Developing Remote Plug-ins with the vSphere Client SDK

Update 1
16 October 2018
VMware vSphere 6.7
vSphere Client SDK 6.7

vmware®

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

https://docs.vmware.com/

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

docfeedback@vmware.com

Contents

About This Book

Developing Remote Plug-ins with the vSphere Client SDK provides information about developing and deploying HTML-5 extensions to the vSphere Client user interface.

VMware provides many APIs and SDKs for different applications and goals. This documentation provides information about the extensibility framework of the vSphere Client for developers who are interested in extending the Web application with custom functionality.

Intended Audience

This information is intended for anyone who wants to extend the vSphere Client with custom functionality. Users typically are software developers who use HTML and JavaScript to create graphical user interface components that work with VMware vSphere$^{®}$.

VMware Technical Publications Glossary

VMware Technical Publications provides a glossary of terms that might be unfamiliar to you. For definitions of terms as they are used in VMware technical documentation, go to http://www.vmware.com/support/pubs.

Revision History

This book, *Developing Remote Plug-ins with the vSphere Client SDK*, is updated with each release of the product or when necessary.

This table provides the update history of *Developing Remote Plug-ins with the vSphere Client SDK*.

Revision	Description
12 FEB 2019	Minor updates and corrections.
	Expanded material about Deployment.
16 OCT 2018	Initial release.

About the vSphere Client SDK

1

The VMware vSphere® Client provides a means for connecting to VMware vCenter Server® systems and managing the objects in the vSphere infrastructure. The VMware vSphere Client is an HTML5-based web application with a modular architecture that supports plug-in extensions. The vSphere Client SDK provides tools and examples that help you create custom plug-ins to extend the functionality of the vSphere Client.

This chapter includes the following topics:

- Knowledge Requirements for Using the vSphere Client SDK

- SDK Version Compatibility

- Overview of Remote Plug-in Architecture

- vSphere Client SDK Contents

- Where To Get the vSphere Client SDK

Knowledge Requirements for Using the vSphere Client SDK

Developing extensions for the vSphere Client by using the vSphere Client SDK, requires familiarity with HTML and JavaScript for the user interface components. The server components of remote plug-ins can be written using any technology you choose.

- You can extend the vSphere Client if you have a good understanding in Web application development by using JavaScript and HTML. You can use any user interface technology to create views for the vSphere Client user interface layer. The sample provided within the SDK uses Angular, TypeScript, and the Clarity Design System.

- You can use any coding language you choose for server components. A remote plug-in generally places these functional requirements on the server components:

  - A web server that provides HTML, JavaScript, graphic, and localization resources for the user interface.

  - A custom API that provides controller and model functionality to support the view component.

  - Business logic components that establish Web Services API sessions with vCenter Server instances, and retrieve data to satisfy requests from the view component. The Web Services API is language agnostic, but requesters often use Java, C#, or Python bindings.

SDK Version Compatibility

Remote plug-ins are supported by vSphere 6.7 Update 1 and later versions. Earlier versions of vCenter Server are incompatible with remote plug-ins.

Overview of Remote Plug-in Architecture

This diagram shows a simplified view of how remote plug-ins fit into the vSphere Client.

Figure 1-1. A Simplified View of Remote Plug-in Architecture

The circled numbers identify the following data paths:

1. The remote plug-in installer registers the plug-in manifest file with the vCenter Server Extension Manager, by using the Web Services API.

2. A web browser downloads user interface elements of the vSphere Client from the vsphere-ui service in vCenter Server.

3. The vsphere-ui service downloads and parses the plug-in manifest file to determine where the plug-in extends the user interface.

4. The browser downloads user interface elements of the plug-in from the plug-in server.

5. The plug-in user interface requests data from the plug-in server.

6. The plug-in server uses the Web Services API to interact with vCenter Server.

For a more detailed look at the architecture of remote plug-ins and their environment, see Chapter 3 Remote Plug-in Architecture in the vSphere Client.

vSphere Client SDK Contents

The vSphere Client SDK contains the following directories to aid developers who create plug-ins.

docs
- Detailed instructions for setting up IDEs.
- Documentation for JavaScript API used by UI components of plug-ins.
- FAQ with troubleshooting and development advice for plug-in developers.

libs   Run-time libraries for Spring framework and vSphere API RPC. This directory is not relevant for remote plug-ins.

samples   A complete remote plug-in sample that demonstrates both client-side and server-side modules, as well as accompanying metadata.

tools   Scripts to assist with development tasks.

vsphere-ui   A complete version of the vSphere Client, both client and server modules. This directory is not relevant for remote plug-ins.

Where To Get the vSphere Client SDK

You can download the vSphere Client SDK from the VMware {code} landing page. The landing page also has links to the latest documentation, including release notes.

See the latest landing page at https://code.vmware.com/sdk/client.

Using the vSphere Client Remote Plug-in Sample

<div style="text-align: right; font-size: 3em; color: #999;">2</div>

For an introduction to the vSphere Client Remote Plug-in SDK, you can install and run the remote plug-in sample. The remote plug-in sample demonstrates a secure, efficient, remote plug-in that was developed according to recommended practices. The sample illustrates several key features that you can adapt to develop your own plug-ins.

This chapter assumes that you have access to a vCenter Server instance and a development machine where you build and run the sample plug-in server.

This chapter includes the following topics:

- About the Remote Plug-in Sample
- Components in the Remote Plug-in Sample
- Build the vSphere Client Remote Plug-in Sample
- Find the SSL Thumbprint and GUID of vCenter Server
- Find the SSL Thumbprint of the Remote Plug-in Server
- Register the vSphere Client Remote Plug-in Sample
- Viewing the vSphere Client Remote Plug-in Sample
- Remote Plug-in Sample Directory Structure

About the Remote Plug-in Sample

The Remote Plug-in sample supports a Global View extension to the vSphere Client. The user interface component creates a modal dialog and several portlets that extend a vSphere inventory object view.

The user interface code also shows how to:

- Initialize the Client API.
- Retrieve session authentication information and pass it to the plug-in server.
- Perform a data retrieval request to the plug-in server.
- Define a context action for a VirtualMachine object.

The plug-in server code shows how to:

- Respond to a data retrieval or modification request.

- Clone and cache a user session with a vCenter Server.

The Remote Plug-in sample does the following actions:

- Creates a global view.

- Opens a modal dialog.

- Authenticates to vCenter Server

- Performs a data retrieval call.

- Creates a relation between a Chassis object and a HostSystem object.

- Creates several views that extend a context object view.

- Defines an action on a VirtualMachine context object.

Components in the Remote Plug-in Sample

The remote plug-in sample in the vSphere Client SDK shows how to design and implement, deploy and register a remote plug-in. The sample is functionally simple, to focus on displaying the infrastructure rather than the business logic.

The sample remote plug-in package contains several components:

- The user interface is written in JavaScript, using the Angular and Clarity libraries to maintain compatibility with the vSphere Client user interface.

- The sample plug-in server is written in Java, but Java is not a requirement. The server includes the following:

    - In-memory data storage for fictitious Chassis objects.

    - Controller logic to handle user interface requests for Chassis objects and vSphere HostSystem objects.

    - Service interfaces for operations on both kinds of objects.

    - A library layer to interface to vCenter Server, including logic to handle delegated authentication.

- The `plugin.json` file specifies the vSphere Client extension points that the plug-in extends.

- The `spring-context.xml` file contains the Spring bean definitions for the plug-in server.

- The `pom.xml` file specifies how Maven will install dependencies and build the plug-in deliverable.

- The `application.properties` file specifies properties that the Spring application server uses to deploy the plug-in server.

The following diagram illustrates the basic architecture of the remote plug-in sample, when installed in a simple vSphere environment.

Figure 2-1. Remote Plug-in Sample Environment
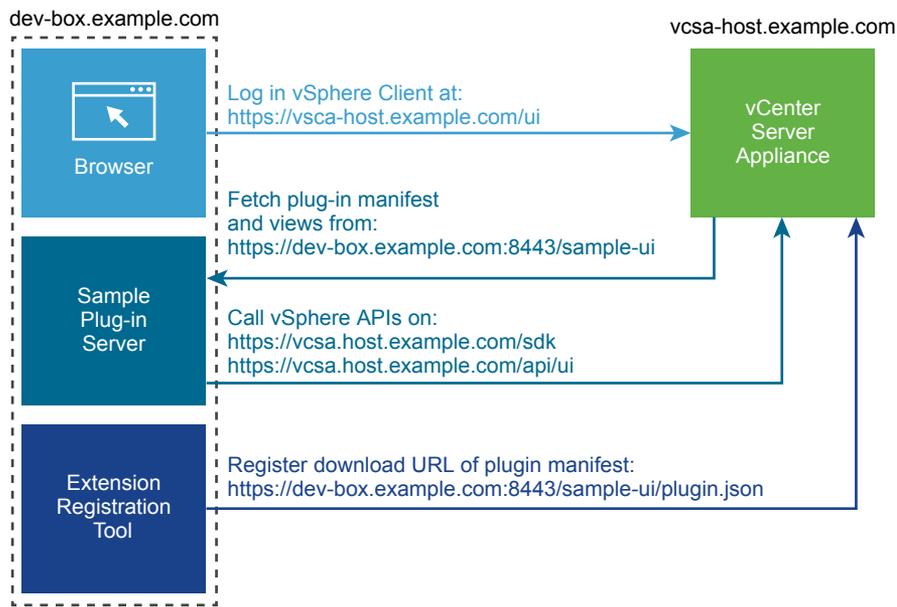
Build the vSphere Client Remote Plug-in Sample

You deploy the remote plug-in sample with a Maven build directed by the sample's `pom.xml` file. This procedure builds the sample and deploys it on the plug-in server.

After you download the vSphere Client SDK, you must build the remote plug-in sample before you can run it.

Prerequisites

Before registering the remote plug-in sample, you must do the following:

- You need Java 8 to compile the code for this sample.

- You need Maven 3 to build the plug-in package.

- You need to download and unzip the SDK. See Chapter 1 About the vSphere Client SDK for information about the structure of the SDK archive.

Procedure

1 Change to the root directory of the remote plug-in sample.

For example: `cd /HD/sdk/html-client-sdk/samples/remote-plugin-sample`

2 `mvn validate`

3 `mvn clean install`

These steps install the Web Services API library into the local Maven repository, download and build the sample executable, download the Clarity design system, and build the JAR file that contains the sample components. The sample is a Spring Boot Application that will start an embedded Tomcat server when you run the sample in a command shell.

What to do next

After you build the remote plug-in sample, you need to run the plug-in server and register the plug-in with vCenter Server.

Find the SSL Thumbprint and GUID of vCenter Server

Before you start the remote plug-in server, you need to find the thumbprint and the GUID of the vCenter Server where you want to register the plug-in.

You need to find out the certificate thumbprint and the GUID of a vCenter Server instance.

Prerequisites

vCenter Server must be running while you do this procedure.

Procedure

1   Connect a browser to the vCenter Server.

    The URL for vCenter Server looks similar to this: `https://vcenter-1.example.com`

    The browser displays a launch screen, with a small padlock icon in the address field.

2   Click **LAUNCH VSPHERE CLIENT (HTML5)** and log in to the vSphere Client.

    The browser displays the default Hosts and Clusters view.

3   If you connected to a vCenter Server instance in an extended linked mode environment, you must select the chosen vCenter Server instance in the navigation pane on the left of the vSphere Client screen.

    The URL in the browser address box contains an embedded managed object reference, similar to the following:

    `Folder:group-d1:56d373bd-4163-44f9-a872-9adabb008ca9`. This is an extended managed object reference that ends with the GUID of the vCenter Server instance. The GUID is a string of 32 hexadecimal digits, organized in groups of 4, 8, or 12 digits, separated by hyphens.

4   Copy the 32 hexadecimal digits of the GUID, along with the inset hyphens, and save this into a shell variable or a text file.

    You will use the GUID when you start the plug-in server.

5   Click the padlock icon in the browser address field to access a certificate information window.

    The browser displays a brief summary of browser properties.

6   Click **Details** to display more certificate information.

    The browser displays full details of the vCenter Server certificate.

7   Scroll through the certificate details to find the SHA1 fingerprint.

    The SHA1 fingerprint is a string of 40 hexadecimal digits, usually in pairs separated by spaces or other non-alphanumeric delimiters.

8   Select the SHA1 fingerprint and copy it to a text file.

9   Edit the text file to remove all spaces or other delimiters from the SHA1 fingerprint.

    You now have a string of 40 contiguous hexadecimal digits. This is the *thumbprint* of the vCenter
    Server instance.

What to do next

The thumbprint and GUID of the vCenter Server instance are needed to start the remote plug-in server.
You can start the server to determine its certificate thumbprint.

Find the SSL Thumbprint of the Remote Plug-in Server

The remote plug-in sample has an embedded application server with a self-signed certificate that is used
in encrypted communications. The certificate and its thumbprint are stored in a Java keystore file.

To register a remote plug-in with vCenter Server, you need to determine the thumbprint of the plug-in
application server's identity certificate. You use this thumbprint in the arguments to the registration
command.

Prerequisites

Before you can find the thumbprint of the application server, you must install the SDK and build the
sample code.

Procedure

1   In a shell window, change to the root directory of the remote plug-in sample and run the JAR file in
    the `target` directory.

    The command to run the plug-in JAR file requires several arguments, including the thumbprint, GUID,
    DNS name, and HTTPS port number of the vCenter Server instance. Use a command similar to the
    following example, but substitute the details that pertain to your vCenter Server:

    ```
    java    -jar target/remote-plugin-sample-6.7.0.jar \
      --vcenter.guid=223b94f2-af15-4613-5d1a-a278b19abc09 \
      --vcenter.thumbprint=3fccb81448ca98bbbf2b75dbb3ab199886accb1f \
      --vcenter.fqdn=vcenter-1.example.com --vcenter.port=443
    ```

    The plug-in application server runs. It might take a few minutes to initialize, and the console displays
    a number of lines of output. When the server is ready, the console displays two lines saying `Tomcat`
    `started` and `Started SpringbootApplication`.

2   Connect a browser to the application server, for example, using the URL of the plug-in manifest.

    The default URL for the manifest file is `https://localhost:8443/sample-ui/plugin.json`.

3   Examine the certificate presented by the application server.

    The way to examine the certificate depends on the browser. For example, you can view a server
    certificate in Firefox by clicking the padlock icon next to the URL, then selecting **More Information >**
    View Certificate. The thumbprint is the field labelled `SHA1 Fingerprint`.

4 Save the certificate thumbprint to a text file.

If the thumbprint contains colon separators, do not remove them.

What to do next

Use the application server thumbprint when you register the plug-in with vCenter Server.

Register the vSphere Client Remote Plug-in Sample

Before you run the remote plug-in sample, you must register it with a vCenter Server instance to which you want to connect. The vSphere Client SDK contains a vCenter Server plug-in registration tool.

You have installed the SDK and you are ready to run the remote plug-in sample.

Prerequisites

Before registering the remote plug-in sample, you must do the following:

- Build the remote plug-in sample.

- If needed, start vCenter Server.

- If needed, change permissions on the plug-in registration tool to allow execute access.

In addition, you need to know the following parameters:

- The host name or IP address of the machine where you are running the sample plug-in server. This address must be accessible to the vCenter Server instance so that it can download the plug-in manifest file.

- The port number where the plug-in server receives HTTPS requests. The sample serves port 8443 by default. The port must be open on the firewall of your development machine and accessible to the vCenter Server.

- The host name or IP address of the vCenter Server where you want to register the remote plug-in sample.

- The username and password of a vSphere user that has permission to access the ExtensionManager on the vCenter Server where you want to register the remote plug-in sample. For example, `administrator@vsphere.local` normally has the necessary `Extension.Register` permission.

- The SHA1 thumbprint of the plug-in server, so that vCenter Server can retrieve the plug-in manifest file.

  Note   The thumbprint must contain colon separators, unlike the thumbprint format used when starting the plug-in server. vCenter Server accepts only thumbprints that have a colon between each pair of characters.

- The version number of the remote plug-in sample.

- The key of the remote plug-in sample, defined in the plug-in manifest. By default, this is `com.vmware.sample.remote`.

- The path from the plug-in web server root to the plug-in manifest file. By default, this is `/sample-ui/plugin.json`.

Procedure

1   In a command shell, change to the `tools/vCenter plugin registration/prebuilt` directory.

    ```
    cd html-client-sdk/tools/*plugin*/prebuilt
    ```

    The directory contains both an `extension-registration.bat` script for Windows DOS shells, and an `extension-registration.sh` script for Unix or Linux shells.

2   Run the extension-registration script appropriate for your operating system, specifying the prerequisite parameters.

    For a Unix or Linux shell, use this syntax:

    ```
    ./extension-registration.sh    -action registerPlugin -remote \
      -url https://myvcenter/sdk \
      -username    administrator@vsphere.local -password mysecret \
      -key com.vmware.sample.remote -version 1.0.0 \
      -pluginUrl https://mydevbox:8443/sample-ui/plugin.json \
      -serverThumbprint 19:FD:2B:0E:62:5E:0E:10:FF:24:34:7A:81:F1:D5:33:19:A7:22:A0 \
      -c 'Example, Inc.' -n 'Remote Plug-in' -s 'This is a sample plug-in'
    ```

    For a DOS command shell, use this syntax:

    ```
    ./extension-registration.bat    -action registerPlugin -remote ^
      -url https://myvcenter/sdk ^
      -username    administrator@vsphere.local -password mysecret ^
      -key com.vmware.sample.remote -version 1.0.0 ^
      -pluginUrl https://mydevbox:8443/sample-ui/plugin.json ^
      -serverThumbprint 19:FD:2B:0E:62:5E:0E:10:FF:24:34:7A:81:F1:D5:33:19:A7:22:A0 ^
      -c 'Example, Inc.' -n 'Remote Plug-in' -s 'This is a sample plug-in'
    ```

    For a PowerShell prompt, use this syntax:

    ```
    ./extension-registration.bat    -action registerPlugin -remote `
      -url https://myvcenter/sdk `
      -username    administrator@vsphere.local -password mysecret `
      -key com.vmware.sample.remote -version 1.0.0 `
      -pluginUrl https://mydevbox:8443/sample-ui/plugin.json `
      -serverThumbprint 19:FD:2B:0E:62:5E:0E:10:FF:24:34:7A:81:F1:D5:33:19:A7:22:A0 `
      -c 'Example, Inc.' -n 'Remote Plug-in' -s 'This is a sample plug-in'
    ```

    Note   If the password contains special characters, use the appropriate escaping sequences for your shell.

