Paper 285-2011

Visualize Your Cloud Data Using the Graph Template Language

Krishnan Raghupathi, SAS R&D, Pune, Maharashtra, India

ABSTRACT

As the adoption of cloud computing by enterprise companies picks up, there will be an exponential growth in the volume of data that gets stored on the cloud. Under such a scenario, there will always be a need to analyze your cloud data in a seamless manner. This paper demonstrates how one can write a SAS® program, using a combination of PROC SOAP and PROC TEMPLATE, that can provide an effective visual representation of data that is available in the cloud.

INTRODUCTION

Research by different analysts has shown that cloud adoption is accelerating as businesses, especially those in the small and mid-sized technology segments, become more comfortable with the benefits of cloud computing and its security features. A natural consequence of this will be an exponential growth in the usage of cloud storage services to meet the ever increasing storage requirements for data analysis.

This paper describes how one can write a SAS® program that provides a visualization of cloud data in a seamless manner. The program uses a combination of PROC SOAP, PROC TEMPLATE, and some custom Java programs that access and manipulate data that is stored on Amazon Simple Storage Service (Amazon S3), a popular cloud data storage service.

This paper provides a step-by-step approach to assembling the final SAS program by breaking it up into the following parts:

- hosting the data on Amazon S3 cloud
- using PROC SOAP to access the data
- extracting the data from the SOAP response
- using PROC TEMPLATE to visualize the data

The complete SAS program is provided at the end of the paper.

HOSTING THE DATA ON AMAZON S3

Amazon S3 (AS3) is a cloud storage service that provides a Web services interface to store or retrieve data from anywhere on the Web. It uses standard REST and SOAP interfaces designed to work with any Internet-development toolkit. The following steps are required to set up the data that will be accessed by the SAS program.

STEP 1 – CREATE YOUR AS3 ACCOUNT

You need to first create an AS3 account before you can upload data to AS3. Refer to http://aws.amazon.com/s3/ for more details about how to sign up for your AS3 account.

STEP 2 – UPLOAD YOUR DATA TO AS3

Once your AS3 account has been created, log in to your Amazon Web Service (AWS) management console and create a bucket. Every object in AS3 is stored in a bucket and you need to create a bucket before you can store any data in it.

Once you have created a bucket, you are ready to add an object to it. An object can be any type of file: a data file, a photo, a video, and so on. Select any data file (it could even be a SAS data set) as your object and add it to the bucket that you created. You can add multiple objects to a bucket, but they all need to have unique names. Refer to http://docs.amazonwebservices.com/AmazonS3/latest/gsg/ for more details about how to go about creating buckets and adding objects to your bucket.

Note: The sample SAS program assumes that there is an object called cars.sas7bdat inside a bucket called CarDataSet. The cars.sas7bdat object is the file corresponding to the well-known SAS data set sashelp.cars.

USING PROC SOAP TO ACCESS THE DATA

Since AS3 provides SOAP interfaces to access the data, you can use PROC SOAP to access the data that is hosted on AS3. PROC SOAP is a SAS procedure that invokes a SOAP-based Web service through JNI. The following steps take you through creating the correct SOAP request using PROC SOAP.

STEP 1 – CONSTRUCT THE SOAP REQUEST

You need to construct the SOAP request for retrieving the object that was uploaded. Here is an example of a SOAP request that gets the cars.sas7bdat object from the CarDataSet bucket.

```
<GetObject xmlns="http://doc.s3.amazonaws.com/2006-03-01">
  <Bucket>CarDataSet</Bucket>
  <Key>cars.sas7bdat</Key>
  <GetMetadata>true</GetMetadata>
  <GetData>true</GetData>
  <InlineData>true</InlineData>
  <AWSAccessKeyId>1D9FVRAYCP1VJEXAMPLE=</AWSAccessKeyId>
  <Timestamp>2011-03-01T12:00:00.000Z</Timestamp>
  <Signature>Iuyz3d3P0aTou39dzbqaEXAMPLE=</Signature>
</GetObject>
```

The different elements of the SOAP request are explained below:

- GetObject: The SOAP operation that returns the latest version of an object.
- Bucket: The bucket from which to retrieve the object.
- Key: The key that identifies the object.
- GetMetadata: The metadata is returned with the object if this is true.
- GetData: The object data is returned if this is true.
- InlineData: If this is true, then the data is returned, base64-encoded, as part of the SOAP body of the response. If false, then the data is returned as a SOAP attachment.

