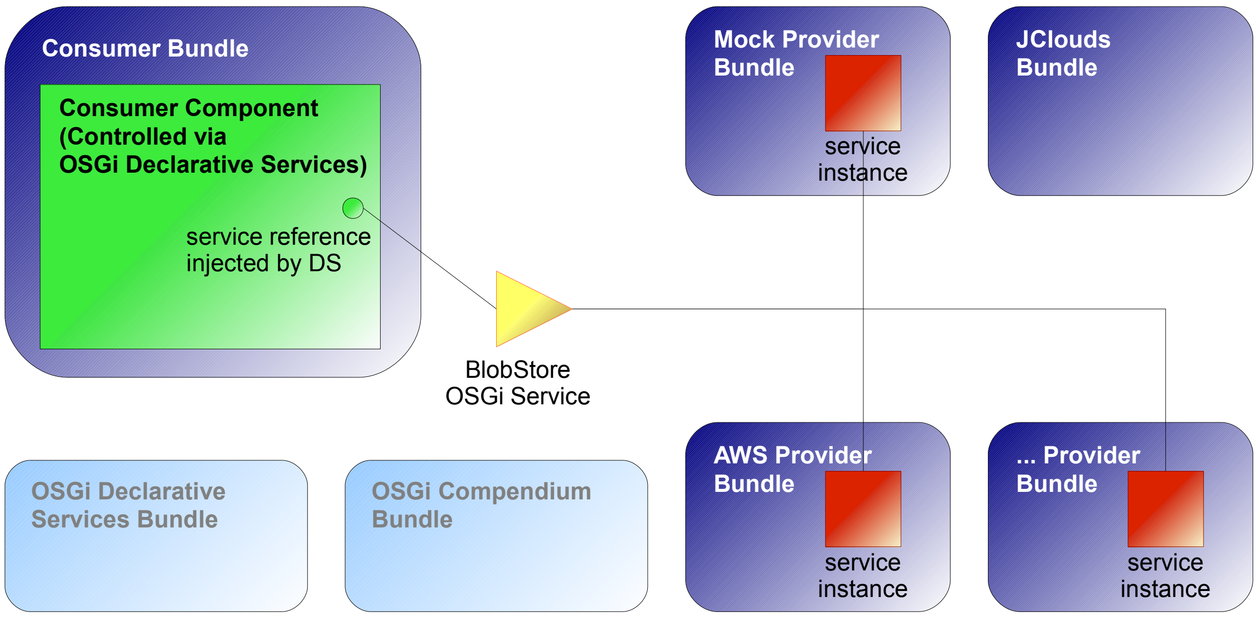
JClouds – OSGi

This demo shows how JClouds could be used in an OSGi context. The demo currently doesn't work yet as work needs to be carried out in JClouds to make it work.

This is just a writeup of how I think it *should* work – so feel free to comment :)

The demo consists of 3 bundles.
There is a consumer bundle which is really the important bundle. This bundle *consumes* (i.e. uses) the JClouds functionality. The demo uses the BlobStore API to store and retrieve data.
The consumer has no configuration wrt to which BlobStore is used. The BlobStore is obtained from the OSGi Service Registry; the consumer takes whatever BlobStore it can find there.

The actual BlobStore instance is put in the Service Registry by another bundle. In the demo there are two BlobStore provider bundles. One that provides a Mock BlobStore (nice for testing) and another one that provides a BlobStore for Amazon S3. So the BlobStore used by the consumer bundle depends on which provider bundle is installed.



The Consumer Bundle gets a reference to the BlobStore Service injected into its main component through OSGi Declarative Services (DS) [1]. It's not essential to use DS, you can also use plain vanilla OSGi ServiceTrackers but DS makes for a really nice and simple way to use the OSGi service, the entire code of the demo consumer is [2]:

```
public class Component {
  private BlobStore blobStore;

  public void activate() {
    try {
      // Create Container
      String containerName = "some-container";
      blobStore.createContainerInLocation(null, containerName);

      // Add Blob
      Blob blob = blobStore.newBlob("test");
      blob.setPayload("testdata");
      blobStore.putBlob(containerName, blob);

      // List Container
      for (StorageMetadata resourceMd : blobStore.list()) {
        if (resourceMd.getType() == StorageType.CONTAINER
            || resourceMd.getType() == StorageType.FOLDER) {
          // Use Map API
          Map<String, InputStream> containerMap = blobStore.getContext()
              .createInputStreamMap(resourceMd.getName());
          System.out.printf("  %s: %s entries%n", resourceMd.getName(),
              containerMap.size());
        }
      }
    } catch (Exception e) {
      e.printStackTrace();
    }
  }

  public void deactivate() {
    // can do cleanup here
  }

  // OSGi Declarative Services calls this to inject our BlobStore to use
  public void setBlobStore(BlobStore bs) {
    blobStore = bs;
  }

  // Used by OSGi Declarative Service to indicate a change in BlobStore
  // service
  public void unsetBlobStore(BlobStore bs) {
    blobStore = null;
  }
}
```

The BlobStore provider bundles currently use the following API to create a BlobStore instance:

```
  context = new BlobStoreContextFactory().createContext(type, accesskeyid, secretkey);
```

While this will probably work, there are potentially different approaches. One such approach could be to use the OSGi Configuration Admin Service [3] where a ManagedServiceFactory would be registered by the JClouds-OSGi integration which could make a BlobStore instance appear once the appropriate configuration is provided for it. This would give you a standard API to say something like: I'm creating a configuration instance for an S3 BlobStore with configuration parameters x, y and z. This could then automatically get the appropriate BlobStore service registered for you. Effectively this would mean that the provider bundles as mentioned in the demo are not needed any more. I think a Configuration Admin-based approach is *much* nicer than the current demo setup so I would suggest to ultimately move towards that...

The code

Source code is in the OSGi branch on my fork in github:
http://github.com/bosschaert/jclouds/tree/OSGi/demos/osgi

To run this demo...

Well, currently the demo doesn't work yet :) So it's really an attempt to focus the JClouds-OSGi work in order to get it to work.

Obviously, besides a plain vanilla OSGi Framework you need a JClouds bundle (or multiple bundles). AFAIK this doesn't exist yet.
Besides that you need the OSGi Declarative Services Bundle [4] and possibly the OSGi Compendium interfaces bundle [5]. If we go for a Config Admin approach we also need that bundle, but let's keep it small initially.

[1] Chapter 112 in the OSGi 4.2 compendium spec: http://www.osgi.org/Download/Release4V42
[2] https://github.com/bosschaert/jclouds/blob/OSGi/demos/osgi/osgi-demo-consumer/src/main/java/org/jclouds/demo/osgi/consumer/Component.java
[3] Chapter 104 in the OSGi 4.2 compendium spec
[4] http://felix.apache.org/site/downloads.cgi
[5] From http://www.osgi.org/Download/Release4V42 or via http://repo1.maven.org/maven2/org/osgi/org.osgi.compendium/4.2.0/org.osgi.compendium-4.2.0.jar