***Getting started guide for AWS (Amazon Web Services) Spring Integration adapters***

**Little about Amazon Web Services (AWS)**

Launched in 2006, Amazon Web Services (AWS) started providing key infrastructure for business as web services, which, also known as cloud computing. Using cloud computing businesses can adopt a new business model whereby they do not have to plan and invest in procuring their own IT infrastructure. They can use the infrastructure and services provided by the cloud service provider and pay as they use the services.

Visit <http://aws.amazon.com/products/> for more details about various products offered by Amazon as a part their cloud computing services.

**Introduction**

This guide intends to explain in brief about various adapters available for the AWS Services and a sample xml tag definition for each of them. We will also provide some simple java client code used with the outbound adapters.

The library currently has adapters for the following Services

1. Amazon S3 (outbound and inbound)
2. Amazon SES (outbound)
3. Amazon SQS (outbound and inbound)

Amazon SES (Simple Email Service) is used to send out mails and cannot receive mails, hence can only have an outbound adapter.

**Amazon S3 adapters**

**Introduction**

Amazon S3 (Simple Storage Service) is a storage for internet. It provides a simple web service that can be used to store and retrieve any amount of data from internet. The files (or objects) stored in S3 also supports access control information to be attached to it. For more details about S3 and its pricing visit <http://aws.amazon.com/s3/>

**1. Outbound Channel Adapter**

An outbound adapter will be used to upload files from our application or local file system to Amazon S3. This would be done by sending a message to a channel with a payload of a *String, a byte array* or a *java.util.File* instance. This channel would be the source of the messages for the AWS S3 outbound adapter. The message will be read, payload extracted and the file would be uploaded to the Amazon S3.   
This adapter has got name space support similar to other adapters in spring integration.  
  
 Below is a sample xml definition in the configuration file to configure an outbound S3 adapter

<beans xmlns=<http://www.springframework.org/schema/beans>  
 xmlns:xsi=<http://www.w3.org/2001/XMLSchema-instance>  
 xmlns:integration=<http://www.springframework.org/schema/integration>  
 xmlns:aws-s3=<http://www.springframework.org/schema/integration/aws/s3>  
 xsi:schemaLocation=  
 "http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-  
 beans-3.0.xsd  
 http://www.springframework.org/schema/integration/aws/s3 http://www.springframework.org/schema/integration/aws/s3/spring-integration-aws-s3-2.1.xsd  
 http://www.springframework.org/schema/integration http://www.springframework.org/schema/integration/spring-integration-2.0.xsd">

<integration:channel id="s3OutboundChannel"/>  
  
 <aws-s3:outbound-channel-adapter propertiesFile="classpath:awscredentials.properties"   
 channel="s3OutboundChannel"  
 remote-directory="test"  
 multipart-upload-threshold="10240"  
 bucket="test.bucket.by.amol">  
 </beans>

The following java snippet sends a file from local file system to Amazon S3

File file = new File("<some file on local file system>");  
 Message<File> message = MessageBuilder  
 .withPayload(file)  
 .build();  
 channel.send(message);

Where *channel* is a channel on which the S3 outbound adapter subscribes or polls depending on the type of the channel.

The following code snippet will be uploading a file with contents taken from the *String* message payload. The name of the file will be taken from a header of the message. The message also contains the Object ACLs (object permissions on S3)

Map<String, Object> headers = new HashMap<String, Object>();  
headers.put("file\_name", "FileUploadedFromS3OutboundAdapter.txt");   
Map<String, Collection<String>> acl = new HashMap<String, Collection<String>>();  
acl.put("amolnayak.spring@gmail.com ", Collections.singletonList("write acp"));  
acl.put("f854da004ee08cf4f8664334d288561c8512c508db9785388de7319ded85f8f3", Arrays.asList("read","read acp"));  
acl.put("http://acs.amazonaws.com/groups/global/AuthenticatedUsers", Collections.singletonList("read acp"));  
headers.put("object\_acls",acl);  
Message<String> message =   
 MessageBuilder  
 .withPayload("This is the content of the file FileUploadedFromS3OutboundAdapter.txt")  
 .copyHeaders(headers)  
 .build();  
channel.send(message);

