Amazon SimpleDB Typica Tutorial

Finnbarr P. Murphy
(fpm@fpmurphy.com)

While experimenting on how to prototype a Cloud XAM Storage System using Amazon Web Services (AWS), I experimented with the Amazon SimpleDB service using Java and the Typica AWS client library. I looked though the Internet for easy examples of how to use this library but found none that I liked – so here is my attempt at a tutorial on using the Typica library with the Amazon SimpleDB service.

To use the Typica library, you should download the latest version of the library. As of the date of this post it is typica.1.6. Install this library wherever you like; just make sure to add the location to your CLASSPATH. In Fedora, the standard location is */usr/share/java*. In addition to this library, the following Java libraries (jars) must also be installed if not already installed and in your CLASSPATH.

- *commons-httpclient*
- *commons-logging*
- *commons-codec*
- *jaxb*

To run the examples you need an Amazon Web Services SimpleDB account . You will need a credit card for this. Usage fees apply for using the service but they are relatively low. At present SimpleDB users pay no usage fees on the first 25 Machine Hours, 1 GB of Data Transfer, and 1 GB of Storage consumed every month. In most use cases, approximately 2 million GET or SELECT API requests can be completed per month before incurring any usage fees. Note that you have to specifically sign up for each of the Amazon Web Service products, i.e. Simple Storage Service (S3), Elastic Compute Cloud (EC2), Simple Queue Service (SQS), SimpleDB, etc. However, if you have signed up for one AWS product, adding another product is just a couple of key clicks.

If you cannot sign up for an SimpleDB account, there are a couple of free alternatives which claim to be *plug-compatible* with SimpleDB. I have not yet tried any of these services and the examples in this post will obviously have to be modified somewhat to use these services. I will discuss this further in the first example below. M/Gateway Developments M/DB is probably the most well known of such services.

Before we go any further, here is a quick review of what SimpleDB is and is not. According to Amazon, SimpleDB is a web service which provides the *core database functions of data indexing and querying in the cloud*. It is not a relational database; it is more of a metadata store. It is a form of distributed database written in Erlang. It requires no schema and automatically indexes your data. It consists of *domains*, *items* and *attributes*. A domain is a collection of items. Items are little hash tables containing attributes i.e. key-value pairs. As an aside, M/DB supports unlimited attributes per item and unlimited number and size of domains.

SimpleDB supports both SOAP and REST (Representational State Transfer). The Typica library uses REST. SimpleDB REST calls are made using HTTP GET requests. The Action query parameter provides the method called and the URI specifies the target of the call. Additional call parameters are specified as HTTP query parameters. A response message is generated for every request message. The response is an XML document that conforms to a schema. Check out the Getting Started Guide and the Developer Guide for further information,

SimpleDB has some major limitations which you should be aware of. The principal ones are a

maximum of 100 domains per account, 10GB maximum size for a domain, 256 attributes per item and a maximum attribute size of 1024 bytes. Queries also have limitations including a maximum of 2500 items returned per query, a 5 second maximum query runtime, and a maximum query response size of 1Mb for certain types of queries. See the SimpleDB Developers Guide for a full list of the limits.

Eventual Consistency is another interesting aspect of SimpleDB. Here is what Amazon have to say about this issue:

> Amazon SimpleDB keeps multiple copies of each domain. When data is written or updated [...] and Success is returned, all copies of the data are updated. However, it takes time for the update to propagate to all storage locations. The data will eventually be consistent, but an immediate read might not show the change. Consistency is usually reached within seconds, but a high system load or network partition might increase this time. Repeating a read after a short time should return the updated data.

Whether this is a real issue in practice, I do not know but it sure sounds like weasel words to me!

