SAP Customer Experience

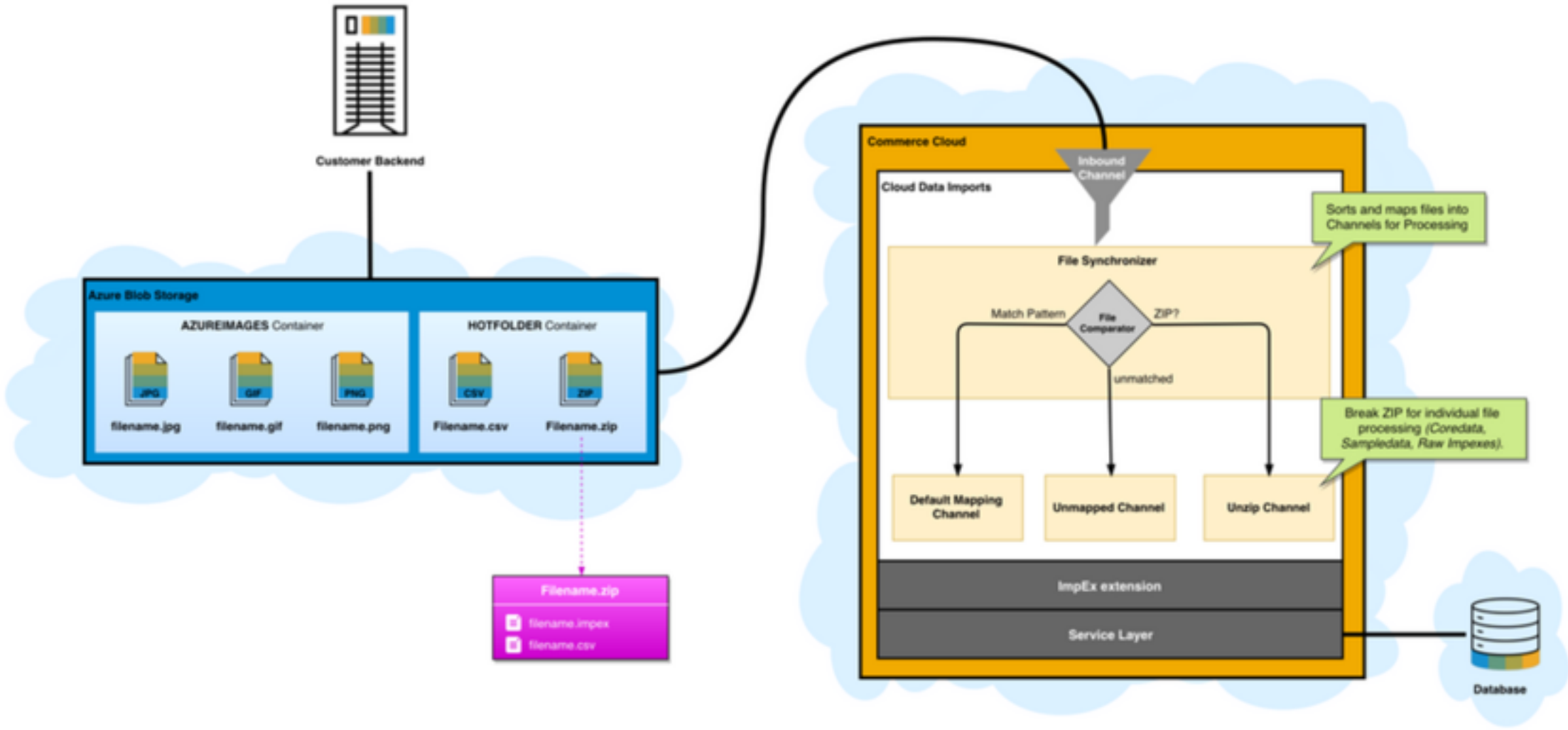
Cloud Hot Folder

THE BEST RUN SAP

Architecture

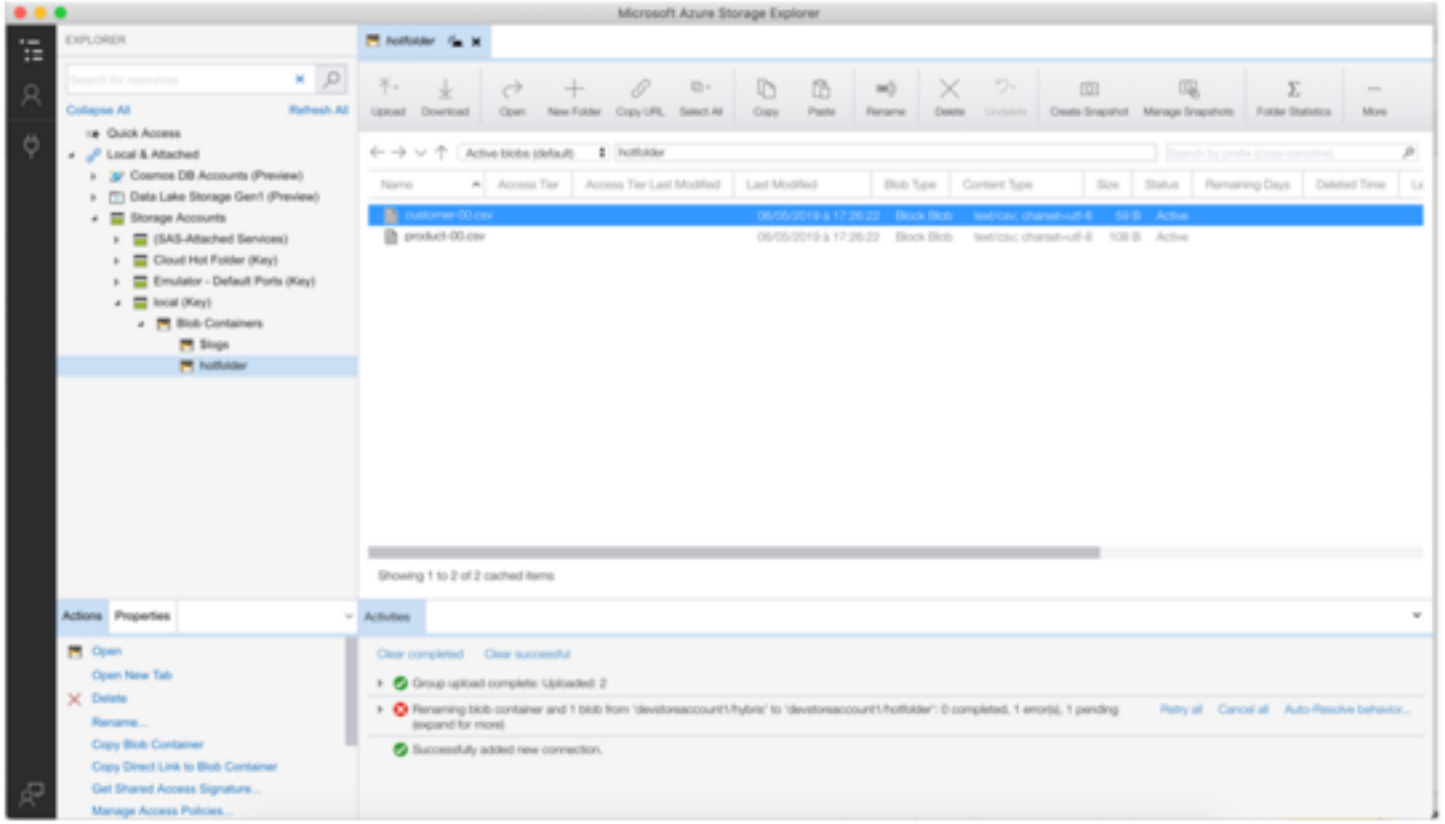
- At a high-level, the following diagram shows how the solution has evolved from SAP Commerce Cloud on SAP Infrastructure to the new SAP Commerce Cloud.
- Major updates are
  1. Remote Storage Support (Azure Cloud Storage)
  2. Support for ZIP Files (Core Data, Sample Data and Raw ImpEx Files)
  3. Support for URL Media Files
  4. File Sorting

Architecture …



Architecture …

- Improved Monitoring Within SAP Commerce Cloud, the Hot Folders module was extended to include the above improvement
- No SSH file transfer protocol (SFTP)server (for uploading media) or data files (for importing) is no longer provided.
- Instead, you have a Cloud Hot Folder that makes use of the Azure Blob Store as a file source.