The last three elements are related to authentication and are explained below.

Authenticated SOAP requests must be sent to AS3 over SSL. AS3 expects the authentication information to be put in the following elements of the SOAP request:

- AWSAccessKeyId: Your AWS Access Key ID provided to you as part of your AS3 account.
- Timestamp: This must be a dateTime (go to http://www.w3.org/TR/xmlschema-2/#dateTime) in the Coordinated Universal Time (Greenwich Mean Time) time zone, such as 2009-01-01T12:00:00.000Z. Authorization will fail if this timestamp is more than 15 minutes away from the clock on Amazon S3 servers.
- Signature: The RFC 2104 HMAC-SHA1 digest (go to http://www.ietf.org/rfc/rfc2104.txt) of the concatenation of "AmazonS3" + OPERATION + Timestamp, using your AWS Secret Access Key as the key. For example, in the following GetObject sample request, the signature element would contain the HMAC-SHA1 digest of the value "AmazonS3CreateBucket2009-01-01T12:00:00.000Z".

Note: The AWS Access Key and AWS Secret Access Key are available in the Security Credentials section of your AWS account.

STEP 2 – GENERATE THE AUTHENTICATED SOAP REQUEST

Since the authentication elements of the SOAP request are sensitive to time you need to be able to generate the SOAP request on the fly. For this you can use a custom Java program, XMLUtilities (source code provided at the end of the paper), that generates the required timestamp and signature. The following SAS program uses javaobj to invoke the Java program to generate the required information.

```
data _null_;
length timestamp $50;
length signature $50;
declare javaobj j ('XMLUtilities');
j.callStringMethod ('getTimeStamp', timestamp);
j.callStringMethod ('getSignature', 'GetObject','7oB2EkYAnCUGOfSsOpCkZTPUrBav4mBWKDbpOGZR', signature);
j.delete();
run;
```

The getTimeStamp() method returns the current time in the required format and the getSignature() method returns the signature based on the operation and the timestamp. The getSignature() method takes the following two parameters in order:

1. the name of the AS3 SOAP method like GetObject
2. the AWS secret access key

STEP 3 – ASSEMBLE THE PROC SOAP PROGRAM

You can now access the data hosted on AS3 using the following SAS program:

```
FILENAME REQUEST "c:\temp\GetCarDataSet_REQUEST.xml";
data _null_;
length timestamp $50;
length signature $50;
declare javaobj j ('XMLUtilities');
j.callStringMethod ('getTimeStamp', timestamp);
j.callStringMethod ('getSignature', 'GetObject', '7oB2EkYAnCUGOfSsOpCkZTPUrBav4mBWKDbpOGZR', signature);
j.delete();

FILE REQUEST;
put "<soap:Envelope xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
xmlns:xsd='http://www.w3.org/2001/XMLSchema' xmlns:soap='http://schemas.xmlsoap.org/soap/envelope/'>";
put "<soap:Body>";
put "<GetObject xmlns='http://doc.s3.amazonaws.com/2006-03-01'>";
put "<Bucket>CarDataSet</Bucket>";
put "<Key>cars.sas7bdat</Key>";
put "<GetMetadata>true</GetMetadata>";
put "<GetData>true</GetData>";
put "<InlineData>true</InlineData>";
put "<AWSAccessKeyId>AKIAJWJ3GV2YEX4UC7QQ</AWSAccessKeyId>";
put "<Timestamp>" timestamp "</Timestamp>";
put "<Signature>" signature "</Signature>";
put "</GetObject>";
put "</soap:Body>";
put "</soap:Envelope>";
run;

FILENAME RESPONSE "c:\temp\GetCarDataSet_RESPONSE.xml";
proc soap
  in=REQUEST
  out=RESPONSE
  url="https://s3.amazonaws.com/soap"
  soapaction="http://soap.amazon.com"
  proxyhost="inetgw.unx.sas.com"
  proxyport=80
  envelope;
run;
```