For our *Hello World* example, we will just create a SimpleDB domain.

```
import com.xerox.amazonws.sdb.Domain;
import com.xerox.amazonws.sdb.ListDomainsResult;
import com.xerox.amazonws.sdb.SDBException;
import com.xerox.amazonws.sdb.SimpleDB;

public class SDBexample1 {

   public static void main(String[] args) {
      // NOTE - replace AAA with your access key and SSS with your secret key
      SimpleDB simpleDB = new SimpleDB("AAA","SSS");

      try {
          simpleDB.createDomain("example1");
          System.out.println("created example1 domain");

          simpleDB.createDomain("example2");
          System.out.println("created example2 domain");

          ListDomainsResult list = simpleDB.listDomains();

          Iterator iterator = list.getDomainList().iterator();
          while (iterator.hasNext())
          {
              Domain domain = (Domain) iterator.next();
              System.out.println("List domain: " + domain.getName());
              simpleDB.deleteDomain(domain.getName());
          }

          System.out.println("deleted all domains");
      } catch (SDBException ex) {
          System.err.println("message : " + ex.getMessage());
          System.err.println("requestID : " + ex.getRequestId());
      } catch (Exception e) {
              System.err.println("Error occured: " + e.getMessage());
              e.printStackTrace();
      }
   }
}
```


Documentation for all the classes and methods in the Typica library can be found online. The application connects to the Amazon SimpleDB service by means of the *SimpleDB(accessIdString, secretKeyString)* constructor. It then creates a new domain by called the *createDomain(nameString)* method. Any Typica SimpleDB exceptions are handled by the *SDBException()* exceptions constructor.

If you plan to use this example and the subsequent examples with the M/DB service, you will need to modify the *SimpleDB* constructor to supply the server host string and possibly the port number. See the Typica SimpleDB class documentation for further information.

Here is the output. when compiled and run.

```
$ javac SDBexample.java
$ java SDBexample
created example domain
$
```

For our next example, we will create two SimpleDB domains, list all our domains, and delete the domains.

```
import java.util.Iterator;

import com.xerox.amazonws.sdb.Domain;
import com.xerox.amazonws.sdb.ListDomainsResult;
import com.xerox.amazonws.sdb.SDBException;
import com.xerox.amazonws.sdb.SimpleDB;


public class SDBexample1 {

   public static void main(String[] args) {
      // NOTE - replace AAA with your access key and SSS with your secret key
      SimpleDB simpleDB = new SimpleDB("AAA","SSS");

      try {
          simpleDB.createDomain("example1");
          System.out.println("created example1 domain");

          simpleDB.createDomain("example2");
          System.out.println("created example2 domain");

          ListDomainsResult list = simpleDB.listDomains();

          Iterator iterator = list.getDomainList().iterator();
          while (iterator.hasNext())
          {
              Domain domain = (Domain) iterator.next();
              System.out.println("List domain: " + domain.getName());
              simpleDB.deleteDomain(domain.getName());
          }

          System.out.println("deleted all domains");
      } catch (SDBException ex) {
          System.err.println("message : " + ex.getMessage());
          System.err.println("requestID : " + ex.getRequestId());
      } catch (Exception e) {
              System.err.println("Error occured: " + e.getMessage());
              e.printStackTrace();
      }
   }
```



```
}
```

It uses the *listDomains()* method to return a list of all the current domains, the *getDomainList()* to set up iteration of the returned list of domains and the *deleteDomain()* method to delete a domain.

Here the the output from running this example.

```
created example1 domain
created example2 domain
List domain: example
List domain: example1
List domain: example2
deleted all domains
```

Note that it listed and deleted the *example* domain which we created in the previous example.