The above code snippet does the following

1. Creates a message with payload *This is the content of the file FileUploadedFromS3OutboundAdapter.txt*.
2. Adds a message header *file\_name* with value *FileUploadedFromS3OutboundAdapter.txt*. This will thus be the name of the file uploaded from the adapter to Amazon S3.
3. Add a message header *object\_acls* with a value as a *Map.* This map contains the object permissions granted to various individuals or groups. In the above example we have used an email, canonical and a group identifier to identify the grantees. The identifier is the key in the map and the values is a Collection of the permissions.
4. Send this message over the channel to which the adapter is subscribed to or is polling for messages (defining the xml in the config file for the outbound adapter should take care of delivering this message to the adapter).

**Configuring the S3 outbound adapter.**

**Below are the attributes supported by the AWS S3 outbound adapter tag** (attributes marked bold are mandatory)

|  |  |
| --- | --- |
| **Attribute Name** | **Attribute Description** |
| accessKey | The AWS access key |
| secretKey | The AWS secret key |
| propertiesFile | The properties file containing the AWS access key and secret key. Use any one of propertiesFile or the accessKey, secretKey attributes. Specifying one of these is ***mandatory*** |
| **bucket** | The bucket to which the files are to be uploaded |
| **channel** | The channel over which the messages will be sent to the outbound adapter |
| charset | If the payload of the message sent over the channel to the adapter is String, this attribute indicates the encoding to expect for this String payload. |
| multipart-upload-threshold | The size in bytes above which multipart part upload will be used to upload files to S3. All files of size below this value will upload the files in a single thread. |
| temporary-suffix | A file may be written to a temporary directory in some cases before being uploaded to S3, this attribute will indicate the extension of the temporary file |
| temporary-directory | The temporary directory that would be used for writing the files just in case they need to be written to temporary directory. |
| thread-pool-executor | The thread pool executor that would be used to upload the files to S3. |
| remote-directory | The remote directory on S3 where the files would be uploaded to. If none specified then the object will be uploaded to the bucket root. |
| remote-directory-expression | The expression that would be evaluated on the incoming message to find the remote directory on S3. Either of this or the remote-directory is to be specified. It is however not mandatory to specify either of them. |
| file-name-generator | The file name generation strategy that will be used to generate a file name for the file to be uploaded on S3. |
| file-name-generation-expression | The expression that would be evaluated on the incoming message and find a name for the file uploaded on S3. |

**2. Inbound Channel Adapter**

An S3 inbound channel adapter is used to synchronize the files on the Amazon S3 with a local directory. The configured Amazon S3 bucket and the given folder will be polled regularly and the files in it will be synchronized with the files in the local directory. New files will be downloaded from the remote bucket and files modified or outdated on local file system will be synchronized. Whenever a new file is created or updated on the local file system, the corresponding *java.io.File* instance is sent over the channel as the message payload.

Below is a sample configuration for an S3 inbound adapter

<beans xmlns=<http://www.springframework.org/schema/beans>  
 xmlns:xsi=<http://www.w3.org/2001/XMLSchema-instance>  
 xmlns:integration=<http://www.springframework.org/schema/integration>  
 xmlns:aws-s3=<http://www.springframework.org/schema/integration/aws/s3>  
 xsi:schemaLocation=  
 "http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-  
 beans-3.0.xsd  
 http://www.springframework.org/schema/integration/aws/s3 http://www.springframework.org/schema/integration/aws/s3/spring-integration-aws-s3-2.1.xsd  
 http://www.springframework.org/schema/integration http://www.springframework.org/schema/integration/spring-integration-2.0.xsd">

<integration:channel id=" s3InboundChannel "/>  
  
 <aws-s3:inbound-channel-adapter  
 propertiesFile="classpath:awscredentials.properties"  
 channel="s3InboundChannel"  
 bucket="test.bucket.by.amol"  
 remote-directory="test"  
 directory="/home/amoln/AWSSync"  
 file-wildcard="\*.txt">  
 <integration:poller fixed-rate="1000"/>   
 </aws-s3:inbound-channel-adapter>  
 </beans>

The above spring configuration polls the Amazon S3 bucket *test.bucket.by.amol* and the folder *test* in it with a fixed rate of 1000ms. Please note, since we have used fixed-rate, the polling will happen irrespective of whether the previous synchronization is complete or not. The adapter is developed to prevent multiple threads to sync the same folder. That is, if a synchronize operation of the previous poll Is still in progress, the synchronize request from the current poll is ignored.