Cloud Hot Folder Processing Flow

The following steps take place:

- Cloud hot folders continually monitor a specific directory in your Blob storage for the presence of new files.
- The cloud hot folder moves any files placed in the Blob directory to a temporary processing directory that it creates in the Blob storage.
- The cloud hot folder downloads files from the processing directory to a standard hot folder in SAP Commerce.
- Standard hot folders unzip files and convert them into compatible formats for SAP Commerce import. This process follows the mapping and processing rules that you define. Cloud hot folders play no part in the process pipeline of standard hot folders.
- When the hot folder finishes processing files, the cloud hot folder moves them from the temporary processing directory to either an:
- Archive directory, which contains files successfully imported, or an
- Error directory, which contains files that threw one or more exceptions during processing.
- Files moved to the Archive or Error directory remain there until you delete them. To reprocess 'failed files', move them back to the Blob directory for the cloud hot folder to pick up.

What are hot folders and Cloud Hot Folders?

Standard hot folders are not the same as Cloud Hot Folders.

Standard hot folders

- are part of SAP Commerce (OOTB)
- import product data etc, in the SAP commerce system/DB

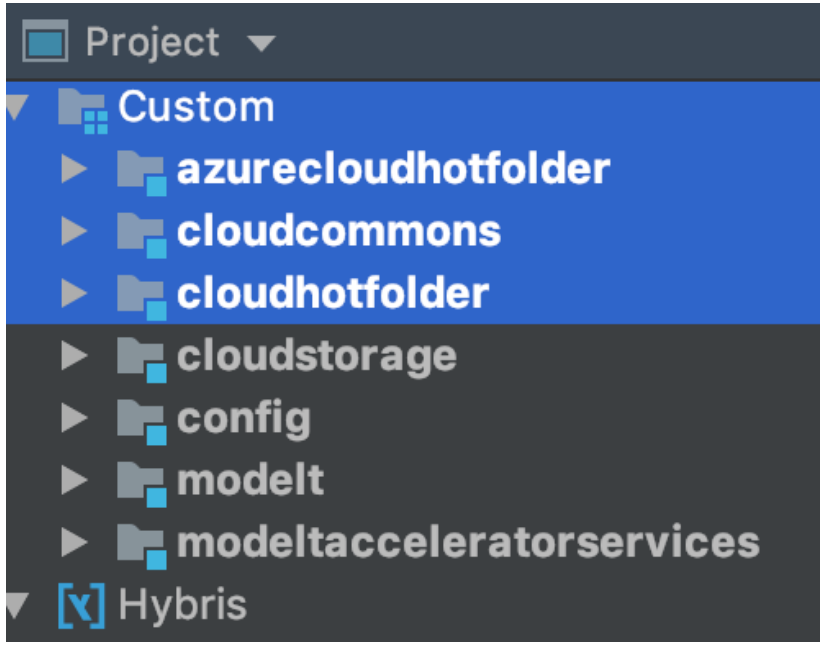
Cloud Hot Folders

- How do they differ from standard HFs?
- Why do I need them?
- What improvements, if any, do they add to existing HFs?

Example Commerce environment manifest

manifest.json

```
{
    "commerceSuiteVersion": "1811",
    "extensions": [
        "kiwi",
        "tiger",
        "yacceleratorstorefront",
        "backoffice",
        "cmsbackoffice",
        "platformbackoffice",
        "modeltacceleratorservices",
        "electronicsstore",
        "yacceleratorfulfilmentprocess",
        "albinoaddon",
        "azurecloudhotfolder",
```



Project structure

CHF Ext.

