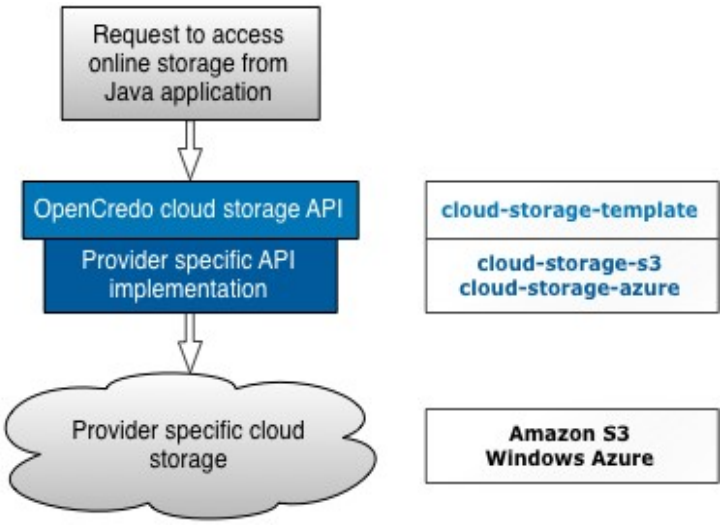
OpenCredo Cloud Storage v.1.2

This document assumes that the reader has at least a basic understanding of storage in the cloud and Spring-Integration. Please refer to JClouds, Amazon Simple Storage Service (Amazon S3), Windows Azure online storage and Spring-Integration for details.

OpenCredo Cloud Storage is motivated by the following goals:

- To provide a simple Spring template style model for implementing applications interacting with cloud storage from Java™.
- To be able to change cloud storage provider with minimal effort.
- To facilitate interaction with cloud storage from Spring Integration.

OpenCredo Cloud Storage consist of five parts: '*cloud-storage-template*', '*cloud-storage-jcloud*', '*cloud-storage-s3*', '*cloud-storage-azure*' and '*cloud-storage-spring-integration-support*'. The diagram below shows how those parts fit together.



cloud-storage-spring-integration-support is not shown on the diagram above because it is an extension specifically for Java applications that use Spring-Integration and will be discussed later in this document.

Naming conventions in OpenCredo Cloud Storage projects

Different cloud providers use different names for similar concepts in the cloud storage domain. OpenCredo tried to unify such names so as to provide a common abstract. The following taxonomy will be extensively used in this reference documentation:

- **container** (bucket in Amazon S3) – Think of this as a folder or directory on “disk” in cloud storage with one restriction; a container can't contain other containers.

- **blob** or **container-object** – Something that can be added(saved)/retrieved/deleted from a *container*. A file in a folder or directory is similar to a blob in a container.

OpenCredo Cloud Storage template (cloud-storage-template)

The *cloud-storage-template* specifies a simple API for interaction with cloud storage.

```
public interface StorageOperations {

    // Get default container name from implementation class.
    public String getDefaultContainerName();

    // List container names in online storage provider
    public List<String> listContainerNames();

    // Operations with storage containers
    public void createContainer(String containerName);
    public void deleteContainer(String containerName);
    public ContainerStatus checkContainerStatus(String containerName);
    public List<BlobDetails> listContainerObjectDetails(String containerName);

    // Operations with objects (blobs) in container
    public void send(String containerName, String objectName, String stringToSend);
    public void send(String containerName, String objectName, File fileToSend);
    public void send(String containerName, String objectName, InputStream isToSend);
    public String receiveAsString(String containerName, String objectName);
    public void receiveAndSaveToFile(String containerName, String objectName, File toFile);
    public InputStream receiveAsInputStream(String containerName, String objectName);
    public void deleteObject(String containerName, String objectName);
    ...
}
```

`StorageOperations` defines core methods to deal with data in the cloud.

`ContainerStatus` represents the status of the container. It is defines as enum with three possible values:

```
public enum ContainerStatus {
    MINE, DOES_NOT_EXIST, ALREADY_CLAIMED
}
```

- MINE – the container was created by current user and it has all access rights to it.
- DOES_NOT_EXIST – the container with specified name does not exist.
- ALREADY_CLAIMED – the container does exist, but the current user is not the owner of it. This container status is specific to Amazon S3 storage's global namespace for its concept of *buckets*. Amazon S3 requires that each container (bucket) has a globally unique name, whereas Windows Azure requires uniqueness per account.

