTTP request.

```
GET https://<server>/api/cis/tasks/<task_ID>
```

If the operation is successful, the Trusted ESXi hosts can no longer attest that their configuration is secure by using that Attestation Service instance.

7 Retrieve detailed information, including the certificates, for a configured Key Provider Service instance used by a Trusted Cluster.

```
GET https://<server>/api/vcenter/trusted-infrastructure/trusted-clusters/<cluster>/kms/
services/<service>
```

You receive the details in the response body. You can use the retrieved information to verify the Key Provider Service instance.

8 Retrieve detailed information, including the certificates, for a registered Attestation Service instance used by a Trusted Cluster.

```
GET https://<server>/api/vcenter/trusted-infrastructure/trusted-clusters/<cluster>/
attestation/services/<service>
```

You receive the details in the response body. You can use the retrieved information to verify the Attestation Service instance.

Establish Trust Between Hosts in a vSphere Trust Authority Cluster and a Workload vCenter Server

You can use HTTP requests to perform trust management operations.

You can establish and remove trust between a Workload vCenter Server and the hosts in a vSphere Trust Authority Cluster. You can also list all Workload vCenter Server instances that have established trust with the host in a vSphere Trust Authority Cluster. Some operations require you to specify parameters in the body of the HTTP request according to your vSphere Trust Authority environment. For details about the syntax of each HTTP request body, see the *API Reference* documentation.

Prerequisites

- Verify that you have access to a working vSphere Trust Authority environment.
- Verify that you have Trusted Infrastructure administrative privileges.

Procedure

1. Establish trust between a vSphere Trust Authority Cluster and a Workload vCenter Server by creating a profile, so that the Workload vCenter Server can retrieve the health status of the vSphere Trust Authority components.

   ```
   POST https://<server>/api/vcenter/trusted-infrastructure/trust-authority-clusters/
   <cluster>/consumer-principals?vmw-task=true
   ```

   You receive the task ID in the response body. You can use the task ID to check the status of the task by running the following HTTP request.

   ```
   GET https://<server>/api/cis/tasks/<task_ID>
   ```

2. Remove the trust between a Workload vCenter Server and the hosts in the vSphere Trust Authority Cluster, so that the Workload vCenter Server stops using the hosts for attestation.

   ```
   DELETE https://<server>/api/vcenter/trusted-infrastructure/trust-authority-clusters/
   <cluster>/consumer-principals/<profile>?vmw-task=true
   ```

   You receive the task ID in the response body. You can use the task ID to check the status of the task by running the following HTTP request.

   ```
   GET https://<server>/api/cis/tasks/<task_ID>
   ```

3. List all profiles which the vSphere Trust Authority Cluster trusts.

   ```
   POST https://<server>/api/vcenter/trusted-infrastructure/trust-authority-clusters/
   <cluster>/consumer-principals?action=query&vmw-task=true
   ```

You receive the task ID in the response body. You can use the task ID to check the status of the task by running the following HTTP request.

```
GET https://<server>/api/cis/tasks/<task_ID>
```

Check Trusted Cluster Health

You can use HTTP requests to retrieve information about the health of the applied vSphere Trust Authority component configurations in a Trusted Cluster.

You can retrieve basic and detailed information about the health of Key Provider Service or Attestation Service configurations applied to a Trusted Cluster with respect to the desired state. You can also retrieve detailed information about the health of all applied vSphere Trust Authority component configurations in a Trusted Cluster. The operations require you to specify parameters in the body of the HTTP request according to your vSphere Trust Authority environment. For details about the syntax of each HTTP request body, see the *API Reference* documentation.

Prerequisites

- Verify that you have access to a working vSphere Trust Authority environment.
- Verify that you have Trusted Infrastructure administrative privileges.

Procedure

1. Retrieve a summary about the health status of all Key Provider Service instances configured for use in a Trusted Cluster.

   ```
   POST https://<server>/api/vcenter/trusted-infrastructure/trusted-clusters/<cluster>/kms/
   services-applied-config?action=query&vmw-task=true
   ```

   You receive the task ID in the response body. You can use the task ID to check the status of the task by running the following HTTP request.

   ```
   GET https://<server>/api/cis/tasks/<task_ID>
   ```

   If the operation is successful, you can verify whether all Key Provider Service configurations of the Trusted Cluster are applied successfully and every host in the cluster is consistent with the desired state.

2. Retrieve detailed information about the health status of a specific Key Provider Service instance configured for use in a Trusted Cluster.

   ```
   GET https://<server>/api/vcenter/trusted-infrastructure/trusted-clusters/<cluster>/kms/
   services-applied-config?vmw-task=true
   ```

   You receive the task ID in the response body. You can use the task ID to check the status of the task by running the following HTTP request.

   ```
   GET https://<server>/api/cis/tasks/<task_ID>
   ```

If the operation is successful, you can verify whether the specified Key Provider Service configuration of the Trusted Cluster is applied successfully and every host in the cluster is consistent with the desired state.

3 Retrieve a summary about the health status of all Attestation Service instances configured for use in a Trusted Cluster.

```
POST https://<server>/api/vcenter/trusted-infrastructure/trusted-clusters/<cluster>/
attestation/services-applied-config?action=query&vmw-task=true
```

You receive the task ID in the response body. You can use the task ID to check the status of the task by running the following HTTP request.

```
GET https://<server>/api/cis/tasks/<task_ID>
```

If the operation is successful, you can verify whether all Attestation Service configurations of the Trusted Cluster are applied successfully and every host in the cluster is consistent with the desired state.

4 Retrieve detailed information about the health status of a specific Attestation Service instance configured for use in a Trusted Cluster.

```
GET https://<server>/api/vcenter/trusted-infrastructure/trusted-clusters/<cluster>/
attestation/services-applied-config?vmw-task=true
```

You receive the task ID in the response body. You can use the task ID to check the status of the task by running the following HTTP request.

```
GET https://<server>/api/cis/tasks/<task_ID>
```

If the operation is successful, you can verify whether the specified Attestation Service configuration of the Trusted Cluster is applied successfully and every host in the cluster is consistent with the desired state.

5 Retrieve detailed information about the health status of all vSphere Trust Authority components configured for use in a Trusted Cluster.

```
GET https://<server>/api/vcenter/trusted-infrastructure/trusted-clusters/<cluster>/
services-applied-config?vmw-task=true
```

You receive the task ID in the response body. You can use the task ID to check the status of the task by running the following HTTP request.

```
GET https://<server>/api/cis/tasks/<task_ID>
```

If the operation is successful, you can verify whether the vSphere Trust Authority component configuration is applied successfully and every host in the cluster is consistent with the desired state.

What to do next

If there are errors, you can try to remediate the Trusted Cluster. See Remediate a Trusted Cluster.

Remediate a Trusted Cluster

You can use HTTP requests to remediate vSphere Trust Authority component configurations in a Trusted Cluster or remove the configurations.

You can update the applied Key Provider Service or Attestation Service configurations in a Trusted Cluster to become consistent with the desired state or you can remove the applied Key Provider Service or Attestation Service configurations. You can also update all applied vSphere Trust Authority component configurations in a Trusted Cluster or remove the configurations. By removing the configurations, you can move hosts from a Trusted Cluster to another cluster. The operations require you to specify parameters in the body of the HTTP request according to your vSphere Trust Authority environment. For details about the syntax of each HTTP request body, see the *API Reference* documentation.

Prerequisites

- Verify that you have access to a working vSphere Trust Authority environment.
- Verify that you have Trusted Infrastructure administrative privileges.

Procedure

1. Remediate all Key Provider Service instances configured for use in a Trusted Cluster.

   ```
   PATCH https://<server>/api/vcenter/trusted-infrastructure/trusted-clusters/<cluster>/kms/
   services-applied-config?vmw-task=true
   ```

   You receive the task ID in the response body. You can use the task ID to check the status of the task by running the following HTTP request.

   ```
   GET https://<server>/api/cis/tasks/<task_ID>
   ```

   If the operation is successful, the Key Provider Service configuration of every host in the cluster is consistent with the desired state.

2. Remove all Key Provider Service configurations from a Trusted Cluster.

   ```
   DELETE https://<server>/api/vcenter/trusted-infrastructure/trusted-clusters/<cluster>/kms/
   services-applied-config?vmw-task=true
   ```

   You receive the task ID in the response body. You can use the task ID to check the status of the task by running the following HTTP request.

   ```
   GET https://<server>/api/cis/tasks/<task_ID>
   ```

   If the operation is successful, the applied Key Provider Service configurations are removed from the configuration of every host in the cluster without affecting the desired state.

3 Remediate all Attestation Service instances configured for use in a Trusted Cluster.

```
PATCH https://<server>/api/vcenter/trusted-infrastructure/trusted-clusters/<cluster>/
attestation/services-applied-config?vmw-task=true
```

You receive the task ID in the response body. You can use the task ID to check the status of the task by running the following HTTP request.

```
GET https://<server>/api/cis/tasks/<task_ID>
```

If the operation is successful, the Attestation Service configuration of every host in the cluster is consistent with the desired state.

4 Remove all Attestation Service configurations from a Trusted Cluster.

```
DELETE https://<server>/api/vcenter/trusted-infrastructure/trusted-clusters/<cluster>/
attestation/services-applied-config?vmw-task=true
```

You receive the task ID in the response body. You can use the task ID to check the status of the task by running the following HTTP request.

```
GET https://<server>/api/cis/tasks/<task_ID>
```

If the operation is successful, the applied Attestation Service configurations are removed from the configuration of every host in the cluster without affecting the desired state.

5 Remediate all vSphere Trust Authority components configured for use in a Trusted Cluster.

```
PATCH https://<server>/api/vcenter/trusted-infrastructure/trusted-clusters/<cluster>/
services-applied-config?vmw-task=true
```

You receive the task ID in the response body. You can use the task ID to check the status of the task by running the following HTTP request.

```
GET https://<server>/api/cis/tasks/<task_ID>
```

If the operation is successful, the vSphere Trust Authority component configuration of every host in the cluster is consistent with the desired state.

