Migrating/Upgrading applications to vSphere 6.0

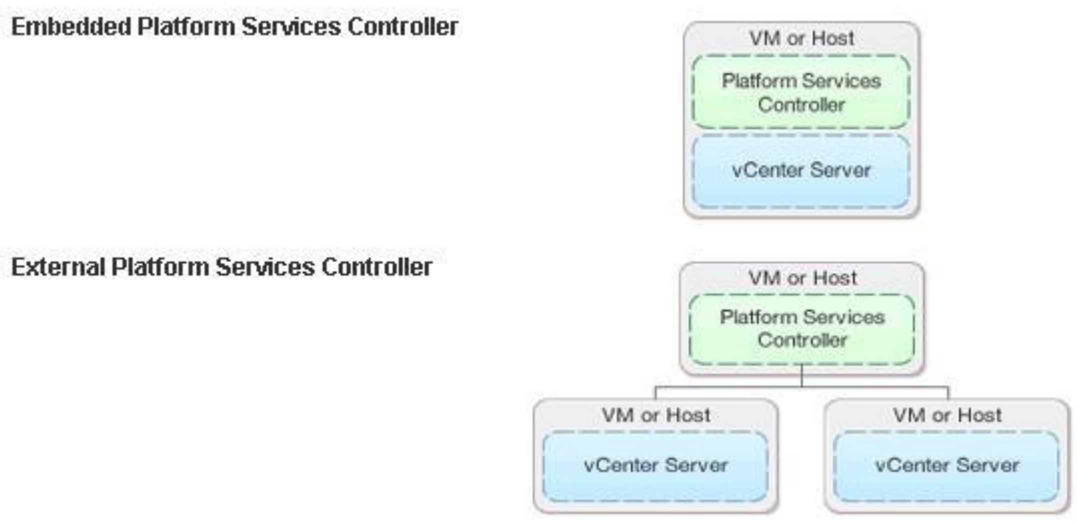
Intended Audience

Users who want to migrate/upgrade existing applications/scripts etc to vSphere 6.0 setup.

Upgrade to vSphere 6.0

Please refer to the vSphere 6.0 Upgrade document from https://www.vmware.com/support/pubs/vsphere-esxi-vcenter-server-pubs.html. You can find them under **ESXi and vCenter Server 6.0 Documentation > vSphere Upgrade**

An upgraded setup or a newly installed setup will look like either of the one below. Embedded or External.


Embedded Platform Services Controller
VM or Host
Platform Services Controller
vCenter Server
External Platform Services Controller
VM or Host
Platform Services Controller
VM or Host
vCenter Server
VM or Host
vCenter Server

New Features in vSphere 6.0

Tagging

Tags allow you to attach metadata to objects in the vSphere inventory to make these objects more sortable and searchable. A tag is a label that you can apply to objects in the vSphere inventory. When you create a tag, you assign that tag to a category. Categories allow you to group related tags together. When you define a category, you can also specify which object types its tags can be applied to and whether more than one tag in the category can be applied to an object. For example, if you wanted to tag your virtual machines by guest operating system type, you could create a category called 'operating system', and specify that it applies to virtual machines only and that only a single tag can be applied to a virtual machine at any time. The tags in this category could be "Windows", "Linux", and "Mac OS". If multiple vCenter Server instances are configured to use Enhanced Linked Mode, tags and tag categories are replicated across all these vCenter Server instances. Tagging replaces the custom attributes functionality found in previous versions of vCenter Server. If you have existing custom attributes, you can convert them into tags.

Content Library

Content libraries are container objects for VM templates, vApp templates, and other types of files. vSphere administrators can use the templates in the library to deploy virtual machines and vApps in the vSphere inventory. Sharing templates and files across multiple vCenter Server instances in same or different locations brings out consistency, compliance, efficiency, and automation in deploying workloads at scale. A content library is managed from a single vCenter Server instance, but can be shared across multiple vCenter Server systems.

Lookup Service

vCenter Lookup Service is a component of Platform Service Controller. Lookup Service registers the location of vSphere components so they can securely find and communicate with each other.

Upgrading applications to vSphere 6.0

The good news is, all of your existing applications/scripts are completely backward compatible and no code change is required. Now given that, the vSphere 6.0 setup has been migrated and also the applications/scripts which you have also seems to be working fine without any code change.