The `BlobDetails` class encapsulates minimal data about a blob.

```
public class BlobDetails {
    private final String containerName;
    private final String name;
    private final String eTag;
    private final Date lastModified;
    ...
}
```

OpenCredo Cloud Storage Template for Amazon S3 (cloud-storage-s3)

cloud-storage-s3 module is an implementation of the *cloud-storage-template* for Amazon S3. An

Amazon Web Services (AWS) account is required to start using *cloud-storage-s3*.

The `S3Template` is the implementation of `StorageOperations` specifically to provide access to Amazon's Simple Storage Service (S3). It is based on the free, open-source Java™ toolkit and application suite JetS3t v.0.8.1.

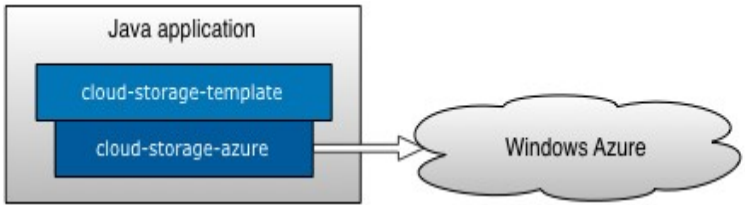
You can use Spring to configure an `S3Template` as shown below, providing your own AWS account details:

```
<!-- Credentials required to access Amazon S3 -->
<bean id="awsCredentials" class="org.opencredo.cloud.storage.s3.AwsCredentials">
       <constructor-arg value="AmazonS3-AWS-key" />
       <constructor-arg value="AmazonS3-AWS-secret-key" />
</bean>

<!-- Template for accessing Amazon S3 -->
<bean id="s3Template" class="org.opencredo.cloud.storage.s3.S3Template">
       <!-- Reference to credentials used by this template to access Amazon S3 -->
       <constructor-arg ref="awsCredentials" />
</bean>
```

OpenCredo cloud storage template backed by JClouds (cloud-storage-jcloud)

The *cloud-storage-jcloud* module is an implementation of the *cloud-storage-template* backed by the opensource project Jclouds which provides a single interface for a range of cloud providers.



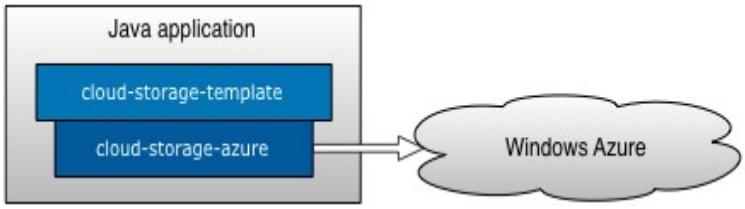
The `JCloudsTemplate` is an implementation of `StorageOperations` which allows *opencredo-cloud-storage* to interact with any provider supported by JClouds. The supported providers are listed in `CloudProvider`.

You can configure the `JCloudTemplate` in the following way

```
private JCloudCredentials credentials = new JCloudCredentials(awsKey, awsSecretKey);
private StorageOperations template =
    new JCloudTemplate(CloudProvider.AWS_S3, credentials, defaultContainerName);
```

OpenCredo cloud storage template for Windows Azure (cloud-storage-azure)

The *cloud-storage-azure* module is an implementation of the *cloud-storage-template* for Windows Azure. *A* Windows Azure platform account is required to use *cloud-storage-azure*.



The `AzureTemplate` is an implementation of `StorageOperations` specifically for Windows Azure. *cloud-storage-azure* interacts with the Windows Azure Storage Services REST API. A REST client is implemented inside the module (this is briefly explained below) and it uses the free and open source Apache HttpComponents v.4.0.1.

You can configure the `AzureTemplate` using Spring as shown below, replacing the indicated credentials with your own account's Azure credentials:

```
<!-- Credentials required to access Windows Azure -->
<bean id="azureCredentials" class="org.opencredo.cloud.storage.azure.AzureCredentials">
       <constructor-arg value="Windows-Azure-account-name" />
```

```
        <constructor-arg value="Windows-Azure-secret-key" />
</bean>

<!-- Template for accessing Windows Azure -->
<bean id="azureTemplate" class="org.opencredo.cloud.storage.azure.AzureTemplate">
        <!-- Reference to credentials used by this template to access Windows Azure -->
        <constructor-arg ref="azureCredentials" />
</bean>
```

Windows Azure REST client

REST clients core interface is `AzureRestService` and it has a single implementation – `DefaultAzureRestService`. This interface mirrors all methods specified in `StorageOperations`.

```
public interface AzureRestService {

    void createContainer(String containerName);
    void deleteContainer(String containerName);
    List<String> listContainerNames();
    void deleteObject(String containerName, String blobName);
    void putObject(String containerName, Blob<?> blob);
    InputStreamBlob getObject(String containerName, String blobName);
    List<BlobDetails> listContainerObjectDetails(String containerName);
    ContainerStatus checkContainerStatus(String containerName);
}
```

It is not intended that `DefaultAzureRestService` will be used directly, the intention being that developers will develop against the `AzureTemplate` instead. *cloud-storage-azure* v.1.0 is based on Azure storage version '2009-09-19'. At the moment it supports only 'Put Blob' operation to create entire Block Blob up to 64 MB.