6 Remove all vSphere Trust Authority component configurations from a Trusted Cluster.

```
DELETE https://<server>/api/vcenter/trusted-infrastructure/trusted-clusters/<cluster>/
services-applied-config?vmw-task=true
```

You receive the task ID in the response body. You can use the task ID to check the status of the task by running the following HTTP request.

```
GET https://<server>/api/cis/tasks/<task_ID>
```

If the operation is successful, the applied vSphere Trust Authority component configurations are removed from the configuration of every host in the cluster without affecting the desired state.

What to do next

You can recheck the Trusted Cluster health after the remediation. See Check Trusted Cluster Health.

vSphere with Tanzu Configuration and Management

14

Starting with vSphere 7.0, you can use the vSphere Automation APIs to configure and manage vSphere with Tanzu.

This chapter provides a brief description of the system architecture and services involved in the process of enabling and disabling vSphere with Tanzu on a vSphere cluster, creating and managing namespaces.

The information in this chapter is intended for vSphere administrators who want to use the vSphere Automation APIs to configure their environment to run Kubernetes workloads in vSphere. To take fully advantage of these vSphere Automation APIs, you must have basic knowledge about the Kubernetes technology and containers.

For more information about how to configure and manage vSphere with Tanzu through the vSphere Client, see the *vSphere with Tanzu Configuration and Management* documentation.

This chapter includes the following topics:

vSphere with Tanzu Terminology

You must understand the basic terminology in this chapter to be able use the vSphere with Tanzu automation APIs effectively.

vSphere with Tanzu Basic Terms

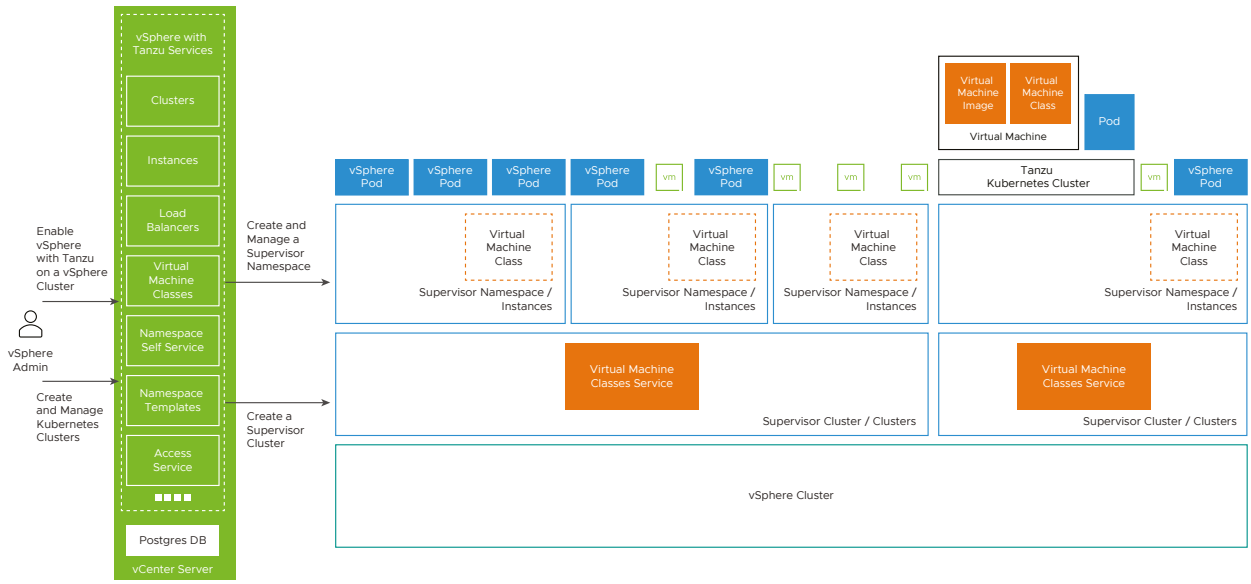
Term	Description
Supervisor Cluster	A vSphere cluster that has the vSphere with Tanzu enabled.
Tanzu Kubernetes cluster	An upstream Kubernetes cluster provisioned and managed by using the VMware Tanzu™ Kubernetes Grid™ Service. A Tanzu Kubernetes resides in a Supervisor Namespace. You can deploy workloads and services to such clusters in the same way as you do with standard Supervisor Cluster.
Supervisor Namespace	A namespace that is created within a Supervisor Cluster. Each namespace sets the resource boundaries for CPU, memory, storage, and also the number of Kubernetes objects that can run within the namespace. After a namespace is configured, you can run Kubernetes workloads within the namespace.
vSphere Pod	A virtual machine with a small footprint that runs one or more Linux containers. A vSphere Pod is equivalent to a Kubernetes pod. vSphere Pods are compatible with the Open Container Initiative (OCI) and can run OCI compatible containers regardless the operating system.
Spherelet	A spherelet is an implementation of the kubelet functionality ported natively on each host in the Supervisor Cluster.
Kubernetes Workload	Workloads are applications that consist of containers running inside vSphere Pods or inside the Tanzu Kubernetes clusters.
Supervisor Cluster control plane	vSphere with Tanzu creates a Kubernetes control plane directly on the hypervisor layer. The control plane manages the worker nodes and the vSphere Pods in the Supervisor Cluster.
Supervisor Cluster worker nodes	ESXi hosts that are part of a Supervisor Cluster are considered as worker nodes. You run your Kubernetes workloads on the worker nodes.
Container Runtime Executive (CRX)	CRX is an isolated Linux execution environment similar to a VM that works together with ESXi.
VM Service	The VM Service functionality allows DevOps engineers to deploy and manage virtual machines in their Kubernetes environment through standard Kubernetes APIs. vSphere administrators are responsible for providing VM Classes and VM Images for the DevOps engineers to choose from, as well as managing resource allocations to self-service provisioned VMs.
Self-Service Namespace	vSphere administrators can activate the Self-Service Namespace service on a Supervisor Cluster and create namespace templates for DevOps engineers to create a Supervisor Namespace themselves.

vSphere with Tanzu Components and Services

Before you can automate some of the administrative tasks for using vSphere with Tanzu, you must first familiarize yourself with the high-level system architecture and components involved.

The vSphere with Tanzu API consists of two packages, `namespace_management` and `namespaces`. In the `namespace_management` package, you can find APIs for enabling and disabling a vSphere cluster with vSphere with Tanzu, configuring the network and storage policies of the Supervisor Cluster, upgrading a cluster to the desired version of vSphere with Tanzu, and so on. In the `namespaces` package, you can find APIs for creating, configuring, and deleting a Supervisor Namespace, and also for setting the necessary permissions for accessing the namespace.

Figure 14-1. Services and Components Involved in Using vSphere with Tanzu



The vSphere Kubernetes Services component runs on vCenter Server and communicates the vSphere admin requests to the Supervisor Cluster control plane. The component comprises of several services which vSphere Automation endpoints you can use to enable vSphere with Tanzu on a vSphere cluster and create Kubernetes workloads.

You can use the Cluster Compatibility service to query a vCenter Server instance about the available clusters that meet the requirements for enabling vSphere with Tanzu.

You can use the Clusters service to enable or disable vSphere with Tanzu on a cluster. You can also reconfigure the settings of a Supervisor Cluster.

You can use the Instances service to create, edit, and delete a Supervisor Namespace from a Supervisor Cluster. You can also change all or some of the settings of an existing namespace.

Starting with vSphere 7.0 Update 1, a Supervisor Cluster backed by a vSphere Distributed Switch uses the HAProxy load balancer to provide connectivity to DevOps and external service. The Load Balancer service represents the user provisioned load balancers.

Starting with vSphere 7.0 Update 2a, vSphere administrators can use the VM Service functionality to enable DevOps engineers to deploy and run VMs and containers in one shared Kubernetes environment through a single Kubernetes native interface. Use the vSphere with Tanzu APIs to define VM Classes and content libraries to allocate resources to virtual machines provisioned by DevOps engineers.

As of vSphere 7.0 Update 2a, vSphere administrators can also configure a Supervisor Namespace as a template on a cluster. Then the DevOps engineers can use it to self-service the creation of Supervisor Namespaces and deploy workloads within them.

Configuring and Managing a Supervisor Cluster

You use the `Clusters` service to enable and disable a Supervisor Cluster, or edit the configuration of an existing Supervisor Cluster. The `Clusters` service is provided within the `namespace_management` package.

You can enable a vSphere cluster to manage Kubernetes workload objects, only after you enable vSphere DRS in a fully automated mode and enable HA on the cluster.

Before you enable programmatically a vSphere with Tanzu on a vSphere cluster, you must prepare your environment to meet the specific networking, storage, and infrastructure requirements. See the *vSphere with Tanzu Configuration and Management* documentation.

For more information about how to configure the storage settings to meet the requirements of vSphere with Tanzu, see Creating Storage Policies for vSphere with Tanzu.

For more information about how to configure the networking settings for Supervisor Clusters that are configured with the VMware NSX-T™ Data Center as the networking stack, see Configuring NSX-T Data Center for vSphere with Tanzu.

Starting with vSphere 7.0 Update 1, you can enable a Supervisor Cluster with vSphere networking or NSX-T Data Center, to provide connectivity between control planes, services, and workloads. A Supervisor Cluster that is configured with vSphere networking uses a vSphere Distributed Switch to provide connectivity to Kubernetes workloads and control planes. The cluster also requires a third-party load balancer that provides connectivity to DevOps users and external services. You can install in your vSphere environment the HAProxy load balancer implementation that VMware provides. See Configuring the vSphere Networking Stack for vSphere with Tanzu and Installing and Configuring the HAProxy Load Balancer.

Staring with vSphere 7.0 Update 2, if you are using vSphere networking, you can use the VMware NSX® Advanced Load Balancer™ to support Tanzu Kubernetes clusters provisioned by the Tanzu Kubernetes Grid Service. See Using the NSX Advanced Load Balancer with vSphere Networking.

Creating Storage Policies for vSphere with Tanzu

Before you enable vSphere with Tanzu, you must set up the storage to provision the Kubernetes infrastructure. You achieve this task by creating storage policies to be used in the Supervisor Cluster and namespaces.