The channel *s3InboundChannel* will carry messages with a payload of type *java.io.File.* This file represents the newly created file or a file that is being updated with the version present in S3 bucket.

**Configuring the S3 inbound channel adapter**

**Below are the attributes supported by the AWS S3 inbound adapter tag** (attributes marked bold are mandatory)

|  |  |
| --- | --- |
| **Attribute Name** | **Attribute Description** |
| accessKey | The AWS access key |
| secretKey | The AWS secret key |
| propertiesFile | The properties file containing the AWS access key and secret key. Use any one of propertiesFile or the accessKey, secretKey attributes. Specifying one of these is ***mandatory*** |
| **bucket** | The bucket to which the files are to be uploaded |
| **channel** | The channel over which the *java.io.File* payload will be sent when a file is synchronized or copied from the remote S3 bucket. |
| temporary-suffix | This is the suffix for the file that is being created or synchronized with the file on remote S3 bucket. Once the synchronize is completed, the temporary suffix is removed from the file’s name. By default the value is *.writing* |
| thread-pool-executor | The thread pool executor that will be used by the underlying AWS Transfer Manager to perform the multipart file uploads. If none is provided, the Transfer Manager uses the default one it creates. |
| remote-directory | The directory on S3 that will be polled and synchronized with a provided local directory, if none is provided, the root of the bucket is considered. Please note that the synchronization is done only for the provided directory and not of any of its subdirectories on S3. |
| **directory** | This local directory that will be used to synchronize the files from the remote-directory on S3. |
| queue-size | The files handles of files being synchronized are internally stored in a queue before they are being polled and returned on a subsequent receive operation on the message source. The number provided against this attribute is the maximum size of the queue. If none is provided, the default value of 1024 is used. |
| max-objects-per-batch | While synchronizing with the local directory, the contents of remote directory on S3 are listed and compared with the contents of the files on local directory. If the number of files in an S3 bucket is large and we desire to paginate the listing, use this variable to set the max objects listed per call. By default no pagination is used and all objects are listed. |
| file-wildcard | The wildcard pattern of the file names to be matched while synchronization, only the files matching this wildcard will be synchronized with the local directory. |
| file-name-regex | The regular expression to be used to match the file name. This is an alternative to the file-wildcard attribute. Use only one of the two attributes. If none of the above two attributes are specified, all the files in the bucket are synchronized. |

**Amazon SES (Simple Email Service) adapters**

**Introduction**

Amazon Simple Email Service is a web service for sending emails from the cloud. It supports two types of mails currently, the simple mail and raw email. Use simple mail if your application just needs to send out emails with some html formatting and without embedded images or attachments. Raw emails give us power to send complex emails with embedded images and attachments. For more details about Amazon SES and its pricing visit <http://aws.amazon.com/ses/>

We have a Spring Integration outbound channel adapter for the Amazon SES Service for sending out simple and raw emails with complete namespace support for configuring the adapter.

**1. Outbound Channel Adapter**

Below is a the spring configuration for configuring the SES outbound adapter

<?xml version="1.0" encoding="UTF-8"?>  
<beans xmlns=<http://www.springframework.org/schema/beans>  
 xmlns:xsi=<http://www.w3.org/2001/XMLSchema-instance>  
 xmlns:integration=<http://www.springframework.org/schema/integration>  
 xmlns:aws-ses=<http://www.springframework.org/schema/integration/aws/ses>  
 xsi:schemaLocation=   
 "http://www.springframework.org/schema/beans   
 http://www.springframework.org/schema/beans/spring-beans-3.0.xsd http://www.springframework.org/schema/integration/aws/ses   
 http://www.springframework.org/schema/integration/aws/ses/spring-integration-aws-ses-2.1.xsd  
 http://www.springframework.org/schema/integration   
 http://www.springframework.org/schema/integration/spring-integration-2.0.xsd">  
  
