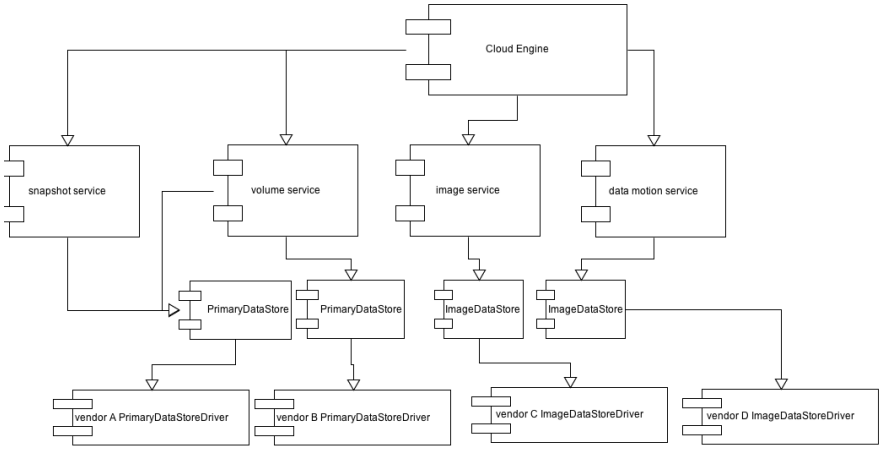
Storage subsystem 2.0

Contents

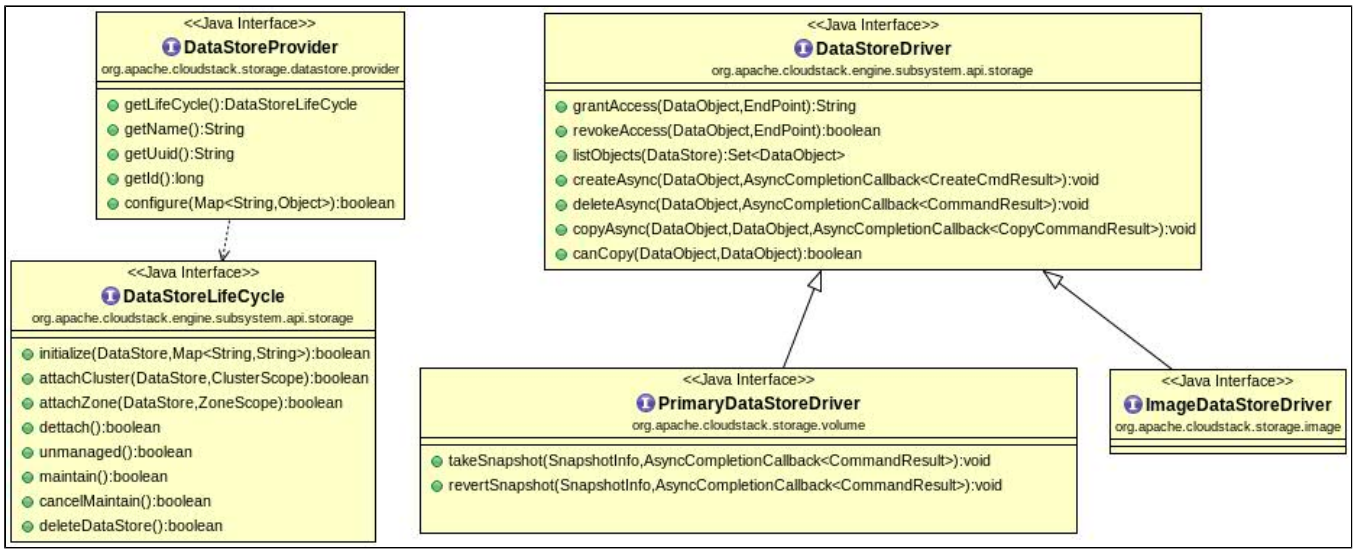
The new storage subsystem:



The functionality of each component:

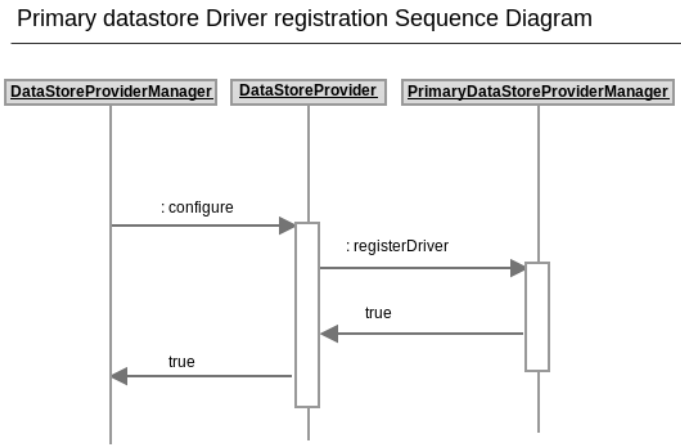
- Cloud Engine is the core component of CloudStack management server
- ImageService will handle image(template, iso) register/download/cache/ on image(secondary) storage(s3/nfs/ etc)
- MotionService will handle moving image from image storage to primary storage
- VolumeService will handle how to create/destroy volume on primary storage
- SnapshotService will handle how to take/destroy snapshot from volume

Data store provider:



Above are all the apis related to storage provider. At the minimal, storage provider should implement these three interfaces:

DataStoreProvider is the entry point of a provider, which will provide DataStoreLifeCycle for all the storages managed by this provider, and also register DataStoreDriver into CloudStack storage subsystem. The provider will be initialized during CloudStack management server startup. The sequence diagram about how DataStoreProvider is initialized, how to register DataStoreDriver into CloudStack storage subsystem is:



DataStoreDriver is the interface for the driver. There are two categories of driver: one for primary storage(PrimaryDataStoreDriver), another one for image storage(ImageDataStoreDriver). The difference between the two categories is quite small: PrimaryDataStgoreDriver has two extra APIs than ImageDataStoreDriver, which are special APIs related to snapshot.

The description of each API:

grantAccess(DataStream, StorageClient client): make the DataStream accessible for a client, and return an URI represent the DataStream. It serves two purposes: one is to get an URI representation of DataStream, different storage vendor may have different way to encode a data created on storage. Another purpose is to make the DataStream accessible for a client, if necessary. For example, an ISCSI LUN may only open to certain hosts at one time, when calling this API will make sure the LUN will be accessed by specified host.

revokeAccess(DataStream, StorageClient client): the reverse operation of above API.

listObjects(DataStream), list data on datastore

createAsync(DataStream): create a data on datastore, the driver shouldn't care about what's the object it is, but should only care about the size of the object, the data store of the object, all of these information can be directly inferred from DataStream. If the driver needs more information about the object, driver developer can get the id of the object, query database, then find about more information. And this interface has no assumption about the underneath storage, it can be primary storage, or s3/swift, or a ftp server, or whatever writable storage.

deleteAsync(DataStream): delete an object on a datastore, the opposite of createAsync

copyAsync(DataStream, DataStream): copy src object to dest object. It's for storage migration. Some storage vendor or hypervisor has its own efficient way to migrate storage from one place to another. Most of the time, the migration across different vendors or different storage types(primary <=> image storage), needs to go to datamotionservice, which will be covered later.