To automate the creation of a tag-based storage policy, use the VMware® vSphere Management SDK. For more information about how to create a tag-based storage policy through the Web Services API, see the *VMware Storage Policy SDK Programming Guide* and *vSphere Web Services SDK Programming Guide* documentations.

Optionally, you can use the vSphere Automation APIs to create and add a tag to the datastore. See the Chapter 10 vSphere Tag Service chapter. Currently, you can create a tag-based storage policy only through the Web Services APIs.

Use the vSphere Automation APIs to retrieve the default storage policy of a specific datastore by calling the `get(datastore_ID)` function of the `DefaultPolicy` service. You can also retrieve commonly used information about the storage policies available in the vCenter Server instance filtered by a specific criteria. Use the `list(policy_filter)` function of the `Policies` service.

You can use the storage policies retrieved through the vSphere Automation APIs to perform the following tasks:

- Assign the storage policies to the Supervisor Cluster. The storage policies set within the Supervisor Cluster enable specification ensure that the Supervisor Cluster control plane, the ephemeral disks of all vSphere Pods, and the container images are placed on the datastores that the policies represent. See Configuring NSX-T Data Center for vSphere with Tanzu.
- Assign the storage policies to the Supervisor Namespace. The storage policies associated with a namespace determine which datastores the namespace can access and use for persistent volumes for the vSphere Pod and the pods inside a Tanzu Kubernetes cluster. See Create a Supervisor Namespace.

Supervisor Cluster Networking

You can enable a Supervisor Cluster with vSphere networking or NSX-T Data Center to provision connectivity to Kubernetes control planes, services, and workloads.

A Supervisor Cluster that uses the vSphere networking stack is backed by a vSphere Distributed Switch and requires a load balancer to provide connectivity to DevOps users and external services. The NSX Advanced Load Balancer and the HAProxy load balancers are supported for vSphere 7.0 Update 2.

A Supervisor Cluster that is configured with NSX-T Data Center, uses the software-based networks of the solution and an NSX Edge load balancer to provide connectivity to external services and DevOps users.

Configuring NSX-T Data Center for vSphere with Tanzu

vSphere with Tanzu requires specific networking configuration to allow you to connect to the Supervisor Clusters, Supervisor Namespaces, and all objects that run inside the namespaces.

Follow the instructions for installing and configuring the NSX-T Data Center for managing Kubernetes workloads documented in the *vSphere with Tanzu Configuration and Management* guide.

First, you need to create a vSphere Distributed Switch and a distributed port group for each NSX Edge uplink. To programmatically achieve this step, use the Web Services APIs as described in the *vSphere Web Services SDK Programming Guide*. Then, you can use the NSX-T Data Center REST APIs to add a compute manager, create transport zones, and perform other steps required for configuring the NSX-T Data Center for vSphere with Tanzu.

Configuring the vSphere Networking Stack for vSphere with Tanzu

To configure a Supervisor Cluster with the vSphere networking stack, you must connect all hosts from the cluster to a vSphere Distributed Switch. Depending on your topology, you must create one or more distributed port groups on the switch and configure them as workload networks to the Supervisor Namespaces on the cluster.

Workload networks provide connectivity to the nodes of Tanzu Kubernetes clusters and to the Supervisor Cluster control planes. The workload network that provides connectivity to Supervisor Cluster control planes is called primary workload network. Each Supervisor Cluster must have one primary workload network represented by a distributed port group.

The Supervisor Cluster control planes on the cluster use three IP addresses from the IP address range that is assigned to the primary workload network. Each node of a Tanzu Kubernetes cluster has a separate IP address assigned from the address range of the workload network that is configured with the namespace where the Tanzu Kubernetes cluster runs.

To create a vSphere Distributed Switch and port groups for configuring the vSphere networking stack of a Supervisor Cluster, you can use the vSphere Web Services APIs as described in the *vSphere Web Services SDK Programming Guide* documentation. When you create a distributed virtual switch, vCenter Server automatically creates one distributed virtual port group. You can use this port group as the primary workload network and use it to handle the traffic for the Supervisor Cluster control planes. Then you can create as many distributed port groups for the workload networks as your topology requires. For a topology with one isolated workload network, create one distributed port group that you will use as a network for all namespaces on the Supervisor Cluster. For a topology with isolated networks for each Supervisor Namespace, create the same number of distributed port groups as the number of namespaces.

To list all workload networks available for a Supervisor Cluster and retrieve information about the configuration of a specific workload network, use the `Networks` service from the vSphere Automation APIs. To associate a vSphere Distributed port group to a workload network, set the necessary information through the `vsphere_network` / `setVsphereNetwork(NetworksTypes.VsphereDVPGNetworkInfo vsphereNetwork)` parameter of the workload network `Info` object. Use the `vsphere_DVPG_network_info` / `NetworksTypes.VsphereDVPGNetworkInfo` structure to describe the configuration or retrieve information about the current configuration of the vSphere Distributed port group of a specific workload network.

If you want to retrieve a list of the distributed switches compatible with vSphere with Tanzu on a vCenter Server system, use the `DistributedSwitchCompatibility` service and filter the available switches by using `VSPHERE_NETWORK` as the networking provider.

Installing and Configuring the HAProxy Load Balancer

You can use the vSphere Automation APIs to customize the HAProxy control plane VM after you install the HAProxy in your vSphere with Tanzu environment.

If you use the vSphere networking stack in your vSphere with Tanzu environment, you need to supply your own load balancer. You can use the open source implementation of the HAProxy load balancer that VMware provides.

For more information about the prerequisites for installation and the deployment procedure through the vSphere Client, see the *vSphere with Tanzu Configuration and Management* documentation.

You can use the vSphere Automation APIs to install and configure the HAProxy load balancer. You can download the latest version of the HAProxy OVA file from the VMware-HAProxy site to a content library item. For more information about how to achieve this task, see Upload an OVF or OVA Package from a Local File System to a Library Item . Then you can create a new VM from the OVA template in the content library as described in Deploy a Virtual Machine or vApp from an OVF Template in a Content Library.

Create a `LoadBalancers.HAProxyConfigCreateSpec` instance to capture the runtime configuration of the HAProxy load balancer. You must set the following configuration parameters:

Parameter	Description
`servers` / `setServers(java.util.List<LoadBalancersTypes.Server> servers)`	A list of `Server`s that represent the endpoints for configuring the HAProxy load balancers. Each endpoint is described by a load balancer IP address and a Data Plane API management port. Each endpoint must be described with the port on the HAProxy VM on which the Data Plane API service listens. The Data Plane API service controls the HAProxy server and runs inside the HAProxy VM. The default port is *5556*. Port *22* is reserved for SSH.
`username` / `setUsername(java.lang.String username)`	The administrator user name that is configured with the HAProxy OVA file and is used to authenticate to the HAProxy Data Plane API server.
`password` / `setPassword(char[] password)`	The password for the administrator user name.
`certificate_authority_chain` / `setCertificateAuthorityChain(java.lang.String certificateAuthorityChain)`	The certificate in PEM format that is signed or is a trusted root of the server certificate that the Data Plane API server presents.

Using the NSX Advanced Load Balancer with vSphere Networking

If you use the vSphere networking stack for workload management, you can install and configure the NSX Advanced Load Balancer, also known as Avi Load Balancer, Essentials Edition, to support the Tanzu Kubernetes clusters.

For more information about how to install and configure the NSX Advanced Load Balancerthrough the vSphere Client, see the *vSphere with Tanzu Configuration and Management* documentation.

You can use the vSphere Automation APIs to deploy the Avi Controller on your vSphere Management network. You can upload the latest version of the NSX Advanced Load Balancer to a library item from your local file system or from a URL. For more information about how to achieve this task, see Upload an OVF or OVA Package from a Local File System to a Library Item . Then you can deploy the Controller VM on your vSphere Management network from the OVA template in the content library as described in Deploy a Virtual Machine or vApp from an OVF Template in a Content Library.

Create a `LoadBalancers.AviConfigCreateSpec` instance to configure the NSX Advanced Load Balancer settings:

Parameter	Description
`server` / `setServer(LoadBalancersTypes.Server server)`	The address of the Avi Controller that is used to configure virtual services.
`username` / `setUsername(java.lang.String username)`	The administrator user name that is used for accessing the Controller VM of the NSX Advanced Load Balancer.
`password` / `setPassword(char[] password)`	The password for the administrator user name.
`certificate_authority_chain` / `setCertificateAuthorityChain(java.lang.String certificateAuthorityChain)`	The certificate in PEM format that is used by the Controller. You can use the certificate that you assigned during the configuration of the NSX Advanced Load Balancer.

Enable vSphere with Tanzu on a Cluster with NSX-T as the Networking Stack

Through the vSphere Automation APIs, you can enable a vSphere cluster for managing Kubernetes workloads. A cluster configured with NSX-T Data Center supports running vSphere Pod and Tanzu Kubernetes clusters.

To enable a vSphere cluster for Kubernetes workload management, you use the services under the `namespace_management` package.

Prerequisites

- Verify that your environment meets the system requirements for enabling vSphere with Tanzu on the cluster. For more information about the requirements, see the *vSphere with Tanzu Configuration and Management* documentation.
- Verify that the NSX-T Data Center is installed and configured. See Configuring NSX-T Data Center for vSphere with Tanzu.
- Create storage policies for the placement of pod ephemeral disks, container images, and Supervisor Cluster control plane cache.
- Verify that DRS is enabled in fully automated mode and HA is also enabled on the cluster.
- Configure shared storage for the cluster. Shared storage is required for vSphere DRS, HA, and storing persistent volumes of containers.
- Verify that the user who you use to access the vSphere Automation services has the **Modify cluster-wide configuration** privilege on the cluster.

- Create a subscribed content library on the vCenter Server system to accommodate the VM image that is used for creating the nodes of the Tanzu Kubernetes clusters.

Procedure

1 Retrieve the IDs of the tag-based storage policies that you configured for vSphere with Tanzu.

   Use the `Policies` service to retrieve a list of all storage policies and then filter the policies to get the IDs of the policies that you configured for the Supervisor Cluster.