    The registration script displays a message that the plug-in `has been successfully registered in vCenter`.

What to do next

In a web browser, connect to the vCenter Server URL and verify that the remote plug-in displays a Global View.

Viewing the vSphere Client Remote Plug-in Sample

After you build and register the remote plug-in sample, you can view it in the vSphere Client.

To view the remote plug-in sample, you must run the plug-in server. If the server is not running yet, change to the root directory of the remote plug-in sample and use a command similar to the following example, but substitute the details that pertain to your vCenter Server:

```
java    -jar target/remote-plugin-sample-6.7.0.jar \
  --vcenter.guid=223b94f2-af15-4613-5d1a-a278b19abc09 \
  --vcenter.thumbprint=3fccb81448ca98bbbf2b75dbb3ab199886accb1f \
  --vcenter.fqdn=vcenter-1.example.com --vcenter.port=443
```

To view the plug-in user interface, open a browser window and connect to the vCenter Server where you registered the plug-in. From the welcome screen you can launch the vSphere Client.

Figure 2-2. vCenter Server Welcome Screen
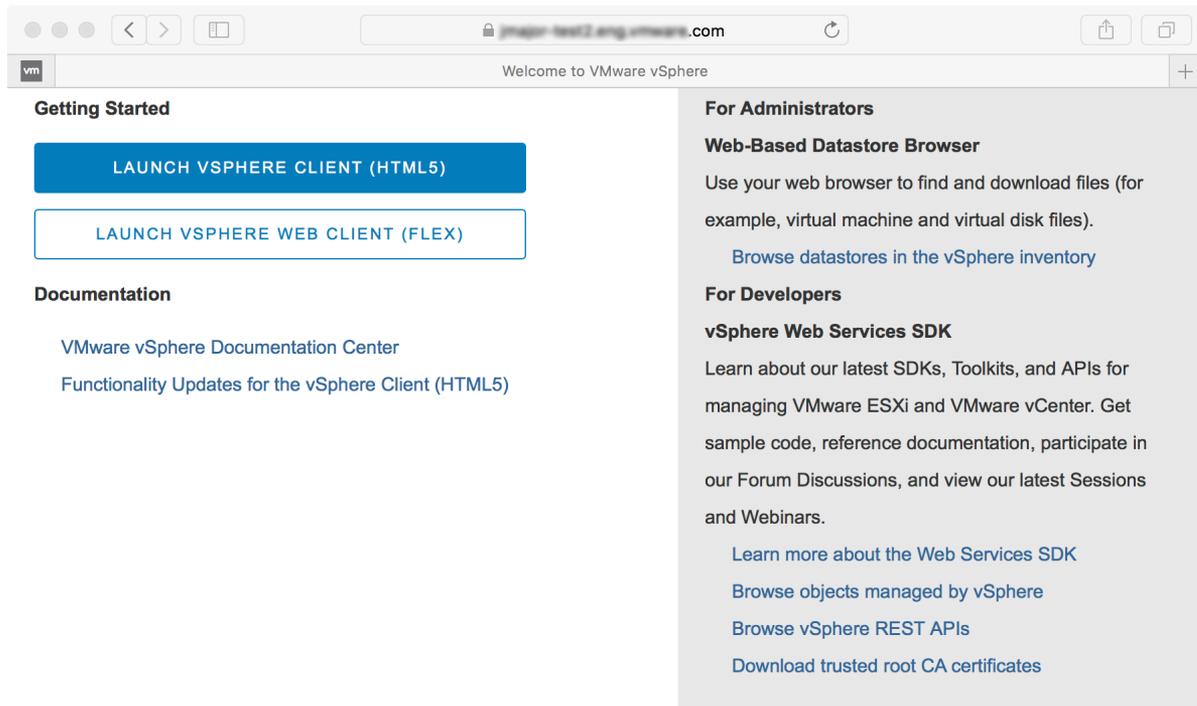
After you select **LAUNCH VSPHERE CLIENT (HTML5)** and authenticate with vCenter Server, the vSphere Client displays the home page with a link to the remote plug-in global view in the navigation column on the left.

The plug-in global view includes the following subviews:

- A welcome page is the default page for the global view.

- A settings page allows you to change the number of items displayed in the Chassis List page.

- A Chassis List page displays summary information about the Chassis objects currently in the store. The Chassis store is initialized with several random Chassis objects for display. On the Chassis List page you can do the following actions:

  - Edit the Chassis object properties in a modal dialog.

  - Display related Host objects.

  - Display the Monitor subview or the Configure subview.

The code that supports these actions demonstrates how to use a modal dialog, how to create, delete, and update Chassis objects, and how to make calls to the plug-in server and the vCenter Server. The following illustrations show some of the features of the global view.

Figure 2-3. Selecting the Sample Plug-in Global View in the Navigation Pane
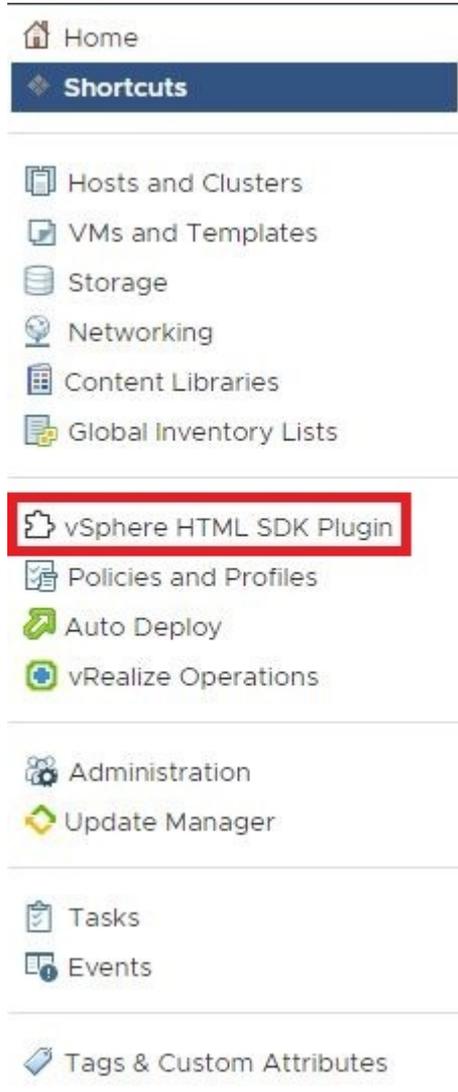
The Global View extension point is specified by the following lines from the manifest file, `plugin.json`:

```
"global": {
   "view": {
      "navigationId": "com.vmware.remote.sample.entrypoint",
      "uri": "index.html?view=entry—point",
      "navigationVisible": false
   }
}
```

The Global View content is specified in the file `entry.point.component.html`.

Figure 2-4. Sample Plug-in Global View Welcome Page

Figure 2-5. Sample Plug-in Global View Chassis List Page
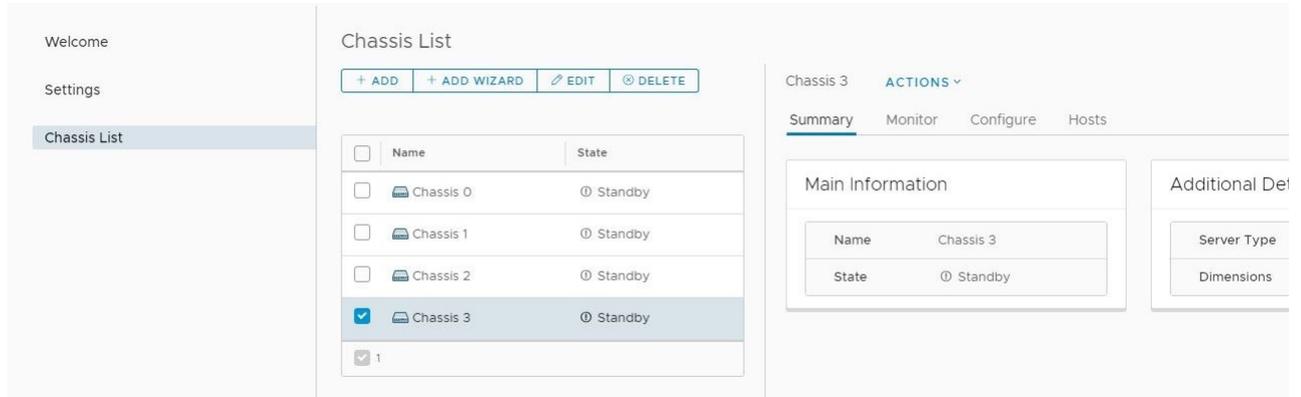
Figure 2-6. Sample Plug-in Edit Chassis Dialog
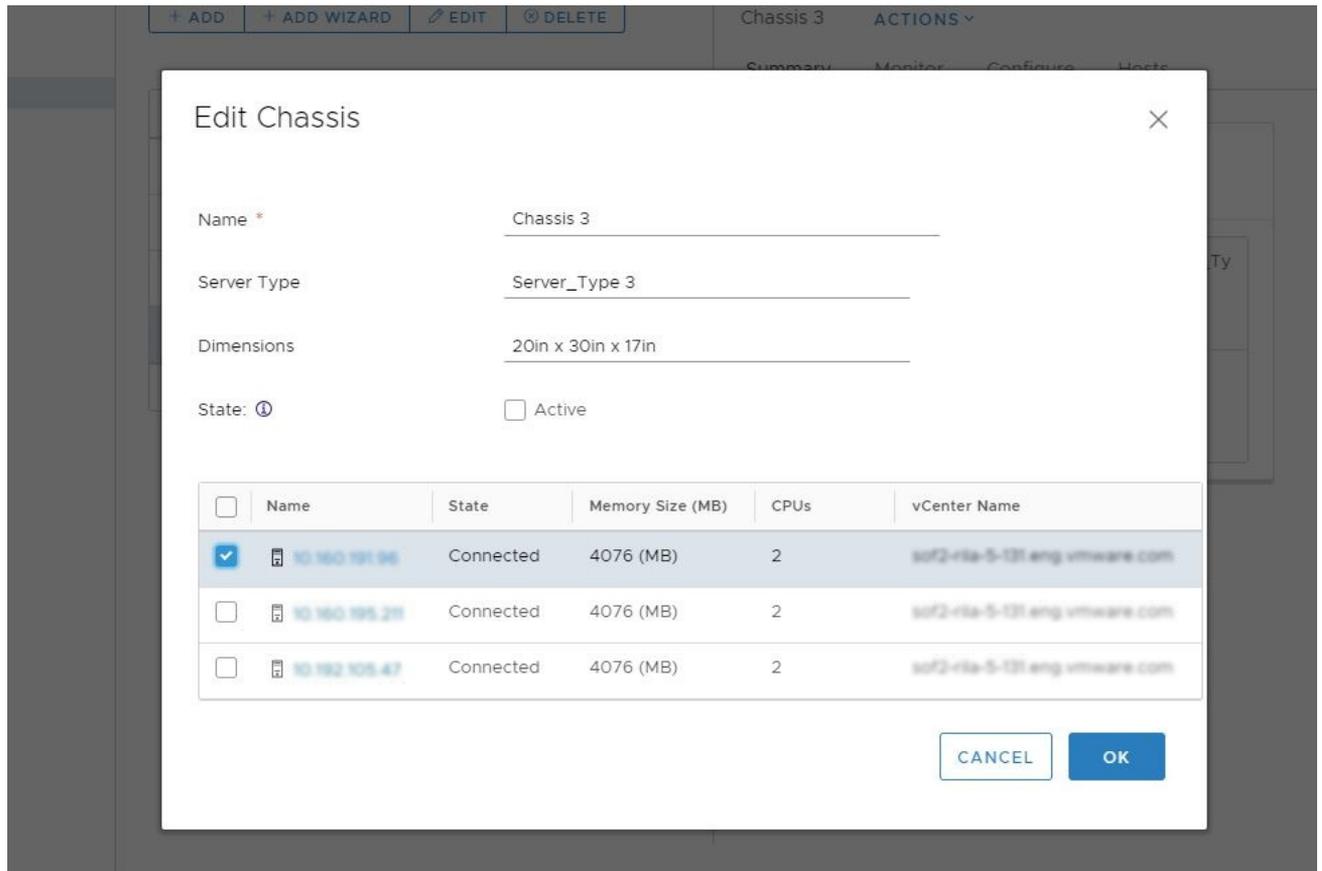
Figure 2-7. Sample Plug-in Chassis with Related Hosts
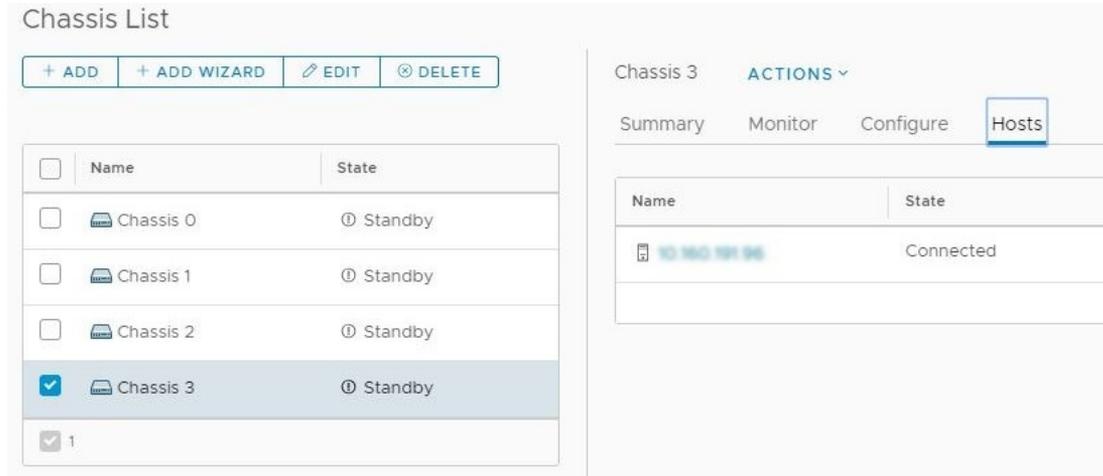
In addition to the global view and its subviews, the remote plug-in sample provides portlets that show how to extend the VirtualMachine vSphere object. You can see the portlets in context in the following illustrations.

Figure 2-8. Portlet Extending the VirtualMachine Summary View
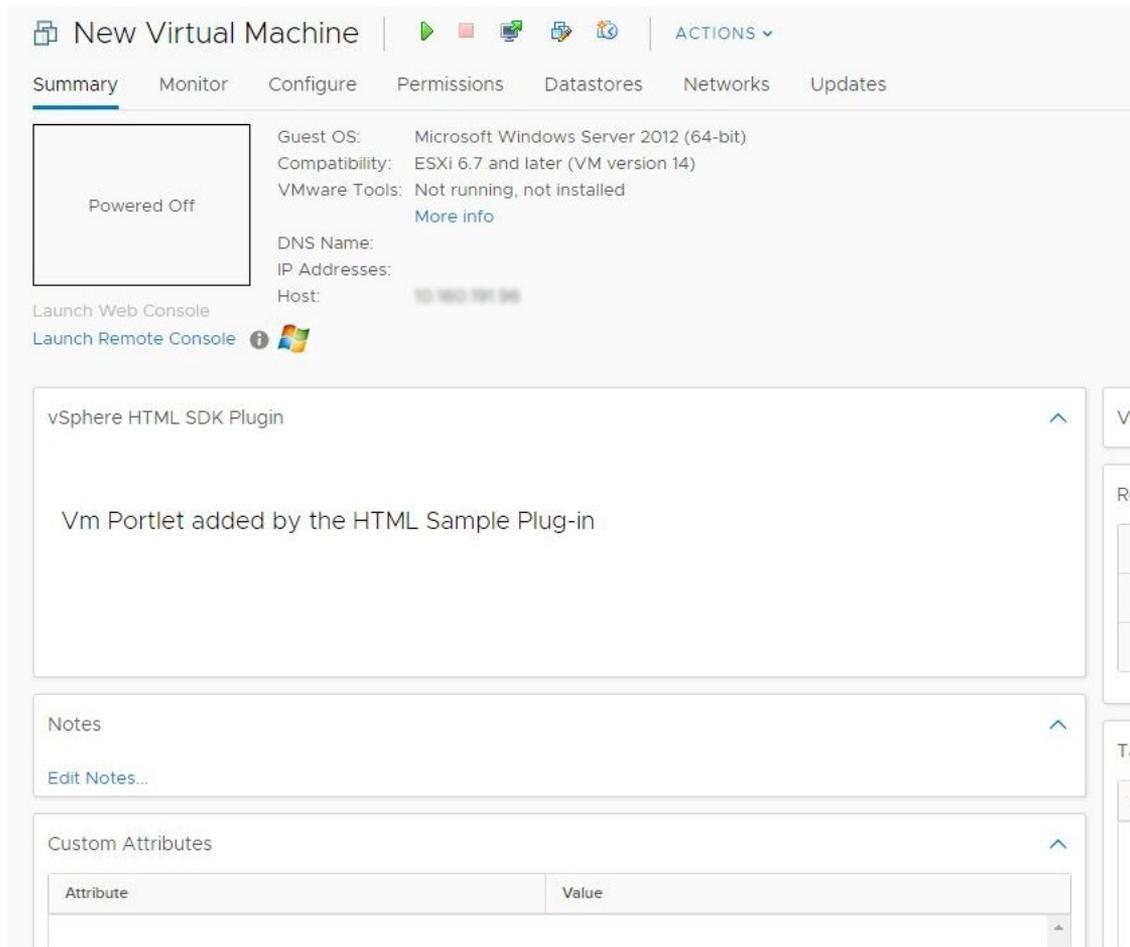
The VirtualMachine Summary View extension point is specified by the following lines from the manifest file, `plugin.json`:

```
"objects": {
    "VirtualMachine": {
        "summary": {
            "view": {
                "uri": "index.html?view=vm-portlet",
            }
        },
        ...
    }
}
```

Figure 2-9. Portlet Extending the VirtualMachine Monitor View

The VirtualMachine Monitor View extension point is specified by the following lines from the manifest file, `plugin.json`:

```
"objects": {
    "VirtualMachine": {
        ...
        "monitor": {
            "views": {
                "navigationId": "com.vmware.remote.sample.monitor",
                "labelKey": "RemoteSample:vm.monitor.view.title",
                "uri": "index.html?view=vm-monitor",
            }
        }
    }
}
```

Figure 2-10. Portlet Extending the VirtualMachine Configure View

The VirtualMachine Configure View extension point is specified by the following lines from the manifest file, `plugin.json`:

```
"objects": {
    "VirtualMachine": {
        "configure": {
            "views": {
                "navigationId": "com.vmware.remote.sample.configure",
                "labelKey": "RemoteSample:vm.configure.view.title",
                "uri": "index.html?view=vm-congfigure",
            }
        }
    }
}
```

The remote plug-in sample shows how to add an action item to a context menu for a VirtualMachine object. You can see how this appears in the following illustration.

