SouthBound SDK Driver Programmer's Guide

Introduction

In order to make CoprHD's new storage system support be developed more easier and efficient, the CorpHD DEV team creates the CoprHD southbound SDK, with which the developers can create southbound SDK drivers for each type of storage system based on the unified API set defined in the southbound SDK framework without updating CoprHD code. As the developer of the VMAX v3 southbound SDK driver, I'd like to introduce how to develop a southbound SDK driver based on my knowledge of the VMAX v3 SBSDK driver development In this wiki page.

There are two types of SBSDK driver:

1. In-tree driver: a driver released as part of the CoprHD product and its source code is part of the CoprHD project source code, it is also deployed as part of CoprHD when CoprHD is deployed;
2. Out-of-tree driver: a driver whose source code is not part of the CoprHD source code, it will be built separately and deployed to CoprHD after CoprHD is deployed and running.

The VMAX v3 SBSDK driver is an in-tree driver. All content of this wiki page is based on the knowledge of VMAX v3 SBSDK driver and in-tree driver unless a special declaration is made.

Fundamental

Create The Driver Project

In-tree Driver

For an in-tree driver, you need to do the following things to create the driver project:

1. Get the source code of CoprHD:

```
git clone https://review.coprhd.org/scm/ch/coprhd-controller.git
```

2. Create the driver project directory under "coprhd-controller/exportLibraries/storagedrivers/" directory, such as "coprhd-controller/exportLibraries/storagedrivers/vmaxv3driver/" for the VMAX v3 driver.

3. Create the project structure. As CoprHD use Gradle to build, you should create a "build.gradle" file in your project directory, as well as the "src" directory. More details about "build.gradle" can be found in the existing driver projects in CoprHD under the "coprhd-controller/exportLibraries/storagedrivers/" directory.

The "build.gradle" file of "vmaxv3driver" project

```
apply plugin: 'java'

group = 'com.emc.storageos.driver.vmaxv3driver'
version = '1.0-SNAPSHOT'
description = 'VMAX V3 Southbound SDK driver implementation'

dependencies {
    compile project(":storagedriver")
    compile library(slf4j)
    compile library(log4j)
    compile library(commons_logging)
    compile library(httpclient4)
    compile library(google_gson)
    compile library(json_sanitizer)
    testCompile group: 'org.testng', name: 'testng', version:'6.8.7'
}
```

Note: the "compile project(":storagedriver")" line is required since the driver needs the interfaces/classes defined in the project.

4. Update the "coprhd-controller/settings.gradle" file by adding your driver project into the "addProjects" part. You can following the existing driver projects content in this file.

Part of the "settings.gradle" of the "coprhd-controller" project

```
rootProject.name = "ViPR"
addProjects(
        "apisvc",
        "authsvc",
        "dbclient",
        "dbsvc",
        "coordinatorsvc",
        "controllersvc",
        "syssvc",
        "exportLibraries/cimadapter",
        "exportLibraries/discoveryplugins",
        "exportLibraries/vnx",
        "exportLibraries/netapp",
        "exportLibraries/netappc",
        "exportLibraries/recoverpoint",
        "exportLibraries/isilon",
        "exportLibraries/datadomain",
        "exportLibraries/vplex",
        "exportLibraries/vnxe",
        "exportLibraries/hds",
        "exportLibraries/compute",
        "exportLibraries/cinder",
        "exportLibraries/glance",
        "exportLibraries/keystone",
        "exportLibraries/scaleio",
        "exportLibraries/xtremio",
        "exportLibraries/ecs",
        "exportLibraries/ceph",
        "exportLibraries/xiv",
        "exportLibraries/storagedrivers/storagedriversimulator",
        "exportLibraries/storagedrivers/vmaxv3driver",
        "geodbsvc",
...
```

Note the ""exportLibraries/storagedrivers/vmaxv3driver"," line in this file which makes the "vmaxv3driver" project can be built as part of the CoprHD building process.

Out-of-tree Driver

1. Download the "storagedriver-xxx.jar" and "storagedriver-xxx-source.jar" files("xxx" means version name) from the attachments section of the "Storage Driver SDK for Array Integration to CoprHD" wiki page. The "storagedriver-xxx.jar" file contains all the classes needed for the SDK driver implementation, and the "storagedriver-xxx-source.jar" file contains the Java source of these classes. The "storagedriver-xxx.jar" file has the following content:

   1. Set of interfaces which driver have to implement;
   2. Set of object model classes which are used in interface methods;
   3. Interfaces for CoprHD services available to storage driver;
   4. Several supporting classes (DriverTask, CapabilityInstance, etc.);
   5. Public libraries which are used by SDK;
   6. Abstract base class for storage driver.