2 Retrieve the IDs of the vSphere Distributed Switch and the NSX Edge cluster that you created when configuring the NSX-T Data Center for vSphere with Tanzu.

   Use the `DistributedSwitchCompatibility` service to list all vSphere Distributed Switches associated with the specific vSphere cluster and then retrieve the ID of the Distributed Switch that you configured to handle overlay networking for the Supervisor Cluster. Use the `EdgeClusterCompatibility` service to retrieve a list of the created NSX Edge clusters for the specific vSphere cluster and associated with the specific vSphere Distributed Switch. Retrieve the ID of the NSX Edge cluster that has the tier-0 gateway that you want to use for the namespaces networking.

3 Retrieve the ID of the port group for the management network that you configured for the management traffic.

   Use the `Networks` service to list the visible networks available on the vCenter Server instance that match some criteria and then retrieve the ID of the management network you previously configured.

4 Create a `Clusters.EnableSpec` / `ClustersTypes.EnableSpec` instance and define the parameters of the Supervisor Cluster that you want to create.

   You must specify the following required parameters of the enable specification:

   - Storage policies settings. The storage policy you set for each of the following parameters ensures that the respective object is placed on the datastore referenced in the storage policy. You can use the same or different storage policy for the different inventory objects.

Parameter	Description
`ephemeral_storage_policy` / `setEphemeralStoragePolicy(java.lang.String ephemeralStoragePolicy)`	Specify the ID of the storage policy that you created to control the storage placement of the vSphere Pods.
`image_storage` / `setImageStorage(ClustersTypes.ImageStorageSpec imageStorage)`	Set the specification of the storage policy that you created to control the placement of the cache of container images.
`master_storage_policy` / `setMasterStoragePolicy(java.lang.String masterStoragePolicy)`	Specify the ID of the storage policy that you created to control the placement of the Supervisor Cluster control plane cache.

- Management network settings. Configure the management traffic settings for the Supervisor Cluster control plane.

Parameter	Description
`network_provider` / `setNetworkProvider(ClustersTypes.NetworkProvider networkProvider)`	Specify the networking stack that must be used when the Supervisor Cluster is created. To use the NSX-T Data Center as the network solution for the cluster, select `NSXT_CONTAINER_PLUGIN`.

`master_management_network` / `setMasterManagementNetwork(ClustersTypes.NetworkSpec masterManagementNetwork)`	Enter the cluster network specification for the Supervisor Cluster control plane. You must enter values for the following required properties:
	■ `network` / `setNetwork(java.lang.String network)` - Use the management network ID retrieved in Step 3.
	■ `mode` / `setMode(ClustersTypes.NetworkSpec.Ipv4Mode mode)` - Set `STATICRANGE` or `DHCP` for the IPv4 address assignment mode. The `DHCP` mode allows an IPv4 address to be automatically assigned to the Supervisor Cluster control plane by a DHCP server. You must also set the floating IP address used by the HA primary cluster by using `floating_IP` / `setFloatingIP(java.lang.String floatingIP)`. Use the DHCP mode only for test purposes. The `STATICRANGE` mode, allows the Supervisor Cluster control plane to have a stable IPv4 address. You can use it in a production environment.
	■ `address_range` / `setAddressRange(ClustersTypes.Ipv4Range addressRange)` - Optionally, you can configure the IPv4 addresses range for one or more interfaces of the management network. Specify the following settings:
	  ■ The starting IP address that must be used for reserving consecutive IP addresses for the Supervisor Cluster control plane. Use up to 5 consecutive IP addresses.
	  ■ The number of IP addresses in the range.
	  ■ The IP address of the gateway associated with the specified range.
	  ■ The subnet mask to be used for the management network.

`master_DNS` / `setMasterDNS(java.util.List<java.lang.String> masterDNS)`	Enter a list of the DNS server addresses that must be used from the Supervisor Cluster control plane. If your vCenter Server instance is registered with an FQDN, you must enter the IP addresses of the DNS servers that you use with the vSphere environment so that the FQDN is resolvable in the Supervisor Cluster. The list of DNS addresses must be specified in the order of preference.

Parameter	Description
`master_DNS_search_domains` / `setMasterDNSSearchDomains(java.util.List<java.lang.String> masterDNSSearchDomains)`	Set a list of domain names that DNS searches when looking up for a host name in the Kubernetes API server. Order the domains in the list by preference.
`master_NTP_servers` / `setMasterNTPServers(java.util.List<java.lang.String> masterNTPServers)`	Specify a list of IP addresses or DNS names of the NTP server that you use in your environment, if any. Make sure that you configure the same NTP servers for the vCenter Server instance, all hosts in the cluster, the NSX-T Data Center, and vSphere with Tanzu. If you do not set an NTP server, VMware Tools time synchronization is enabled.

- Workload network settings. Configure the settings for the networks for the namespaces. The namespace network settings provide connectivity to vSphere Pods and namespaces created in the Supervisor Cluster.

Parameter	Description
`ncp_cluster_network_spec` / `setNcpClusterNetworkSpec(ClustersTypes.NCPClusterNetworkEnableSpec ncpClusterNetworkSpec)`	Set the specification for the Supervisor Cluster configured with the NSX-T Data Center networking stack. Specify the following cluster networking configuration parameters for `NCPClusterNetworkEnableSpec`: ■ `cluster_distributed_switch` / `setClusterDistributedSwitch(java.lang.String clusterDistributedSwitch)` - The vSphere Distributed Switch that handles overlay networking for the Supervisor Cluster. ■ `nsx_edge_cluster` / `setNsxEdgeCluster(java.lang.String nsxEdgeCluster)` - The NSX Edge cluster that has tier-0 gateway that you want to use for namespace networking. ■ `egress_cidrs` / `setEgressCidrs(java.util.List<Ipv4Cidr> egressCidrs)` - The external CIDR blocks from which the NSX-T Manager assigns IP addresses used for performing source NAT (SNAT) from internal vSphere Pods IP addresses to external IP addresses. Only one egress IP address is assigned for each namespace in the Supervisor Cluster. These IP ranges must not overlap with the IP ranges of the vSphere Pods, ingress, Kubernetes services, or other services running in the data center. ■ `ingress_cidrs` / `setIngressCidrs(java.util.List<Ipv4Cidr> ingressCidrs)` - The external CIDR blocks from which the ingress IP range for the Kubernetes services is determined. These IP ranges are used for load balancer services and Kubernetes ingress. All Kubernetes ingress services in the same namespace share a common IP address. Each load balancer service is assigned a unique IP address. The ingress IP ranges must not overlap with the IP ranges of the vSphere Pods, egress, Kubernetes services, or other services running in the data center. ■ `pod_cidrs` / `setPodCidrs(java.util.List<Ipv4Cidr> podCidrs)` - The internal CIDR blocks from which the IP ranges for vSphere Pods are determined. The IP ranges must not overlap with the IP ranges

Parameter	Description
	of the ingress, egress, Kubernetes services, or other services running in the data center. All vSphere Pods CIDR blocks must be of at least /23 subnet size.
`worker_DNS` / `setWorkerDNS(java.util.List<java.lang.String> workerDNS)`	Set a list of the IP addresses of the DNS servers that must be used on the worker nodes. Use different DNS servers than the ones you set for the Supervisor Cluster control plane.
`service_cidr` / `setServiceCidr(Ipv4Cidr serviceCidr)`	Specify the CIDR block from which the IP addresses for Kubernetes services are allocated. The IP range must not overlap with the ranges of the vSphere Pods, ingress, egress, or other services running in the data center. For the Kubernetes services and the vSphere Pods, you can use the default values which are based on the cluster size that you specify.

- Supervisor Cluster size. You must set a size to the Supervisor Cluster which affects the resources allocated to the Kubernetes infrastructure. The cluster size also determines default maximum values for the IP addresses ranges for the vSphere Pods and Kubernetes services running in the cluster. You can use the `ClusterSizeInfo.get()` / `GET https://<server>/api/vcenter/namespace-management/cluster-size-info` calls to retrieve information about the default values associated with each cluster size.
- Optional. Associate the Supervisor Cluster with the subscribed content library that you created for provisioning Tanzu Kubernetes clusters. See Creating and Managing Content Libraries for Tanzu Kubernetes Releases.

  To set the library, use `default_kubernetes_service_content_library` / `setDefaultKubernetesServiceContentLibrary(java.lang.String defaultKubernetesServiceContentLibrary)` and pass the subscribed content library ID.

5 Enable vSphere with Tanzu on a specific cluster by passing the cluster enable specification to the `Clusters` service.

Results

A task runs on vCenter Server for turning the cluster into a Supervisor Cluster. Once the task completes, Kubernetes control plane nodes are created on the hosts that are part of the cluster enabled with vSphere with Tanzu. Now you can create Supervisor Namespaces.

What to do next

Create and configure namespaces on the Supervisor Cluster. See Create a Supervisor Namespace.

Java Example of Enabling vSphere with Tanzu on a Cluster with NSX-T Networking

This example enables vSphere with Tanzu on a cluster that has NSX-T configured as the networking stack.

The following code snippet is part of the `EnableSupervisorCluster.java` sample. Some parts of the original code sample are omitted to save space. You can view the complete and up-to-date version of this sample in the `vsphere-automation-sdk-java` VMware repository at GitHub.

```
        (...)
@Override
    protected void run() throws Exception {

        System.out.println("We are building the Spec for enabling vSphere supervisor
cluster");

        ClustersTypes.EnableSpec spec = new ClustersTypes.EnableSpec();
        (...)
        spec.setSizeHint(SizingHint.TINY);
        (...)
        spec.setServiceCidr(serCidr);
        spec.setNetworkProvider(ClustersTypes.NetworkProvider.NSXT_CONTAINER_PLUGIN);
        (...)
        spec.setNcpClusterNetworkSpec(NCPSpec);
        (...)
        spec.setMasterManagementNetwork(masterNet);
        (...)
        spec.setMasterDNS(masterDNS);
        (...)
        spec.setWorkerDNS(workerDNS);
        (...)
        spec.setMasterNTPServers(NTPserver);

        spec.setMasterStoragePolicy(this.storagePolicyId); // Storage policy identifier
        spec.setEphemeralStoragePolicy(this.storagePolicyId);// Storage policy identifier
        spec.setLoginBanner("This is your first Project pacific cluster");
        (...)
        spec.setImageStorage(imageSpec);

        this.ppClusterService.enable(clusterId, spec);
        System.out.println("Invocation is successful for enabling vSphere supervisor cluster,
check H5C");

    }
        (...)
```

Enable vSphere with Tanzu on a Cluster with the vSphere Networking Stack

Starting with vSphere 7.0 Update 1, you can select between creating a Supervisor Cluster with the vSphere networking stack or with NSX-T Data Center as the networking solution. A Supervisor

Cluster that is configured with the vSphere networking stack only supports Tanzu Kubernetes clusters. vSphere Pods are not supported.