In the next example we use a properties file *aws.properties* to store the AWS *accessid* and *secretkey* strings. We encrypt communications between the application and AWS by adding a third parameter *true* to the *SimpleDB()* constructor. We also retrieve and print out the domain metadata.

```
import java.io.FileInputStream;
import java.io.IOException;
import java.io.FileNotFoundException;
import java.util.Iterator;
import java.util.Properties;

import com.xerox.amazonws.sdb.Domain;
import com.xerox.amazonws.sdb.DomainMetadataResult;
import com.xerox.amazonws.sdb.ListDomainsResult;
import com.xerox.amazonws.sdb.SDBException;
import com.xerox.amazonws.sdb.SimpleDB;

public class SDBexample3 {
   private static String AWS_PROPERTIES = "aws.properties";
   private static String AWS_ACCESSID = "aws.accessId";
   private static String AWS_SECRETKEY = "aws.secretKey";

   private static String domainName = "test1";
   private static String awsAccessId = null;
   private static String awsSecretKey = null;

   private static Properties props;

   public static void main(String[] args) {
      /* get the accessid and secretkey strings from the properties file*/
      try {
           props = new Properties();
           props.load(new FileInputStream(AWS_PROPERTIES));

           awsAccessId = props.getProperty(AWS_ACCESSID);
           if (awsAccessId == null || awsAccessId.trim().length() == 0)
           {
               System.out.println("Access key found or not set");
               System.exit((int) 1);
           }
           awsSecretKey = props.getProperty(AWS_SECRETKEY);
           if (awsSecretKey == null || awsSecretKey.trim().length() == 0)
           {
               System.out.println("Secret key either not found or not set");
               System.exit((int) 1);
           }
       } catch (FileNotFoundException e) {
```



```
           System.err.println("Could not find properties file");
           System.exit((int) 1);
        } catch (IOException e) {
           System.err.println("Could not access properties file");
           System.exit((int) 1);
        }

      /* encrypt communications */
      SimpleDB simpleDB = new SimpleDB(awsAccessId, awsSecretKey, true);
      try {
          simpleDB.createDomain(domainName);
          System.out.println("created " + domainName +  " domain");

          ListDomainsResult list = simpleDB.listDomains();

          Iterator iterator = list.getDomainList().iterator();
          while (iterator.hasNext())
          {
              Domain domain = (Domain) iterator.next();
              DomainMetadataResult metadata = domain.getMetadata();

              System.out.println("\nDomain Metadata for : " + domain.getName());
              System.out.println("     ItemCount: " + metadata.getItemCount());
              System.out.println("     AttributeNameCount: " + metadata.getAttributeNameCo
unt());
              System.out.println("     AttributeValueCount: " + metadata.getAttributeValue
Count());
              System.out.println("     ItemNamesSizeBytes: " + metadata.getItemNamesSizeBy
tes());
              System.out.println("     AttributeNamesSizeBytes: " + metadata.getAttributeN
amesSizeBytes());
              System.out.println("     AttributeValuesSizeBytes: " + metadata.getAttribute
ValuesSizeBytes());
              System.out.println("     Timestamp: " + metadata.getTimestamp() + "\n");
          }

          simpleDB.deleteDomain(domainName);
          System.out.println("deleted " + domainName + " domain");
      } catch (SDBException ex) {
          System.err.println("message : " + ex.getMessage());
          System.err.println("requestID : " + ex.getRequestId());
      } catch (Exception e) {
              System.err.println("Error occured: " + e.getMessage());
              e.printStackTrace();
      }
   }
}
```

Here the the output from running this example. Note that most of the values are zero because we have not yet added any items or attributes.

```

created test1 domain

Domain Metadata for : test1
     ItemCount: 0
     AttributeNameCount: 0
     AttributeValueCount: 0
     ItemNamesSizeBytes: 0
     AttributeNamesSizeBytes: 0
     AttributeValuesSizeBytes: 0
     Timestamp: Sat Aug 08 19:18:28 GMT-05:00 2009

deleted test1 domain
```