The different elements of the program are explained below:

- The timestamp and signature required for generating authenticated SOAP requests are created by invoking the Java program XMLUtilities via javaobj.
- REQUEST refers to the generated XML file that contains the authenticated SOAP request.
- The series of PUT statements creates the SOAP request XML in the required format.
- RESPONSE refers to the file where the SOAP response output XML will be written.
- http://s3.amazonaws.com/soap is a secure URL of the Web service endpoint for AS3 SOAP requests.
- http://soap.amazon.com is the SOAPAction element to invoke on the Web service.

EXTRACTING THE DATA FROM THE SOAP RESPONSE

The result of executing the SAS program described in Step 3 of the previous section is the creation of an XML file containing the SOAP response that includes the requested data in its body. However, this data is base64-encoded and so needs to be decoded back to its original form before it can be used. The custom Java program XMLUtilities (referred to Step 2 of the previous section) contains a utility method that decodes base64-encoded data. The following SAS program uses javaobj to invoke the utility method that extracts the requested data from the SOAP response and decodes it.

```
data _null_;
declare javaobj j ('XMLUtilities');
j.callStaticVoidMethod('extractSOAPObjectResponse','c:\temp\GetCarDataSet_RESPONSE.xml','c:\temp\cars.sas7bdat'
);
j.delete();
run;
```

The extractSOAPObjectResponse() method extracts the inline data from the body of the SOAP response, decodes it, and then stores the decoded data in the specified location. The extractSOAPObjectResponse() method takes the following two parameters in order:

1. fully qualified path of the SOAP response XML.
2. fully qualified path where the decoded data should be stored. In the above program the value of this parameter is a path to a SAS data set file since that is what was added to the bucket in AS3.

USING PROC TEMPLATE TO VISUALIZE THE DATA

Now that the requested data has been extracted from the SOAP response body, you can visualize the data using the Graph Template Language (GTL) as demonstrated in the following program:

```
libname temp ‘c:\temp’;
proc template;
  define statgraph barchart;
    begingraph;
      entrytitle "Average Horsepower by Vehicle Type";
      layout overlay;
        barchart x=type y=horsepower /
            stat=mean orient=horizontal;
      endlayout;
    endgraph;
  end;
run;

proc sgrender data=temp.cars template=barchart;
run;
```

The different elements of the program are explained below:

- The PROC TEMPLATE statement is GTL syntax that defines a STATGRAPH template called SGPLOT. This template defines a graph containing a bar chart with category role (X role) as type and response role (Y role) as horsepower.
- The PROC SGRENDER statement associates the SGPLOT template with the SAS data set temp.mycars resulting in the creation of the graph