Lets see how to change your existing code to access the new features of vSphere 6.0 (tagging, content library, lookup service etc).

Changes to consume new API's

- My application/script contains hard coded VIM and SSO API URL's.

=>

- Start using the **Lookup Service API's** to get the VIM, SSO and other new Service URL's.

```
// Configure and login to the platform service controller containing the lookup
service.
PlatformServiceController pscNode = new PlatformServiceController(lsUrl);
pscNode.login(ssoUsername, ssoPassword);
String ssoUrl = pscNode.getSsoUrl();
// Get access to the lookup service in the platform service controller.
LookupServiceHelper lsHelper = pscNode.getLsServiceHelper();
// Get access to the vCenter Server or management node. The below call to get the
default
// management node works only if there is only 1 vCenter Server or Management Node.
String mgmtNodeId = lsHelper.getDefaultMgmtNode();

// Get access to the service manager using the platform service controller and mgmt
node ID.
ServiceManager serviceManager = ServiceManagerFactory.getServiceManager(pscNode,
mgmtNodeId);
// Get access to the legacy VIM API's.
String vimUrl = serviceManager.getVimUrl();
```

- My application/script contains the legacy username password based login mechanism.

=>

- Start using the **SSO Token based authentication** across all the API's.

```
// Configure and login to the platform service controller containing the lookup
service.
PlatformServiceController pscNode = new PlatformServiceController(lsUrl);
// Login using the SSO username and password and retrieve the SAML token.
pscNode.login(ssoUsername, ssoPassword);
SSOConnection ssoConnection = pscNode.getSsoConnection();
ssoConnection.getSamlBearerToken();
```

- My application/script contains **multiple VIM/vCenter API** endpoints.

=>

- Start using the **Lookup Service API's** and get access to these multipe VIM/vCenter API endpoints. Refer to the ListServices and print_services.py from VMware vCloud Suite SDK for Java and Python respectively.

```
// Configure and login to the platform service controller containing the lookup
service.
PlatformServiceController pscNode = new PlatformServiceController(lsUrl);
pscNode.login(ssoUsername, ssoPassword);
// Get access to the lookup service in the platform service controller.
LookupServiceHelper lsHelper = pscNode.getLsServiceHelper();
// Get access to the vCenter Server or management node.
String vcenter1Id = lsHelper.getMgmtNodeId("vcenter1.acme.com");
String vcenter2Id = lsHelper.getMgmtNodeId("vcenter2.acme.com");

// Get access to the service manager using PSC and respective mgmt node ID's.
ServiceManager vCenter1Manager = ServiceManagerFactory.getServiceManager(pscNode,
vcenter1Id);
ServiceManager vCenter2Manager = ServiceManagerFactory.getServiceManager(pscNode,
vcenter2Id);
// Get access to the legacy VIM API's.
VimPortType vCenter1VimPort = vCenter1Manager.getVimPortType();
VimPortType vCenter2VimPort = vCenter2Manager.getVimPortType();

// Get access to the new Tagging API's.
Tag vCenter1TagService = vCenter1Manager.getVapiService(Tag.class);
Tag vCenter2TagService = vCenter2Manager.getVapiService(Tag.class);
```

- My application/script **persists the VIM moref's** in the context of a VIM/vCenter.

=>

- Even in the new **embedded/external PSC node** setup. The **VIM/vCenter API's are only in the context of a vCenter**. The API's do not span/cross vCenter's. Your persisted moref's and the applications relying on it should still work the same way with no issues. Note: All of the new API's (Tagging, Content Library etc) uses UUID's as identifiers.

- My application/script was working with both vCenter Server Appliance 5.5 and vCenter Server for Windows 5.5 setup.

=>

- As described earlier, as part of the upgrade process. The end result of an upgraded setup should not break any of your legacy applications/scripts. Ex:

  - If you had pointed to the SSO and VIM URL's which were part of a vCenter Server Appliance 5.5 in the same node. The upgrarde can only be done to a vCenter Server with an Embedded Platform Services Controller. Which would still preserve your SSO and VIM URL's in the same node.

  - If you had pointed to the SSO and VIM URL's residing in different nodes using a vCenter Server for Windows 5.5 and vCenter Single Sign-On server setup. The upgrade can only be done to a vCenter Server with an external Platform Services Controller. Which would still preserve your SSO and VIM URL's in different nodes.