To enable a cluster configured with the vSphere networking stack for Kubernetes workloads management, you must use the services under the `namespace_management` package.

Prerequisites

- Verify that your environment meets the system requirements for enabling vSphere with Tanzu on the cluster. For more information about the requirements, see the *vSphere with Tanzu Configuration and Management* documentation.
- Verify that DRS is enabled in fully automated mode and HA is also enabled on the cluster.
- Configure shared storage for the cluster. Shared storage is required for vSphere DRS, HA, and storing persistent volumes of containers.
- Create storage policies for the placement of Kubernetes control planes.
- Create a subscribed content library on the vCenter Server system to accommodate the VM image that is used for creating nodes of Tanzu Kubernetes clusters. See Creating and Managing Content Libraries for Tanzu Kubernetes Releases.
- Add all hosts from the cluster to a vSphere Distributed Switch and create port groups for workload networks. See Configuring the vSphere Networking Stack for vSphere with Tanzu.
- Configure an HAProxy load balancer instance that is routable to the vSphere Distributed Switch that is connected to the hosts from the vSphere cluster.
- Verify that the user who you use to access the vSphere Automation services has the **Namespaces.Manage** privilege on the cluster.

Procedure

1. Retrieve the ID of the cluster which hosts were added to the vSphere Distributed Switch.

   Use the `ClusterCompatibility` service to filter the clusters by using their network providers. To retrieve a list of all clusters in the vCenter Server system which are configured with the vSphere networking stack, set the network provider in the filter specification to `VSPHERE_NETWORK`.

2. Retrieve the IDs of the tag-based storage policies that you configured for vSphere with Tanzu.

   Use the `Policies` service to retrieve a list of all storage policies and then filter the policies to get the IDs of the policies that you configured for the Supervisor Cluster.

3. Retrieve the ID of the port group for the management network that you configured for the management traffic.

   Use the `Networks` service to list the visible networks available on the vCenter Server instance that match some criteria and then retrieve the ID of the management network you previously configured.

4 Create a Supervisor Cluster enable specification and define the parameters of the Supervisor Cluster that you want to enable.

   You must specify the following required parameters of the enable specification:

   - Supervisor Cluster size. You must set a size to the Supervisor Cluster which affects the resources allocated to the Kubernetes infrastructure. The cluster size also determines default maximum values for the IP addresses ranges for the vSphere Pod and Kubernetes services running in the cluster. You can use the `ClusterSizeInfo.get()` / `GET https://<server>/api/vcenter/namespace-management/cluster-size-info` calls to retrieve information about the default values associated with each cluster size.
   - Storage policy settings. To specify the ID of the storage policy that you created to control the placement of the Supervisor Cluster control plane cache, use the `master_storage_policy` / `setMasterStoragePolicy(java.lang.String masterStoragePolicy)` parameter.
   - Load balancer. To specify the user-provisioned load balancer configuration for the cluster, use the `load_balancer_config_spec` / `setLoadBalancerConfigSpec(LoadBalancersTypes.ConfigSpec loadBalancerConfigSpec)` parameter of the enable specification. You must specify the following parameters of the `LoadBalancersTypes.ConfigSpec` specification:

Parameter	Description
`id` / `setId(java.lang.String id)`	A user-friendly name of the load balancer. The name must be an alphanumeric string with a maximum length of 63 characters which is unique across the namespaces in the vCenter Server instance.
`provider` / `setProvider(LoadBalancersTypes.Provider provider)`	The type of the load balancer that you want to use. In vSphere 7.0 Update 2, you can choose between the HAProxy load balancer and the NSX Advanced Load Balancer. Pass as a value to this parameter one of the following constants: `HA_PROXY` or `AVI`.
`address_ranges` / `setAddressRanges(java.util.List<IPRange> addressRanges)`	The IP address ranges in CIDR format from which HAProxy allocates the IP addresses for the virtual servers. You must provide at least one IP range which is reserved by HAProxy. The CIDR range specified with this parameter must not overlap with the IPs allocated for the Kubernetes control planes and workloads. The IP range that you configure must be on a separate subnet.
`ha_proxy_config_create_spec` / `setHaProxyConfigCreateSpec(LoadBalancersTypes.HAProxyConfigCreateSpec haProxyConfigCreateSpec)`	The HAProxy runtime configuration. See Installing and Configuring the HAProxy Load Balancer.
`avi_config_create_spec` / `setAviConfigCreateSpec(LoadBalancersTypes.AviConfigCreateSpec aviConfigCreateSpec)`	The NSX Advanced Load Balancer configuration. See Using the NSX Advanced Load Balancer with vSphere Networking.

- Management network settings. Configure the network parameters for the Kubernetes control planes.

Parameter	Description
`network_provider` / `setNetworkProvider(ClustersTypes.NetworkProvider networkProvider)`	Specify the networking stack that must be used when the Supervisor Cluster is created. To use the vSphere network as the solution for the cluster, select `VSPHERE_NETWORK`.
`master_management_network` / `setMasterManagementNetwork(ClustersTypes.NetworkSpec masterManagementNetwork)`	Enter the cluster network specification for the Supervisor Cluster control plane. You must enter values for the following required properties: ■ `network` / `setNetwork(java.lang.String network)` - Use the management network ID retrieved in Step 3. ■ `mode` / `setMode(ClustersTypes.NetworkSpec.Ipv4Mode mode)` - Set `STATICRANGE` or `DHCP` for the IPv4 address assignment mode. The `DHCP` mode allows an IPv4 address to be automatically assigned to the Supervisor Cluster control plane by a DHCP server. You must also set the floating IP address used by the HA primary cluster by using `floating_IP` / `setFloatingIP(java.lang.String floatingIP)`. Use the DHCP mode only for test purposes. The `STATICRANGE` mode, allows the Supervisor Cluster control plane to have a stable IPv4 address and can be used in a production environment.
`master_DNS` / `setMasterDNS(java.util.List<java.lang.String> masterDNS)`	Enter a list of the DNS server addresses that must be used from the Supervisor Cluster control plane. If your vCenter Server instance is registered with an FQDN, you must enter the IP addresses of the DNS servers that you use with the vSphere environment so that the FQDN is resolvable in the Supervisor Cluster. The list of DNS addresses must be specified in the order of preference.
`master_DNS_search_domains` / `setMasterDNSSearchDomains(java.util.List<java.lang.String> masterDNSSearchDomains)`	Set a list of domain names that DNS searches inside the Kubernetes control plane nodes, so that the DNS server can resolve them. Order the domains in the list by preference.
`master_NTP_servers` / `setMasterNTPServers(java.util.List<java.lang.String> masterNTPServers)`	Specify a list of IP addresses or DNS names of the NTP server that you use in your environment, if any. Make sure that you configure the same NTP servers for the vCenter Server instance, all hosts in the cluster, and vSphere with Tanzu. If you do not set an NTP server, VMware Tools time synchronization is enabled.

- Workload network settings. Configure the settings for the network that will handle the networking traffic for Kubernetes workloads running on the Supervisor Cluster.

Parameter	Description
`service_cidr` / `setServiceCidr(Ipv4Cidr serviceCidr)`	Specify the CIDR block from which the IP addresses for Kubernetes services are allocated. The IP range must not overlap with the ranges of the vSphere Pods, ingress, egress, or other services running in the data center. For the Kubernetes services and the vSphere Pods, you can use the default values which are based on the cluster size that you specify.
`workload_networks_spec` / `setWorkloadNetworksSpec(ClustersTypes.WorkloadNetworksEnableSpec workloadNetworksSpec)`	Enter the workload networks specifications for the cluster. To configure the primary workload network that is used to expose the Supervisor Cluster control plane to DevOps and other workloads, create a `create_spec` / `NetworksTypes.CreateSpec` instance. Enter the following parameters of the vSphere Distributed Switch: ■ `network` / `setNetwork(java.lang.String network)`. The name of the vSphere Distributed Switch that is associated with the hosts in the cluster. The name must be a unique alphanumeric string that does not exceed 63 characters. ■ `network_provider` / `setNetworkProvider(ClustersTypes.NetworkProvider networkProvider)`. Pass `VSPHERE_NETWORK` as value to this parameter. ■ `vsphere_network` / `setVsphereNetwork(NetworksTypes.VsphereDVPGNetworkCreateSpec vsphereNetwork)`. Optionally, you can create a `vsphere_DVPG_network_create_spec` / `NetworksTypes.VsphereDVPGNetworkCreateSpec` instance to describe the configuration of the namespace network backed by the vSphere Distributed port group. You must define the following parameters for the vSphere Distributed port group specification: ■ `portgroup` / `setPortgroup(java.lang.String portgroup)`. Specify the port group that serves as the primary network to the Supervisor Cluster. ■ `address_ranges` / `setAddressRanges(java.util.List<IPRange> addressRanges)`. Set the IP range for allocating IP addresses for the Kubernetes control planes and workloads. You must use unique IP ranges for each workload network. ■ `gateway` / `setGateway(java.lang.String gateway)`. Set the gateway for the primary network.

Parameter	Description
	■ `subnet_mask` / `setSubnetMask(java.lang.String subnetMask)`. Specify the subnet mask of the network.

- Content library settings. Add the subscribed content library that contains the VM images for deploying the nodes of Tanzu Kubernetes clusters. See Creating and Managing Content Libraries for Tanzu Kubernetes Releases.