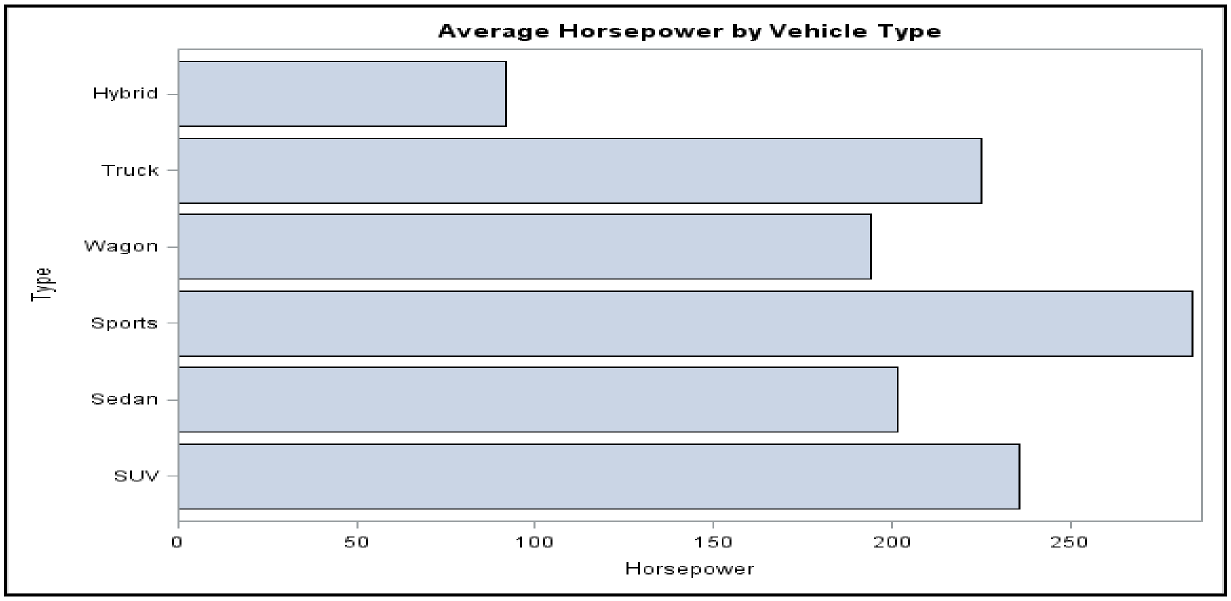

Figure 1: Bar chart created by the above program

CONCLUSION

With the increasing adoption of cloud computing platforms (both private and public) by enterprises, there will be an exponential growth in the usage of cloud data storage services for storing data required for data analysis. The need to analyze such cloud data in a seamless manner will always exist. There exist a number of cloud data storage providers like Amazon, Rackspace Cloud, Windows Azure, Google Storage, and so on. This paper demonstrates that it is possible to write a SAS program using existing Base SAS procedures that provides a visual representation of cloud data that is hosted on Amazon Simple Storage Service (AS3). A full-blown implementation of data analysis of cloud data needs to consider additional aspects like

- **Securing the access keys:** Access to the data on AS3 is controlled by two access keys – a public access key and a secret access key. The public access key is included in the SOAP request body between the AWSAccessKeyID tags, while the secret access key is used to generate the signature that should be included in every authenticated SOAP request. Since the secret access keys should not be shared publicly, it is recommended to separate the SAS code that generates the authentication signature from the main program. The main program only requires the signature, which can be provided on a need basis – note that the signature depends on the nature of the operation and expires after 15 minutes.
- **Cost implications:** Since AS3 is a paid storage service, a cost is incurred every time the program is executed depending on the number of read/write requests and the volume of data transferred in/out of AS3. However, the cost can be minimized by using a combination of Amazon EC2 (cloud computing service) and Amazon S3 (cloud data storage service) in the same region since there are no data transfer charges in such scenarios.
- **Private clouds:** For enterprises that are keen to leverage the benefits of cloud computing but are wary of using public clouds due to security or regulatory concerns, deploying private cloud software is a viable alternative. One such example is Eucalyptus - private cloud software that enables enterprises to establish their own cloud computing environments within the safety of their firewall. Since Eucalyptus is interface-compatible with Amazon Web Services cloud infrastructure, Amazon S3 implementations can be easily migrated to private cloud environments that are created using Eucalyptus.

REFERENCES

- Amazon Simple Storage Service - http://aws.amazon.com/s3
- SAS documentation for PROC SOAP
- SAS documentation for PROC TEMPLATE

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Name: Krishnan Raghupathi
Enterprise: SAS Research and Development (India)
Address: Level 2A & Level 3, Cyber City Tower 5, Margarpatta City
City, State ZIP: Pune, Maharashtra 411013
E-mail: Krishnan.PR@sas.com

Sample Code Source Files

NOTE: This source code is provided for purpose of illustrating the points made in this paper. Readers are encouraged to evaluate and test this source code thoroughly, before deciding to use it in their own SAS programs.