In the subsequent section below, we will take an migration scenario of a simple application to use the new tagging feature.

Sample Scenario - Java

Existing Sample

Fetch an existing cluster and perform some operations on it.

The code below demonstrates the steps that needs to be done in performing a simple fetch of an existing cluster.

- Configure and connect to the SSO endpoint. Get the SAML token using the SSO Username and Password.
- Configure and connect to the VIM endpoint. Use the SAML token to login and extract the session cookie.
- Use the session cookie in the subsequent VIM API calls.

```
import com.vmware.vcloud.suite.samples.vim.helpers.VimUtil;
import com.vmware.vim25.InvalidPropertyFaultMsg;
import com.vmware.vim25.ManagedObjectReference;
import com.vmware.vim25.NotFoundFaultMsg;
import com.vmware.vim25.RuntimeFaultFaultMsg;
import com.vmware.vim25.ServiceContent;
import com.vmware.vim25.VimPortType;
import com.vmware.vim25.VimService;

public class ExistingSample {

    public static VimPortType vimPort;
    public static ServiceContent serviceContent;

    public static void login(String vimUrl, String ssoUsername, String ssoPassword) {
        VimService vimService = new VimService();
        vimPort = vimService.getVimPort();
        // retrieve the service content.
        // get the SAML token from SSO.
        // login to VC using vimPort.loginByToken()
        // extract the cookie.
        // reuse the cookie in the subsequent calls via vimPort.
    }

    public static ManagedObjectReference findCluster(String clusterName)
            throws InvalidPropertyFaultMsg, RuntimeFaultFaultMsg, NotFoundFaultMsg {
        return VimUtil.getCluster(vimPort, serviceContent, clusterName);
    }

    public static void main(String args[]) throws InvalidPropertyFaultMsg,
RuntimeFaultFaultMsg,
            NotFoundFaultMsg {

        String vimUrl = "https://10.161.18.127/sdk";
        String ssoUsername = "Administrator@vsphere.local";
        String ssoPassword = "Admin!23";
        String clusterName = "vAPISDKCluster";

        login(vimUrl, ssoUsername, ssoPassword);

        ManagedObjectReference clusterMoref = findCluster(clusterName);
    }
}
```

Upgraded Sample

Note: Requires JDK 1.7 or above.

Fetch an existing cluster, **tag it** (new vSphere 6.0 feature) and do some operations on it.

The code below demonstrates the same functionality as above. But in addition to it. It demonstrates on how to consume the new tagging service and tag the retrieved cluster.

Changes:

- **SSO endpoint and VIM endpoint URL's** were hardcoded and need to be known earlier. Now, you can use the PlatformServiceController's Lookup Service feature to get the SSO, VIM and other new service/feature URL's.
- **PlatformServiceController** class when constructed with the Lookup Service URL(https://pscnode/lookupservice/sdk) acts as an entry point to your application. When there are multiple platform service controllers(federated) and

to each of which multiple VC's are attached. You need not worry about keeping track of the various VC's across all of these platform service controllers. Since they are federated, just connecting to any one of the Lookup Serivce in the platform service controller will provide the details of all the VC's in the setup. Also if the SSO's across the platform service controller's belong to the same authentication domain (AD/LDAP). You can get the SAML token from any one of the platform service controller node and use it to login to any of the VC's.

- **ServiceManager** class when constructed with the proper management node id provides access to the different services seamlessly. Ex: VIM API's, Tagging, Content Library etc. ServiceManager's -> getVimPortType() provides you access to the legacy VIM API's. Similary ServiceManager -> getVapiService(Tag.class) provides access to the new Tagging feature. So, once you get access to these various existing and new services handles you can start accessing the features.
- **ID magic** - The cluster moref and other resources in the legacy VIM API's contains the ID's in the managed object reference format. This format contains the type and value (not a UUID, just a string) information. The new API's/features are now exposing UUID's. For Ex: Tagging though is a feature by itself. It needs the resources to be specified for it to tag them. In this case, the cluster moref is understood by the tagging feature via the DynamicID. All you have to do is, construct a **Dynamic ID** using ManagedObjectReference.
    - **ManagedObjectReference clusterMoref = VimUtil.getCluster(clusterName);**
    - **DynamicID clusterId = new DynamicID(clusterMoref.getType(), clusterMoref.getValue());**

Now you can start to use this constructed clusterId from the clusterMoref onto the new API's.