Figure 2-11. Action Added to VirtualMachine Context Menu
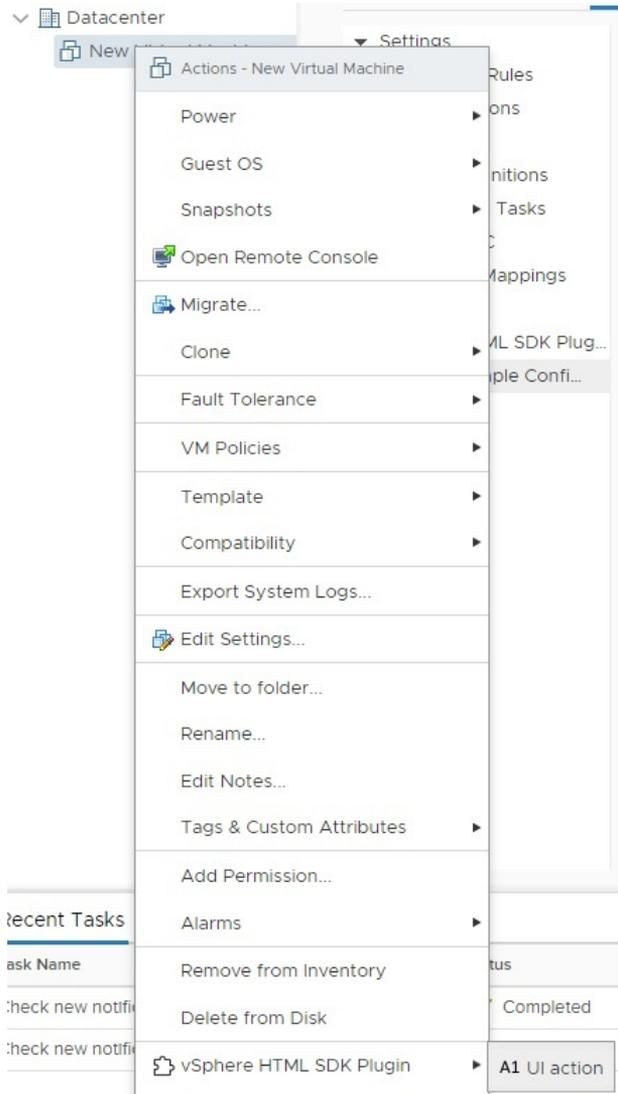
The VirtualMachine context menu extension point is specified by the following lines from the manifest file, `plugin.json`:

```
"objects": {
    "VirtualMachine": {
        "menu": {
            "actions": {
                "labelKey": "RemoteSample:vm.action.label",
                "icon": {
                    "name": "action-1"
                },
                "trigger": {
                    "type"; "modal",
                    "uri": "index.html?view=vm-action-modal",
                    "titleKey": "RemoteSample:vm.action.model.title",
                    "size": {
                        "width": 600,
                        "height": 250
```

```
                    }
                }
            }
        }
    }
}
```

The remote plug-in sample displays a HostSystem Summary view and a HostSystem Monitor view also. The Summary view shows the number of related Chassis objects. You click the number to reach the Monitor view. The Monitor view displays a datagrid listing all available Chassis objects. To create a relation between a HostSystem and a Chassis object, select the checkbox beside the Chassis object and click **Update**.
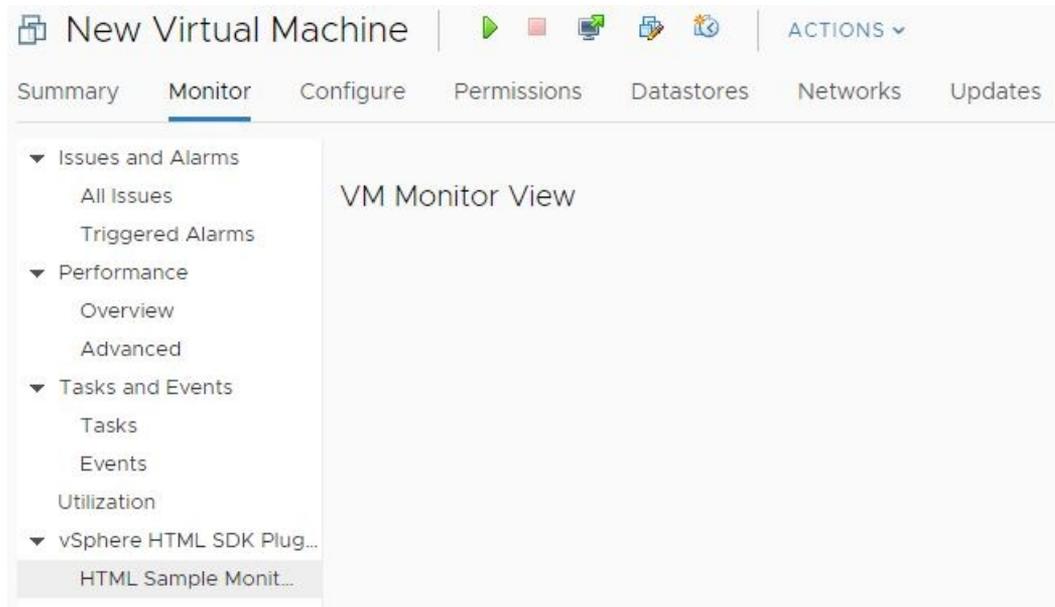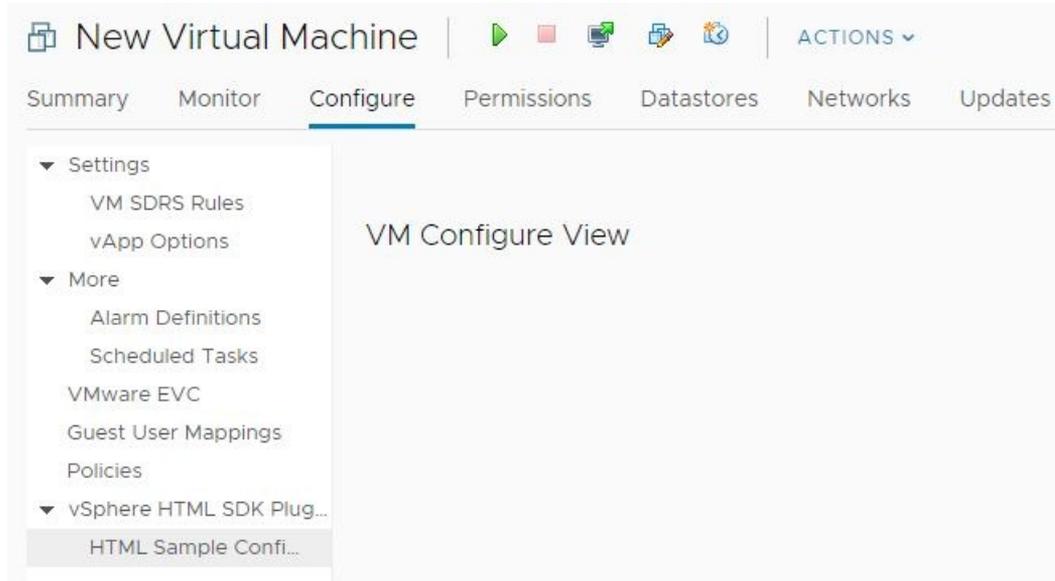
Figure 2-12. Portlet Extending the HostSystem Summary View

The HostSystem Summary View extension point is specified by the following lines from the manifest file, `plugin.json`:

```
"objects": {
    "HostSystem": {
        "summary": {
            "view": {
                "uri": "index.html?view=host-summary"
            }
        }
    }
}
```

Figure 2-13. Portlet Extending the HostSystem Monitor View
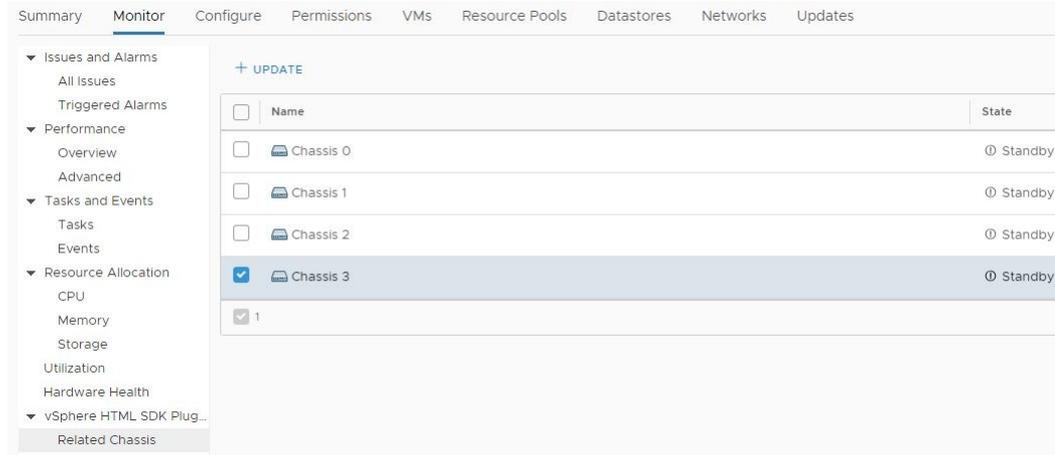
The HostSystem Monitor View extension point is specified by the following lines from the manifest file, `plugin.json`:

```
"objects": {
    "HostSystem": {
        "monitor": {
            "views": [
                {
                    "navigationId": "hostMonitor",
                    "labelKey": "host.monitor.view.title",
                    "uri": "index.html?view=host-monitor"
                }
            ]
        }
    }
}
```

Remote Plug-in Sample Directory Structure

The following illustrations show the directory structure of the remote plug-in sample code.

Top Level Directory

Figure 2-14. Remote Plug-in Sample Top Level Directory
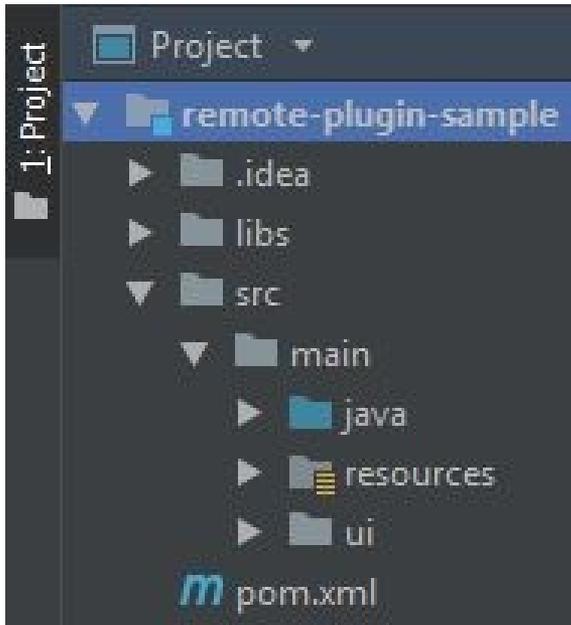
Table 2-1. Contents of Top Level Directory

Subdirectory	Description
remote-plugin-sample/libs	The external library (vim25.jar) that needs to be installed as a local dependency during the project build and deploy process.
remote-plugin-sample/src	The root of the plug-in source code. It contains three main subdirectories: ■ `java` ■ `resources` ■ `ui` Descriptions of these subdirectories follow.

`java` Directory for Server-Side Code

The server-side code for the remote plug-in sample is written in Java. It contains several packages.

Figure 2-15. Remote Plug-in Sample Server-Side Directory
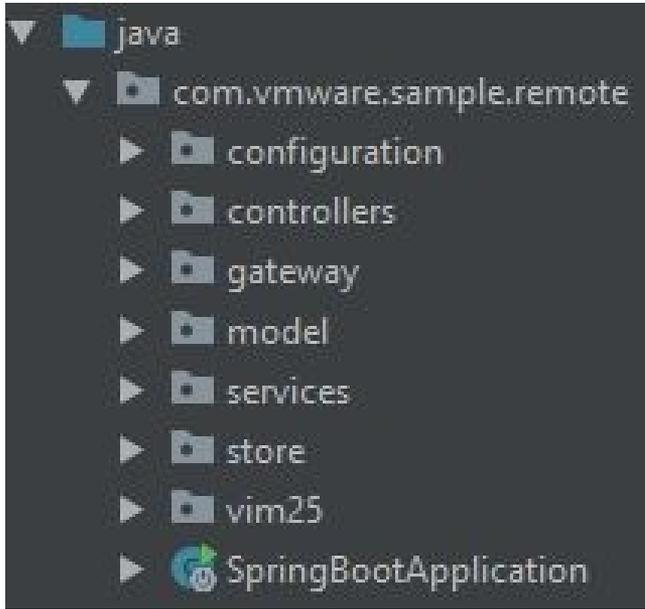
Table 2-2. Contents of Server-Side Directory

Subdirectory	Description
configuration	The Configuration service reads connection properties of vCenter Server, such as FQDN, thumbprint, port, and GUID, and exposes them to Spring Boot components through public accessors.
controllers	The Controller components have the logic to keep open user sessions to the vCenter Server.
gateway	The Session Service interacts with vCenter Server on behalf of the plug-in server. The Session service authenticates the plug-in server with a clone ticket.
model	Object definitions for plug-in server use.
services	Services that implement the plug-in server-side business logic.
store	In-memory database to store custom data for the plug-in.
vim25	Components that communicate with vCenter Server: ■ VimObjectService retrieves data from the vCenter Server, using the Web Services API. ■ ThumbprintTrustManager helps to authenticate the remote side of a secure socket, using the public interface javax.net.ssl.X509TrustManager interface.
SpringBootApplication.java	The Spring Boot Application bootstraps, deploys, and runs the plug-in on an embedded Tomcat application server.

resources Directory

The resources directory of the remote plug-in sample contains application configuration files and a certificate file for encrypted communications.

Figure 2-16. Remote Plug-in Sample Resources Directory
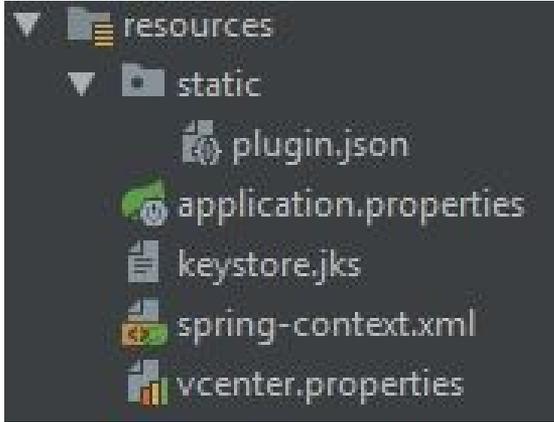
Table 2-3. Contents of Resources Directory

Subdirectory	Description
application.properties	Settings specific to the plug-in server, including port number for incoming HTTPS connections, context root of the web application, log level setting, and settings pertaining to the keystore.
keystore.jks	Java keystore for self-signed certificate of vCenter Server.
spring-context.xml	Context configuration for plug-in server Spring application.
static/plugin.json	Descriptors for UI components the plug-in adds to the vSphere Client.

`ui/src` **Directory**

The `ui/src` directory of the remote plug-in sample contains the client-side code for the plug-in. The main source files are in the `app` subdirectory, which contains an Angular application, including the AppModule, services, and component files.

Figure 2-17. Remote Plug-in Sample ui/src Directory

.
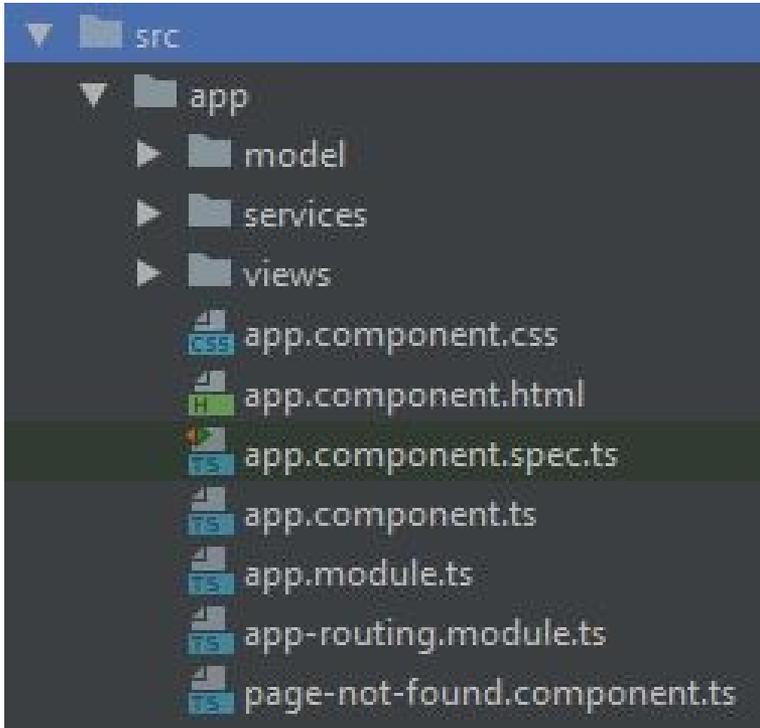
Table 2-4. Contents of ui/src Directory

Subdirectory	Description
app/model	Object definitions for plug-in client-side use.
app/services	Services that implement the plug-in client-side business logic
app/views	UI views (Angular components) that implement the visible functionality of the plug-in. Each view has its own subdirectory that contains HTML, CSS, and Angular code in separate files.
app/app.component	The main component that bootstraps the Angular Application and initializes the `htmlClientSdk` service.
app/app-routing.module	The routing module of the plug-in client-side application.
index.html	Static HTML page that loads by default when the plug-in server context path is loaded. The `index.html` page loads `htmlClientSkd.js` and `app.component`, which is the entry point of the Angular application.