`Blob<?>` is an abstract class encapsulating container object (blob) data to be sent to Windows Azure. This class has 3 implementations: `StringBlob` (sends string as blob), `FileBlob` (sends file content as blob), and `InputStreamBlob` (sends stream as blob). A received blob from Windows Azure cloud storage is always a `InputStreamBlob`. It's up to the caller to convert the stream to anything it expects. The `AzureTemplate` has convenience methods that encapsulate stream conversion to String or saving to the file.

OpenCredo Cloud Storage SI adapters (cloud-storage-spring-integration-support)

The *cloud-storage-spring-integration-support* is a link between Spring-Integration (SI) and cloud storage. It provides several SI end point adapters.

An *Outbound adapter* sends blob to the cloud storage. An *Inbound adapter* polls cloud storage for blob details (please note it is not a full blob, but just the details) and forwards those details to a blob transformer. The *Transformer's* job is to download the full blob and forward that manipulated blob content for further processing. *Transformer* might need to delete the blob specified in blob details if it is configured to do so. See the diagram below that explains this process:

You can configure your Cloud storage adapters in Spring using custom namespace elements, which will be covered in the following individual adapter sections.

The Outbound adapter

The Outbound adapter takes a message from an SI channel and sends its content to the specified container in cloud storage. Currently the outbound adapter can handle SI messages with String or File payloads. A new blob name for the blob in the container can be defined by providing an implementation of the `BlobNameBuilder` strategy.

```
public interface BlobNameBuilder {

    String createBlobName(Message<?> message);
}
```

The Default strategy is `DefaultBlobNameBuilder`, which returns a blob name in the pattern "`object.<current_date-time_in_milliseconds>`".

Outbound Adapter Namespace Support

The outbound adapter can be defined in Spring using the custom namespace:

```
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:cloud="http://www.opencredo.com/schema/cloud-storage/integration"
       xsi:schemaLocation="...
              http://www.opencredo.com/schema/cloud-storage/integration
              http://www.opencredo.com/schema/cloud-storage/integration/opencredo-si-cloud-
storage-1.2.xsd">

<!-- Outbound adapter -->
<cloud:outbound-channel-adapter
       template="template" container="${containerName}" channel="blobToSendChannel"
       name-builder="blobNameBuilder" />
```

The Outbound adapter custom element includes the following attributes:

- 'template' – a reference to implementation of `StorageOperations` bean.
- 'container' – the default name of the cloud storage container. This container will be used for blob upload/download if other container not specified explicitly.

- 'channel' – a reference to the SI channel from where blobs are coming.
- 'name-builder' – a reference to an implementation of `BlobNameBuilder` bean. This attribute is optional as the `DefaultBlobNameBuilder` will be used by default.

The Inbound adapter

The Inbound adapter polls a specified cloud storage container and receives a list of `BlobDetails` (see `BlobDetails` description in previous sections). It is important to note that the Inbound adapter never downloads the full blob. It filters the received `BlobDetails` list to identify those blobs that shall be downloaded. Filtered `BlobDetails` are added to the queue (queue ordering can be customised to allow for prioritisation). Each subsequent poll will then remove the head of that queue with the `BlobDetails` instance then becoming the payload of the sent Spring Integration message. It is then the job of the transformer to download the full blob and transform its data to the expected form for further processing.

The Inbound adapter is state-full, that means it will not check online storage container on each polling request, but will return the first `BlobDetails` from the queue if there is any. The online storage container is checked for new `BlobDetails` if the queue is empty. The following diagram walks through this behaviour in more detail:

As mentioned before you can change the `BlobDetails` filtering criteria and the sequence in which filtered `BlobDetails` appear in the queue to be processed.