The next example is our first example to get (create) an *item* using the *getItem()* method and add *attributes* to that *item* using the *putAttrributes()* method. It includes a 2 second delay (sleep) to guard against the *eventual consistency* issue before attempting to retrieve the domain metadata. I found that a Java exception was sometimes thrown unless I added this delay. The delay I picked was arbitrary; a smaller delay may suffice. It also includes a different way of retrieving the *accessid* and *secretkey* strings using *getClassLoader()*.

```
import java.util.ArrayList;
import java.util.Iterator;
import java.util.Properties;
import java.util.List;

import com.xerox.amazonws.sdb.Item;
import com.xerox.amazonws.sdb.ItemAttribute;
import com.xerox.amazonws.sdb.Domain;
import com.xerox.amazonws.sdb.DomainMetadataResult;
import com.xerox.amazonws.sdb.ListDomainsResult;
import com.xerox.amazonws.sdb.SDBException;
import com.xerox.amazonws.sdb.SimpleDB;

public class SDBexample4 {
   private static String AWS_PROPERTIES = "aws.properties";
   private static String AWS_ACCESSID = "aws.accessId";
   private static String AWS_SECRETKEY = "aws.secretKey";

   private static String domainName = "test4";

   public static void main(String[] args) {

      try {
           /* get the accessid and secretkey strings from the properties file*/
           Properties props = new Properties();
           props.load(SDBexample4.class.getClassLoader().getResourceAsStream(AWS_PROPERTIE
S));

           String awsAccessId = props.getProperty(AWS_ACCESSID);
           String awsSecretKey = props.getProperty(AWS_SECRETKEY);

           if ((awsAccessId == null || awsAccessId.trim().length() == 0) ||
               (awsSecretKey == null || awsSecretKey.trim().length() == 0))
           {
               System.out.println("Either access key or Secret key not found or not set");
               System.exit((int) 1);
           }
          SimpleDB simpleDB = new SimpleDB(awsAccessId, awsSecretKey, true);

          Domain dom = simpleDB.createDomain(domainName);
          System.out.println("created " + dom.getName() +  " domain");

          // new item (called 'xset1')
          Item i = dom.getItem("xset1");

          List list = new ArrayList();

          list.add(new ItemAttribute("test1", "value1", false));
          list.add(new ItemAttribute("test2", "value2", false));
          list.add(new ItemAttribute("test3", "value3", false));

          // add the 3 attributes to xset1
          i.putAttributes(list);

          // sleep for 2 seconds to handle "eventual consistency'
          Long stoptime = 2000L;
          System.out.print("going to sleep for 2 seconds ");
          try {
```



```
            Thread.sleep(stoptime);
          } catch (InterruptedException e) {
               e.printStackTrace();
               System.exit((int) 1);
          }
          System.out.println("Done!");

          // get the domain metadata
          DomainMetadataResult metadata = dom.getMetadata();

          System.out.println("\nDomain Metadata for : " + dom.getName());
          System.out.println("     ItemCount: " + metadata.getItemCount());
          System.out.println("     AttributeNameCount: " + metadata.getAttributeNameCount()
);
          System.out.println("     AttributeValueCount: " + metadata.getAttributeValueCoun
t());
          System.out.println("     ItemNamesSizeBytes: " + metadata.getItemNamesSizeBytes()
);
          System.out.println("     AttributeNamesSizeBytes: " + metadata.getAttributeNames
SizeBytes());
          System.out.println("     AttributeValuesSizeBytes: " + metadata.getAttributeValu
esSizeBytes());
          System.out.println("     Timestamp: " + metadata.getTimestamp() + "\n");

          simpleDB.deleteDomain(domainName);
          System.out.println("deleted " + domainName + " domain");
      } catch (SDBException ex) {
          System.err.println("message : " + ex.getMessage());
          System.err.println("requestID : " + ex.getRequestId());
      } catch (Exception e) {
              System.err.println("Error occured: " + e.getMessage());
              e.printStackTrace();
      }
   }
}
```

Here is the output for this example:

```

created test5 domain
going to sleep for 2 seconds Done!

Domain Metadata for : test4
     ItemCount: 1
     AttributeNameCount: 3
     AttributeValueCount: 3
     ItemNamesSizeBytes: 5
     AttributeNamesSizeBytes: 15
     AttributeValuesSizeBytes: 18
     Timestamp: Sat Aug 08 21:48:07 GMT-05:00 2009

deleted test5 domain
```