2. Create your driver project and configure the "storagedriver-xxx.jar" jar as part of the library of this project.

3. When building your project, if you use any third-party libraries(jar files), please remember to bundle them into the output driver jar file, since

when uploading the driver jar file to CoprHD, only one jar file is allowed to be uploaded. And also note that the "storagedriver-xxx.jar" is unnecessary to be bundled into the driver jar since CoprHD has it already.

Build

For the in-tree drivers, you can use the following commands to build the driver:

```
cd coprhd-controller
./gradlew :vmaxv3driver:assemble
ll build/gradle/exportLibraries/storagedrivers/vmaxv3driver/libs/
```

Notes:

1. Please run the "./gradlew ..." command under the "coprhd-controller" directory;
2. Replace the "vmaxv3driver" project name with your own project name;
3. After the building process finishes, the output driver jar can be found as "coprhd-controller/build/gradle/exportLibraries/storagedrivers/vmaxv3driver/libs/storageos-vmaxv3driver.jar"(for the "vmaxv3driver" project).

In some cases, you may want to modify the "southbound SDK framework" source code in the "controllersvc" project, for an example, adding some logging lines, you can modify the source code and then build the "storageos-controllersvc.jar":

```
cd coprhd-controller
./gradlew :controllersvc:assemble
ll build/gradle/controllersvc/libs
```

After that you can replace the same jar file in CoprHD with the one you build here, to make it generate more logging information.

For the out-of-tree drivers, the building process depends on how you implement the project, such as "Maven" or "Gradle", just make sure the driver jar file can be created successfully.

Deploy

Note: the steps described here is the manual steps of the current driver deployment process. Currently " Cao, Shiliang " is working on a feature to make the steps be automated. After this feature is ready you don't have to use the manual steps here. Please take attention.

1. Update CoprHD configuration files

Note:

1. The instructions here are the ones for the "vmaxv3driver" project, you need to change them to fit your own driver project.
2. There are 2 types of drivers according to the "scope" perspective: "storage system" type and "storage provider" type. The "storage system" type means one discovery process can discover one storage array, while the "storage provider" type means one discovery process can discover multiple storage arrays. The 2 types need different configuration files modifications. The below ones are for the "vmaxv3driver" driver which is a "storage provider" type driver, for a "storage system" type driver, the modifications will be different. More information about "storage provider" and "storage system" types drivers can be found here.

Login CoprHD VM via SSH, and update the following configuration files:

1. Add storage provider type: update the "/opt/storageos/conf/driver-conf.xml" file on CoprHD:

   a. Add the following line in "storageSystemMap":

      <entry key="Vmaxv3Driver" value="vmaxv3_system"/>

   b. Add the following line in "storageProvidersMap":

      <entry key="Vmaxv3Driver" value="vmaxv3_provider"/>

   c. Add the following line in "blockSystems":

    <value>vmaxv3_system</value>

  d. Add the following line in "providerManaged":

    <value>vmaxv3_system</value>

2. Update the "/opt/storageos/conf/controller-conf.xml" file on CoprHD:

  a. Add the following line in "<drivers>" under "externalBlockStorageDevice":

    <entry key="vmaxv3_provider" value-ref="vmaxv3StorageDriver"/>

    <entry key="vmaxv3_system" value-ref="vmaxv3StorageDriver"/>

  b. Add the following line beside "<bean id="storageDriverSimulator" ... ></bean>":

    <bean id="vmaxv3StorageDriver" class="com.emc.storageos.driver.vmaxv3driver.Vmaxv3StorageDriver"></bean>

3. Update the "/opt/storageos/conf/discovery-externaldevice-context.xml" file on CoprHD:

  a. Add the following line beside "<bean id="storageDriverSimulator" ... ></bean>":

    <bean id="vmaxv3StorageDriver" class="com.emc.storageos.driver.vmaxv3driver.Vmaxv3StorageDriver"></bean>

  b. Add the following line in "<drivers>" under "externaldevice":

    <entry key="vmaxv3_provider" value-ref="vmaxv3StorageDriver"/>

    <entry key="vmaxv3_system" value-ref="vmaxv3StorageDriver"/>

2. Upload your driver jar file

Upload the driver jar file to CoprHD's "/opt/storageos/lib" directory. After uploading it to CoprHD, need to change the file permission on CoprHD "chmod 644 storageos-vmaxv3driver.jar" and owner "chown storageos:storageos storageos-vmaxv3driver.jar" to make it work.

```
scp
build/gradle/exportLibraries/storagedrivers/vmaxv3driver/libs/storageos-vm
axv3driver.jar root@<CoprHD-IP>:/opt/storageos/lib/
```

3. Update the "classpath" of the controller service

Login CoprHD VM via SSH, update "classpath of controller service" in file "/opt/storageos/bin/controllersvc": add ":${LIB_DIR}/storageos-vmaxv3driver.jar" to the end of the "export CLASSPATH=" line.