Remote Plug-in Architecture in the vSphere Client

<div style="text-align: right">**3**</div>

The vSphere Client remote plug-in architecture is designed to integrate plug-in functionality into the vSphere Client without the need to run inside vCenter Server. This provides plug-in isolation and enables scale-out of plug-ins that operate in large vSphere environments.

The remote plug-in architecture provides the following benefits to plug-in developers:

- Your plug-in is protected from interference by unstable or compromised plug-ins loaded in the same vSphere Client.

- Plug-in compatibility is robust across vCenter Server upgrades.

- An incompatible plug-in does not interfere with vCenter Server operation.

- You can deploy a number of plug-in versions within the same vSphere environment.

- Your remote plug-in user interface needs to communicate with only a single back-end server.

- The topology of deployed plug-ins is well defined and easy to understand. This aids in the process of troubleshooting a problem with your plug-in.

This chapter includes the following topics:

- Components of the vSphere Client Architecture

- vCenter Server Configurations

- Communication Paths in Remote Plug-in Architecture

- Communications Among UI Components in the vSphere Client

- Client-Server Communications with Remote Plug-ins

- Security Concepts for Remote Plug-ins

Components of the vSphere Client Architecture

The vSphere Client architecture enables administrators to manage vSphere environments of varying scale and complexity with a single user interface. It supports environments ranging from a single vCenter Server with an embedded Platform Services Controller to a number of vCenter Server instances in Enhanced Linked Mode or Hybrid Linked Mode.

The user interface component runs in a browser and manages the HTML5 views presented to the user. The user interface communicates with the vsphere-ui service, requesting HTML and Javascript files and vSphere inventory data, and authenticating as needed. The user interface also manages a sandbox for each active plug-in, and provides client library services to the plug-in user interface.

The vsphere-ui service is an OSGi Java application server that runs on every vCenter Server node. The vsphere-ui service communicates with all of the services provided by vCenter Server and the Platform Services Controller, by using a variety of API styles and protocols. The vsphere-ui service maintains an authenticated session connection as a client of each of the services.

The vsphere-ui service also provides REST and Web Sockets APIs to the vSphere Client user interface running in the browser. The service supports authentication for users of the vSphere Client by redirecting the browser to a login user interface provided by the VMware vCenter Single Sign-on service running in the Platform Services Controller.

vCenter Server Configurations

Remote plug-ins can operate in linked mode configurations of vCenter Server as well as in single vCenter Server instances. These illustrations show the components and communication paths of both kinds of configuration.

The following illustration shows a single vCenter Server with an embedded Platform Services Controller.

Figure 3-1. vCenter Server with Embedded PSC
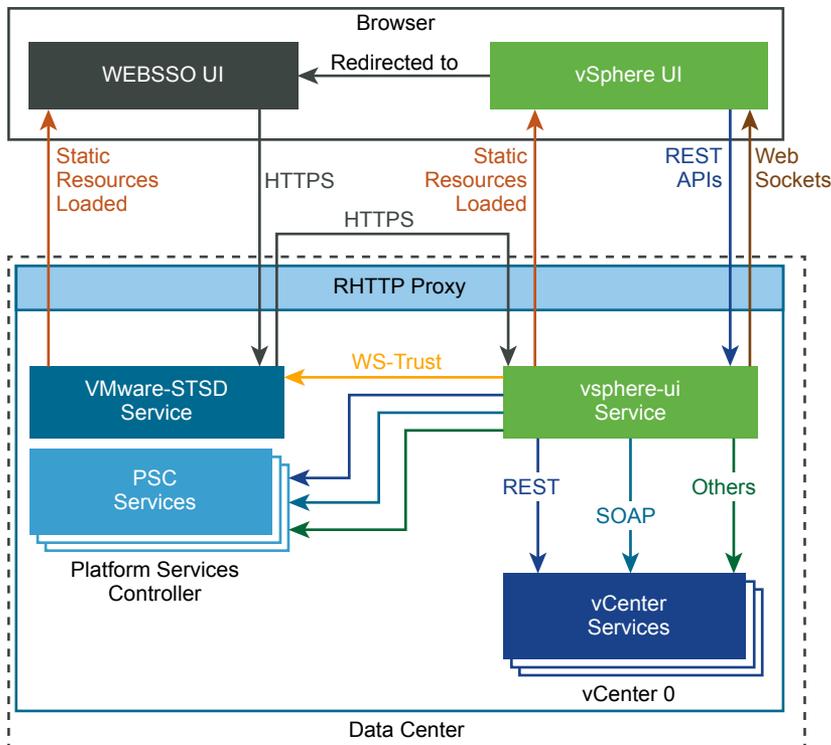
The following diagram shows three vCenter Server instances configured in Enhanced Linked Mode. The Platform Services Controller for this group is external to the vCenter Server instances.

Figure 3-2. vCenter Server Instances Configured in Extended Linked Mode
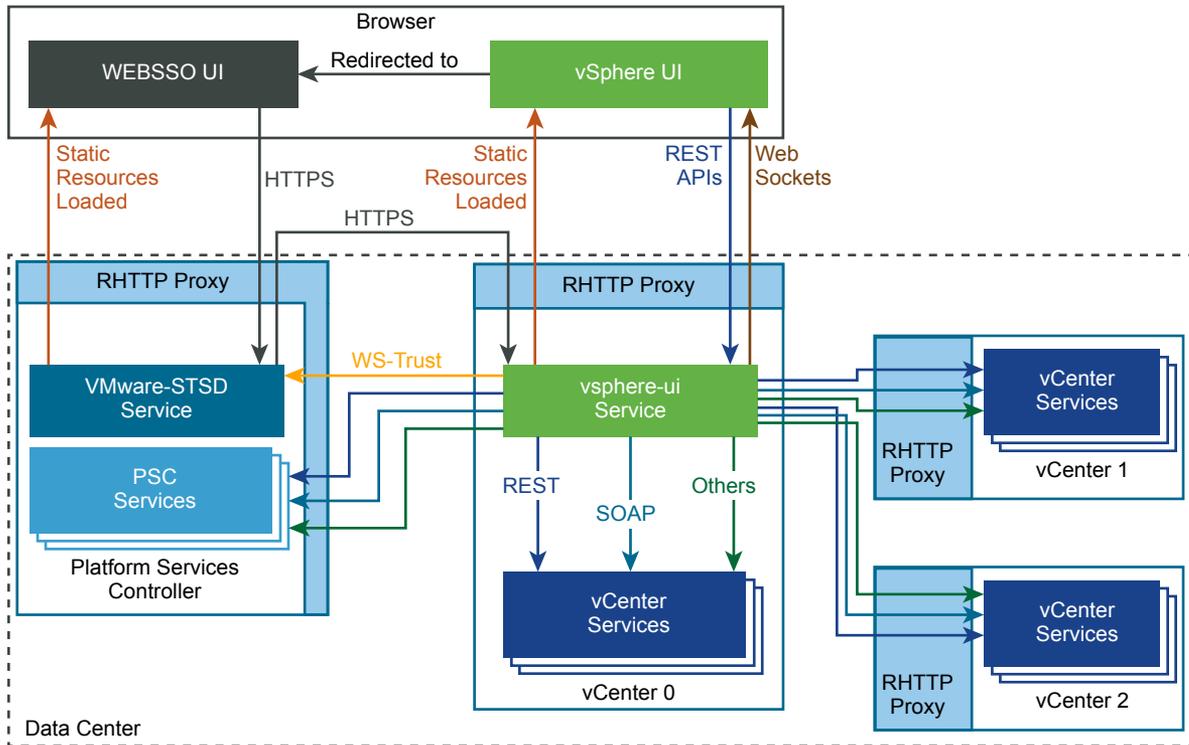
In a linked mode configuration, the vsphere-ui service handles delegation of requests to the linked vCenter Server instances, using the same protocols that it uses when communicating with its own services. From the point of view of the user interface, communications are as simple as in the embedded configuration.

Communication Paths in Remote Plug-in Architecture

This diagram shows some of the communication paths between a plug-in, its user interface, and the vCenter Server to which the user interface is connected.

On the front end, the vsphere-ui service downloads and parses the plug-in manifest and serves UI components to the browser, including references to plug-in components, which are served by the plug-in server. These paths use simple HTTPS communications.

The back end communication paths between the plug-in server and the vCenter Server Web Services API use SOAP messages over HTTPS. These communications are described in more detail in Chapter 8 Remote Plug-in Server Considerations for the vSphere Client.

Figure 3-3. A Simplified View of Remote Plug-in Architecture

The circled numbers identify the following data paths:

1. The remote plug-in installer registers the plug-in manifest file with the vCenter Server Extension Manager, by using the Web Services API.

2. A web browser downloads user interface elements of the vSphere Client from the vsphere-ui service in vCenter Server.

3. The vsphere-ui service downloads and parses the plug-in manifest file to determine where the plug-in extends the user interface.

4. The browser downloads user interface elements of the plug-in from the plug-in server.

5. The plug-in user interface requests data from the plug-in server.

6. The plug-in server uses the Web Services API to interact with vCenter Server.

Communications Among UI Components in the vSphere Client

The vSphere Client user interface loads both its own components and the components belonging to the plug-in user interface. The vsphere-ui service reads the plug-in manifest file to determine where it should insert plug-in components in the user interface.

The user interface components loaded in the browser are organized as shown in the following diagram.

Figure 3-4. User Interface with a Sandboxed Plug-in
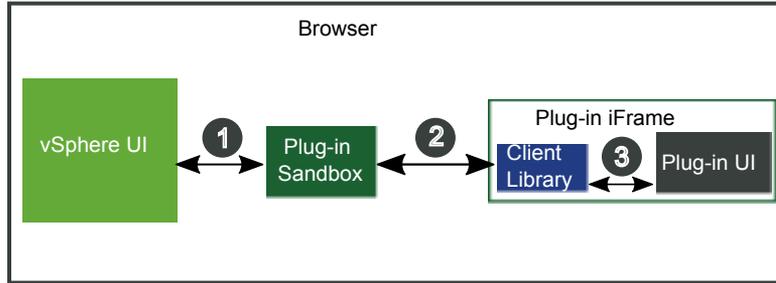
Paths:

1   Internal JavaScript methods

2   `window.postMessage()` method in browser

3   Public JavaScript methods

The plug-in user interface operates within its own IFrame, isolated from other plug-ins. The plug-in loads a copy of the vSphere Client JavaScript API client library, which is its sole connection to other client code. The plug-in code communicates with the client library code using JavaScript method calls.

The client library communicates with the sandbox component that the vSphere Client provides to interface with the plug-in UI components. The communication with the sandbox uses the browser's `window.postMessage()` API. This makes it possible for the IFrame and its parent window to be loaded from different origins.

Note   In the vSphere 6.7 Update 1 release of the vSphere Client, the IFrame and its parent window share the same origin. Do not depend on this to remain the same in future releases.

The plug-in sandbox communicates with other components of the vSphere Client user interface by using internal Javascript APIs.

If the vSphere Client has more than one plug-in active, each plug-in is allocated its own sandbox, and operates within its own IFrame, as shown in the following illustration.

Figure 3-5. User Interface with Two Plug-ins in Separate Sandboxes
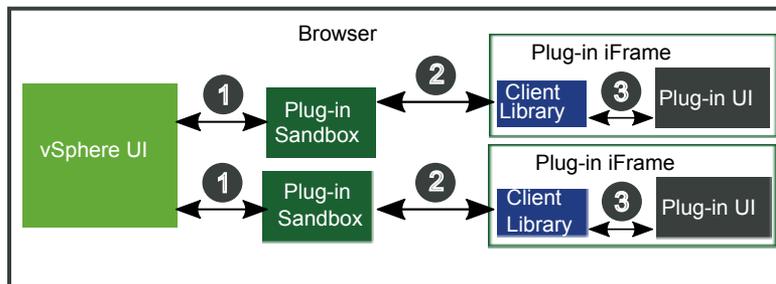
Paths:

1   Internal JavaScript methods

2   `window.postMessage()` method in browser

3   Public JavaScript methods

In this case, each plug-in UI communicates with its own back-end server.

Client-Server Communications with Remote Plug-ins

Client requests to back-end services are handled in a similar way for plug-in user interface components and vSphere Client user interface components.

A plug-in sandbox runs outside the plug-in iframe. The sandbox component, along with the rest of the vSphere Client user interface components, sends requests to the vsphere-ui service of the vCenter Server instance that the browser connected to. Service requests use REST and Web sockets over HTTPS. All requests pass through the vCenter Server reverse proxy, which routes them to the correct server components.

A plug-in user interface sends service requests to the plug-in back end, using any RPC style on top of the HTTPS transport protocol. All requests pass through the vCenter Server reverse proxy, which routes them to the plug-in server. The proxy routing is configured at the time the plug-in is installed; it simplifies dealing with self-signed certificates, and avoids problems with cross-origin resource access.

Security Concepts for Remote Plug-ins

Remote plug-ins typically use the HTTP protocol as a transport for requests, whether using REST or SOAP requests. Authentication methods vary, depending on the target endpoint.

Client-side sessions with REST endpoints are tracked with a session token passed in a custom HTTP header. Server-side sessions with SOAP endpoints are tracked with a session cookie.

A plug-in developer can choose what form of authentication suits the plug-in server component. A best practice is to authenticate by using the session token that the plug-in user interface can get from the client library.

To authenticate requests to vCenter Server, the plug-in server sends the token to a specific REST endpoint of the vsphere-ui service. The vsphere-ui service verifies the authenticating token and returns a session clone ticket. The plug-in server uses the clone ticket with the vSphere Web Services API to obtain a SOAP session. The authentication process is described in more detail in Chapter 8 Remote Plug-in Server Considerations for the vSphere Client.

Creating a Remote Plug-in for the vSphere Client

4

This chapter outlines how to create your own remote plug-in for the vSphere Client. The goal is to illustrate the entire process using simple components. Other chapters deal with some of the steps in more detail.

To create a remote plug-in, you need both front-end components and back-end components. The front-end components constitute a user interface based on HTML 5. The back-end components include a web server and business logic in support of the front end.

For the front end of your plug-in, you can choose any JavaScript-compatible language. The samples in the vSphere Client SDK use Angular and Clarity. A best practice is to use Clarity elements to harmonize with the look and feel of the vSphere Client.

For the back end, you can choose any language and any web server. The SDK samples are written in Java and built with Maven, but there is no restriction on the language or tools that you use for back-end development. The SDK samples use Tomcat as a web server because it is bundled with Spring.

This chapter includes the following topics:

- Code Components to Create a Remote Plug-in for the vSphere Client
- Deployment Requirements for a Remote Plug-in for the vSphere Client
- Sample Manifest File for a Remote Plug-in

Code Components to Create a Remote Plug-in for the vSphere Client

A remote plug-in for the vSphere Client includes several code components that you create.

To create code for a complete plug-in, you must do the following:

- Choose the extension points here your user interface views will extend the vSphere Client.
- Create front-end views that provide the user interfaces for data access.
- Create back-end controllers that interface between services and user interface views.
- Create models for your logical data objects.
- Create back-end services that translate between data models and the objects in storage that back the models.

Deployment Requirements for a Remote Plug-in for the vSphere Client

To prepare a remote plug-in for deployment, you must prepare several files and launch some processes.

Deployment of a remote plug-in takes place at run time, but you must make preparations in advance. You need to do the following to prepare for plug-in deployment:

- Run a web server that provides plug-in components on demand.

- Run your plug-in server binary. The plug-in server runs on a virtual or physical machine of your choice, but it must be on the same machine as the web server.

- Prepare a plug-in manifest file, `plugin.json`, that specifies the plug-in components. The manifest file must be on the same machine as the web server.

- Register your plug-in with a vCenter Server instance. To register, you need the URL and credentials to access vCenter Server. You also need the certificate thumbprint of your plug-in server and the URL of your plug-in manifest file.

To register a plug-in, use the script in the SDK: `tools/vCenter plugin registration/prebuilt/extension-registration.sh`

The script arguments are shown in the following example. The `-key` argument must match the `namekey` element in the plug-in manifest, and the `-version` argument must match the `plugin.api.version` element in the manifest. The path segment of the `-pluginUrl` must be specified relative to the directory where you run the plug-in server.

```
./extension-registration.sh    -action registerPlugin -remote \
  -url https://myvcenter/sdk \
  -username    administrator@vsphere.local -password mysecret \
  -key com.mycompany.myplugin -version 1.0.0 \
  -pluginUrl https://mydevbox:8443/myplugin/plugin.json \
  -serverThumbprint 19:FD:2B:0E:62:5E:0E:10:FF:24:34:7A:81:F1:D5:33:19:A7:22:A0 \
  -c 'Example, Inc.' -n 'Remote Plug-in' -s 'This is a remote plug-in'
```

Sample Manifest File for a Remote Plug-in

The following sample manifest file contains the minimum elements required for a very simple plug-in.

```
{
  "manifestVersion" : "1.0.0",
  "requirements": {
    "plugin.api.version": "1.0.0"
  },
  "configuration": {
    "nameKey": "plugin.name",
    "icon": {
```

```
            "name": "main"
        }
    },
    "definitions": {
        "i18n": {
            "locales": ["en-US"],
            "definitions": {
                "plugin.name": {
                    "en-US": "Hello World"
                }
            }
        }
    },
    "global": {
        "view": {
            "uri": "index.html"
        }
    }
}
```

Note   The URIs specified in the manifest file are relative to the location of the manifest file itself. That is, the directory containing `plugin.json` should be considered the server root directory for plug-in resources.

Deploying Remote Plug-ins for the vSphere Client

<div style="text-align: right">

5

</div>

A remote plug-in must be deployed by the vSphere Client before it can be used. Deployment is the process of preparing components of vCenter Server to accept communications from the plug-in and to display plug-in views in the browser.

This chapter includes the following topics:

- Remote Plug-in Life Cycle

- Remote Plug-in Discovery by the vSphere Client

- Remote Plug-in Deployment

- Remote Plug-in Topologies

Remote Plug-in Life Cycle

Remote plug-in content becomes visible in the vSphere Client user interface after you prepare and register the plug-in with a vCenter Server instance. The plug-in displays when you connect a browser to the vSphere Client URL of the vCenter Server instance, or of a vCenter Server linked to the instance where the plug-in is registered, and log in.