1. SAS Program

```
FILENAME REQUEST "c:\temp\GetCarDataSet_REQUEST.xml";

data _null_;
length timestamp $50;
length signature $50;
declare javaobj j ('XMLUtilities');
j.callStringMethod ('getTimeStamp', timestamp);
j.callStringMethod ('getSignature', 'GetObject', '7oB2EkYAnCUGOfSsOpCkZTPUrBav4mBWKDbpOGZR', signature);
j.delete();

FILE REQUEST;
put "<soap:Envelope xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
xmlns:xsd='http://www.w3.org/2001/XMLSchema' xmlns:soap='http://schemas.xmlsoap.org/soap/envelope/'>";
put "<soap:Body>";
put "<GetObject xmlns='http://doc.s3.amazonaws.com/2006-03-01'>";
put "<Bucket>CarDataSet</Bucket>";
put "<Key>cars.txt</Key>";
put "<GetMetadata>true</GetMetadata>";
put "<GetData>true</GetData>";
put "<InlineData>true</InlineData>";
put "<AWSAccessKeyId>AKIAJWJ3GV2YEX4UC7QQ</AWSAccessKeyId>";
put "<Timestamp>" timestamp "</Timestamp>";
put "<Signature>" signature "</Signature>";
put "</GetObject>";
put "</soap:Body>";
put "</soap:Envelope>";
run;


FILENAME RESPONSE "c:\temp\GetCarDataSet_RESPONSE.xml";

proc soap
  in=REQUEST
  out=RESPONSE
  url="https://s3.amazonaws.com/soap"
  soapaction="http://soap.amazon.com"
  proxyhost="inetgw.unx.sas.com"
  proxyport=80
  envelope;
run;

data _null_;
declare javaobj j ('XMLUtilities');
j.callStaticVoidMethod ('extractSOAPObjectResponse','c:\temp\GetCarDataSet_RESPONSE.xml','c:\temp\cars.txt');
j.delete();
run;

PROC IMPORT OUT= WORK.MYCARS
            DATAFILE= "C:\temp\cars.txt"
            DBMS=TAB REPLACE;
     GETNAMES=YES;
     DATAROW=2;
RUN;

proc template;
  define statgraph barchart;
    begingraph;
      entrytitle "Average Horsepower by Vehicle Type";
      layout overlay;
        barchart x=type y=horsepower /
            stat=mean orient=horizontal;
      endlayout;
    endgraph;
  end;
run;

proc sgrender data=sashelp.cars template=barchart;
run;
```