  To set the library, use `default_kubernetes_service_content_library` / `setDefaultKubernetesServiceContentLibrary(java.lang.String defaultKubernetesServiceContentLibrary)` and pass the subscribed content library ID.

5 Enable the Supervisor Cluster by passing the enable specification to the `Clusters` service.

Results

A task runs on vCenter Server for enabling vSphere with Tanzu on the cluster. Once the task completes, three Kubernetes control planes are created on the hosts that are part of the cluster.

What to do next

Create and configure namespaces on the Supervisor Cluster.

Upgrading a Supervisor Cluster

You can use the vSphere with Tanzu APIs to upgrade a single or a group of clusters to a specific version.

vSphere with Tanzu supports rolling upgrades through the vSphere Automation APIs for Supervisor Clusters and for the infrastructure supporting these clusters. This model ensures that there is minimal downtime for the cluster workloads during the upgrade process.

To retrieve a list of all available vSphere with Tanzu upgrade versions for a specific vCenter Server system, use the `ClusterAvailableVersions` service. You can get information about the release version, name, description, release date, and release notes for each available upgrade.

You must use the `Clusters` service in the `namespace_management.software` package for upgrading a Supervisor Cluster. You can retrieve upgrade information about all Supervisor Clusters enabled on a vCenter Server system by using the `list()` Java method or the `GET https://<server>/api/vcenter/namespace-management/software/clusters` HTTP request. You receive a list of basic upgrade-related information for each cluster, such as the current software version, the date of the last successful upgrade, the upgrade status of the cluster, and so on. In case some of the clusters are in the process of upgrading, you can retrieve also information about their desired upgrade version. If you want to view a more detailed upgrade-related information about a cluster, you must use the `get(cluster_ID)` method or the `GET https://<server>/api/vcenter/namespace-management/software/clusters/<cluster_ID>`.

After you view the details about the upgrade versions that you can apply on a single or multiple Supervisor Clusters, you can create upgrade specifications that define the versions you want to upgrade to. When you upgrade a batch of Supervisor Cluster and for some reason one of the clusters fails to upgrade, you receive information about the pre-check exceptions that led to that cluster upgrade failure.

Java Example of Upgrading a Supervisor Cluster

This example upgrades a Supervisor Cluster.

The following code snippet is part of the `UpgradeSupervisorCluster.java` sample. Some parts of the original code sample are omitted to save space. You can view the complete and up-to-date version of this sample in the `vsphere-automation-sdk-java` VMware repository at GitHub.

```
@Override
    protected void run() throws Exception {

        System.out.println("We are building the Spec for upgrading vSphere supervisor
cluster");
        com.vmware.vcenter.namespace_management.software.ClustersTypes.UpgradeSpec spec = new
UpgradeSpec();
        spec.setIgnorePrecheckWarnings(true);
        spec.setDesiredVersion(this.desiredVersion);
        this.wcpUpdateService.upgrade(clusterId, spec);
        System.out.println(
                "Invocation is successful for updating vSphere supervisor cluster, check H5C,
track the status using GET API");
    }
```

Monitoring the Enable and Upgrade Supervisor Cluster Operations

When you run the Supervisor Cluster enable and upgrade operations, the status of the tasks is not returned. Since these operations might be time-consuming but critical when automating second- and third-party products with vSphere with Tanzu, you can write a logic to track the task status.

To monitor the status of the enable and upgrade Supervisor Cluster operations, get the `Info` / `ClustersTypes.Info` instance and query the Kubernetes and the configuration statuses of the cluster. Track the statuses every two minutes or so, until you receive `READY` for `kubernetes_status` / `getKubernetesStatus()` and `RUNNING` for `config_status` / `getConfigStatus()`. These statuses indicate that the Supervisor Cluster reached the desired configuration status and is ready for running Kubernetes workloads.

Reconfiguring a Supervisor Cluster

You can change some or all the predefined settings of a Supervisor Cluster through the vSphere Automation APIs.

To update only some of the Supervisor Cluster settings, you must create an instance of `UpdateSpec` and pass it to the `update(cluster_ID, update_spec)` method or in the body of `PATCH https://<server>/api/vcenter/namespace-management/clusters/<cluster_ID>` HTTP request. To reconfigure entirely the Supervisor Cluster, you must create an instance of `SetSpec` and pass it to the `set(cluster_ID, set_spec)` method or to the body of the `PUT https://<server>/api/vcenter/namespace-management/clusters/<cluster_ID>` HTTP request.

The settings you can configure with the `UpdateSpec` and `SetSpec` specifications are the same as the ones that you used for enabling the Supervisor Cluster. For example, you can change the storage settings on the Supervisor Cluster. Note that the changes that you make to the storage settings after the initial cluster configuration, apply only to the newly created Supervisor Cluster control planes, vSphere Pods, and container images.

Disabling a Supervisor Cluster

You can programmatically disable vSphere with Tanzu on a vSphere cluster by using the vSphere Automation APIs.

When you disable a Supervisor Cluster, the vSphere Kubernetes Service forcefully deletes from the cluster all objects and configurations part of the Kubernetes infrastructure. To disable Kubernetes workloads on a cluster, call the disable operation of the `Clusters` service and pass as parameter the ID of the specific Supervisor Cluster.

Content Libraries in vSphere with Tanzu

vSphere with Tanzu uses content libraries as centralized repositories for templates, VM images, Tanzu Kubernetes release distributions, and other files related to their deployment.

Creating and Managing Content Libraries for Tanzu Kubernetes Releases

VMware Tanzu distributes Kubernetes software versions as Tanzu Kubernetes releases. To obtain and use these releases on your Tanzu Kubernetes clusters, you create subscribed or local content libraries.

A Tanzu Kubernetes release provides the VMware Kubernetes distribution which can be used with Tanzu Kubernetes clusters. Each Tanzu Kubernetes release is distributed as an OVA package. The Tanzu Kubernetes Grid Service uses the OVA package to deploy the virtual machine nodes for Tanzu Kubernetes clusters.

A Tanzu Kubernetes release is supported on Photon OS. The virtual machine nodes that are built from the OVA package have a 16 GB disk size. You specify the CPU and RAM resource reservations when you use a virtual machine class to size the Tanzu Kubernetes cluster.

Depending on your need for synchronization frequency and on the access to the published content libraries storing the Tanzu Kubernetes releases, you can use two approaches for storing Tanzu Kubernetes releases.

Automated Synchronization of Tanzu Kubernetes Releases

VMware publishes a content library that contains the latest VMware distributions of Kubernetes as an OVA package. If you want to provision Tanzu Kubernetes clusters, you can create a subscribed content library on the vCenter Server instance where vSphere with Tanzu is enabled. When configuring the content library subscription, use the following subscription URL of the publisher : https://wp-content.vmware.com/v2/latest/lib.json. For more information about how to create a subscribed content library, see Subscribe to a Content Library.

When you create the subscription, you configure the synchronization mechanism for downloading the content of the published library. You can select between on demand and automatic download of the virtual machine image for the Tanzu Kubernetes cluster nodes. If you choose to synchronize the subscribed library on demand, only the metadata for the library content is updated and as a result storage space is saved. This approach is an important consideration as more images containing different Kubernetes versions are published. However, the first time you decide to use a new virtual machine image version, you have to wait for it to download.

You associate the subscribed content library with the Supervisor Cluster on which you want to create a Tanzu Kubernetes cluster, when you first enable vSphere with Tanzu on a cluster. See Enable vSphere with Tanzu on a Cluster with NSX-T as the Networking Stack.

The size of the content library can grow over time as new Kubernetes versions and images are published. If the underlying storage runs out of space, you will need to move to a new subscribed content library. After you create a new subscribed content library that has sufficient capacity for the target cluster, update the library association of the Supervisor Cluster. See Reconfiguring a Supervisor Cluster.

Manual Synchronization of Tanzu Kubernetes Releases

In an air-gapped network environment, you can use the storing functionality provided by a local content library for the needed Tanzu Kubernetes releases. You must first create a local content library, then download the OVA package for each Tanzu Kubernetes release that you want to import to the library. See Create a Local Content Library.

You can find the latest versions of the Kubernetes distribution by navigating to the https://wp-content.vmware.com/v2/latest URL. You must download the `photon-ova.ovf` and `photon-ova-disk1.vmdk` for each distribution you want and then upload these files from your local file system to your local content library. See Upload an OVF or OVA Package from a Local File System to a Library Item .

Note Make sure that you use as a name for each library item the Photon image version and the Kubernetes version from the directory where you downloaded the files. For example: `photon-3-k8s-v1.20.2---vmware.1-tkg.1.1d4f79a`.

Creating and Managing Content Libraries for VM Provisioning in vSphere with Tanzu

To provision new virtual machines in a vSphere with Tanzu environment, the DevOps engineers rely on VM templates and images. The role of the vSphere administrator is to make sure the DevOps engineers have access to these VM templates and images by using the Content Library service.

As a vSphere administrator, you can create a local content library and populate it with VM templates in OVF or OVA file format, or other types of files. For more information and a sample of how to create a local content library, see Create a Local Content Library.

You can also create a subscription to download the content of a published local content library as described in the following topic: Subscribe to a Content Library.

After you create the content library, you must populate it with content either from your local file system or from a Web server. You must use only the VM images available on the VMware Cloud Marketplace web site. For example, download or subscribe to VM Service Image for Ubuntu if you want to enable a DevOps engineer to deploy a VM using this image. For more information about the available ways to populate a content library with content, see Library Items.

You must give the DevOps users access to the VM templates stored in the content libraries, so that they can use these templates to provision VMs through the VM Service functionality. To give access, you must associate one or more content libraries to the namespace where the VM Service is present. See Associating a Content Library with a Namespace and Virtual Machines in vSphere with Tanzu.

Associating a Content Library with a Namespace

You must give access to a source of VM templates, so that the DevOps engineers can use them to provision VMs in a self-service manner. To give access, you associate a content library with VM templates to the namespace used by the DevOps engineers.

