By Bipin Chandra & Maneesh Sharma

Introduction:

In today's world, multiple businesses need to communicate with each other to meet their needs. As the transactional communication between different applications grows, there is a proportional growth in the data that needs to be stored by each of them

There could be certain limitations/use cases with each application/enterprise which could force them to find an alternate plan for their data storage. For e.g.

a. Low scale enterprises not having enough bandwidth to accommodate for storing data over network.

b. Data associated with the actual transactional data which could be repetitive in nature and not accessed frequently.

c. Paying a lump sum amount each year for storage but actual data stored is lesser.

d. Big data analysis and many more.

Amazon hosts a solution to address such problems via providing web services called AWS3 i.e. Simple storage service.

Amazon S3 is easy to use object storage, with a simple web service interface to store and retrieve any amount of data from anywhere on the web. With Amazon S3, you pay only for the storage you actually use. There is no minimum fee and no setup cost.

This enables the companies to focus more on the scalability and performance of their application than the cost and other factors involved in storing data.

This step by step documentation provides a seamless integration of AWS3 with IBM Sterling B2B Integrator. Sterling Integrator is a business process-centric transaction engine which enables enterprises to server all of their B2B integration needs through a single, flexible B2B gateway. We make use of business processes and services to integrate with AWS3 which can be enhanced to perform all the AWS3 supported operations, like – put files, get files, delete files, create a directory and many more.

Assessing current technology can help companies understand basic functional needs of various IT initiatives, discover new capabilities they can already deliver. In many cases, existing B2B Gateway can extend itself to enable the success of other non-Ecommerce IT initiatives.

Benefits:

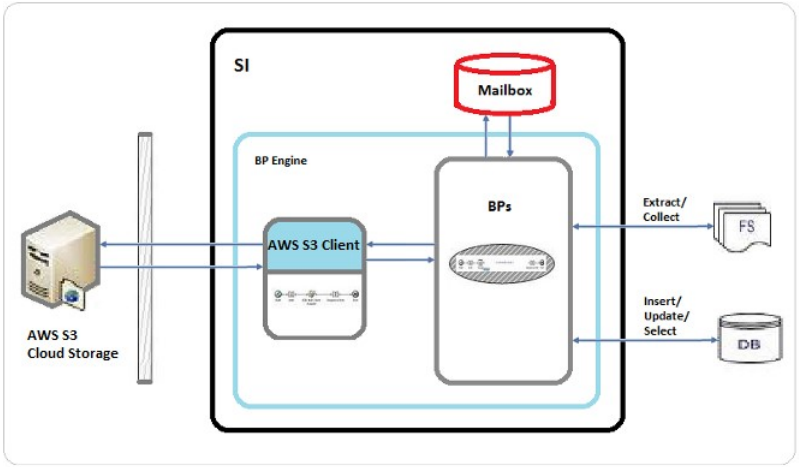
- Secure, durable, highly-scalable cloud based storage solution.
- Store and retrieve any amount of data.
- Various range of storage classes designed for different use cases.
- Supports event notification in case of upload/download.
- Easy to use as it comes with an interactive web based UI console.

AWS S3 Client Service:

AWS3 Client Service from Sterling Integrator creates seamless connectivity to the cloud storage environment hosted by Amazon. It is a scalable, reliable, and portable storage solution.

SI users can easily get, put and delete files and can perform many more operations.

AWS S3-SI Integration Architecture Diagram:



We will explain how to use Amazon's S3 storage with the Java API provided by Amazon. The example shows you how to create a bucket, list it's content, create a folder into a bucket, upload a file, give the file a public access and finally how to delete all this items.

Setting Up Your Project

> You will need the AWS SDK for Java for this example to work. If you didn't downloaded the SDK by now, please download it here. You will also need to integrate the .JAR files from the archive into your project.

> Create a user in Amazon IAM (https://console.aws.amazon.com/iam) if you don't have one. Here you will get a "Access key" and "Secret Access Key". You will need this credentials to connect to S3.

Authenticate with Amazon S3

There are 4 different methods to authenticate your requests against Amazon S3

1. **Use the default credential profiles file –** this is the recommend way by Amazon. Create a file with following structure and fill in the access keys:

```
# Move this credentials file to (~/.aws/credentials)
# after you fill in your access and secret keys in the default profile

# WARNING: To avoid accidental leakage of your credentials,
# DO NOT keep this file in your source directory.
[default]
aws_access_key_id=
aws_secret_access_key=
```

Save this file under the file-name "credentials" in your .aws folder by default C:\Users\user\.aws\credentials for Windows users or your home directory in Linux.