A remote plug-in has the following stages in its life cycle:

Build	The plug-in developer assembles the resources and builds the package to implement the plug-in.
Run	The plug-in developer starts the back-end server for the plug-in.
Registration	The plug-in developer registers the plug-in as a vCenter Server extension.
Discovery	vCenter Server discovers the plug-in registration at the time of a trigger event., such as a user login
Deployment	vCenter Server downloads the plug-in manifest file and configures the reverse proxy to route server requests.
Use	The plug-in operates within an iFrame of the vSphere Client user interface.
Uninstallation	The developer unregisters the plug-in, and vCenter Server deletes the routing configuration.

Remote Plug-in Discovery by the vSphere Client

The vSphere Client service checks for new remote plug-ins on three different events:

- vSphere Client service startup. Any new remote plug-ins registered in the Extension Manager before this event will be installed and visible in the vSphere Client after the server startup finishes and the user logs in.

- User login. User login triggers a check for remote plug-ins that were registered after the server was started. The vSphere Client does not wait for the plug-ins to be fully installed before displaying the vSphere Client user interface. After the newly discovered remote plug-ins are deployed and ready for use, the user sees a notification, and a subsequent refresh will display the plug-in in the user interface. The notification displays as a banner at the top of the vSphere Client window, similar to this illustration:

Figure 5-1. Banner notification that plug-in is installed
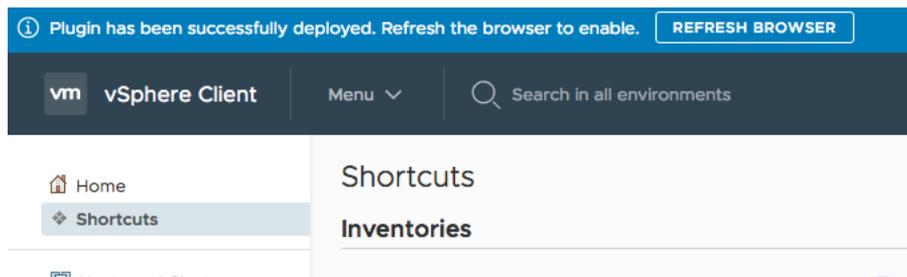
- Browser refresh, while in developer mode. To select developer mode, you can set `local.development=true` in `webclient.properties`. A browser refresh while logged in to the vSphere Client triggers a check for newly registered remote plug-ins. Similar to user login, the vSphere Client will not wait for the plug-ins to be fully installed before proceeding. An additional refresh may be needed after the notification is displayed.

Remote Plugin Uninstallation

Unregistering a plug-in package on vCenter Server causes the vSphere Client service to attempt to delete the plug-in from the environment. A vCenter Server instance discovers plug-in installations on the same three events that trigger discovery of a new plug-in registration:

- Restarting the vsphere-ui service.

- Logging into the vSphere Client.

- Refreshing the browser, while in development mode.

Note   These actions trigger the vSphere Client to begin the process of uninstallation, but the vSphere Client does not wait for the process to complete. You might need to refresh the browser window after enough time has elapsed to complete the uninstallation. The vSphere Client currently does not display notifications after uninstalling remote plug-ins.

Remote Plug-in Deployment

When the vSphere Client discovers a remote plug-in, it schedules the plugin for deployment. vCenter Server takes the following steps to deploy a remote plugin in the vSphere Client:

- Download the remote plug-in manifest, `plugin.json`, from the location specified in the `ClientInfo` property registered in the ExtensionManager. When you use the extension-registration script provided in the SDK this is the value of the `-pu` or `-pluginUrl` argument. vCenter Server downloads your plug-in manifest into `/etc/vmware/vsphere-ui/vc-packages/vsphere-client-serenity/`*your_plugin_name-your_plugin_version*.

- Parse the plug-in manifest to determine what views will be shown in the user interface, and at what extension points.

- Configure the vCenter Server reverse proxy to allow traffic to the remote plug-in server.

After these steps complete, the vSphere Client displays a notification that the remote plugin is deployed.

Remote Plug-in Topologies

Enhanced linked mode (ELM) enables you to manage all linked vCenter Server instances from a single vSphere Client connected to any one of the linked vCenter Server instances. The way that the vSphere Client visualizes the vSphere resources on a vCenter Server instance depends on the set of remote plug-ins installed on that instance. The topology of remote plug-ins is the overall configuration of plug-ins that determines how the vSphere Client interacts with the resources on different vCenter Server instances.

Remote Plug-in Terminology

The following terms are useful to understand the concepts in this chapter.

- Plug-in - A product that extends the vSphere user interface with additional functionality.

- Plug-in Version - The version of the plug-in specified at the time of registration..

- Plug-in Instance - A plug-in server registered with a vCenter Server instance. The plug-in instance is defined by the plug-in product, version, and plug-in manifest URL.

- Plug-in UI - The part of the plug-in loaded inside an IFrame within the vSphere Client UI in the browser.

- Plug-in Server - the back-end process to which the plug-in UI talks.

Visibility of Remote Plug-in Views

In an Extended Linked Mode (ELM) environment, the vSphere Client is capable of loading views from plug-in instances registered with any of the linked vCenter Server instances. For the following examples, suppose you are working with three linked vCenter Server instances: VC1, VC2, VC3. You register instances of plug-in A with VC1 and VC2, but not with VC3.

When you log in to a vCenter Server instance, it checks for plug-ins available on other linked vCenter Server instances. However, plug-in views are visible in your vSphere Client only when appropriate, as defined by these rules:

- Global views are always present in the vSphere Client, regardless of which linked instance of vCenter Server you connect to. A plug-in instance selector enables you to choose between plug-in servers. For example, if you connect to VC3 you can choose between the global views of plug-in A instances registered with VC1 and VC2.

- The visibility of an object-specific plug-in view depends on which vCenter Server instance manages the object. When you select an object managed by a given vCenter Server instance, you have access to object-specific views served by the plug-ins registered with that vCenter Server instance. For example, if you connect to VC3 and select an object managed by VC1, the vSphere Client displays the object-specific view served by plug-in A. However, if you connect to VC1 and select an object managed by VC3, the vSphere Client does not display the object-specific view because VC3 has no instance of plug-in A registered.

- Plug-in visibility is also limited by version compatibility between the plug-in and your vSphere Client. For example, suppose you connect to VC3 and select an object managed by VC2, but the instance of plug-in A registered with VC2 uses a new method not recognized by the older version of the JavaScript API served by VC3. Your version of the vSphere Client is unable to display the view served by plug-in A.

Remote Plug-in Multi-Instance Support

The remote plug-in architecture allows for multiple instances of the same remote plug-in to be deployed within an ELM environment. Instances of a remote plug-in provide distinct server instances that are completely equivalent in function, as long as they have the same version.

Figure 5-2. One Plug-in instance per vCenter Server
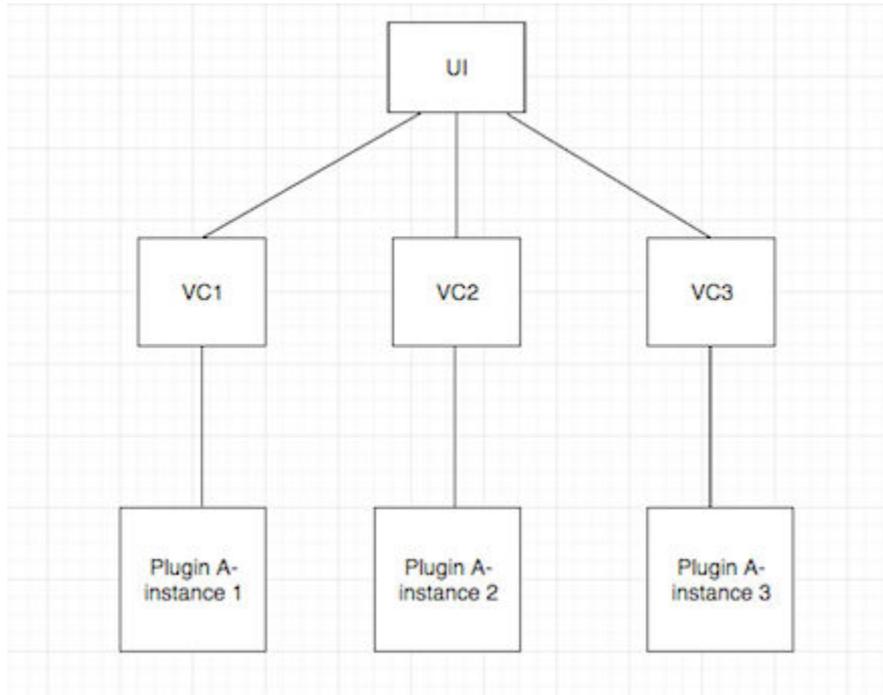
This topology has three instances of PluginA, such that the first plug-in server is connected to VC1, the second is connected to VC2, and the third server is connected to VC3. These plug-in servers are completely equivalent in function: the plug-in manifests that they host and the plug-in specific bits they execute are identical.

Object views will be loaded from the plug-in instance connected with the vCenter Server instance that the object belongs to. For example, if the plug-in declares a portlet on the VM summary page, then browsing VMs on VC2 will load the portlet from the second plug-in server, and browsing VMs on VC3 will load the portlet from the third plug-in server. Calls to the plug-in back end will be routed to the corresponding plug-in server instance.

Global views, however, will be aggregated in a single global view with an additional instance selector component that allows the user to choose between the global views of the different instances. A sample instance selector is shown below:

Figure 5-3. Selector for Single Plug-in Instance per vCenter Server
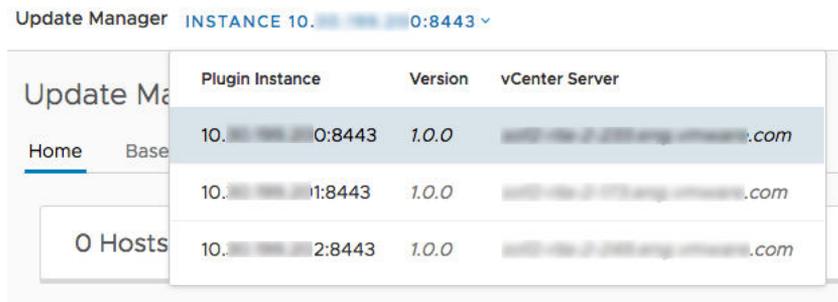
The instance selector shown above displays three back-end servers that represent the three instances of PluginA with version 1.0.0. The **Plugin Instance** column displays the IP address or fully qualified domain name of each plug-in instance, and the **vCenter Server** column displays the vCenter Server that each plug-in instance is connected to. Switching between the items in this drop-down will load the global view of the remote plug-in with version "1.0.0" from the specified plug-in instance.

Differentiating Plug-in Instances

When the vSphere Client checks the vCenter Server's Extension Manager for new remote plug-ins it also checks for new instances of a remote plug-in. If the vSphere Client finds two extension registrations in two different vCenter Server instances such that the extension ID and extension version are the same but the plug-in manifest URL (ExtensionClientInfo.url) is different, then these two extensions are considered to be different instances of the same remote plug-in.

For example, consider an ELM environment with three vCenter Server instances. Suppose that the following extension registration commands are executed:

```
./extension-registration.sh -action registerPlugin -url https://vc1.example.com/sdk
-u "Administrator@vsphere.local" -p 'Admin!23' -c 'Example, Inc.' -n 'Remote Plugin
Test' -s 'A test plugin demonstrating plugin instances' -k
com.example.remoteplugin.test -pu
"https://pluginserver1.example.com/path/to/plugin.json" -v "1.0.0" -st
plugin1_server_thumbprint -remote
```

```
./extension-registration.sh -action registerPlugin -url https://vc2.example.com/sdk
-u "Administrator@vsphere.local" -p 'Admin!23' -c 'Example, Inc.' -n 'Remote Plugin
Test' -s 'A test plugin demonstrating plugin instances' -k
com.example.remoteplugin.test -pu
"https://pluginserver1.example.com/path/to/plugin.json" -v "1.0.0" -st
plugin1_server_thumbprint -remote
```

Both VC1 and VC2 have the same plug-in manifest, and thus the same plug-in server. This is considered to be a single plug-in instance, registered with two vCenter Server instances.

Now suppose that we register the following extension in VC3:

Developing Remote Plug-ins with the vSphere Client SDK

```
./extension-registration.sh -action registerPlugin -url https://vc3.example.com/sdk
-u "Administrator@vsphere.local" -p 'Admin!23' -c 'Example, Inc.' -n 'Remote Plugin
Test' -s 'A test plugin demonstrating plugin instances' -k
com.example.remoteplugin.test -pu
"https://pluginserver2.example.com/path/to/plugin.json" -v "1.0.0" -st
plugin2_server_thumbprint -remote
```

The extension registered in VC3 has the same ID and version as the one registered in VC1 and VC2 but has a different manifest URL. The next time a user logs in, the vSphere Client will detect that this is a different instance of the remote plugin with ID `com.example.remoteplugin.test`, version `1.0.0`. The UI will show the following behavior:

- Object views declared by plug-in instance 2 (registered in VC3) will be shown for the corresponding objects from VC3. However, the views declared by instance 1 (registered in VC1 and VC2) will be shown for objects from VC1 and VC2.

- For global views, there will be a single entry point in the object navigator that takes the user to a plug-in instance selector view where the user will be able to switch between the global views of the two plugin instances.

Remote Plug-in Deployment Example with Simultaneous Users

After being detected, a remote plug-in will be scheduled for deployment in the vSphere Client. The deployment of a remote plug-in, on a high level, consists of the following stages:

- vCenter Server downloads the remote plug-in manifest (plugin.json) from the location specified in the Extension ClientInfo registered in the ExtensionManager. If you use the extension-registration script provided in the SDK, this is the value of the -pu argument. The plug-in manifest is downloaded to a file named as follows: `/etc/vmware/vsphere-ui/vc-packages/vsphere-client-serenity/your_plugin_name-your_plugin_version`

- vCenter Server parses the plug-in manifest to determine what views will be shown in the UI and where they will appear.

- vCenter Server configures the VMware reverse HTTP proxy to route plug-in UI traffic to the remote plug-in server.

After these stages complete successfully, the vSphere Client UI displays a notification message that the remote plug-in is installed.

This example shows in more detail how the deployment process works, in a situation involving three users simultaneously accessing the data center. The initial state consists of the following:

- Three vCenter Server instances in an ELM environment: vCENTER-0, vCENTER-1, and vCENTER-2.

- Three users are accessing the data center: Alpha, Blue, and Claire.

- Blue and Claire are already browsing the vSphere UI loaded from `vCENTER-0`

  - Blue is looking at the summary page of `VM-1` managed by `vCENTER-1`.

VMware, Inc.   46

- ■ Claire is looking at the summary page of VM–2 managed by vCENTER–2.