The next example creates two items (*xset1* and *xset2*) and adds 10 attributes to each item. It also outputs the *request identifier* and *box usage* for each of the two *putAttributes* requests. Every request generates a response message and these two items are included as part of this message. Box usage is the measure of machine resources consumed by a request. It does not include bandwidth or storage. It is reported as the portion of a machine hour used to complete a particular request.

```

import java.util.ArrayList;
import java.util.Iterator;
import java.util.Properties;
```



```
import java.util.List;

import com.xerox.amazonws.sdb.Item;
import com.xerox.amazonws.sdb.ItemAttribute;
import com.xerox.amazonws.sdb.Domain;
import com.xerox.amazonws.sdb.DomainMetadataResult;
import com.xerox.amazonws.sdb.ListDomainsResult;
import com.xerox.amazonws.sdb.SDBException;
import com.xerox.amazonws.sdb.SDBResult;
import com.xerox.amazonws.sdb.SimpleDB;

public class SDBexample5 {
   private static String AWS_PROPERTIES = "aws.properties";
   private static String AWS_ACCESSID = "aws.accessId";
   private static String AWS_SECRETKEY = "aws.secretKey";

   private static String domainName = "test5";

   public static void main(String[] args) {
     try {

           /* get the accessid and secretkey strings from the properties file*/
           Properties props = new Properties();
           props.load(SDBexample4.class.getClassLoader().getResourceAsStream(AWS_PROPERTIE
S));

           String awsAccessId = props.getProperty(AWS_ACCESSID);
           String awsSecretKey = props.getProperty(AWS_SECRETKEY);

           if ((awsAccessId == null || awsAccessId.trim().length() == 0) ||
               (awsSecretKey == null || awsSecretKey.trim().length() == 0))
           {
               System.out.println("Either access key or Secret key not found or not set");
               System.exit((int) 1);
           }

          SimpleDB simpleDB = new SimpleDB(awsAccessId, awsSecretKey, true);

          Domain dom = simpleDB.createDomain(domainName);
          System.out.println("created " + dom.getName() +  " domain");

          Item i = dom.getItem("xset1");

          List list = new ArrayList();

          list.add(new ItemAttribute("test10", "value10", false));
          list.add(new ItemAttribute("test11", "value11", false));
          list.add(new ItemAttribute("test12", "value12", false));
          list.add(new ItemAttribute("test13", "value13", false));
          list.add(new ItemAttribute("test14", "value14", false));
          list.add(new ItemAttribute("test15", "value15", false));
          list.add(new ItemAttribute("test16", "value16", false));
          list.add(new ItemAttribute("test17", "value17", false));
          list.add(new ItemAttribute("test18", "value18", false));
          list.add(new ItemAttribute("test19", "value19", false));

          SDBResult result = i.putAttributes(list);
          System.out.println("Item Identifier: " + i.getIdentifier());
          System.out.println("     Request ID: " + result.getRequestId());
          System.out.println("      Box Usage: " + result.getBoxUsage());
          i = dom.getItem("xset2");
          list = new ArrayList();

          list.add(new ItemAttribute("test20", "value20", false));
          list.add(new ItemAttribute("test21", "value21", false));
          list.add(new ItemAttribute("test22", "value22", false));
          list.add(new ItemAttribute("test23", "value23", false));
          list.add(new ItemAttribute("test24", "value24", false));
```



```
          list.add(new ItemAttribute("test25", "value25", false));
          list.add(new ItemAttribute("test26", "value26", false));
          list.add(new ItemAttribute("test27", "value27", false));
          list.add(new ItemAttribute("test28", "value28", false));
          list.add(new ItemAttribute("test29", "value29", false));

          result = i.putAttributes(list);
          System.out.println("Item Identifier: " + i.getIdentifier());
          System.out.println("     Request ID: " + result.getRequestId());
          System.out.println("      Box Usage: " + result.getBoxUsage());

          // sleep for 2 seconds to handle "eventual consistency'
          Long stoptime = 2000L;
          System.out.print("going to sleep for 2 seconds ");
          try {
              Thread.sleep(stoptime);
          } catch (InterruptedException e) {
               e.printStackTrace();
               System.exit((int) 1);
          }
          System.out.println("Done!");

          ListDomainsResult list1 = simpleDB.listDomains();
          Iterator iterator = list1.getDomainList().iterator();
          while (iterator.hasNext())
          {
              Domain domain = (Domain) iterator.next();
              DomainMetadataResult metadata = domain.getMetadata();

              System.out.println("\nDomain Metadata for : " + domain.getName());
              System.out.println("     ItemCount: " + metadata.getItemCount());
              System.out.println("     AttributeNameCount: " + metadata.getAttributeNameCo
unt());
              System.out.println("     AttributeValueCount: " + metadata.getAttributeValue
Count());
              System.out.println("     ItemNamesSizeBytes: " + metadata.getItemNamesSizeBy
tes());
              System.out.println("     AttributeNamesSizeBytes: " + metadata.getAttributeN
amesSizeBytes());
              System.out.println("     AttributeValuesSizeBytes: " + metadata.getAttribute
ValuesSizeBytes());
              System.out.println("     Timestamp: " + metadata.getTimestamp() + "\n");
          }

      } catch (SDBException ex) {
          System.err.println("message : " + ex.getMessage());
          System.err.println("requestID : " + ex.getRequestId());
      } catch (Exception e) {
          System.err.println("Error occured: " + e.getMessage());
          e.printStackTrace();
      }
   }
}
```