If you use this method you can create a Credentials object in your code like this:

```
AWSCredentials credentials = new ProfileCredentialsProvider().getCredentials();
```

2. **Use Environment Variables –** set the values of following environment variables in your system AWS_ACCESS_KEY_ID and AWS_SECRET_ACCESS_KEY

3. **Java System Properties –** aws.accessKeyId and aws.secretKey. UseSystemPropertiesCredentialsProvider to load the variables in your program

4. **Programmatically set credentials –** in this example I will use this method, because it is easier to follow

We will need to set following two properties in integrationservices.properties file under <INSTALL_DIR>/properties folder.

awss3.accessKey=<Your_Access_Key>

awss3.secretKey=<Your_Secret_Key>

Use following in your code:

```
AWSCredentials credentials = new BasicAWSCredentials("YourAccessKeyID", "YourSecretAccessKey");
```

Create S3 Client

To be able to communicate with S3 you have to use an implementation of AmazonS3. You will use the instance to address requests to the server

```
AmazonS3 s3client = new AmazonS3Client(credentials);
```

Create Bucket

Buckets must have unique names within the whole S3 realm

```
String bucketName = "javatutorial-net-example-bucket";
s3client.createBucket(bucketName);
```

List Buckets

You can get the list off all buckets like this

```
for (Bucket bucket : s3client.listBuckets()) {
System.out.println(" - " + bucket.getName());
}
```

Create Folder in S3 Bucket

Use this code to create an empty folder in your bucket

```
public static void createFolder(String bucketName, String folderName, AmazonS3 client) {
// create meta-data for your folder and set content-length to 0
ObjectMetadata metadata = new ObjectMetadata();
metadata.setContentLength(0);

// create empty content
InputStream emptyContent = new ByteArrayInputStream(new byte[0]);
```

```
// create a PutObjectRequest passing the folder name suffixed by /
PutObjectRequest putObjectRequest = new PutObjectRequest(bucketName,
folderName + SUFFIX, emptyContent, metadata);

// send request to S3 to create folder
client.putObject(putObjectRequest);
}
```

Upload file in S3

If you want to upload a file into a folder use this

```
String fileName = folderName + SUFFIX + "testvideo.mp4";
s3client.putObject(new PutObjectRequest(bucketName, fileName,
new File("C:\\Users\\user\\Desktop\\testvideo.mp4")));
```

Just remove the folder and the suffix in the file-name to upload directly to the bucket. If you want to make your file public (files are private by default in Amazon S3) set this to your PutObjectRequest (see the complete example below for more details)

```
.withCannedAcl(CannedAccessControlList.PublicRead)
```

Deleting Files, Folders and Buckets

To delete a bucket use this. Buckets must be empty or you can't delete them

```
s3client.deleteBucket(bucketName);
```

To delete files use:

```
s3client.deleteObject(bucketName, fileName);
```

To delete a folder you have to delete all files in it first. Please look at the complete example below for more info.

Complete Java implementation:

This is how we create a java service for AWS S3 Client. Customer can use this as a default implementation and can build their requirement on top of it.

```
package com.sterlingcommerce.woodstock.services.integration;

import jarjar.orgapachecommonsio.IOUtils;

import java.io.BufferedOutputStream;
import java.io.ByteArrayInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.sql.SQLException;
import java.util.List;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3Client;
import com.amazonaws.services.s3.model.Bucket;
import com.amazonaws.services.s3.model.CannedAccessControlList;
import com.amazonaws.services.s3.model.GetObjectRequest;
import com.amazonaws.services.s3.model.ObjectMetadata;
import com.amazonaws.services.s3.model.PutObjectRequest;
import com.amazonaws.services.s3.model.S3Object;
import com.amazonaws.services.s3.model.S3ObjectSummary;
import com.sterlingcommerce.woodstock.mailbox.MailboxException;
import com.sterlingcommerce.woodstock.mailbox.MailboxFactory;
import com.sterlingcommerce.woodstock.mailbox.repository.IRepository;
import com.sterlingcommerce.woodstock.mailbox.repository.Message;
import com.sterlingcommerce.woodstock.services.IService;
import com.sterlingcommerce.woodstock.util.frame.log.LogService;
import com.sterlingcommerce.woodstock.util.frame.log.Logger;
import com.sterlingcommerce.woodstock.workflow.Document;
import com.sterlingcommerce.woodstock.workflow.DocumentNotFoundException;
import com.sterlingcommerce.woodstock.workflow.WorkFlowContext;
import com.sterlingcommerce.woodstock.workflow.WorkFlowException;
import com.sterlingcommerce.woodstock.util.frame.Manager;


public class AWSS3ClientService implements IService {

    private Document primaryDoc = null;

    private InputStream docAsStream = null;

    protected static Logger log;
```

```
static {
  log = LogService.getLogger("integrationservices");
}

private static boolean isOK = true;
   private static final String SUFFIX = "/";

public WorkFlowContext processData(WorkFlowContext wfc)
      throws WorkFlowException {

   isOK = true;

   try {
      wfc.harnessRegister();
      String action = wfc.getParm("operation");//PUT, GET, DELETE, MKDIR
      String local_path = wfc.getParm("local_path");//Path to read file from for uploading
      String messageId = wfc.getParm("messageId");//Message ID for mailbox
      String s3_bucketname = wfc.getParm("s3_bucketname");//Bucket name in S3 area
      String s3_key = wfc.getParm("s3_key");  //Name with which resource resides in S3 area
      String s3_folderName = wfc.getParm("s3_foldername");//Folder name OR structure

      log.log("AWSS3ClientService.processData(): local_path:" + local_path);
      log.log("AWSS3ClientService.processData(): s3_key:" + s3_key);
      log.log("AWSS3ClientService.processData(): action:" + action);
      log.log("AWSS3ClientService.processData(): messageId:" + messageId);
      log.log("AWSS3ClientService.processData(): s3_bucketname:" + s3_bucketname);
      log.log("AWSS3ClientService.processData(): s3_folderName:" + s3_folderName);

      String accessKey = Manager.getProperties("integrationservices").getProperty("awss3.accessKey");
      String secretKey = Manager.getProperties("integrationservices").getProperty("awss3.secretKey");

      log.log("AccessKey for AWS S3 area: " + accessKey);
      log.log("SecretKey for AWS S3 area: " + secretKey);
         AWSCredentials credentials = new BasicAWSCredentials(
                  accessKey, // access key
                  secretKey); // secret key

         // create a client connection based on credentials
         AmazonS3 s3Client = new AmazonS3Client(credentials);

         // create bucket - name must be unique for all S3 users
         s3Client.createBucket(s3_bucketname);

         // list buckets
         for (Bucket bucket : s3Client.listBuckets()) {
               log.log(" - " + bucket.getName());
         }


      if ("put".equalsIgnoreCase(action)) {

         if ( wfc.getPrimaryDocument() != null ){
            docAsStream = wfc.getPrimaryDocument().getBodyInputStream();
            putFileToS3(docAsStream, s3_key, s3Client, s3_bucketname);
         }
         else if (local_path != null && !"".equalsIgnoreCase(local_path)) {// put file from FS to S3
            docAsStream = new FileInputStream( new File( local_path ));
            putFileToS3(docAsStream, s3_key, s3Client, s3_bucketname);
         } else if (messageId != null && !"".equalsIgnoreCase(messageId) && !"null".equalsIgnoreCase(messageId)) {
            putFileFromMailboxToS3(messageId, s3_key, s3Client, s3_bucketname);
         }

      } else if ("get".equalsIgnoreCase(action)) {
         log.log("Usage: s3client read <s3_key>");
         readFileFromS3(s3_key, s3Client, s3_bucketname);
         wfc.putPrimaryDocument(primaryDoc);
      } else if ("delete".equalsIgnoreCase(action)) {
         log.log("Usage: s3client delete <s3_key>");
         delete(s3_bucketname, s3_folderName, s3Client, s3_key);
      } else if ("mkdir".equalsIgnoreCase(action)) {// create folder into bucket
         log.log("Usage: s3client mkdir <s3_key>");
         createFolder(s3_bucketname, s3_folderName, s3Client);
      } else {
         log.log("Usage: s3client put/read/delete/mkdir" + " [<local_path> <s3_key>]");
         //System.exit(1);
      }

      log.log("Done!");
   } catch (Exception e) {
      isOK = false;
      log.log(e.getMessage());