2. Custom Java program to be invoked via javaobj:

Note: This program depends on commons-codec.jar, an Apache implementation of common encoders and decoders. Refer to http://commons.apache.org/codec/ for more details.

```
import java.io.ByteArrayInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Iterator;
import java.util.TimeZone;

import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import javax.xml.XMLConstants;
import javax.xml.namespace.NamespaceContext;
import javax.xml.namespace.QName;
import javax.xml.xpath.XPath;
import javax.xml.xpath.XPathConstants;
import javax.xml.xpath.XPathExpression;
import javax.xml.xpath.XPathExpressionException;
import javax.xml.xpath.XPathFactory;

import org.apache.commons.codec.binary.Base64;
import org.w3c.dom.Node;
import org.xml.sax.InputSource;


public class XMLUtilities {

        private Date date;
        private String timestamp;
        private String signature;


        public XMLUtilities() {
                date = new Date();
                SimpleDateFormat df = new SimpleDateFormat("yyyy-MM-dd'T'HH:mm:ss");
                df.setTimeZone(TimeZone.getTimeZone("GMT"));
                timestamp = df.format(date)+".000Z";

        }
        /**
         * Decodes a base64 encoded string to ascii format
         * @param istream - Input stream containing the bas64 encoded string
         * @param ostream - Output stream that will contain the decoded ascii string
         * @throws FileNotFoundException
         * @throws IOException
         */
        public static void decodeBase64ToAscii(InputStream istream, OutputStream ostream) throws
FileNotFoundException, IOException {


                 sun.misc.BASE64Decoder base64 = new sun.misc.BASE64Decoder();
                 base64.decodeBuffer(istream, ostream);
                 int ch = 0;

                 while ((ch = istream.read()) >= 0) {
                  ostream.write(ch);
                 }

                 ostream.flush();
                 ostream.close();

        }

        /**
         * Evaluates an XPath expression against the specified xml document and returns the
result
         * @param xmlDocumentLoc - XML doc
         * @param xpathExpression - XPath expression
         * @param returnType - Type of object that is to be returned
         * @return - Returns the result of the xpath expression evaluation
         * @throws FileNotFoundException
         * @throws XPathExpressionException
         */
        public static Object evaluateXPathExpression(String xmlDocumentLoc, String
xpathExpression, QName returnType) throws FileNotFoundException, XPathExpressionException {


               InputSource xmldoc = new InputSource(new FileInputStream(new
File(xmlDocumentLoc)));

               XPathFactory xPathFactory = XPathFactory.newInstance();
               XPath xPath = xPathFactory.newXPath();
               xPath.setNamespaceContext(new SOAPNamespaceContext());
               XPathExpression xPathExpression = xPath.compile(xpathExpression);

               return xPathExpression.evaluate(xmldoc,returnType);

        }

        public String getTimeStamp() {
               return timestamp;
        }
```

```
    public String getSignature(String operation, String key) {

        String s = null;

        try {

                        String data = "AmazonS3"+operation+timestamp;
                        Mac mac = Mac.getInstance("HmacSHA1");
                        mac.init(new SecretKeySpec(key.getBytes(), "HmacSHA1"));
                        byte[] signature = Base64.encodeBase64(mac.doFinal(data.getBytes("UTF-
8")));

                        s = new String(signature);

        } catch (Exception e) {
                s = "";
        }

        return s;
    }



        public static void extractSOAPObjectResponse(String xmlResponseDoc, String objectLoc) {

                String xPathExpression = 
"/soapenv:Envelope/soapenv:Body/pre:GetObjectResponse/ns1:GetObjectResponse/ns1:Data";

                try {
                        Object result = 
evaluateXPathExpression(xmlResponseDoc,xPathExpression,XPathConstants.NODE);
                        Node n = (Node)result;
                        String s = n.getTextContent();
                        ByteArrayInputStream bis = new ByteArrayInputStream(s.getBytes());

                        FileOutputStream fos = new FileOutputStream(new File(objectLoc));

                        decodeBase64ToAscii(bis, fos);

                } catch (FileNotFoundException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                } catch (XPathExpressionException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                } catch (IOException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }

        }


}
```

```
class SOAPNamespaceContext implements NamespaceContext {

    public String getNamespaceURI(String prefix) {
        if (prefix == null) throw new NullPointerException("Null prefix");
        else if ("pre".equals(prefix)) return "http://doc.s3.amazonaws.com/2006-03-01";
        else if ("ns1".equals(prefix)) return "http://s3.amazonaws.com/doc/2006-03-01/";
        else if ("soapenv".equals(prefix)) return "http://schemas.xmlsoap.org/soap/envelope/";
        else if ("xml".equals(prefix)) return XMLConstants.XML_NS_URI;
        return XMLConstants.NULL_NS_URI;
    }

    // This method isn't necessary for XPath processing.
    public String getPrefix(String uri) {
        throw new UnsupportedOperationException();
    }

    // This method isn't necessary for XPath processing either.
    public Iterator getPrefixes(String uri) {
        throw new UnsupportedOperationException();
    }

}
```