■ Alpha is about to install a plug-in from Example Company.
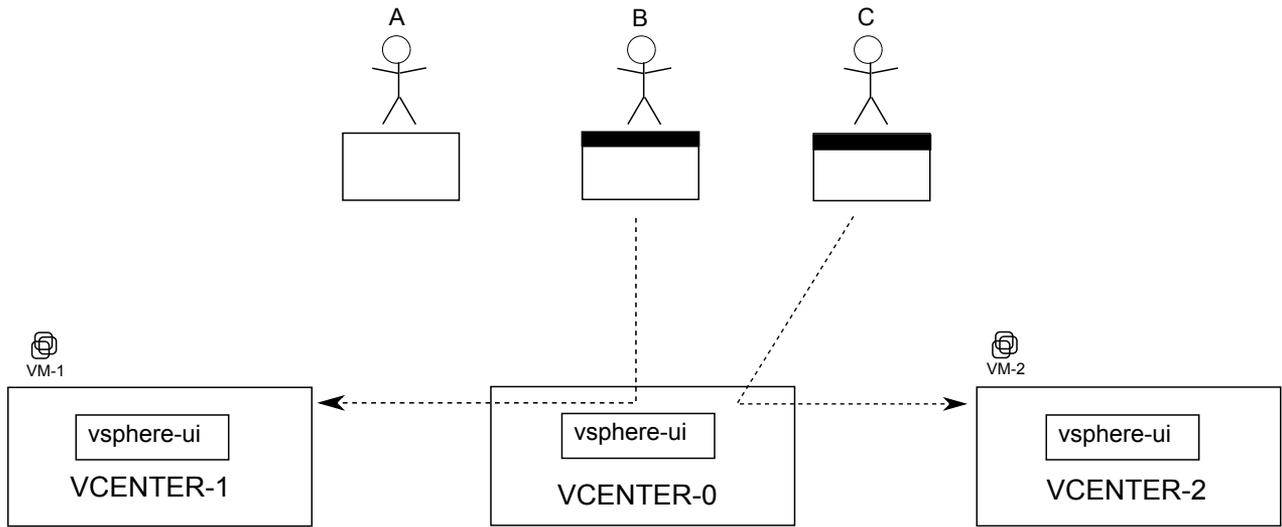
1 Alpha installs and configures the back-end server for the ExampleCo plug-in.
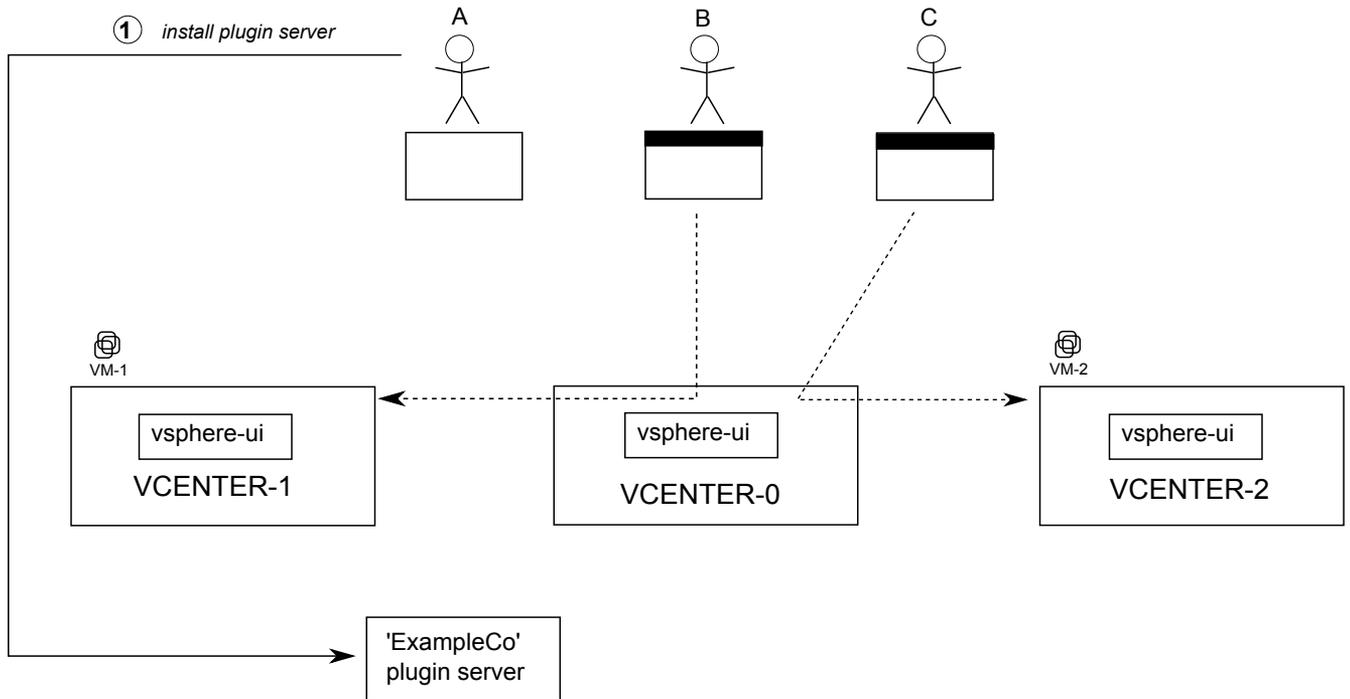
2   Alpha registers the `ExampleCo` plug-in with the `vCENTER-1` ExtensionManager using Example
Company's plug-in installer.
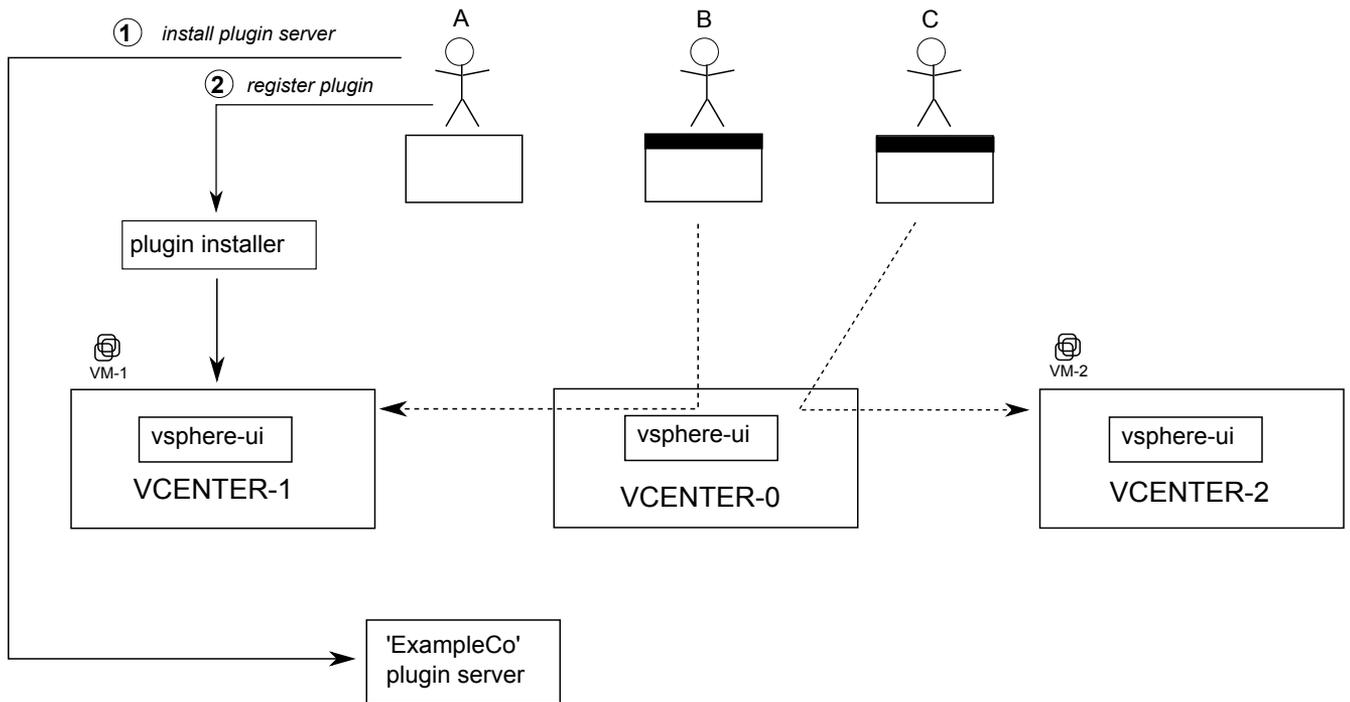
3   Alpha logs in to the vSphere UI connected to `vCENTER-0`.
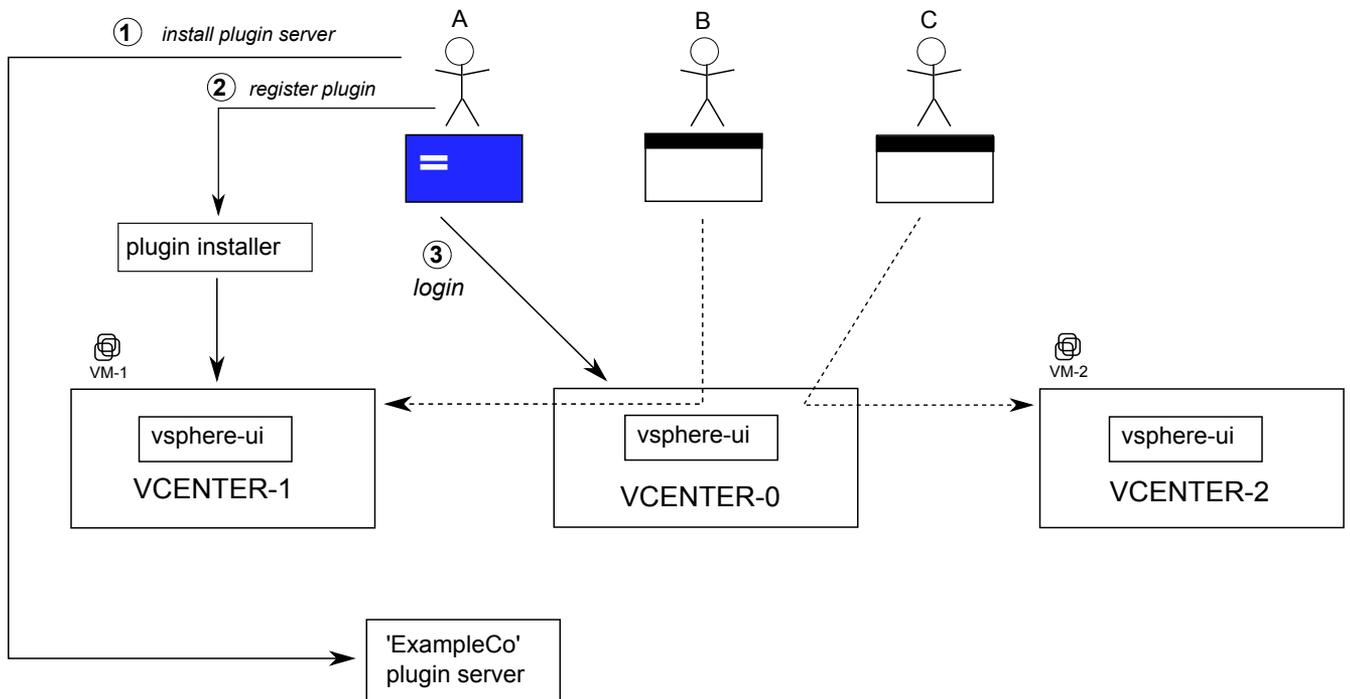
4   Alpha's successful login triggers the vsphere-ui service to contact the ExtensionManager of each vCenter Server instance to discover new plug-in registrations.
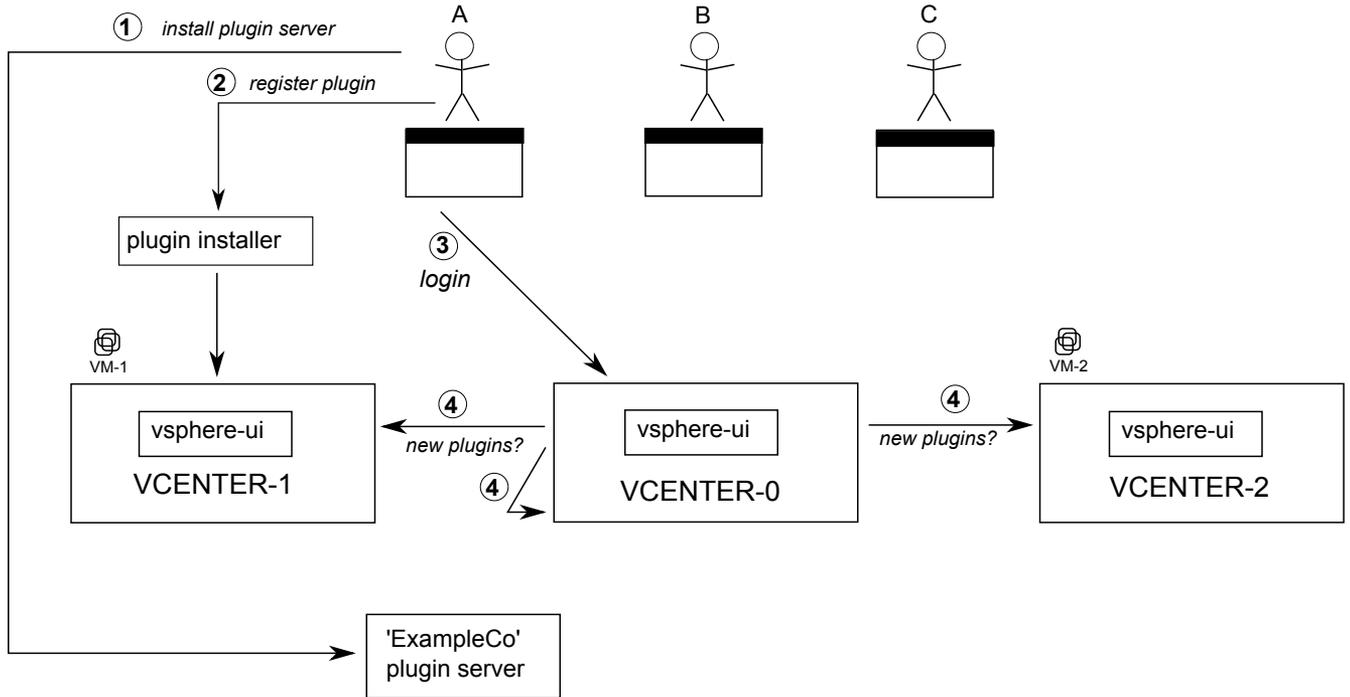
5   The vsphere-ui service discovers the new ExampleCo plug-in registered with vCENTER–1.
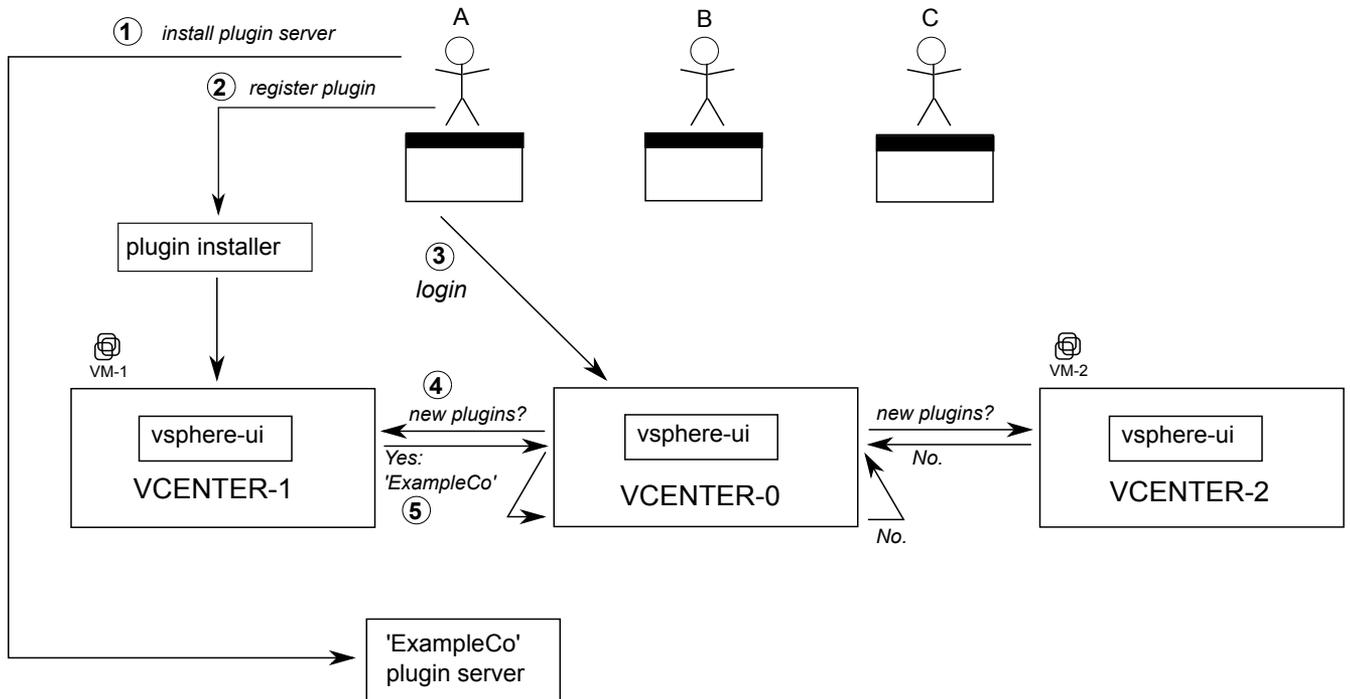
6   The vsphere-ui service downloads the plug-in manifest JSON of the `ExampleCo` plug-in from its back-
     end server.
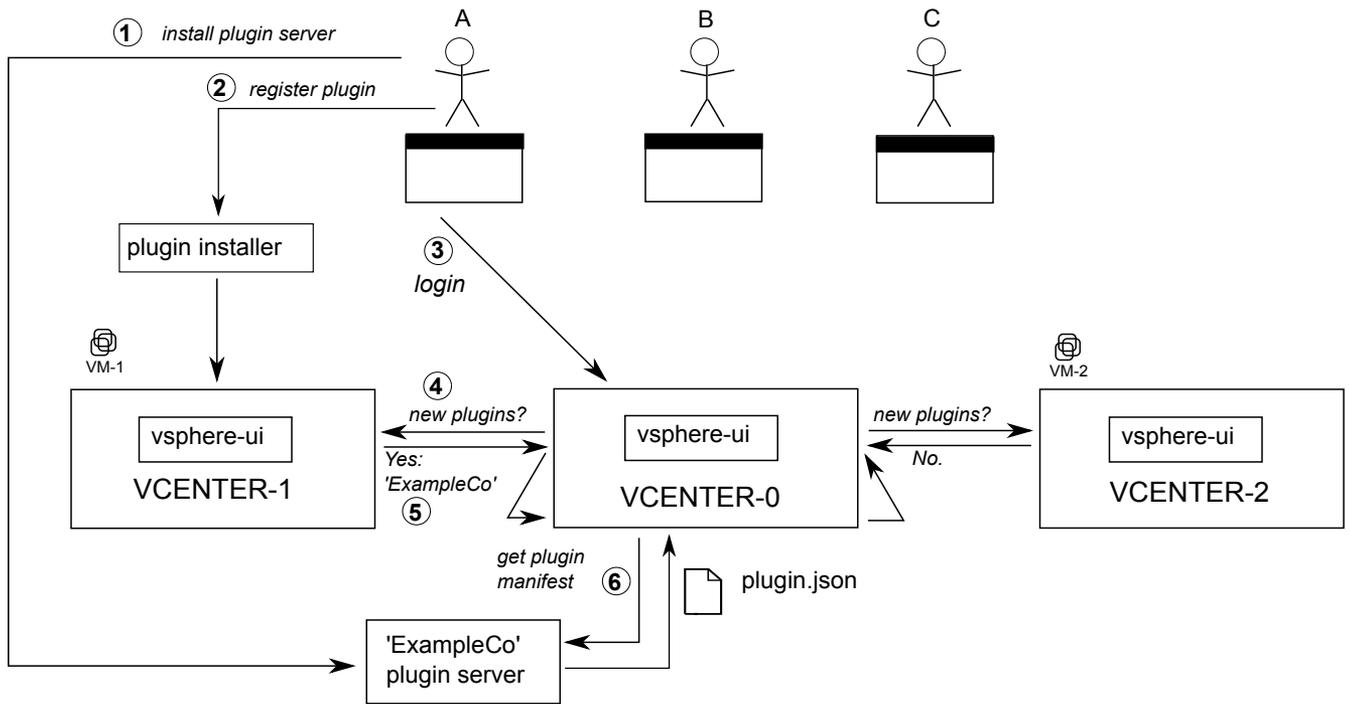
Alpha has completed the login and has loaded the vSphere Client UI, which displays the home
screen of the vSphere Client.