      log.logException(e);
      wfc.setBasicStatus(1);
      throw new WorkFlowException(e.toString());
```

```
        } finally {
            wfc.unregisterThread();
            if (isOK) {
                wfc.setBasicStatusSuccess();
            } else {
                wfc.setBasicStatusError();
            }
        }
        return wfc;
    }

        public void createFolder(String bucketName, String folderName, AmazonS3 client) {
                // create meta-data for your folder and set content-length to 0
                ObjectMetadata metadata = new ObjectMetadata();
                metadata.setContentLength(0);

                // create empty content
                InputStream emptyContent = new ByteArrayInputStream(new byte[0]);

                // create a PutObjectRequest passing the folder name suffixed by /
                PutObjectRequest putObjectRequest = new PutObjectRequest(bucketName,
                                folderName + SUFFIX, emptyContent, metadata);

                // send request to S3 to create folder
                client.putObject(putObjectRequest);
        }

    public void putFileToS3(InputStream source, String key, AmazonS3 s3client, String bucketName) throws IOException {
        s3client.putObject(new PutObjectRequest(bucketName, key, source,
null).withCannedAcl(CannedAccessControlList.PublicRead));
}

    public void putFileFromMailboxToS3(String messageId, String key, AmazonS3 s3client, String bucketName)
            throws IOException {        IRepository repos = MailboxFactory.getRepository();
            String filename = null;
            Message message = null;
            Document batchDoc = null;
            try {
                log.log("AWSS3Client.putFileFromMailboxToS3():messageId:" + messageId);
                message = repos.getMessage(Long.parseLong(messageId));// source here is messageid
                log.log("AWSS3Client.putFileFromMailboxToS3(): message:" + message.toString());
                String docId = message.getDocumentId();
                log.log("AWSS3Client.putFileFromMailboxToS3(): docId:" + docId);
                filename = message.getMessageName();
                log.log("AWSS3Client.putFileFromMailboxToS3():filename:" + filename);
                if (docId != null) {
                    batchDoc = new Document(docId, true);
                }
                log.log("AWSS3Client.putFileFromMailboxToS3(): batchDoc:" + batchDoc);
            } catch (NumberFormatException nfe) {
                log.logException(nfe);
            } catch (MailboxException me) {
                log.logException(me);
            } catch (DocumentNotFoundException dnfe) {
                log.logException(dnfe);
            } catch (SQLException sqle) {
                log.logException(sqle);
            }

            InputStream in = batchDoc.getInputStream();
            final File aFile = File.createTempFile("tmp", "txt");
            aFile.deleteOnExit();
            try{
              FileOutputStream out = new FileOutputStream(aFile);
              IOUtils.copy(in, out);
            } catch (IOException ioe) {
                log.logException(ioe);
            }
                s3client.putObject(new PutObjectRequest(bucketName, key,
aFile).withCannedAcl(CannedAccessControlList.PublicRead));
    }

    public void readFileFromS3(String key, AmazonS3 s3Client, String bucketName) throws IOException {

        S3Object object = s3Client.getObject(
                new GetObjectRequest(bucketName, key));
                InputStream in = object.getObjectContent();
                //Process the objectData stream.

        String filename = key.substring(key.lastIndexOf('/') + 1, key.length());

        OutputStream out = new BufferedOutputStream(new FileOutputStream(new File(filename)));
```

```
        byte[] b = new byte[1024];
        int numBytes = 0;
        while ((numBytes = in.read(b)) > 0) {
            out.write(b, 0, numBytes);
        }
                in.close();
        out.close();
        primaryDoc = new Document();
        primaryDoc.setBody(b);

    }


        /**
         * If key is null - This method first deletes all the files in given folder and then the folder itself
         * If key is not null - This method deletes the given fileName
         */
        public void delete(String bucketName, String folderName, AmazonS3 client, String fileName) {
                List<S3ObjectSummary> fileList =
                                client.listObjects(bucketName, folderName).getObjectSummaries();
                if(fileName!=null){
                        for (S3ObjectSummary file : fileList) {
                                log.log("AWSS3ClientMain.delete(): fileName:"+fileName);
                                log.log("AWSS3ClientMain.delete(): file.getKey():"+file.getKey());
                                if (file.getKey().contains(fileName)) {
                                        client.deleteObject(bucketName, file.getKey());
                                }
                        }
                }else{
                        log.log("AWSS3ClientMain.delete(): fileName:"+fileName);
                        for (S3ObjectSummary file : fileList) {
                                client.deleteObject(bucketName, file.getKey());
                        }
                        client.deleteObject(bucketName, folderName);
                }

        }


}
```