Here is the output for this example:

```
created test5 domain
Item Identifier: xset1
     Request ID: 34e6df86-5f2b-4bcf-312d-22b595a87fe7
      Box Usage: 0.0000221907
Item Identifier: xset2
     Request ID: dd66f222-de70-3d06-8ce5-7842e5d22d1d
      Box Usage: 0.0000221907
going to sleep for 4 seconds Done!
```



```
Domain Metadata for : test5
     ItemCount: 2
     AttributeNameCount: 20
     AttributeValueCount: 20
     ItemNamesSizeBytes: 10
     AttributeNamesSizeBytes: 120
     AttributeValuesSizeBytes: 140
     Timestamp: Sun Aug 09 08:44:19 GMT-05:00 2009
```

For our final example, we will retrieve and print the attributes stored in the *test5* domain *xset1* item by the previous example.

```
import java.util.ArrayList;
import java.util.Iterator;
import java.util.Properties;
import java.util.List;

import com.xerox.amazonws.sdb.Item;
import com.xerox.amazonws.sdb.ItemAttribute;
import com.xerox.amazonws.sdb.Domain;
import com.xerox.amazonws.sdb.DomainMetadataResult;
import com.xerox.amazonws.sdb.ListDomainsResult;
import com.xerox.amazonws.sdb.SDBException;
import com.xerox.amazonws.sdb.SDBResult;
import com.xerox.amazonws.sdb.SimpleDB;

public class SDBexample6 {
   private static String AWS_PROPERTIES = "aws.properties";
   private static String AWS_ACCESSID = "aws.accessId";
   private static String AWS_SECRETKEY = "aws.secretKey";

   private static String domainName = "test5";

   public static void main(String[] args) {

      try {

           /* get the accessid and secretkey strings from the properties file*/
           Properties props = new Properties();
           props.load(SDBexample4.class.getClassLoader().getResourceAsStream(AWS_PROPERTIE
S));

           String awsAccessId = props.getProperty(AWS_ACCESSID);
           String awsSecretKey = props.getProperty(AWS_SECRETKEY);
          if ((awsAccessId == null || awsAccessId.trim().length() == 0) ||
               (awsSecretKey == null || awsSecretKey.trim().length() == 0))
           {
               System.out.println("Either access key or Secret key not found or not set");
               System.exit((int) 1);
           }

          SimpleDB simpleDB = new SimpleDB(awsAccessId, awsSecretKey, true);

          Domain dom = simpleDB.getDomain(domainName);
          System.out.println("opened " + dom.getName() +  " domain");

          Item i = dom.getItem("xset1");

          List attrs = i.getAttributes();
          for (ItemAttribute attr : attrs) {
              System.out.println(attr.getName() + " = " + attr.getValue());
          }

      } catch (SDBException ex) {
          System.err.println("message : " + ex.getMessage());
```



```
            System.err.println("requestID : " + ex.getRequestId());
        } catch (Exception e) {
                System.err.println("Error occured: " + e.getMessage());
                e.printStackTrace();
        }
    }
}
```

Here is the output from this example:

```
opened test5 domain
test10 = value10
test12 = value12
test11 = value11
test14 = value14
test13 = value13
test16 = value16
test15 = value15
test18 = value18
test17 = value17
test19 = value19
```

Well that is all of the examples for now. If you compile, run and play with these examples, you should end up with a good understanding of how to use the Typica library to access and manipulate Amazon SimpleDB. I plan to discuss how to query Amazon SimpleDB using this library in a future post.

Can the Typica library be used as part of a AWS Cloud VIM implementation? Absolutely. Can Amazon SimpleDB be used to implement a Cloud XAM Storage System? Not by itself. However if Amazon S3 is used to provide persistence for XStreams and Amazon SimpleDB is used to provide persistence for XSet metadata, the answer is yes to a basic prototype of a Cloud XAM Storage System but no for a production quality Cloud XAM Storage System due the *eventual consistency* issue as well as the current limitations in the number of attributes, size of domains, no datatyping and seconds a query may run.