You can add multiple content libraries to a namespace that has the VM Service enabled or the same content library to several namespaces. You associate a content library to a namespace when you create a new namespace, update or reconfigure an existing one.

To make the VM Service aware of the content libraries in your environment that the DevOps engineers can use to self-service VMs, you must use a `VMServiceSpec` / `VM_service_spec` instance and pass it to the namespace configuration. The instance contains a list of content libraries that will be used by the VM Service. You can specify this list with the `content_libraries` attribute or by calling the `setContentLibraries(java.util.Set<java.lang.String> contentLibraries)` method of the VM Service specification.

You can also associate one or more VM classes with the namespace. See Associating a VM Class with a Supervisor Namespace.

Managing Namespaces on a Supervisor Cluster

You can use the vSphere Automation APIs to create namespaces on a Supervisor Cluster and configure them with resource limits and permissions for the DevOps users.

To create and configure a namespace, use the `Instances` service from the `namespaces` package. You can configure the access control to the objects in a namespace by using the `Access` service.

Create a Supervisor Namespace

You can use the vSphere with Tanzu automation APIs to create namespaces on a Supervisor Cluster. You can set resource quotas, storage, as well as permissions for the DevOps users.

Prerequisites

- Enable vSphere with Tanzu on a vSphere cluster.
- Create users and groups for the DevOps engineers who will use the namespace. For more information about how to create users and groups through the Web Services APIs, see the *vSphere Web Services SDK Programming Guide*.
- Create storage policies for persistent storage used by the vSphere Pods and the pods inside a Tanzu Kubernetes cluster.
- Create VM Classes and content libraries for DevOps provisioned VMs. See Create a VM Class in vSphere with Tanzu and Creating and Managing Content Libraries for VM Provisioning in vSphere with Tanzu.
- Required privileges on the Supervisor Cluster:
  - **Namespaces.Modify cluster-wide configuration**
  - **Namespaces.Modify namespace configuration**
  - **Virtual Machine Classes.Manage Virtual Machine Classes**

Procedure

1. Retrieve the Supervisor Cluster ID by filtering the clusters available in the vCenter Server system.

   Call the list operation of the `Cluster` service from the `vcenter` package and retrieve the cluster ID from the returned cluster summary.
2. Retrieve the ID of the storage policy that you configured for placement of the persistent volumes from vSphere Pods and Tanzu Kubernetes clusters.
3. Configure the access control to the objects in the namespace.

   Create an instance of the `Instances.Access` class and specify the following access information:

Parameter	Description
`domain` / `setDomain(domain)`	Set the domain name of the vCenter Server system on which the namespace is created.
`subject_type` / `setSubjectType(subjectType)`	Set the type of the user accounts that are associated with the specific role for the namespace. You must select between the `USER` and `GROUP` options.
`subject` / `setSubject(subject)`	Set the name of the user or group that have permissions to access the namespace objects.
`role` / `setRole(role)`	Set the role that is associated with the predefined set of privileges that you want to grant the specific user or group. You can select between the `EDIT`, `VIEW` and `OWNER` roles. The owner role is introduced in vSphere 7.0 Update 2a. When a DevOps engineer creates a namespace in a self-service manner, the Namespace Self-Service grants the owner role to the namespace creator. See Self-Service Namespace Management.

4 Create a `CreateSpec` instance that holds the namespaces specification.

The namespace specification can contain the following information:

Parameter	Description
`cluster` / `setCluster(cluster)`	Set the ID of the Supervisor Cluster on which the namespace is created.
`namespace` / `setNamespace(namespace)`	Set a name of the namespace following the DNS label standard defined in RFC 1123. The name must be unique across all namespaces in the current vCenter Server system.
`networks` / `setNetworks(java.util.List<java.lang.String> networks)`	Optional. You can set the workload networks used by the Supervisor Namespace. Pass `null` as a value of this parameter, if the Supervisor Cluster is configured to use the NSX-T Data Center as networking solution. The workload networking support for such namespaces is provisioned by the NSX-T Data Center. If the Supervisor Cluster uses the vSphere networking stack, pass the workload network to be associated with the namespace. If you pass `null` as a value of this parameter, the Supervisor Namespaces on the cluster are automatically associated with the cluster primary workload network. See Configuring the vSphere Networking Stack for vSphere with Tanzu.
`description` / `setDescription(description)`	Optional. You can set a description of the namespace.
`access_list` / `setAccessList(accessList)`	Optional. You can set the access control that is associated with the namespace in Step 3.

Parameter	Description
`storage_specs` / `setStorageSpecs(storageSpecs)`	Optional. You can set the amount of storage dedicated to each storage policy associated with the namespace and the maximum amount of storage that is used by the namespace. Use the `StorageSpec` specification to configure the storage quotas on the namespace.
`resource_spec` / `setResourceSpec(resourceSpec)`	Optional. You can set resource limitations to the namespace. You can limit the CPU, memory, the maximum number of pods that can exist on the namespace, and so on.
`creator` / `setCreator(InstancesTypes.Principal creator)`	Optional. The Namespace Self-Service populates this parameter with information about the DevOps user who created the namespace with `cubectl`. The user name and domain of the namespace creator are stored with this parameter.
`vm_service_spec` / `setVmServiceSpec(InstancesTypes.VMServiceSpec vmServiceSpec)`	Optional. The VM Service specification for the Dev-Ops provisioned virtual machines.

5 Create a namespace object on the Supervisor Cluster by using the namespace create specification.

What to do next

Share the namespace with DevOps engineers and provide them with the user or group configured for accessing the namespace.

Java Example of Creating a Supervisor Namespace

This example creates a Supervisor Namespace on a Supervisor Cluster.

The following code snippet is part of the `CreateNameSpace.java` sample. Some parts of the original code sample are omitted to save space. You can view the complete and up-to-date version of this sample in the `vsphere-automation-sdk-java` VMware repository at GitHub.

```
        (...)
    @Override
    protected void run() throws Exception {

        InstancesTypes.CreateSpec spec =new InstancesTypes.CreateSpec();
        spec.setCluster(this.clusterId);
        spec.setDescription("My first namespace, WOW");
        spec.setNamespace(this.namespaceName);
        InstancesTypes.StorageSpec storageSpec=new InstancesTypes.StorageSpec();
        storageSpec.setLimit(Long.valueOf(this.storageLimit).longValue());
        storageSpec.setPolicy(this.storagePolicyId);
        List<InstancesTypes.StorageSpec> storageSpecs = new
ArrayList<InstancesTypes.StorageSpec>();
        storageSpecs.add(storageSpec);
        spec.setStorageSpecs(storageSpecs);
        InstancesTypes.Access accessList= new InstancesTypes.Access();
        accessList.setDomain(this.domainName);
```

```
        if(this.roleName.equalsIgnoreCase("EDIT")) {
        accessList.setRole(AccessTypes.Role.EDIT);
        } else{
            accessList.setRole(AccessTypes.Role.VIEW);
        }
        accessList.setSubject( this.subjectName); //Default is Administrator
        if(this.subjectType.equalsIgnoreCase("USER")) {
            accessList.setSubjectType( AccessTypes.SubjectType.USER);
        } else{
            accessList.setSubjectType( AccessTypes.SubjectType.GROUP);
        }

        List<InstancesTypes.Access> accessLists = new ArrayList<InstancesTypes.Access>();
        accessLists.add(accessList);
        spec.setAccessList(accessLists);
        this.namespaceService.create(spec);
        System.out.println("Invocation is successful for creating supervisor namespace, check
H5C or call GET API to get status");


    }
        (...)
```

Updating the Namespace Configuration

You can change the whole namespace configuration or only some of the namespace settings.

To change the configuration of an existing namespace, you must have the **Namespaces.Configure** privilege on the Supervisor Cluster.

Note Before deleting a storage policy from vCenter Server or a Supervisor Namespace, or changing the storage policy assignment, make sure that no persistent volume claim with the corresponding storage class runs in the namespace. Also, ensure that no Tanzu Kubernetes cluster is using the storage class.

To patch a namespace configuration, create an `UpdateSpec` specification and set new values only to the configuration settings that you want to change. The parameters of the update specification are the same as the ones you configured during the namespace creation. When you call the update operation, only the settings that you configured in the update specification are applied, the other settings are left as they are.

To reconfigure a namespace entirely, you must create an instance of the `SetSpec` class. You can change the description, access controls, storage settings, and resource limitations of the specific namespace.

Configuring the Access to a Namespace

You can use the vSphere with Tanzu APIs to grant access permissions to DevOps engineers on the Supervisor Namespaces.

Use the `Access` service to retrieve information about the access control of the DevOps engineers on a specific namespace. You can also set up or remove an access control for a specific user or group on a specific namespace, and add another access control on the namespace. You set up each access control to allow a user or group to access a namespace in a specific vCenter Server system. You can grant access to a DevOps engineer to more than one namespace.

You must have the **Namespaces.Configure.** privilege to grant permissions to a user. You assign the view and edit access role on the namespace for the user or group.

Starting with vSphere 7.0 Update 2a, you can also assign the owner role to a DevOps engineer. These roles allow the user to deploy workloads, share the namespace with other DevOps engineers, and delete it when it is no longer needed.

Self-Service Namespace Management

You can use the vSphere with Tanzu automation APIs to create a Supervisor Namespace with specific resource quotas, set permissions, and assign storage policies. DevOps engineers can then use the namespace as a template for self-provisioning namespaces on the cluster.

Starting with vSphere 7.0 Update 2a, the Namespace Self-Service feature is available in vSphere with Tanzu. The service enables Kubernetes users to create Supervisor Namespaces from templates configured through the automation APIs or vSphere Client. To activate the Namespace Self-Service on a cluster, use one of the following options:

- Create a self-service namespace template and then activate the Namespace Self-Service on the cluster.
- Create or update a self-service namespace template simultaneously with activating the Namespace Self-Service on the cluster.

Currently, only one namespace self-service template is allowed per Supervisor Namespace. After a DevOps engineer creates a namespace from the template, the namespace can also be deleted through `kubectl`. You can verify whether a namespace is created from a template by retrieving the value of the `self_service_namespace` / `getSelfServiceNamespace()` flag of the namespace instance information object.