After finish the above 3 steps, you need to restart CoprHD services to make it take effect:

```
/etc/storageos/storageos restart
```

4. Add the "Storage System Type" items on CoprHD UI

  a. Go to "System -> Storage System Types", click the "Add" button;

  b. Add 2 items, one for "vmaxv3_provider" and the other for "vmaxv3_system"(note that the names must match the ones used in driver xml files):

```
Name            | Display Name     | Is Provider | Type     | Supports SSL
| Port Number | SSL Port Number | Driver Class Name
vmaxv3_provider | VMAX V3 Provider | true        | block    | true
| 8443        | 8443            |
com.emc.storageos.driver.vmaxv3driver.Vmaxv3StorageDriver
vmaxv3_system   | VMAX V3 System   | false       | block    | true
| 8443        | 8443            |
com.emc.storageos.driver.vmaxv3driver.Vmaxv3StorageDriver
```

Note that the "vmaxv3driver" driver is a "storage provider" type driver. In this case, 2 items are needed here as shown in the table above. For a "storage system" type driver, only one item is needed.

Testing

After you deploy the driver jar on CoprHD, you can login CoprHD, go to "Physical -> Storage Providers" page to add a "VMAX V3 Provider", and then CoprHD will start the discovery process which will make the CoprHD controller service invokes the discovery APIs implemented by the "vmaxv3driver" southbound SDK driver. If the discovery process succeeds, you can go ahead to test the provisioning operations like "create volume" and "export volume".

Note that the "vmaxv3driver" driver is a "storage provider" type driver. For a "storage system" type driver, go to the "Physical -> Storage Systems" page to add the storage system.

Implementation

After a southbound SDK driver is deployed on CoprHD, when there are operations(discovery, provisioning operations) which need the southbound SDK driver, the controller service will look up the given driver entrance class as the given "Driver Class Name" field(for "vmaxv3driver", it's "com.emc.storageos.driver.vmaxv3driver.Vmaxv3StorageDriver") in its classpath. The entrance class should implements the following 2 interfaces provided by the southbound SDK framework:

1. `com.emc.storageos.storagedriver.DiscoveryDriver // This interface defines the set of methods for discovery.`
2. `com.emc.storageos.storagedriver.BlockStorageDriver // This interface defines the set of methods for block provisioning operations.`

Note that both "DiscoveryDriver" and "BlockStorageDriver" interfaces extend from the "StorageDriver " interface, that means the entrance class should also implement the abstract methods defined in the "StorageDriver " interface. For the "vmaxv3driver" driver, since none of the API methods in the "StorageDriver" interface is needed, so the corresponding implementation methods are dummy ones which just return null. This may not be true in another driver implementation, so please consult "roytman, evgeny <evgeny.roytman@emc.com>" who is the author of the southbound SDK framework to know how to implement the API methods defined in the "StorageDriver" interface when implementing your southbound SDK driver.

To implement the "DiscoveryDriver" and "DiscoveryDriver" interfaces, you don't need to create a class to implement them from scratch, instead, you can leverage the existing classes provided by the southbound SDK, such as:

1. **com.emc.storageos.storagedriver.AbstractStorageDriver**
   This abstract class implements the "DiscoveryDriver" interface and add the methods of "Registry" and "LockManager" supports. To leverage this class, you need to create a class(the entrance class of the driver) which extends the "DiscoveryDriver" abstract class and implements the "BlockStorageDriver" interface. In this entrance class, you need to implement all the abstract methods needed for discovery and the block provisioning operations.
2. **com.emc.storageos.storagedriver. DefaultStorageDriver**
   This class extends the "AbstractStorageDriver" abstract class and implements the "BlockStorageDriver" interface. It implements all the abstract methods by using dummy implementations such as throwing an "UnsupportedOperationException" exception. To leverage this class, you need to create the entrance class which extends this class, and override the dummy implementation methods by implementing the real discovery/provisioning operations code lines.

For the "vmaxv3driver" driver, it chose the second approach, which is to create the "com.emc.storageos.driver.vmaxv3driver.Vmaxv3StorageDriver" entrance class by making it extend the " DefaultStorageDriver" class and implement the discovery/provisioning operation methods to override the existing dummy implementation methods in " DefaultStorageDriver".