There are three predefined filters: `AcceptOnceBlobNameFilter` (default filter for Inbound adapter), `PatternMatchingBlobNameFilter` and `CompositeBlobDetailsFilter`. You can also create their own filters by implementing `BlobDetailsFilter` interface or extending `AbstractBlobDetailsFilter`.

```
public interface BlobDetailsFilter {

    List<BlobDetails> filter(List<BlobDetails> objects);
}
```

Implementation of the marker interface `BlobDetailsComparator` gives you the opportunity to customize the sequence of `BlobDetails` in the processing queue. Only one default implementation is provided out of the box at the moment and it is the `BlobLastModifiedDateComparator` that orders `BlobDetails` by last modified date.

Inbound Adapter Namespace Support

Identically to the outbound adapter, an inbound adapter can be defined using a custom Spring namespace:

```
<!-- Inbound adapter -->
<cloud:inbound-channel-adapter channel="blobDetailsChannel"
        template="template" container="${containerName}" filter="blobNameFilter"
        comparator="lastModifiedDateComparator">
        <si:poller>
                <si:interval-trigger interval="5000" />
        </si:poller>
</cloud:inbound-channel-adapter>
```

Inbound adapter custom element attributes:

- 'channel' – a reference to SI channel where incoming `BlobDetails` will be pushed.

- 'template' – a reference to an implementation of `StorageOperations`.
- 'container' – the default name of the cloud storage container. This container will be used for blob upload/download if other container not specified explicitly.
- 'filter' – a reference to an implementation of `BlobDetailsFilter` bean. This attribute is optional; by default the `AcceptOnceBlobNameFilter` will be used.
- 'comparator' – a reference to an implementation of `BlobDetailsComparator`. This attribute is also optional; `BlobLastModifiedDateComparator` will be used by default.

`BlobDetails` and Transformer

Blob data is downloaded by a transformer that can be configured somewhere in the SI pipeline. A transformer expects `BlobDetails` as the SI message payload and uses its information to identify the container and blob to be downloaded. Two transformers are provided: `BlobToByteArrayTransformer` (creates a byte array from blob data) and `BlobToStringTransformer` (creates a string from the downloaded blob data). Both transformers push results to an output SI channel for further processing. Before results are pushed to output channel, transformer might delete the blob from the container. It is possible to create custom transformer by implementing `BlobTransformer` interface or extend `AbstractBlobTransformer` class.

```
public interface BlobTransformer<T> {

    Message<T> transform(Message<BlobDetails> message) throws BlobTransformException;
}
```

A string transformer can be configured as shown below:

```
<!-- Blob to String transformer -->
<bean id="toStringTransformer"
      class="org.opencredo.cloud.storage.si.transformer.internal.BlobToStringTransformer">
       <constructor-arg ref="template" />
       <!-- Optional constructor argument specifying that blob should be deleted after download-->
       <constructor-arg ref="true" />
</bean>

<!-- SI transformer configuration -->
<si:service-activator input-channel="blobDetailsChannel"
       ref="toStringTransformer" method="transform" output-channel="furtherProcessingChannel" />
```

It is important to note that the inbound adapter (providing `BlobDetails`) and transformer (download and, if required, delete the blob specified in `BlobDetails`) are using the same cloud storage provider account. Account details are provided when the 'template' (`S3Template` or `AzureTemplate`) is created.

Custom namespace

The custom SI namespace for OpenCredo cloud storage (http://www.opencredo.com/schema/cloud-storage/integration) can be applied as shown below:

```
<?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:si="http://www.springframework.org/schema/integration"
       xmlns:cloud="http://www.opencredo.com/schema/cloud-storage/integration"
       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
       http://www.springframework.org/schema/integration/spring-integration-2.0.xsd
              http://www.opencredo.com/schema/cloud-storage/integration
       http://www.opencredo.com/schema/cloud-storage/integration/opencredo-si-cloud-storage-
1.2.xsd">

       …

       <!-- Inbound adapter -->
       <cloud:inbound-channel-adapter channel="blobDetailsChannel"
              template="template" container="${containerName}" filter="blobNameFilter"
              comparator="lastModifiedDateComparator">
              <si:poller>
                     <si:interval-trigger interval="5000" />
              </si:poller>
       </cloud:inbound-channel-adapter>

       <!-- Outbound adapter -->
       <cloud:outbound-channel-adapter
              template="template" container="${containerName}" channel="blobToSendChannel"
              name-builder="blobNameBuilder" />
</beans>
```

References:

Amazon Simple Storage Service (Amazon S3): http://aws.amazon.com/s3/

JClouds: http://www.jclouds.org/

Windows Azure: http://www.microsoft.com/windowsazure/

Spring-Integration (SI): http://www.springsource.org/spring-integration

JetS3t: http://jets3t.s3.amazonaws.com/index.html

Apache HttpComponents: http://hc.apache.org/