To create a template for a self-service namespace, call the `create(java.lang.String cluster, NamespaceTemplatesTypes.CreateSpec spec)` method of the `NamespaceTemplates` interface. You can also use the `POST https://<vCenter_server_IP>/api/vcenter/namespaces/namespace-templates/clusters/<cluster_ID>` HTTP request. In both cases you must use as parameters the cluster ID and the namespace template create specification.

You define the following configuration settings and resource limitations of the template:

Parameter	Description
`template` / `setTemplate(java.lang.String template)`	The identifier of the namespace template must be a unique name across all clusters on the vCenter Server instance. The name must be compliant with DNS.
`resource_spec` / `setResourceSpec(Structure resourceSpec)`	The resource quotas, such as CPU and memory, that are reserved for the namespace on the vCenter Server instance. The CPU limit is set in MHz and the minimum value is 10MHz. The memory and storage limits are set in MiB. For more options to configure resource limits for the namespace, see the `ResourceQuotaOptionsV1` class in the API Reference documentation.
`storage_specs` / `setStorageSpecs(java.util.List<InstancesTypes.StorageSpec> storageSpecs)`	The amount of storage in MiB utilized for each storage policy that you associate with the namespace. You must specify at least one policy.
`networks` / `setNetworks(java.util.List<java.lang.String> networks)`	Optional. The networks associated with the namespace. Currently, you can set only one network for the namespace. Pass `null` as argument if the Supervisor Cluster is configured with NSX-T Data Center support. If you pass `null` for a namespace template on a cluster configured with a vSphere networking stack, the namespace is automatically associated with the Supervisor management workload network.
`permissions` / `setPermissions(java.util.List<NamespaceTemplatesTypes.Subject> permissions)`	Optional. The permissions that allow DevOps engineers to use the template to self-provision namespaces through `kubectl`. If set to `null`, only users with the Administrator role can use the template.

Once you have the template created, you can activate the Namespace Self-Service on the cluster. Call the `activate(java.lang.String cluster)` method of the `NamespaceSelfService` interface or use the `POST https://<vCenter_server_IP>/api/vcenter/namespaces/namespace-self-service/<cluster_ID>?action=activate` HTTP request. If you want to restrict DevOps users to use the namespace template on a cluster, you can deactivate the Namespace Self-Service feature. Then users will be able to delete only the namespaces already created from the template.

You can use the `activateWithTemplate` option provided by the `NamespaceSelfService` interface. Call the `activateWithTemplate(java.lang.String cluster, NamespaceSelfServiceTypes.ActivateTemplateSpec spec)` method of the `NamespaceSelfService` interface or use the `POST https://<vCenter_server_IP>/api/vcenter/namespaces/namespace-self-service/<cluster_ID>?action=activateWithTemplate` HTTP request. Depending on the availability of a template on the cluster, this method either creates a new namespace template or activates the deactivated service and at the same time updates the existing template.

Virtual Machines in vSphere with Tanzu

vSphere with Tanzu offers the VM Service functionality to enable DevOps engineers to provision and manage VMs on a namespace in a self-service manner. You use the vSphere with

Tanzu automation APIs to create VM classes that specify the deployment policy and resource reservations of such VMs.

Starting with vSphere 7.0 Update 2a, DevOps engineers can use the VM Service functionality to deploy and run VMs on a namespace through the `kubectl` commands. You can use the vSphere with Tanzu automation APIs to manage the two VM Service components: VM classes and content libraries. For more information about managing content libraries in the context of vSphere with Tanzu, see Content Libraries in vSphere with Tanzu.

You can use the automation APIs to create and manage VM classes. A VM class specification defines the number of CPUs, memory capacity, and resource reservation settings of the desired virtual machine. vSphere with Tanzu currently offers twelve ready-to-use VM classes (T-shirt sizes) that are derived from the most popular VMs in Kubernetes. Based on the resource reservation that a VM specification requests, each predefined VM class has two editions: guaranteed and best effort. The guaranteed VM class fully reserves the configured resources. A best effort VM class does not guarantee any resource reservations and allows their overcommitment.

You associate a VM class with a specific namespace to make it available to the DevOps engineers who have access to that namespace. You can assign any number of existing VM classes or create a custom one. Note that VMs deployed by the DevOps engineers through the VM Service can only be managed with the `kubectl` commands. A VM provisioned by DevOps engineers shares the same resources in a namespace as containers.

Use the `VirtualMachineClasses` interface to create and manage a specification of a VM class object. Through these objects you predefine the number of CPUs, memory capacity, and reservation settings. See Create a VM Class in vSphere with Tanzu. To make a VM class available to the DevOps engineers for self-service VM deployment, you must associate it with a specific namespace. See Associating a VM Class with a Supervisor Namespace.

Create a VM Class in vSphere with Tanzu

You can use the vSphere Automation Kubernetes APIs to create custom VM classes to be used for VM deployment in vSphere with Tanzu.

A VM class specifies the CPU, memory, and resource reservations for a VM. vSphere with Tanzu offers several preconfigured VM classes which you can use as is, edit, or delete. You can also create a custom VM class in your vCenter Server instance and it will be available to all Supervisor Clusters and the namespaces created in these clusters. Note that even though a VM class is available to all namespaces, a DevOps user can only use the VM classes associated with the namespaces that he/she can access.

Prerequisites

Required privileges:

- **Namespaces.Modify cluster-wide configuration**
- **Namespaces.Modify namespace configuration**
- **Virtual Machine Classes.Manage Virtual Machine Classes**

Procedure

1 Create the specification of the VM class object by providing the following attributes:

Attribute	Description
`id` / `setId(java.lang.String id)`	The identifier of the VM class must follow these DNS requirements: ■ A unique name in the current vCenter Server instance. ■ An alphanumeric name with maximum 63 characters. ■ No uppercase letters of spaces. ■ A dash can be used anywhere except as a first or last character. Note that after a VM class is created, you cannot edit its ID.
`cpu_count` / `setCpuCount(long cpuCount)`	The number of virtual CPUs (vCPUs) configured for a VM that will be deployed with this VM class.
`memory_MB` / `setMemoryMB(long memoryMB)`	The memory in MB configured for a VM that will be deployed with this VM class. The value must be between 4 MB and 24 TB and a multiple of 4.
`description` / `setDescription(java.lang.String description)`	Optional. The description of the VM class.
`cpu_reservation` / `setCpuReservation(java.lang.Long cpuReservation)`	Optional. The percentage of total available CPU resources reserved for the VM deployed with the VM class. The percentage you specify with this attribute is multiplied by the minimum CPU available among all cluster nodes to get the CPU resources guaranteed by vSphere for a VM. The resulting value is in MHz.
`memory_reservation` / `setMemoryReservation(java.lang.Long memoryReservation)`	Optional. The percentage of available memory that is reserved for a VM deployed with this VM class. The value can be between 0 and 100%.

2 Create the VM class object.

Call the `create(VirtualMachineClassesTypes.CreateSpec spec)` method of the `VirtualMachineClasses` interface and pass as argument the created VM class specification. You can also call the `POST https://<vCenter_Server_IP>/api/vcenter/namespace-management/virtual-machine-classes` HTTP request and pass the created VM class specification in the request body.

What to do next

After you create the custom VM class, you can edit its parameters or delete it from your environment. See Editing or Removing a VM Class from Your Environment.

You can make your VM class available to DevOps engineers by associating it with a namespace. See Associating a VM Class with a Supervisor Namespace.

Editing or Removing a VM Class from Your Environment

You can use the automation APIs to edit the configuration of a VM class that you created or a predefined VM classes that vSphere with Tanzu offers. When you no longer need an existing VM class, you can remove it from your vCenter Server instance.

Note that editing a VM class specification does not affect the VMs that are already deployed by the DevOps engineers from this class. Only newly deployed VMs will use the reconfigured VM class.

Deleting a VM class results in its removal from all related namespaces. All VMs deployed with this VM class remain unchanged but DevOps engineers can no longer use it.

You can list all VM classes available for a vCenter Server instance. Call the `list` method of the `VirtualMachineClasses` interface or use the `GET https://<vCenter_Server_IP>/api/vcenter/namespace-management/virtual-machine-classes` HTTP request. You receive a list of all VM classes and a detailed information about each one of them. Based on the detailed information, you can narrow the list and retrieve the IDs of the VM classes that you want to edit or delete.

To edit a VM class configuration, call the `update(String vmClass, VirtualMachineClassesTypes.UpdateSpec spec)` method of the `VirtualMachineClasses` interface and pass as arguments the VM class ID and the update specification. You can also use the `PATCH https://<vCenter_Server_IP>/api/vcenter/namespace-management/virtual-machine-classes/<VM_Class_ID>` HTTP request and pass the update specification in the request body. You can edit all VM class attributes except the ID of the class. Only the attributes modified within the update specification will be edited.

To delete a VM class from your environment, call the `delete(java.lang.String vmClass)` method of the `VirtualMachineClasses` interface and pass the ID of the class as argument. You can also use the `DELETE https://<vCenter_Server_IP>/api/vcenter/namespace-management/virtual-machine-classes/<VM_Class_ID>` HTTP request.

Associating a VM Class with a Supervisor Namespace

You must associate a VM class with a namespace to make it available for DevOps engineers to deploy VMs in a self-service manner.

You can associate one or more VM classes with a single namespace or you can add one VM class to several namespaces. You can use the predefined VM classes that vSphere with Tanzu provides or you can create custom ones. See Create a VM Class in vSphere with Tanzu.

Since VM Service is the feature responsible for handling VM classes, you must make that service aware of the VM classes available to the engineers using a specific namespace. You can achieve this when you create a new namespace or edit an existing one. See Managing Namespaces on a Supervisor Cluster.

When you create the VM Service specification set the list of VM classes that must be used to create VMs on the specific namespace. You achieve this by calling the `setVmClasses(java.util.Set<java.lang.String> vmClasses)` method of the `VirtualMachineClasses` object or using the `vm_classes` property of the `vm_service_spec` structure.

You can also associate one or more content libraries with a namespace that has the VM Service enabled. See Associating a Content Library with a Namespace.