7   The vsphere-ui service sends notifications to all vSphere Client users (Alpha, Blue, and Claire). Each user sees a blue notification banner at the top of the screen.
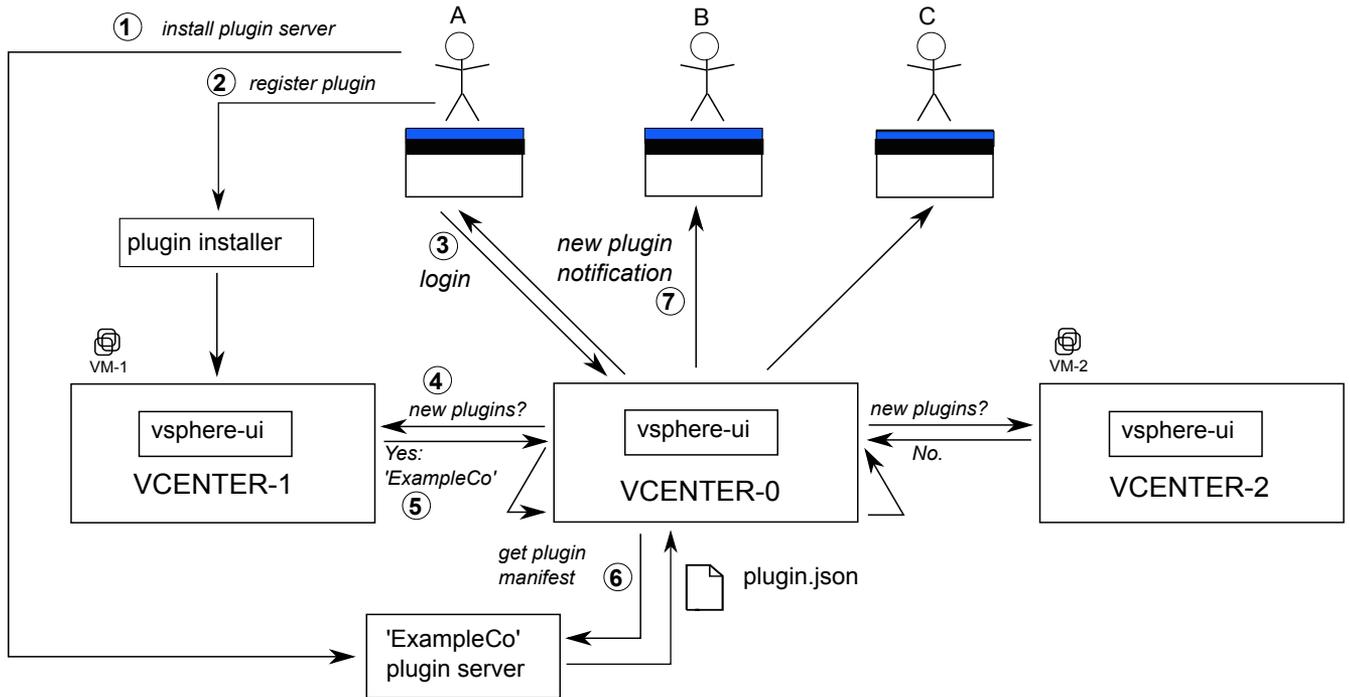
Note   Depending on timing, it is possible that the vsphere-ui service reads the `ExampleCo` manifest before Alpha loads the vSphere Client UI. In this case, Alpha immediately sees the plug-in, and does not see a notification.

8   When Alpha refreshes the vSphere Client UI in the browser, the ExampleCo plug-in is loaded. The
    portlet adds a home menu item and a shortcut link.
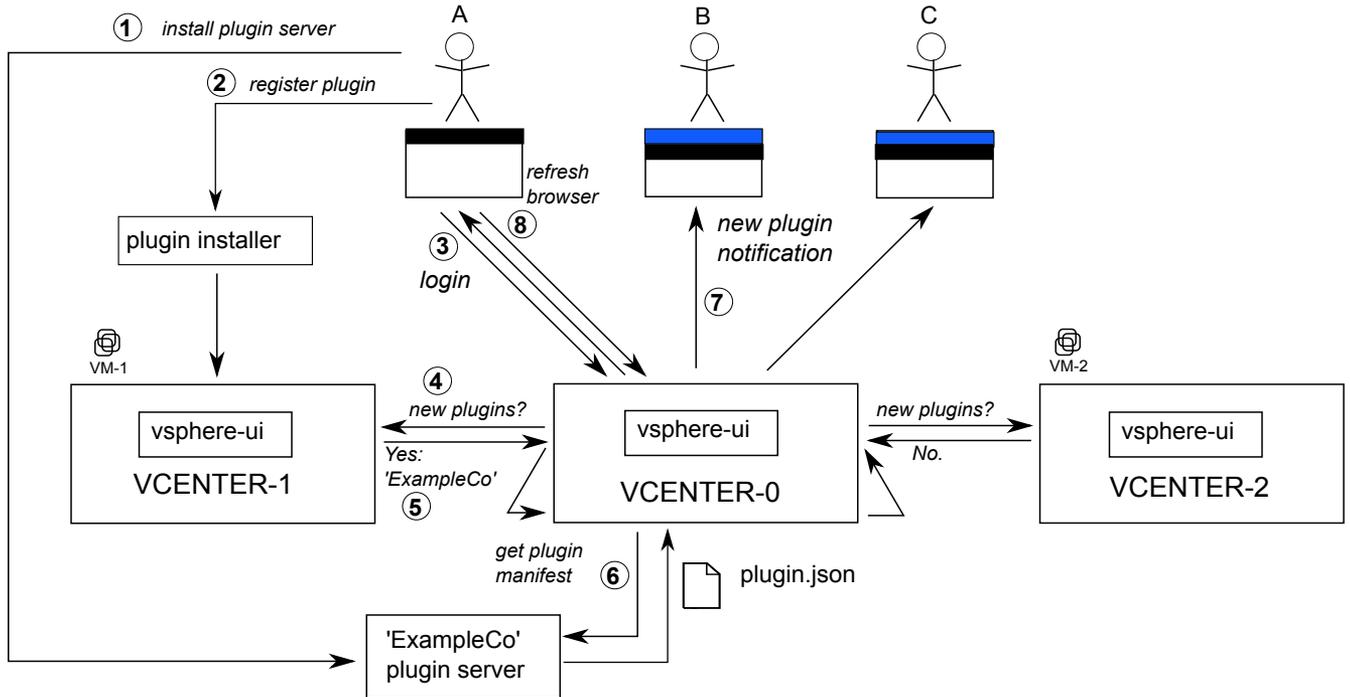
9   When Blue refreshes the vSphere Client UI in the browser, the ExampleCo plug-in is loaded for this
    user. The plug-in adds a portlet on the summary page of VMs. Since VM–1 is managed by
    vCENTER–1, which has the ExampleCo plug-in registered, Blue sees the newly added portlet.
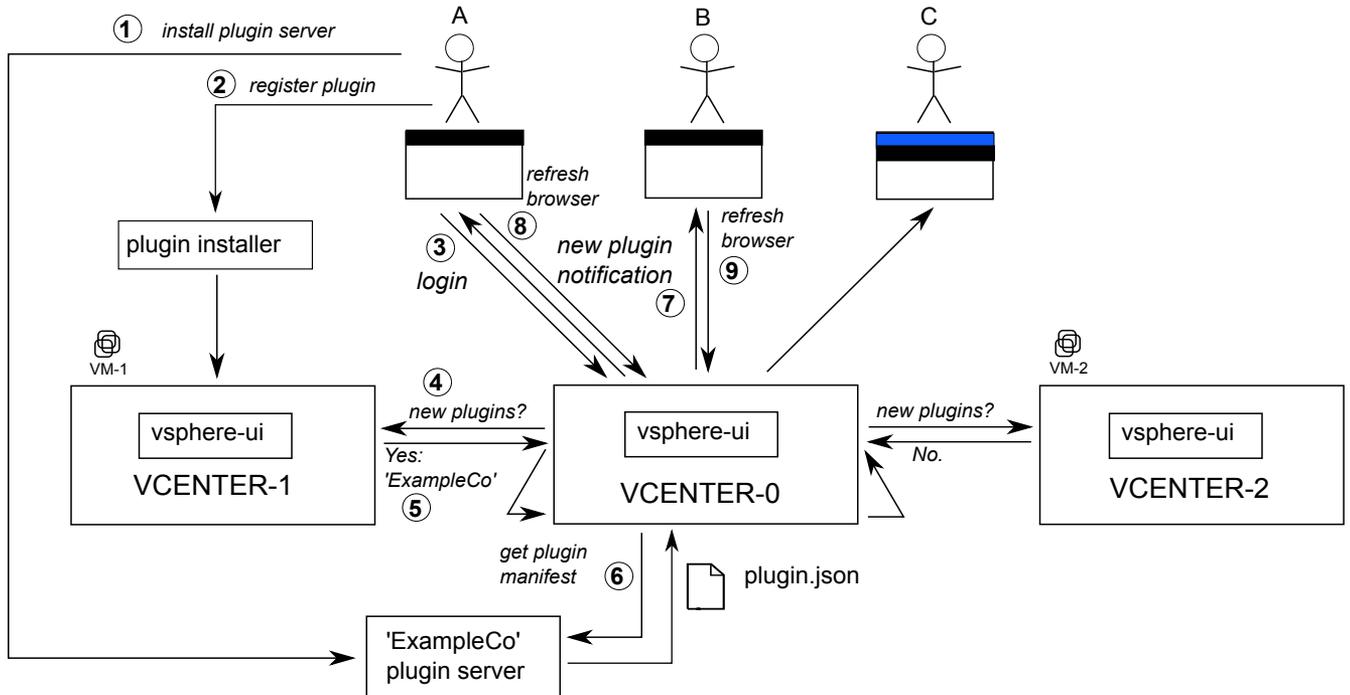
10 When Claire refreshes the vSphere Client UI in the browser, the `ExampleCo` plug-in can now be loaded for this user. However, since `VM-2` is managed by `vCENTER-2`, which does not have the `ExampleCo` plug-in registered, Claire cannot see the newly added portlet.
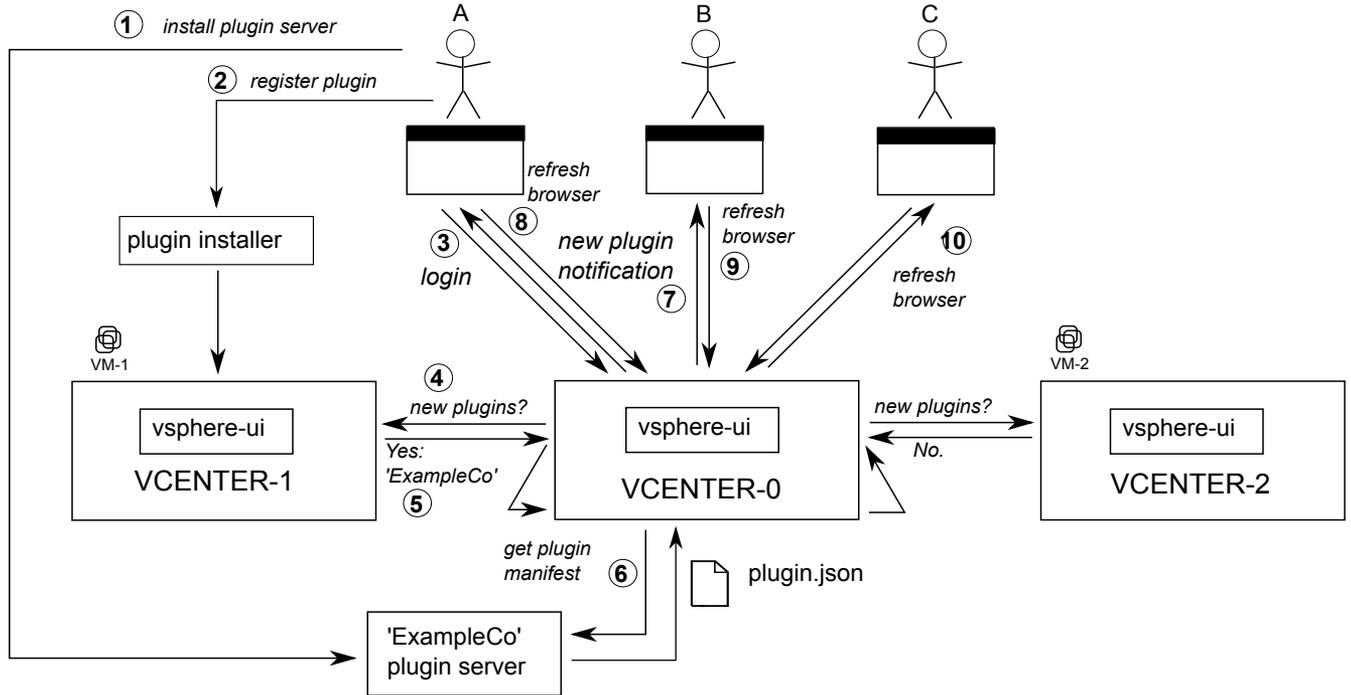
If Claire then navigates to a VM on `vCENTER-1`, the vSphere Client will display the portlet added by the `ExampleCo` plug-in.

Remote Plug-in Multi-Version Support

Unlike local plug-ins, the remote plug-in architecture allows for co-existence of remote plug-ins with the same ID but different versions.

Consider an ELM environment with three vCenter Server instances and the following plug-in registrations:

./extension-registration.sh -action registerPlugin -url https://***vcenter-ip-or-fqdn-of-vc1***/sdk -u "Administrator@vsphere.local" -p 'Admin!23' -c 'Example, Inc.' -n 'ExampleCo' -s 'A test plugin demonstrating plugin instances' -k com.example.exampleco -pu "https://**my-remote-plugin-server-version-1**/*path-to*/plugin.json" -v "**1.0.0**" -st ***plugin_server_1_thumbprint*** -remote

./extension-registration.sh -action registerPlugin -url https://***vcenter-ip-or-fqdn-of-vc2***/sdk -u "Administrator@vsphere.local" -p 'Admin!23' -c 'Example, Inc.' -n 'ExampleCo' -s 'A test plugin demonstrating plugin instances' -k com.example.exampleco -pu "https://**my-remote-plugin-server-version-2**/*path-to*/plugin.json" -v "**2.0.1**" -st ***plugin_server_2_thumbprint*** -remote

```
./extension-registration.sh -action registerPlugin -url https://vcenter-ip-or-fqdn-
of-vc3/sdk -u "Administrator@vsphere.local" -p 'Admin!23' -c 'Example, Inc.' -n
'ExampleCo' -s 'A test plugin demonstrating plugin instances' -k
com.example.exampleco -pu "https://my-remote-plugin-server-version-3/path-
to/plugin.json" -v "3.2.0" -st plugin_server_3_thumbprint -remote
```

These commands register three extensions (one in each of the three vCenter Servers) with the same ID (com.example.exampleco) but different versions - version 1.0.0 on VC1, version 2.0.1 on VC2 and version 3.2.0 on VC3. These are three different versions of the remote plug-in with ID com.example.exampleco. When you log in to the UI you will see the following:

- Object views declared by plug-in com.example.exampleco version 1.0.0 will be shown on applicable objects from VC1. Calls to the plug-in back-end server will be routed to the plug-in server dedicated to version 1.0.0 of the plugin.

- Object views declared by plug-in com.example.exampleco version 2.0.1 will be shown on applicable objects from VC2. Calls to the plug-in back-end server will be routed to the plug-in server dedicated to version 2.0.1 of the plug-in.

- Object views declared by plug-in com.example.exampleco version 3.2.0 will be shown on applicable objects from VC3. Calls to the plug-in back-end server will be routed to the plug-in server dedicated to version 3.2.0 of the plug-in.

- There will be a single entry point in the object navigator that will take the user to a plug-in instance/version selector view where the user will be able to switch between the global views of the different versions and instances of the remote plug-in.

Choosing Extension Points for vSphere Client Plug-ins

<div style="text-align: right">6</div>

The vSphere Client supports adding content at key extension points in the user interface. A plug-in developer can insert custom views that present objects and functions not provided by vSphere.

The available extension points closely follow the navigation experience in the vSphere Client, which facilitates a clear mapping of a plug-in to the views and workflows it owns. The extension points operate at a high level to allow the developer maximum creative space and flexibility.

This chapter includes the following topics:

- Types of Extension Points in the vSphere Client

- Remote Plug-in Manifest Example

- List of Extension Points in the vSphere Client

Types of Extension Points in the vSphere Client

The vSphere Client provides a number of integration points that plug-ins can extend. Extensions are separated into groups by the inventory context selection, and sub-groups by the purpose of the user interface element.

The extension points are grouped as follows:

- **Global extension points**

   These are plug-in UI elements that have global scope. Their context is the entire plug-in, rather than a particular inventory object.

   - **View**

      This is a single global UI element that can consume a large section of the vSphere Client real estate. When multiple global pages are required, they should be implemented within this single global view. Navigation between the nested pages must be handled by the plug-in front end.

      For example, configuration pages or status dashboards.

- **Object extension points**

These are plug-in UI elements within the scope of the currently selected context object in the inventory. They are defined per object type and displayed only for the selected object. Their views can include other inventory objects or external objects, as long as there is a logical relevance to the current object.

- **Summary Section View**

  This is a single plug-in view that is displayed as a small box in the object's summary view. It is fixed in size and should contain primarily simple name-value data. Optionally it can contain action buttons or links to more detailed Monitor or Configure pages.

  For example, a view for a selected host that shows the following:

  - The number of virtual machines that need backup.

  - A button that backs up all virtual machines that need backup.

  - A link to a Monitor view that lists all virtual machines that need backup, and allows a user to back them up separately.

- **Monitor Views**

  This comprises a single Monitor category with one or more views. It can contain detailed monitoring and maintenance data and workflows relevant to the current object.

  For example, the backup status of all virtual machines on a selected host.

- **Configure Views**

  This comprises a single Configure category with one or more views. It can contain detailed configuration data and workflows relevant to the current object.

  For example, a list of virtual machines on a selected host that should be backed up.

- **Menu**

  This is a single plug-in solution menu with one or more actions. It contains actions that apply to the current object, allowing user input before execution.

  For example, a Menu action to back up all virtual machines on a selected host.

Remote Plug-in Manifest Example

The following JSON code is an example of a plug-in manifest file that demonstrates how to specify some of the vSphere Client SDK extension points for your plug-in.

```
{
  "manifestVersion": "1.0.0",
  "requirements": {
    "plugin.api.version": "1.0.0"
  },
  "configuration": {
    "nameKey": "My Plugin",
    "icon": {
```

```
        "name": "main"
      }
    },
    "global": {
      "view": {
        "navigationId": "myGlobalViewId",
        "uri": "myplugin/globalView.html",
        "navigationVisible": false
      }
    },
    "objects": {
      "Datacenter": {
        "summary": {
          "view": {
            "uri": "myplugin/summary.html",
            "icon": {
              "name": "main"
            },
            "size": {
              "widthSpan": 1,
              "heightSpan": 2
            }
          }
        },
        "monitor": {
          "views": [
            {
              "navigationId": "myview1",
              "labelKey": "category.view1",
              "uri": "myplugin/view1.html"
            }
          ]
        },
        "configure": {
          "views": [
            {
              "navigationId": "myview1",
              "labelKey": "category.view1",
              "uri": "myplugin/view1.html"
            }
          ]
        },
        "menu": {
          "actions": [
            {
              "labelKey": "action1",
              "icon": {
                "name": "action-1"
              },
              "trigger": {
                "type": "modal",
                "uri": "myplugin/modal-action.html",
                "size": {
                  "height": 250,
                  "width": 600
```

```
                    }
                }
            }
        ]
    }
  }
},
"definitions": {
  "iconSpriteSheet": {
    "uri": "myplugin/images/icon-sprite.png",
    "definitions": {
      "main": {
        "x": 0,
        "y": 0
      }
    }
  },
  "i18n": {
    "locales": [
      "en-US",
      "de-DE",
      "fr-FR"
    ],
    "definitions": {
      "category.view1": {
        "en-US": "Monitor View 2",
        "de-DE": "Monitoransicht 2",
        "fr-FR": "Vue Surveiller 2"
      }
    }
  }
 }
}
```

List of Extension Points in the vSphere Client

The vSphere Client publishes extension points that you can use to create your extensions, including global extension points and several kinds of object-specific extension points.