azurecloudhotfolder

Pulls from Azure Blob storage, filters and sorts

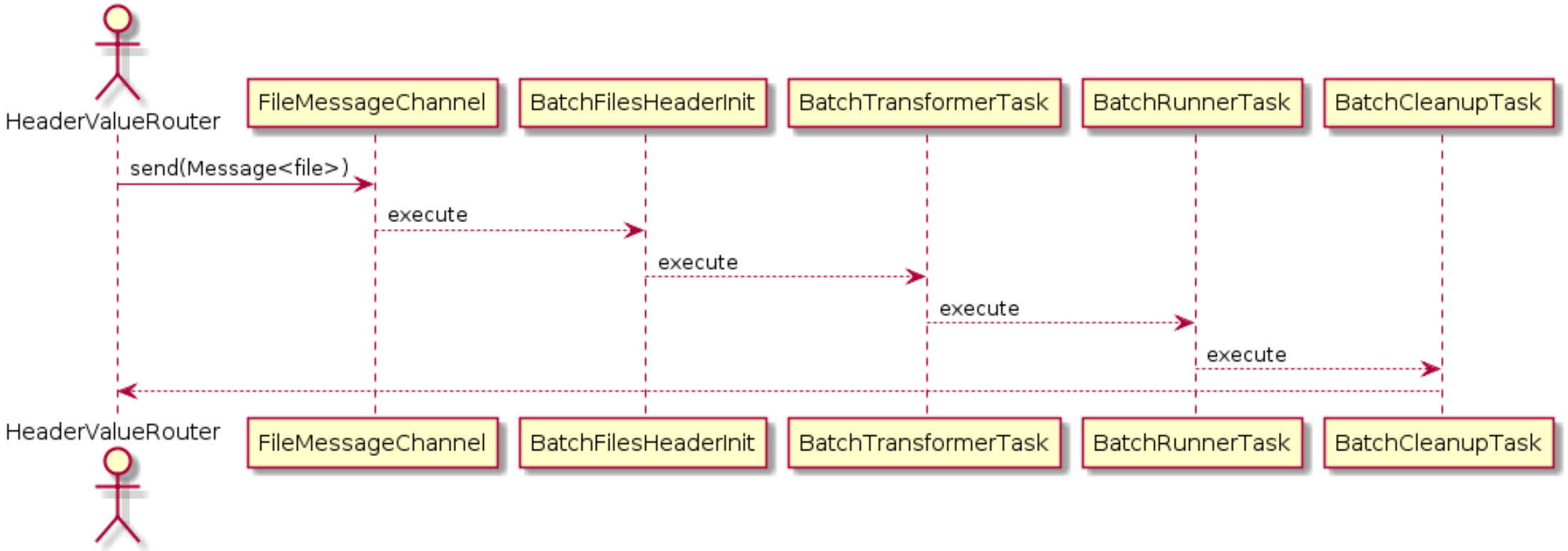
cloudhotfolder

Tasks for processing ZIP folders

cloudcommons

AOP, services for task executors, roles & plugging CHF into HF flow

Standard/OOTB Hot Folders file processing flow



OOTB Impex transformer mappings

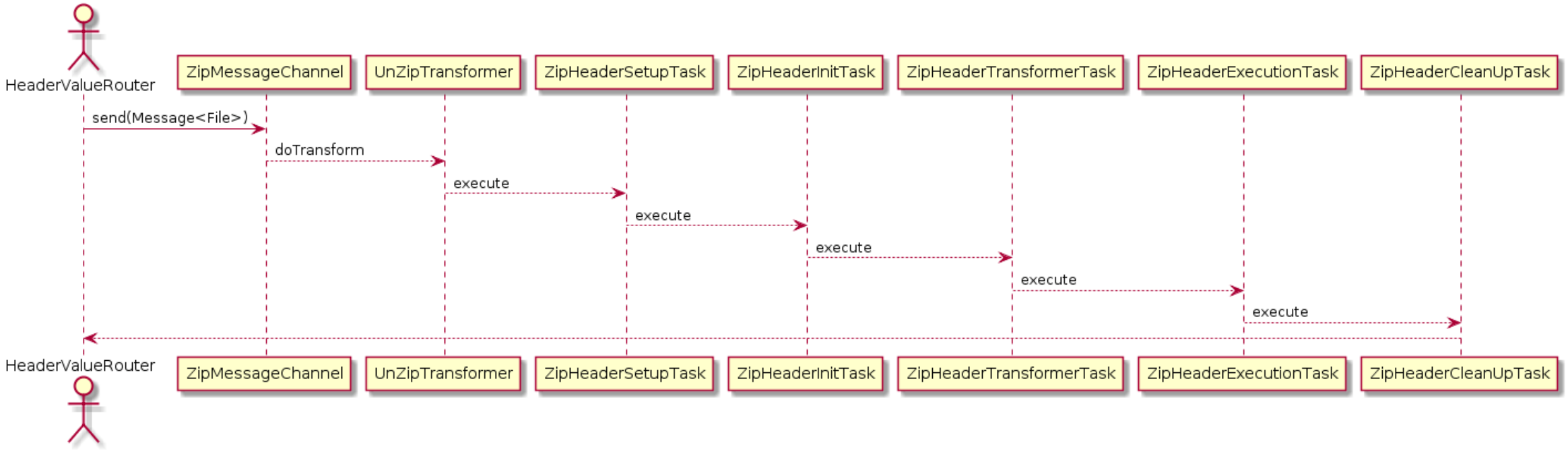
```
<bean id="baseDirectoryApparel" class="java.lang.String">
    <constructor-arg value="#{baseDirectory}/${tenantId}/apparel" />
</bean>
<!-- 1) Scan for files -->
<file:inbound-channel-adapter id="batchFilesApparel" directory="#{baseDirectoryApparel}"
    filename-regex="^(.*)-(\d+)\.csv" comparator="fileOrderComparator">
    <int:poller fixed-rate="1000" />
</file:inbound-channel-adapter>
```

```
<!-- Transformer converters mappings -->
<bean id="batchApparelProductConverterMapping"
      class="de.hybris.platform.acceleratorservices.dataimport.batch.converter.mapping.impl.DefaultConverterMapping"
      p:mapping="base_product"
      p:converter-ref="batchApparelProductConverter"/>

<bean id="batchSizeVariantConverterMapping"
      class="de.hybris.platform.acceleratorservices.dataimport.batch.converter.mapping.impl.DefaultConverterMapping"
      p:mapping="variant"
      p:converter-ref="batchSizeVariantConverter"/>

<bean id="batchStyleVariantConverterMapping"
      class="de.hybris.platform.acceleratorservices.dataimport.batch.converter.mapping.impl.DefaultConverterMapping"
      p:mapping="variant"