Business Processes:

Following is the sample business process consisting of AWS S3 Client Service to perform GET, PUT and DELETE operations on a file, and create a directory on AWS S3 system..

```
<process name="AWSS3ClientProcess">
  <sequence>
        <operation name="Extract File">
          <participant name='TestS3ClientForB2Bi'/>
          <output message='xout'>
            <assign to='.' from='PrimaryDocument' />
          </output>
         <input message="xin">
           <assign to="." from="*"/>
         </input>
        </operation>
  </sequence>
</process>
```

Service Implementation – serviceinstances.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<services>
 <service parentdefname="AWSS3ClientForB2Bi"
name="TestS3ClientForB2Bi"
            displayname="TestS3ClientForB2Bi"

            description="AWS S3 Service for Sterling B2B Integrator"
             targetenv="all" activestatus="1"

             systemservice="0" parentdefid="-1"/>
</services>
```

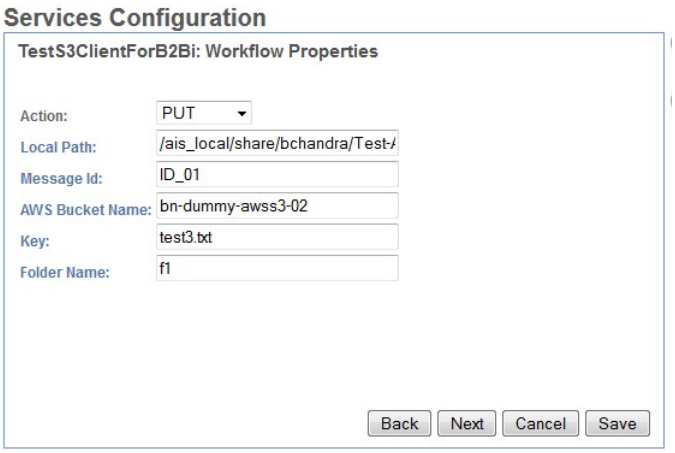
Service Definitions – servicedefs/TestServices.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<SERVICES>

 <SERVICE name="AWSS3ClientForB2Bi" description="AWS Simple Storage Solution with B2Bi" label="AWSS3ClientForB2Bi" implementationType="CLASS"
JNDIName="com.sterlingcommerce.woodstock.services.integration.AWSS3ClientService" type="BASIC" adapterType="N/A" adapterClass="N/A" version="10.0"
SystemService="NO">
  <VARS type="instance">
   <GROUP title="fs.wfd.group1.title" instructions="Instructions">
    <VARDEF varname="operation" type="String" htmlType="select" label="Action" options="actiontype" />
    <VARDEF varname="local_path" type="String" htmlType="text" label="Local Path" maxsize="512" />
    <VARDEF varname="messageId" type="String" htmlType="text" label="Message Id" maxsize="512" />
    <VARDEF varname="s3_bucketname" type="String" htmlType="text" label="AWS Bucket Name" maxsize="512" />
    <VARDEF varname="s3_key" type="String" htmlType="text" label="Key" maxsize="512" />
    <VARDEF varname="s3_foldername" type="String" htmlType="text" label="Folder Name" maxsize="512" />
   </GROUP>
  </VARS>
 </SERVICE>

 <OPTION name="actiontype">
  <ELE value="put" displayname="PUT" />
  <ELE value="delete" displayname="DELETE" />
  <ELE value="get" displayname="GET" />
  <ELE value="mkdir" displayname="MKDIR" />
 </OPTION>
</SERVICES>
```

Service Configuration in B2B UI



Installation of all dependant client jar – list shows all the awss3 client jars need to be in dynamicclasspath.cfg

Go to ./install/bin

```
./install3rdParty.sh awss3 1_0 -j /ais_local/share/bchandra/awss3/jar/aws-java-sdk-1.10.49.jar
```

Installing AWS S3 Client Service jar

Download the attached AWS S3 Client jar to test the sample example. Click this link to download - integrationservices_1010000.jar or download from the attachment tab below.

Now go to ./install/bin

./InstallService.sh /home/bchandra/awss3/client/integrationservices_1010000.jar

Execute CRUD operation.

Execute the above BPs to perform desired action. Any logging info goes to **integrationservices.log** under <INSTALL_DIR>/logs folder.

References:

http://docs.aws.amazon.com/AmazonS3/latest/gsg/GetStartedWithS3.html

https://javatutorial.net/java-s3-example