You can read the existing "vmaxv3driver" driver code for more details: "coprhd-controller/exportLibraries/storagedrivers/vmaxv3driver"

Discovery

The following methods are defined in the "com.emc.storageos.storagedriver.DiscoveryDriver" interface and should be implemented in the driver entrance class. Please note that all the input/output fields information below is gotten from my "vmaxv3driver" development experience without any official document information and just for your reference, this may be vary when you are doing your driver development.

1. discoverStorageProvider

discoverStorageProvider

```
// 1. Original method signature copied from "DiscoveryDriver.java".
/**
 * Discover storage provider and storage systems under this provider
management.
 * This operation is similar to provider scan.
 * For managed storage systems driver should return key connection
properties required to run detailed discovery.

 * @param storageProvider Type: Input/Output.
 * @param storageSystems  Type: Output.
 * @return driver task
 */
public DriverTask discoverStorageProvider(StorageProvider storageProvider,
List<StorageSystem> storageSystems);

// 2. Usage
/*
This method is invoked by the southbound SDK framework when CoprHD starts a
"storage provider" discovery
process. The SDK framework will pass the storage provider information(such
as IP, username & password)
in the "storageProvider" argument. With this information, the
implementation of this method should connect
to the target storage provider and fetch the storage provider information
and the storage system list
information, then set them into the "storageProvider" instance and the
"storageSystems" instance passed in
as the method arguments. After the method returns, the southbound SDK
framework will read the "storage
provider" and the "storage systems" information from the "storageProvider"
and "storageSystems" instances.
*/

// 3. Arguments
/*
1. storageProvider
    The following fields of this instance are set by the southbound SDK
framework as input:
        useSSL, providerHost, portNumber, username, password
    The following fields of this instance should be set by the driver as
output:
        useSSL, providerHost, portNumber, username, password
```

```
2. storageSystems
 This is a list of the storage systems managed by the storage provider
being set by the driver
    as output. Each storage system instance in this list should have the
following field values:
        systemType, serialNumber, systemName, nativeId
*/

// 4. Return value
/*
    A "DriverTask" instance which indicates the result of this operation,
such as "TaskStatus.READY"
    and "TaskStatus.FAILED".
*/
```

2. discoverStorageSystem

discoverStorageSystem

```
// 1. Original method signature copied from "DiscoveryDriver.java".
/**
 * Discover storage system and it's capabilities
 *
 * @param storageSystem StorageSystem to discover. Type: Input/Output.
 * @return
 */
public DriverTask discoverStorageSystem(StorageSystem storageSystem);

// 2. Usage
/*
This method is invoked by the southbound SDK framework when CoprHD starts a
"storage system" discovery
process. The SDK framework will pass the storage system information(such as
IP, username & password)
in the "storageSystem" argument. With this information, the implementation
of this method should connect
to the target storage system and fetch the storage system information, then
set it into the "storageSystem"
instance passed in as the method arguments. After the method returns, the
southbound SDK framework will read
the "storage systems" information from the and "storageSystem" instance.
*/

// 3. Arguments
/*
1. storageSystem
    The following fields of this instance are set by the southbound SDK
framework as input:
        ipAddress, portNumber, username, password
    The following fields of this instance should be set by the driver as
output:
        isSupportedVersion, systemName, nativeId, firmwareVersion, model,
provisioningType,
        serialNumber, deviceLabel
 */

// 4. Return value
/*
    A "DriverTask" instance which indicates the result of this operation,
such as "TaskStatus.READY"
    and "TaskStatus.FAILED".
*/
```

3. discoverStoragePools

discoverStoragePools

```
// 1. Original method signature copied from "DiscoveryDriver.java".
/**
 * Discover storage pools and their capabilities.
 * @param storageSystem Type: Input.
 * @param storagePools  Type: Output.
 * @return
 */
public DriverTask discoverStoragePools(StorageSystem storageSystem,
List<StoragePool> storagePools);

// 2. Usage
/*
This method is invoked by the southbound SDK framework when CoprHD starts
discovering "storage pools"
of a storage system. The SDK framework will pass the storage system
information(such as IP, username & password)
in the "storageSystem" argument. With this information, the implementation
of this method should connect
to the target storage system and fetch the storage pools information, then
set it into the "storagePools"
instance passed in as the method arguments. After the method returns, the
southbound SDK framework will read
the "storage pools" information from the and "storagePools" instance.
*/

// 3. Arguments
/*
1. storageSystem
    The following fields of this instance are set by the southbound SDK
framework as input:
        ipAddress, portNumber, username, password
2. storagePools
 This is a list of the storage pools managed by the storage system being
set by the driver
    as output. Each storage pool instance in this list should have the
following field values:
        nativeId, displayName, deviceLabel, poolName, storageSystemId,
protocols, totalCapacity,
        freeCapacity, subscribedCapacity, operationalStatus,
supportedResourceType,
        poolServiceType, capabilities
*/

// 4. Return value
/*
    A "DriverTask" instance which indicates the result of this operation,
such as "TaskStatus.READY"
    and "TaskStatus.FAILED".
*/
```