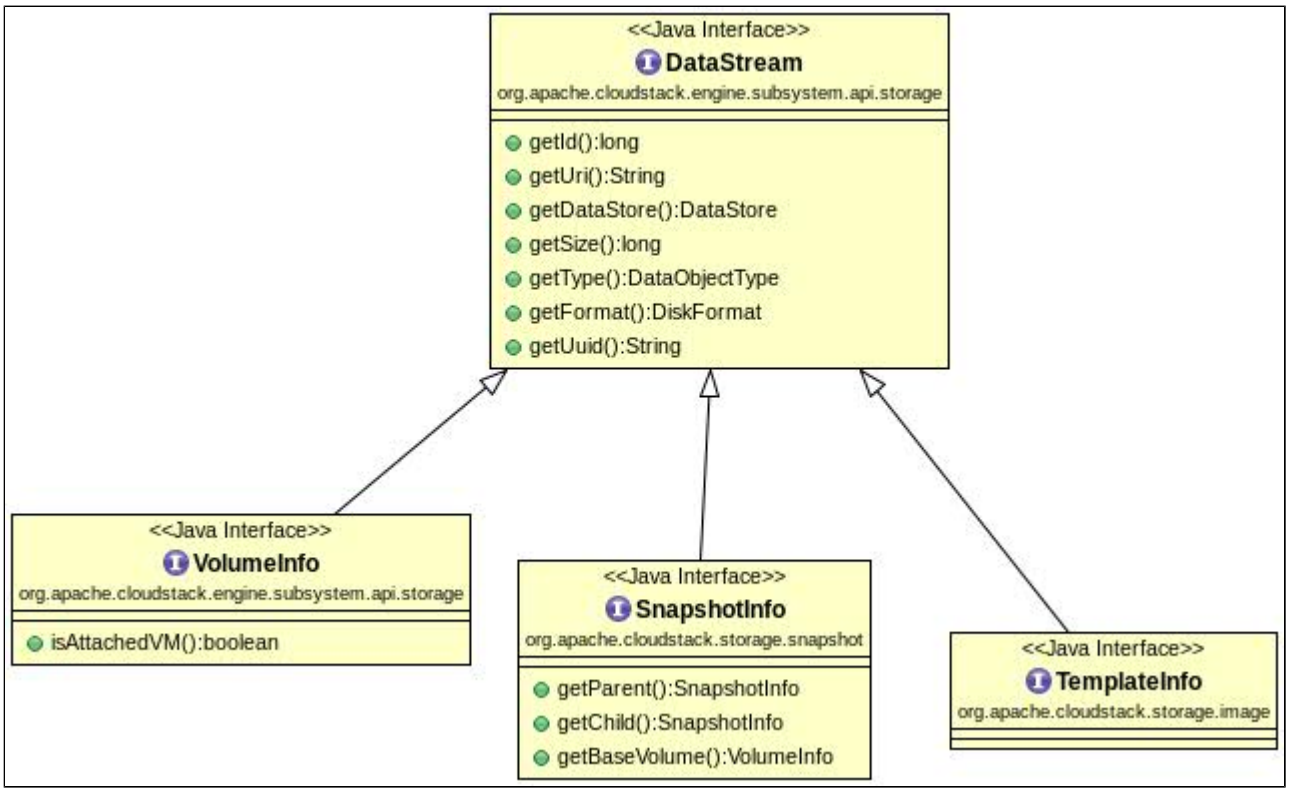
canCopy(DataStream, DataStream): it helps datamotionservice to make the decision on storage migration.

The two extra APIs for snapshot:

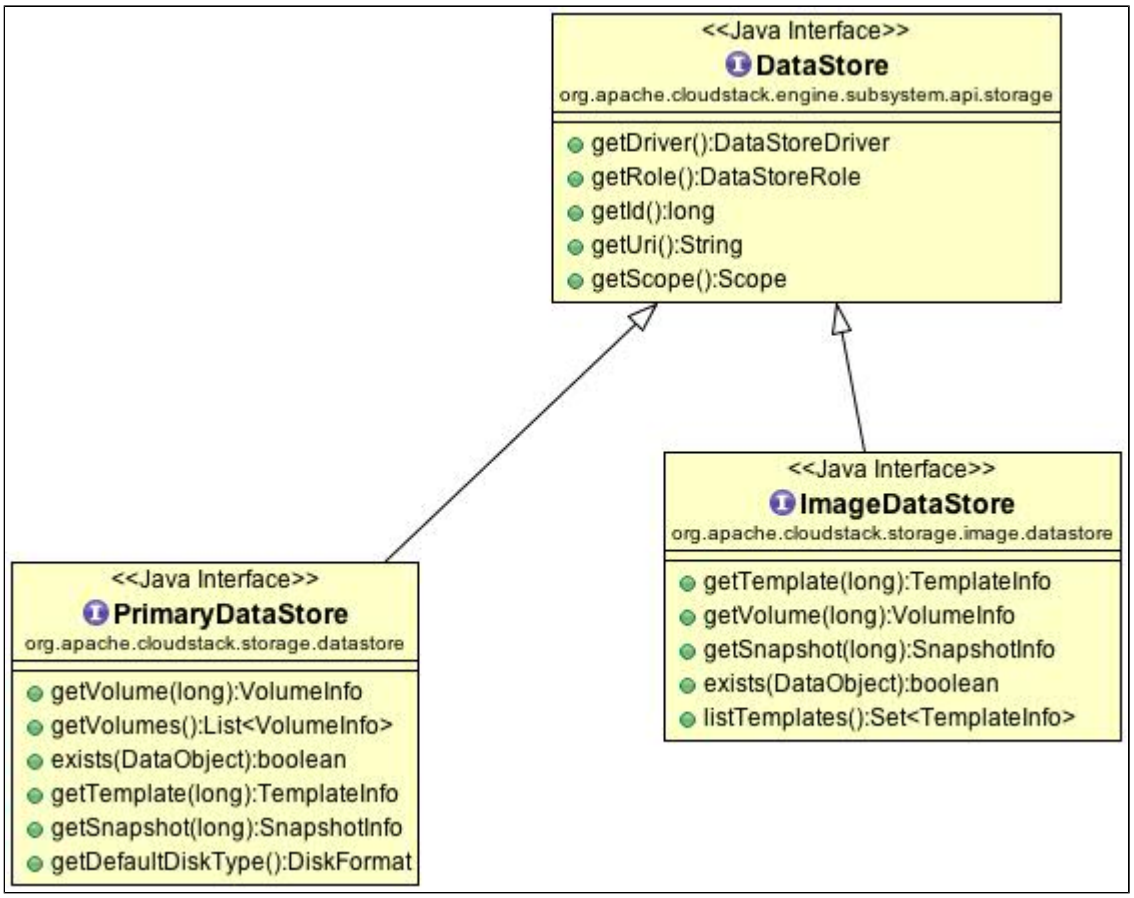
takeSnapshot(SnapshotInfo snapshot): take snapshot

revertSnapshot(SnapshotInfo snapshot): revert snapshot.

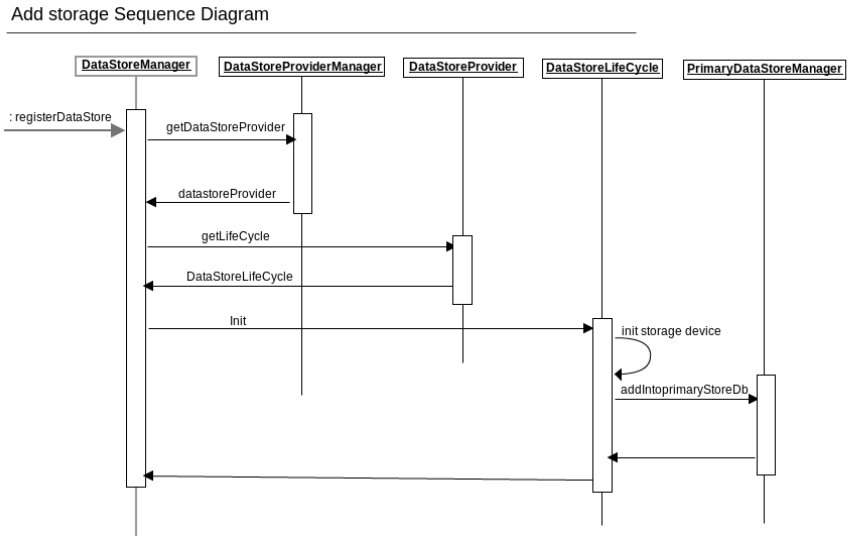
Data model:



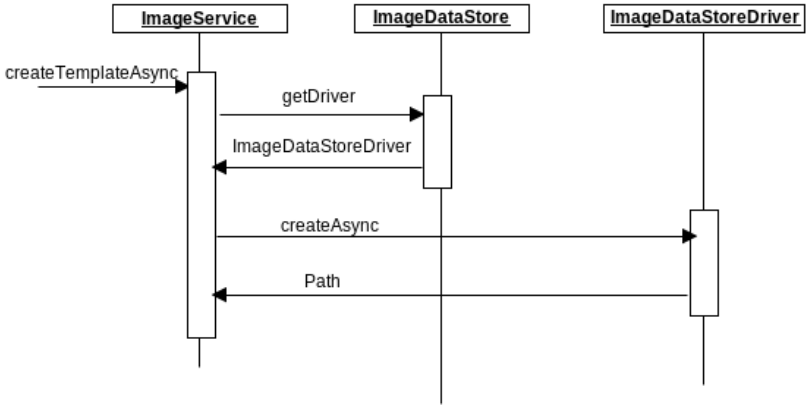
Data store model:



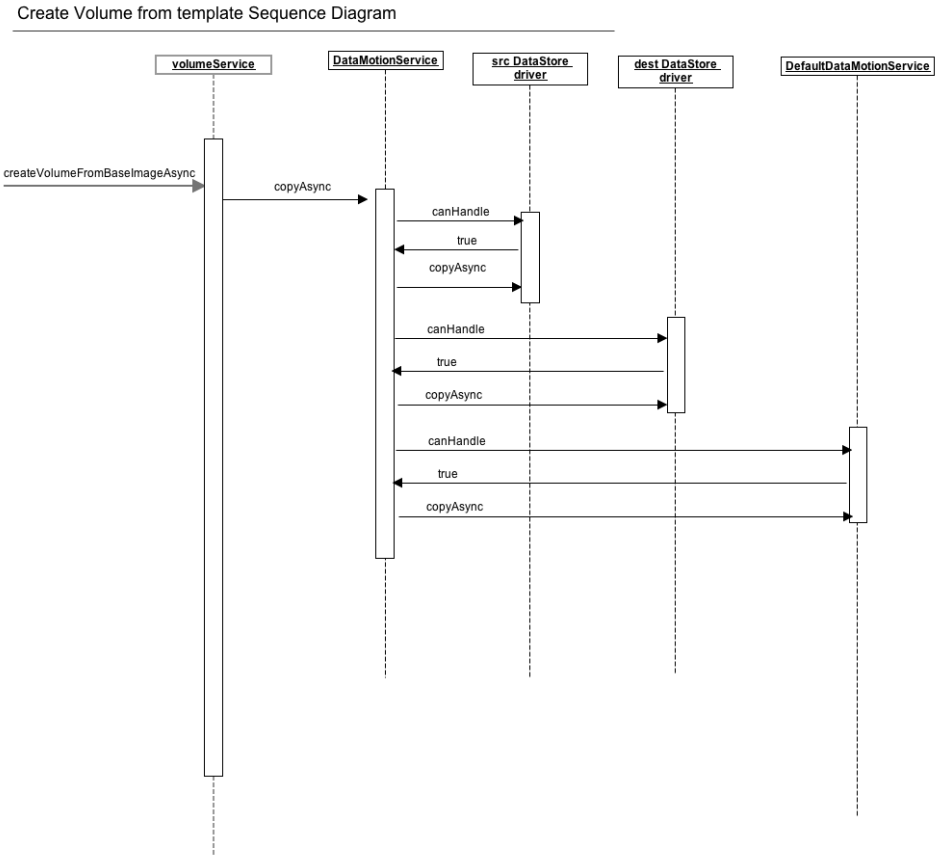
add primary storage:



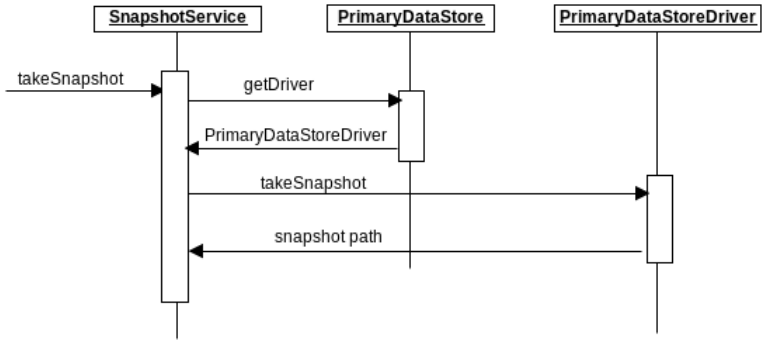
add template on image store:



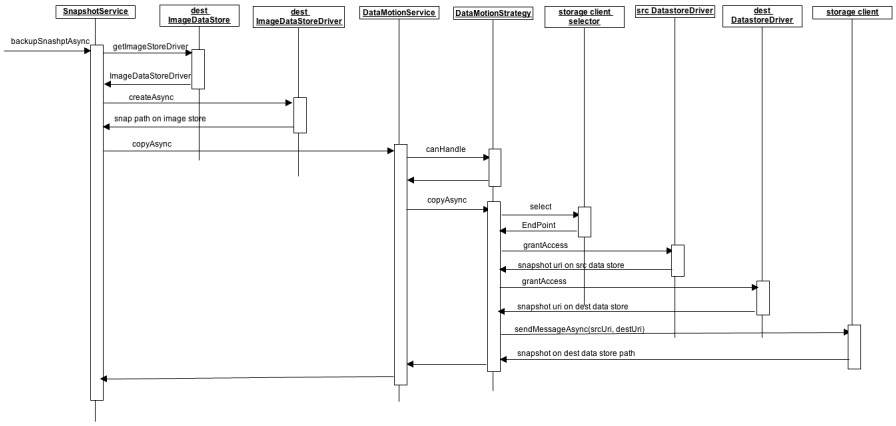
create volume from template:



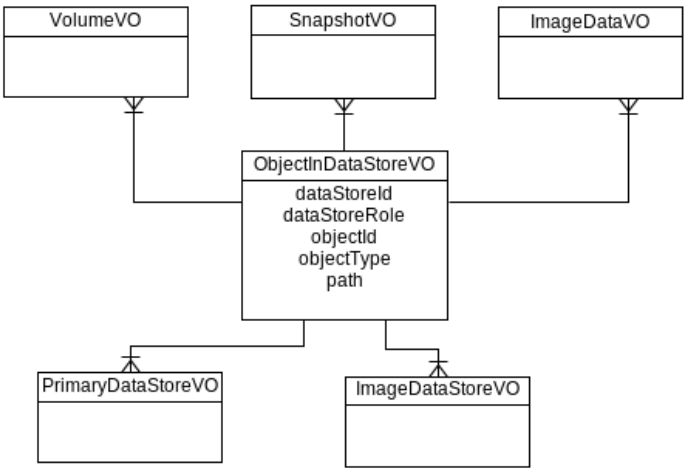
take snapshot



backup snapshot into imagestore



DB relationship:



Which interfaces should be implemented by device driver developer?

The storage subsystem will provide rich extension points for device driver developer.

At the minimal, device driver developer needs to implement three interfaces, as said before: the DataStoreProvider, DataStoreDriver and DataStoreLifeCycle.

There are other places can be extended:

TemplateInstallStrategy: customize how to install template on primary storage

DataMotionStrategy: customize how to move data from one place to another

EndPointSelector: how to select storage client based on src DataStream and dest DataStream, this algorithm will be used during data migration.

How to test

I use devcloud2(https://cwiki.apache.org/confluence/display/CLOUDSTACK/DevCloud) as the test environment.

Preparation before starting test:

1. Start a devcloud2 VM

2. create a nfs primary server inside devcloud vm: e.g. create a directory at /opt/storage/primarynfs

3. find out host uuid(xe host-list |grep uuid), local storage uuid(xe sr-list), devcloud host ip address(e.g. 192.168.56.10), sample template url(e.g http://download.cloud.com/templates/devcloud/defaulttemplates/5/ce5b212e-215a-3461-94fb-814a635b2215.vhd)

4. install testng plugin in eclipse(http://testng.org/doc/eclipse.html)

5. cp engine/storage/integration-test/test/resource/testng.xml engine/storage/integration-test/test/resource/testng2.xml

6. set testng2.xml as testng template: eclipse -> preferences -> testng -> template xml file -> add the path of testng2.xml, then save.

7. modify testng2.xml with the information you get from step 3.

Write an integration test, take an example of volumeServiceTest in storage-integration-test, which can register a template, and create a volume on it. In order to run the test, just right click on volumeServiceTest, select "run as testng"

+ Presentation at collab12: **storage.pdf**