 <integration:channel id="outboundAdapterChannel"/>

<aws-ses:outbound-channel-adapter   
 propertiesFile="classpath:awscredentials.properties"  
 channel="outboundAdapterChannel"/>  
</beans>

**Configuring the SES outbound channel adapter**

**Below are the attributes supported by the AWS SES outbound adapter tag** (attributes marked bold are mandatory)

|  |  |
| --- | --- |
| **Attribute Name** | **Attribute Description** |
| accessKey | The AWS access key |
| secretKey | The AWS secret key |
| propertiesFile | The properties file containing the AWS access key and secret key. Use any one of propertiesFile or the accessKey, secretKey attributes. Specifying one of these is ***mandatory*** |
| **channel** | The channel over which the message is sent to the SES adapter, it can either be of type *javax.mail.internet.MimeMessage, String* or *org.springframework.integration.aws.ses.core.AmazonSESSimpleMailMessage* |

**Sending mail messages**

1. **Simple Mail Messages**

We will now give a sample java program to send Simple Mail Messages using Amazon SES. A Simple Mail Message does not support attachments and embedded contents. It supports basic html content to be sent as the mail body.

Map<String, Object> headers = **new** HashMap<String, Object>();

headers.put("fromEmailId", "xyz@somemail.com");

headers.put("htmlFormat", **true**);

headers.put("subject", "Mail Sent from AWS SES Outbound adapter");

headers.put("toEmailId", "abc@anothermail.com");

Message<String> msg =

MessageBuilder.*withPayload*("<html><i>A Simple Mail Message sent from " +

"Amazon SES Outbound adapter from Spring integration<i></html>")

.copyHeaders(headers)

.build();

channel.send(msg);

The above piece off code is pretty simple.

1. We add four headers to the message for the “to email id”, “from email id”, “subject “and   
a flag to indicate whether the content is an html content or plain text content.

2. Of these headers the “from email id” and “subject” are mandatory. We can specify one or more of “to”, “cc “or “bcc “email addresses.

3. The message needs to have a payload of string which is either a plain text or html content. The content will be rendered a html only if the “htmlFormat” header is set appropriately.

4. The value of the “htmlFormat” header can be Boolean true, or “y”,”yes” or “true” as String. For String the value is case insensitive.

5. The message is sent over the channel to which is the input channel of the outbound channel adapter.

1. **Raw Mail Messages**

Use raw mail when you need more flexibility to send mails, like setting mime types and email headers. As long as the content complies with the standard email format standard you can use this means for sending the mail to your recipients.

Session session = Session.*getDefaultInstance*(**new** Properties());

MimeMessageHelper helper = **new** MimeMessageHelper(**new** MimeMessage(session),**true**);

helper.setTo("abc@somemail.com");

helper.setFrom("xyz@anothermail.com");

helper.setText("A Sample Embedded image");

helper.addAttachment(file.getName(),**new** File("<File path to the attachment>"));

helper.setSubject("Name Pic");

Message<MimeMessage> message =   
 MessageBuilder.*withPayload*(helper.getMimeMessage()).build();

channel.send(message);

We construct a plain Mime message using springs *MimeMessageHelper* and send the message with *MimeMessage* payload over the channel. The messages over this channel are consumed by the outbound SES adapter and the mail is sent out using SES.

**Amazon SQS (Simple Queue Service)**

Amazon Simple Queue Service (SQS) is a highly scalable, reliable messaging platform for storing and receiving messages from the queues. It is the Amazon’s web-scale messaging infrastructure exposed as web service. For more details about SQS, resources and pricing visit <http://aws.amazon.com/sqs/>.

We have an outbound and inbound adapter in spring integration with complete namespace support for sending and receiving messages respectively, to and from Amazon SQS.   
Inbound adapter is transactional which can participate in incoming transactions and supports redelivery of unsuccessful delivery of messages up to a specified max number of times.