4. discoverStoragePorts

discoverStoragePorts

```
// 1. Original method signature copied from "DiscoveryDriver.java".
/**
 * Discover storage ports and their capabilities
 * @param storageSystem Type: Input.
 * @param storagePorts Type: Output.
 * @return
 */
public DriverTask discoverStoragePorts(StorageSystem storageSystem,
List<StoragePort> storagePorts);

// 2. Usage
/*
This method is invoked by the southbound SDK framework when CoprHD starts
discovering "storage ports"
of a storage system. The SDK framework will pass the storage system
information(such as IP, username & password)
in the "storageSystem" argument. With this information, the implementation
of this method should connect
to the target storage system and fetch the storage ports information, then
set it into the "storagePorts"
instance passed in as the method arguments. After the method returns, the
southbound SDK framework will read
the "storage ports" information from the and "storagePorts" instance.
*/

// 3. Arguments
/*
1. storageSystem
    The following fields of this instance are set by the southbound SDK
framework as input:
        ipAddress, portNumber, username, password
2. storagePorts
 This is a list of the storage ports managed by the storage system being
set by the driver
    as output. Each storage port instance in this list should have the
following field values:
        storageSystemId, nativeId, deviceLabel, displayName, portName,
transportType,
        portNetworkId, portGroup, portHAZone, operationalStatus,
endPointID, portType,
 capabilities
*/

// 4. Return value
/*
    A "DriverTask" instance which indicates the result of this operation,
such as "TaskStatus.READY"
    and "TaskStatus.FAILED".
*/
```

5. discoverStorageHostComponents

discoverStorageHostComponents

```
// 1. Original method signature copied from "DiscoveryDriver.java".
/**
 * Discover host components which are part of storage system
 *
 * @param storageSystem Type: Input.
 * @param embeddedStorageHostComponents Type: Output.
 * @return
 */
public DriverTask discoverStorageHostComponents(StorageSystem
storageSystem, List<StorageHostComponent> embeddedStorageHostComponents);

// 2. Usage
/*
The "vmaxv3driver" driver does not use this method, so no more information
about it.
*/
```

Provisioning

The following methods are defined in the "com.emc.storageos.storagedriver.BlockStorageDriver" interface and should be implemented in the driver entrance class. Please note that all the input/output fields information below is gotten from my "vmaxv3driver" development experience without any official document information and just for your reference, this may be vary when you are doing your driver development.

1. createVolumes

createVolumes

```
// 1. Original method signature copied from "BlockStorageDriver.java".
/**
 * Create storage volumes with a given set of capabilities.
 * Before completion of the request, set all required data for provisioned
volumes in "volumes" parameter.
 *
 * @param volumes Input/output argument for volumes.
 * @param capabilities Input argument for capabilities. Defines storage
capabilities of volumes to create.
 * @return task
 */
public DriverTask createVolumes(List<StorageVolume> volumes,
StorageCapabilities capabilities);
```

```
// 2. Usage
/*
This method is invoked by the southbound SDK framework when CoprHD creates
volume. The SDK framework will
pass the volume list and the "capabilities" object. With this information,
the implementation of this
method should be able to create the volume(s) creation request needed by
the target volume(s) creation API
of the target array. After the method returns, the southbound SDK framework
will read the volume(s)
information from the and "volumes" instance.
Note that: In the ViPR UI "Create Block volume" catalog service page, user
can create multiple volume items,
each item can have multiple volumes with the same volume size. In this
case, the volume list passed to the
current method will contain multiple volume items and each one may have
multiple volumes with the same volume
size. For an example, if the user creates 3 volume items in the "Create
Block volume" catalog service page,
the method call to the driver will pass in a volume list with 3 instances
in it, each one is an item the user
created in the UI and may have multiple volumes(if the user set so) with
the same size.
*/

// 3. Arguments
/*
1. volumes
    The following fields of this instance are set by the southbound SDK
framework as input:
        nativeId, wwn, displayName, deviceLabel, provisionedCapability
2. capabilities
 Not used.
*/

// 4. Return value
/*
    A "DriverTask" instance which indicates the result of this operation,
such as "TaskStatus.READY"
    and "TaskStatus.FAILED".
*/
```