- Similarly, there is another type called **Static ID**. This can be used as is i.e the managed object's moref value (not the type) can be used as a static ID in other places. Ex: In the new Content Library feature, in order to create a new library backed by a VC datastore, you need to provide the datastore's moref value as an

ID in the new Content Library creation API's. Refer to the ContentLibraryWorkflow sample.

```
import com.vmware.cis.tagging.Tag;
import com.vmware.cis.tagging.TagAssociation;
import com.vmware.vapi.saml.exception.InvalidTokenException;
import com.vmware.vapi.std.DynamicID;
import com.vmware.vcloud.suite.lookup.RuntimeFaultFaultMsg;
import com.vmware.vcloud.suite.samples.common.MultipleManagementNodeException;
import com.vmware.vcloud.suite.samples.common.PlatformServiceController;
import com.vmware.vcloud.suite.samples.common.ServiceManager;
import com.vmware.vcloud.suite.samples.common.ServiceManagerFactory;
import com.vmware.vcloud.suite.samples.vim.helpers.VimUtil;
import com.vmware.vim25.InvalidLocaleFaultMsg;
import com.vmware.vim25.InvalidLoginFaultMsg;
import com.vmware.vim25.InvalidPropertyFaultMsg;
import com.vmware.vim25.ManagedObjectReference;
import com.vmware.vim25.NotFoundFaultMsg;
import com.vmware.vim25.VimPortType;

public class UpgradedSample {

    public static ServiceManager serviceManager;
    public static Tag tagService;
    public static TagAssociation tagAssociationService;
    public static VimPortType vimService;

    public static void login(String lsUrl, String ssoUsername, String ssoPassword)
            throws RuntimeFaultFaultMsg, MultipleManagementNodeException,
            com.vmware.vim25.RuntimeFaultFaultMsg, InvalidLocaleFaultMsg,
InvalidLoginFaultMsg,
            InvalidTokenException, DatatypeConfigurationException {
        PlatformServiceController psc = new PlatformServiceController(lsUrl);
        psc.login(ssoUsername, ssoPassword);
        String mgmtNodeId = psc.getLsServiceHelper().getDefaultMgmtNode();
        serviceManager = ServiceManagerFactory.getServiceManager(psc, mgmtNodeId);
        vimService = serviceManager.getVimPortType();
        tagService = serviceManager.getVapiService(Tag.class);
        tagAssociationService = serviceManager.getVapiService(TagAssociation.class);
    }

    public static ManagedObjectReference findCluster(String clusterName)
            throws InvalidPropertyFaultMsg, com.vmware.vim25.RuntimeFaultFaultMsg,
NotFoundFaultMsg {
        return VimUtil.getCluster(serviceManager.getVimPortType(),
                serviceManager.getServiceContent(), clusterName);
    }

    public static void tagCluster(ManagedObjectReference clusterMoref, String tagName)
{
        for (String id : tagService.list()) {
            if (tagService.get(id).getName().equals(tagName)) {
                String tagId = tagService.get(id).getId();
                DynamicID clusterId =
                        new DynamicID(clusterMoref.getType(),
clusterMoref.getValue());
                tagAssociationService.attach(tagId, clusterId);
                return;
            }
        }
        System.out.println("Tag Not Found - " + tagName);
    }

    public static void main(String args[]) throws InvalidTokenException,
            MultipleManagementNodeException, com.vmware.vim25.RuntimeFaultFaultMsg,
            InvalidLocaleFaultMsg, InvalidLoginFaultMsg,
DatatypeConfigurationException,
            RuntimeFaultFaultMsg, InvalidPropertyFaultMsg, NotFoundFaultMsg {

        String lsUrl = "https://10.161.18.127/lookupservice/sdk";
        String ssoUsername = "Administrator@vsphere.local";
        String ssoPassword = "Admin!23";
        String clusterName = "vAPISDKCluster";
        String tagName = "IMAGEBACKUP";

        login(lsUrl, ssoUsername, ssoPassword);

        ManagedObjectReference clusterMoref = findCluster(clusterName);

        tagCluster(clusterMoref, tagName);


    }
```