**Outbound Channel Adapter**

Use Amazon SQS outbound channel adapter if you want to send messages to Amazon SQS Queue. Below is an example of configuring an SQS outbound adapter channel adapter.

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xmlns:integration=*"http://www.springframework.org/schema/integration"*

xmlns:aws-sqs=*"http://www.springframework.org/schema/integration/aws/sqs"*

xsi:schemaLocation=

*"http://www.springframework.org/schema/beans   
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd*

*ht*[*tp://www.springframework.org/schema/integration/aws/sqs h*](tp://www.springframework.org/schema/integration/aws/sqs%20h)*ttp://www.springframework.org/schema/integration/aws/sqs/spring-integration-aws-sqs-2.1.xsd*

*ht*[*tp://www.springframework.org/schema/integration h*](tp://www.springframework.org/schema/integration%20h)*ttp://www.springframework.org/schema/integration/spring-integration-2.0.xsd"*>

<integration:channel id=*"sqsOutboundChannel"*/>

<aws-sqs:outbound-channel-adapter

propertiesFile=*"classpath:awscredentials.properties"*

channel=*"sqsOutboundChannel"*

destination-queue=  
 *"https://queue.amazonaws.com/123456789012/MyTestQueue"*/>

</beans>

Above definition is self explanatory and simple to understand. Below section summarizes various attributes supported by SQS outbound adapter definition.

**Configuring the SQS outbound channel adapter**

**Below are the attributes supported by the AWS SQS outbound adapter tag** (attributes marked bold are mandatory)

|  |  |
| --- | --- |
| **Attribute Name** | **Attribute Description** |
| accessKey | The AWS access key |
| secretKey | The AWS secret key |
| propertiesFile | The properties file containing the AWS access key and secret key. Use any one of propertiesFile or the accessKey, secretKey attributes. Specifying one of these is ***mandatory*** |
| **channel** | The channel over which the messagewill be sent to the SQS outbound adapter. The payload of the message can be of any type. However, messages sent to SQS accept only *String* payload, hence, we need to have a conversion service in place for converting the message payload to *String* and from *String* back to message payload type. |
| default-sqs-queue | Relevant only when destination-queue-expression attribute is specified. When the expression evaluation on the message doesn’t yield any destination queue, the message is sent to this default queue. |
| destination-queue-expression | The expression that is evaluated against the incoming message to get a destination SQS queue for the message |
| destination-queue | Statically define a queue to which the messages will be sent. One of the *destination-queue* or *destination-queue-expression* is **mandatory**. |
| verify-sent-messages | If the flag is set to true, the MD5 digest of the request is compared with the MD5 digest of the content received on the server side. By default the value is set to false. ***NOTE:*** *This functionality is currently not implemented and a proper action to be taken in case the MD5 digest doesn’t match is not defined.* |

**Sample client code sample for sending a message to Amazon SQS using SQS Outbound channel adapter.**

Map<String, String> attributes = new HashMap<String, String>();  
 attributes.put("Some Attribute","Some Value");  
  
 Map<String, Object> headers = new HashMap<String, Object>();   
 headers.put("message\_attributes", attributes);  
  
 channel.send(MessageBuilder.withPayload(Integer.valueOf(10))  
 .copyHeaders(headers)  
 .build());

Above client code does the below 3 things

1. Constructs a *Map* or String key and value and adds a key value pair to it. This is a *Map* containing any random user attributes user wants to send to the SQS along with the message.
2. Build a message with a payload of type *java.lang.Integer* and add the *Map* created above to it as a header with name *message\_attributes.*
3. Send the message over the channel, this is the channel whose messages are consumed by the SQS outbound channel adapter (specified in the channel attribute of the config). The adapter performs the necessary conversion and serialization and sends the message over to SQS.

**Inbound Channel Adapter**

Amazon SQS Inbound channel adapter is used to receive messages from the SQS Queues in the cloud. We have namespace support for configuring this adapter and it is configured as below.