For a complete list of the extension points available in the vSphere Client SDK, see the reference guide *vSphere Client Remote Plug-in Extensions Reference* at https://code.vmware.com/web/sdk/client.

vSphere Client Plug-in User Interface Modules

7

The vSphere Client provides several JavaScript interfaces that your plug-in can use to communicate with the vSphere Client platform. These JavaScript methods are documented here as if they have TypeScript signatures, but they run as pure JavaScript, and all complex types are plain old JavaScript objects.

The plug-in web application runs in a separate iframe which is part of the vSphere Client. The iframe content is rendered from the web application server of the plug-in back end.

Note Do not access the `window.parent`, which belongs to the vSphere Client. Such access is unsupported and could cause your plug-in to fail in a future release of the vSphere Client.

This chapter includes the following topics:

- Bootstrapping the JavaScript API
- vSphere Client JavaScript API: htmlClientSdk Interface
- vSphere Client JavaScript API: Modal Interface
- vSphere Client JavaScript API: Application Interface
- vSphere Client JavaScript API: Event Interface
- Example Using the modal API

Bootstrapping the JavaScript API

The vSphere Client loads plug-in resources in a tenant iframe. The plug-in must load a thin JavaScript library to support communication with the parent window in the vSphere Client. The library implements a JavaScript API that the plug-in front-end code uses to manage resources outside its iframe.

To bootstrap the Client Library the following script should be added to all HTML pages in the plug-in:

```
<script type="text/javascript"src="/api/ui/htmlClientSdk.js"></script>
```

After you load the script, you initialize the htmlClientSdk object, by invoking the `htmlClientSdk.initialize()` method. Before you initialize, you can only invoke `htmlClientSdk.initialize()` or `htmlClientSdk.isInitialized()`. After you initialize the htmlClientSdk object, you can invoke any of the methods in the JavaScript API.

If you use frameworks such as jQuery, or zone.js with Angular, you only need to initialize the htmlClientSdk object once. You should initialize as early as possible, so that the htmlClientSdk functions will be available to all plug-in user interface components.

Note Do not use any communication method not provided by the APIs. Doing so is unsupported because the implementation of the htmlClientSdk functions can change in future releases of the vSphere Client.

vSphere Client JavaScript API: htmlClientSdk Interface

The htmlClientSdk object provides access to all the JavaScript API interfaces. You load and initialize the htmlClientSdk first in your plug-in views.

`htmlClientSdk.initialize()` method

Signature	`htmlClientSdk.initialize(callback):void`
Description	Initializes the htmlClientSdk object and invokes the callback function when initialization is complete. After the htmlClientSdk object is initialized, subsequent calls invoke the callback function without needing to initialize the htmlClientSdk object.
Parameter: `callback`	The callback function must have the following signature: function callback():void The callback function has access to all the functions in the JavaScript API. You use the callback to code logic for initializing plug-in state.

`htmlClientSdk.isInitialized()` method

Signature	`htmlClientSdk.isInitialized():boolean`
Description	Tests whether the htmlClientSdk object has been initialized.

vSphere Client JavaScript API: Modal Interface

The `modal` interface enables your plug-in to manage modal dialog windows.

`modal.open()` method

Signature	`modal.open(configObj:ModalConfig):void`
Description	Opens a modal dialog box specified by the `configObj` parameter.
Parameter: `configObj`	Specifies the properties of this modal dialog box.

modal.close() **method**

Signature	`modal.close(data:any):void`
Description	Closes the modal dialog box in the parent iframe.
Parameter: `data`	Passes `data` to the callback function specified by `onClosed` property at dialog open.

modal.setOptions() **method**

Signature	`modal.setOptions(options:DynamicModalOptions):void`
Description	Called by the parent view to modify some properties for a modal dialog box in the parent iframe.
Parameter: `options`	Specifies values for some dialog box properties.

modal.getCustomData() **method**

Signature	`modal.getCustomData():any`
Description	Returns the `customData` object provided when a modal dialog box was opened, or `null` if no `customData` object was provided.

modal.DynamicModalOptions **type**

Description	Specifies values for some properties of a modal dialog box.

Name	Type	Required?	Notes
`title`	`string`	no	Dialog title. May not contain an icon. (If not present, no change to dialog title.)
`height`	`number`	no	Dialog size. Specified in pixels. (If not specified, no change to dialog height.)

modal.ModalConfig **type**

Description	Specifies the properties of a modal dialog box.

Name	Type	Required?	Notes
`url`	`string`	yes	Location of HTML content for the dialog.
`title`	`string`	no	Dialog title. May not contain an icon. (default=`''`)
`size`	`(width:number, height:number)`	yes	Dialog size. Specified in pixels.
`closable`	`boolean`	no	Whether the dialog displays a close button. (default=`true`)
`onClosed`	`callbackFn`	no	Function runs when the dialog closes.
`customData`	`any`	no	Data the calling module passes to the dialog.
`contextObjects`	`string[]`	no	IDs of relevant objects the calling module passes to the dialog.

vSphere Client JavaScript API: Application Interface

The *app* interface provides context object information and helps your plug-in navigate and control the vSphere Client user interface.

app.getApiEndpoints() **method**

Signature	`app.getApiEndpoints():UiApiEndpoint`
Description	Returns the URLs of the vsphere-ui service API endpoints available to plug-ins.

app.getContextObjects() **method**

Signature	`app.getContextObjects():any[]`
Description	Returns the current context objects.

Return value:		
	for global view	Returns empty array. Global views have no context.
	for vSphere object	Returns a context item for the associated vSphere object.
	for dialog opened by `modal.open()`	If dialog opened by `htmlClientSdk.modal.open(configObj)`, returns value of `configObj.contextObjects` (or empty array, if `contextObjects` undefined)
	for dialog opened by `plugin.json` actions	If dialog opened by action defined in `plugin.json`, returns an array of context items.

A context item is a JavaScript object containing a single property, `id:string`. This is the ID of the associated vSphere object.

app.navigateTo() **method**

Signature	`app.navigateTo(options:NavigationOptions):void`
Description	Navigates to a specified view, and optionally passes custom data to the view.
Parameter: `options`	Specifies the destination view and custom data.

app.getNavigationData() **method**

Signature	`app.getNavigationData():any`
Description	Returns the custom data passed to the view by the `navigateTo()` method. (If no custom data passed, returns `null`.)

app.getClientInfo() **method**

Signature	`app.getClientInfo():ClientInfo`
Description	Returns type and version info for the vSphere Client.

app.getClientLocale() **method**

Signature	`app.getClientLocale():string`
Description	Returns the current locale of the vSphere Client.

app.getSessionInfo() **method**

Signature	`app.getSessionInfo(callback):void`
Description	Retrieves information about the client's authentication session. The callback function must have the following signature:

```
function callback(sessionInfo:SessionInfo)
```

app.getTheme() **method**

Signature	`app.getTheme():PluginTheme`
Description	Retrieves information about the theme the plug-in should use.

app.ClientInfo **type**

Description	Documents type and version of vSphere Client.

Name	Type	Required?	Notes
`type`	`string`	info only	The vSphere Client type (`HTML` or `Flex`)
`version`	`string`	info only	The vSphere Client version string.

app.NavigationOptions **type**

Description	Specifies a destination view and custom data for the view.

Name	Type	Required?	Notes
`targetViewId`	`string`	yes	Navigation ID of the destination view. (For a remote plug-in, this property must identify a view created by the same plug-in.)
`objectId`	`string`	no	ID of any object associated with the view. (For a global view, this field is not required.)
`customData`	`any`	no	A custom data structure passed to the view.

app.PluginTheme **type**

Description	Indicates the UI theme that is currently selected.

Name	Type	Required?	Notes
`name`	`string`	info only	Possible values: `light` or `dark`.

`app.QueryParam` **type**

Description	Holds a single query parameter of a URL.

Name	Type	Required?	Notes
name	string	info only	Name of query parameter, as in ?*name*=value.
value	string	info only	Value of query parameter, as in ?name=*value*.

`app.SessionInfo` **type**

Description	Holds information about the current session of the vSphere Client.

Name	Type	Required?	Notes
sessionToken	string	info only	Identifier of the plug-in authentication session with vCenter Server.

`app.UiApiEndpoint` **type**

Description	Holds the parsed elements of the plug-in URL.

Name	Type	Required?	Notes
origin	string	info only	<protocol>://<hostname><port>
pathname	string	info only	
queryParams	Array<QueryParam>	info only	<name>=<value>
fullUrl	string	info only	<origin>/<pathname>?<queryParams>

vSphere Client JavaScript API: Event Interface

The event interface helps your plug-in with event management.

`event.onGlobalRefresh()` **method**

Signature	`event.onGlobalRefresh(pluginCallbackFunc:function):void`
Description	Registers a global refresh handler that the vSphere Client will call when the **Global Refresh** button is clicked.
Parameter: `pluginCallbackFunc`	A reference to a global refresh handler, with signature `function callback():void`

event.onThemeChanged() **method**

Signature	`event.onThemeChanged(themeCalbackFunc:function):void`
Description	Registers an event handler that the vSphere Client will call when the vSphere Client changes the current theme.
Parameter: `themeCallbackFunc`	A reference to a theme change handler, with signature `function themeCallbackFunc(theme:app.PluginTheme):void`

Example Using the modal API

This example shows some basic features of the Model interface of the Client API.

modal.html

```html
<html>
  <head>
    <script src="http://code.jquery.com/jquery-latest.min.js"
            type="text/javascript"></script>
    <script src="/api/ui/htmlClientSdk.js"
            type="text/javascript"></script>
    <script type='text/javascript'>
      function handler(event)
      {
        var choice = $('input[name=heads_or_tails]:checked').val();
        htmlClientSdk.modal.setOptions({title: choice});
        settimeout(function(){htmlClientSdk.modal.close(choice);}, 3000);
      }
    </script>
  </head>
  <body>
    <form name='flip' onSubmit='return handler()'>
      <p><input type='radio' name='heads_or_tails' value='HEADS' />HEADS</p>
      <p><input type='radio' name='heads_or_tails' value='TAILS' />TAILS</p>
      <input type='submit' name='submit' value='Submit' />
    </form>
  </body>
</html>
```

modal.js

```javascript
flipper = function(){
  # Select correct answer.
  correct = ['heads', 'tails'][2*Math.random()-1];

  # Create callback function.
  checker = function(choice){
    var correct = htmlClientSdk.modal.getCustomData();
```

```
    if (choice === correct) {
      alert('You chose wisely.');
    } else {
      alert('Sorry, you lose.');
    }}

  # Configure modal dialog.
  var config ={
    url: "example/dialog.html",
    title: 'Choose!',
    size: { width: 490, height: 240 },
    onClosed: checker,
    customData: correct}

  # Open modal dialog.
  htmlClientSdk.modal.open(config);
}

# Initialize Javascript API.
$(document).ready(htmlClientSdk.initialize(flipper));
```

Remote Plug-in Server Considerations for the vSphere Client

8

A remote plug-in for the vSphere Client has both a server portion and a user interface portion. You can use any coding language or framework you choose for the server portion. Your plug-in server must generally provide the following functionality:

- A web application server that serves both the plug-in manifest file and the plug-in user interface files.

- A fixed Service Provider Interface that responds to vSphere Client requests for dynamic view content, such as menus.

- Data access and computation services on behalf of the plug-in user interface views, if needed.

- Session cloning service to authenticate with the vCenter Server Web Services API, if needed.

This chapter includes the following topics:

- Communication Paths for Authentication in the Remote Plug-in Server

- vSphere Authentication in the Remote Plug-in Server

Communication Paths for Authentication in the Remote Plug-in Server

The remote plug-in server operates outside the vCenter Server instance, and must authenticate with the Web Services API to identify and authorize its access to vSphere resources. The process of authentication requires several steps.

The plug-in user interface communicates with the vsphere-ui service through a plug-in sandbox in the browser. The plug-in sandbox uses the vSphere Client session token to authenticate with the vsphere-ui service in vCenter Server. The plug-in server needs a SOAP client session token to authenticate its operations with the Web Services API. The following diagram shows the communication paths involved in converting the vSphere Client session token to a plug-in server SOAP session token.

Figure 8-1. **Plug-in Server Communication Paths for Authentication**
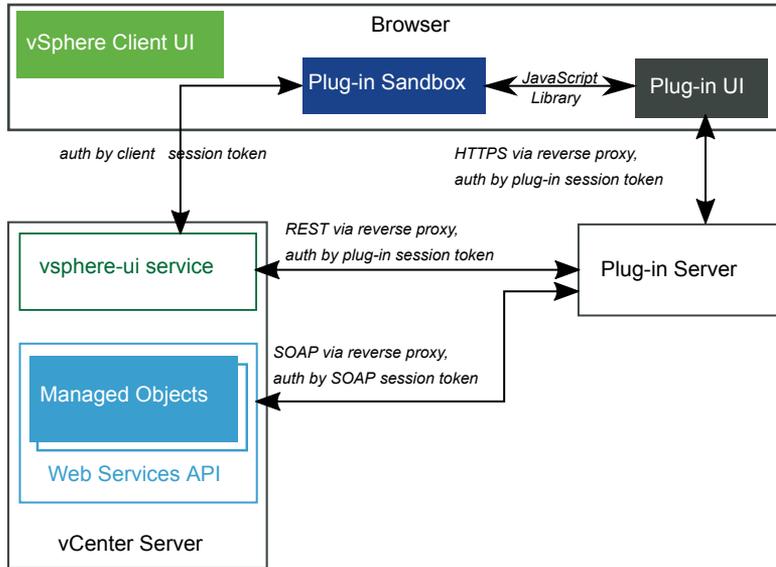
Cloning a session consists of three interactions involving the plug-in server:

1   The plug-in user interface sends its session ID and the vCenter Server endpoint to the plug-in server.

2   The plug-in server sends a REST request to vCenter Server to acquire a ticket that allows it to clone the user session.

3   The plug-in server sends a SOAP request to vCenter Server to clone the user session and acquire a new session ID.

vSphere Authentication in the Remote Plug-in Server

When a plug-in server accesses vSphere data, it needs to authenticate with vCenter Server. To authenticate by using the Web Services API, the plug-in server clones the user session that is currently in use by the plug-in user interface. This gives the plug-in server the same access rights as the user who is logged in with the vSphere Client.

Following are the detailed steps to clone the user session.

Note   The plug-in server must be registered with the vCenter Server instance before it can clone a session with that instance.

1   The plug-in user interface calls the `app.getSessioninfo()` method in the client JavaScript library, which in turn contacts the plug-in sandbox to request session information. The sandbox returns an object containing a `sessionToken` string, which contains a new plug-in session token that can be used for authentication by the plug-in server.

2   The plug-in user interface calls the `app.getApiEndpoints()` method in the client JavaScript library, which returns an object containing a `uiApiEndpoint` property. The value of the `uiApiEndpoint` property is an object containing a `fullUrl` property, which contains the endpoint URL for a plug-in server REST request to the vsphere-ui service.

3   The plug-in user interface removes any query parameters and fragments from the URL, leaving the scheme, host, port, and path segments. The user interface sends both the session token value and the base URL to the plug-in server.

> **Note** Do not hard-code the URL in the server.

4   The plug-in server builds a REST request to the vsphere-ui service. The request contains the following:

  ■ A POST verb.

  ■ The `Content-type` and `Accept` headers both set to `application/json`.

  ■ A custom header named `vmware-api-session-id`, with the session token as its value.

  ■ A JSON object body, containing a `vc-guid` property whose value is the GUID of the vCenter Server instance.

The request will look similar to this:

```
POST /api/ui/vcenter/session/clone-ticket
Content-type: application/json
Accept: application/json
vmware-api-session-id: 12345678

{
   "vc_guid": "223b94f2-af15-4613-5d1a-a278b19abc09"
}
```

5   The plug-in server sends the REST request to the vsphere-ui service, which returns a clone ticket valid for the Web Services API of the vCenter Server instance. This is a single-use key to authenticate a call to the SessionManager.

The response will look similar to this:

```
{
   "session_clone_ticket": "cst-VCT-82cbd981-5f52-0a67-fe55-d995a7347f82--tp-B6-BC-CB-B8-59-89-C0-
F2-E4-F0-C2-91-8F-28-C1-DE-10-5E-24-69"
}
```

6   The plug-in server constructs a SOAP request to obtain a regular session ID from the Web Services API, by using the `cloneSession()` operation on the Session Manager.

The code for the SOAP request will be similar to this Java example:

```
VimService vimService = new VimService();
VimPortType client = vimService.getVimPort();
ManagedObjectReference siRef = new ManagedObjectReference();
siRef.setType("ServiceInstance");
siRef.setValue("Serviceinstance");
```

```
ServiceInstance si = client.createStub(ServiceInstance.class, siRef);
ServiceInstanceContent sic = si.RetrieveContent();
SessionManager mgr = client.createStub(SessionManager.class, sic.getSessionManager());
UserSession wsSession = mgr.cloneSession(cloneTicket);
```

7   The plug-in server uses the `key` property of the `UserSession` object to authenticate subsequent requests to the Web Services API. The server places a `vmware_soap_session` cookie in its SOAP request headers, with the session key as the cookie value.

Response Codes to session/clone-ticket Request

The REST request for a clone-ticket from the vsphere-ui service can produce the following response codes.

Table 8-1. **Response Codes to clone-ticket Request**

Response Code	Description
201	Session clone ticket was successfully created and returned. The body type is `application/json`, and the response body has this format: ``` { "session_clone_ticket": "string" } ```
401	The request is not properly authenticated.
403	The remote plug-in is not registered with the vCenter Server instance identified by the specified UUID.
404	There is no vCenter Server instance identified by the specified UUID.