Sample configuration for SQS inbound channel adapter

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

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xmlns:integration=*"http://www.springframework.org/schema/integration"*

xmlns:aws-sqs=*"http://www.springframework.org/schema/integration/aws/sqs"*

xsi:schemaLocation=

*"http://www.springframework.org/schema/beans   
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd*

*http://www.springframework.org/schema/integration/aws/sqs   
http://www.springframework.org/schema/integration/aws/sqs/spring-integration-aws-sqs-2.1.xsd*

*http://www.springframework.org/schema/integration   
http://www.springframework.org/schema/integration/spring-integration-2.0.xsd"*>

<integration:channel id=*"inboundSQSAdapter"*/>

<aws-sqs:inbound-channel-adapter

propertiesFile=*"classpath:awscredentials.properties"*

channel=*"inboundSQSAdapter"*

sqs-queue=*"https://queue.amazonaws.com/123456789012/MyTestQueue"*

transactional=*"true"*

max-redelivery-attempts=*"3"*>

<integration:poller fixed-rate=*"5000"*>

<integration:transactional isolation=*"DEFAULT"* propagation=*"REQUIRED"*

transaction-manager=*"txManager"*/>

</integration:poller>

</aws-sqs:inbound-channel-adapter>

<bean id=*"txManager"* class=*"SomeTransactionManager"*/>

</beans>

We will summarize the above configuration below.

1. We define a channel over which the inbound channel adapter will place messages retrieved from Amazon SQS Queue in the cloud, in the above configuration the channel is *inboundSQSAdapter.*
2. An inbound channel adapter is configured with the properties file containing the access key and the secret key, the channel over which the messages from Amazon SQS will be placed, the Amazon SQS Queue, max redelivery attempts and a poller sub element configured with the transaction manager configured. In the following section we will see all the possible attributes supported and its significance.
3. A bean definition of the transaction manager.

**Configuring the SQS inbound channel adapter**

**Below are the attributes supported by the AWS SQS inbound adapter tag** (attributes marked bold are mandatory)

|  |  |
| --- | --- |
| **Attribute Name** | **Attribute Description** |
| accessKey | The AWS access key |
| secretKey | The AWS secret key |
| propertiesFile | The properties file containing the AWS access key and secret key. Use any one of propertiesFile or the accessKey, secretKey attributes. Specifying one of these is ***mandatory*** |
| **channel** | The channel over which the messages received from SQS will be placed. The messages payload received from SQS is of type *String,* however, the message is constructed by converting the String message payload to the original payload type (this is the type when the message was sent to the outbound channel adapter) and the *Map* of message attributes if any are added as the header of the message sent over this channel. |
| transactional | Indicates if the adapter participates in an incoming transaction. |
| sqs-queue | The Queue on SQS from which the messages will be polled and received. |
| max-redelivery-attempts | Relevant only if the adapter is transactional. If the message delivery to the channel specified fails for some reason, the message will not be deleted from SQS and will be attempted to be redelivered the given number of times (next redelivery will happen after the visibility timeout of the message completes on SQS) or till successfully delivered which ever happens first before being deleted from SQS. By default redeliver for failed messages is not enabled (value set to 0) and the message will be deleted on transaction completion. |

Apart from the above attributes, the inbound-channel-adapter tag supports the *poller* tag. Refer to spring integration documentation for more details on configuring the poller.

The message sent over the channel configured with the inbound adapter will be exactly identical to the message that would be sent to SQS using the outbound channel adapter.

**Other miscellaneous points**

**Known Issues**

1. In case of inbound adapter, the file if already exists in local directory, a new file with .writing extension if created. However, the renaming doesn’t work after the write to the temp file is done.
2. In case of SQS outbound adapter, the logic for verify messages if the parameter set, is not implemented.

The file testprops.properties contains some instructions about the properties and the values those can be set. Set appropriate values before executing the tests.

The file awscredentials.properties contains the aws credentials of the aws account, set the appropriate credentials before executing any tests

The values in the xml are also hardcoded (need to get these from properties too).Please make appropriate changes to the appropriate config files too before executing the tests.

For any more information, feedback and suggestions feel free to drop me a mail on [amolnayak.spring@gmail.com](mailto:amolnayak.spring@gmail.com)