      p:converter-ref="batchStyleVariantConverter"/>
```

OOTB hardcoded impex

```
<!-- Transformer converters mappings -->
<bean id="batchApparelProductConverterMapping"
      class="de.hybris.platform.acceleratorservices.dataimport.batch.converi
      p:mapping="base_product"
      p:converter-ref="batchApparelProductConverter"/>


<!-- Apparel specific converters -->
<bean id="batchApparelProductConverter" class="de.hybris.platform.acceleratorserv
    <property name="header">
        <value>#{defaultImpexProductHeader}
            # Insert Apparel Products
            INSERT_UPDATE ApparelProduct;code[unique=true];varianttype(code);name
        </value>
    </property>
    <property name="impexRow">
        <value>;{+0};{1};{2};{3};{4};{5};{6};{7};{8};{9};{S}</value>
    </property>
    <property name="rowFilter">
        <bean class="de.hybris.platform.acceleratorservices.dataimport.batch.conv
            <property name="expression" value="row[1]"/>
        </bean>
    </property>
    <property name="type" value="ApparelProduct"/>
</bean>
```

Cloud Hot Folders file processing flow



What is a header/zip header?

```
public class BatchHeader
{
    private Long sequenceId;
    private File file;
    private List<File> transformedFiles;
    private String encoding;
    private String storeBaseDirectory;
    private String catalog;
    private String language;
    private boolean net;
```

1 CSV File ->
1 BatchHeader ->
List or resultant IMPEX files

1 ZIP folder ->
1 ZipBatchHeader with
Collection of CSV files ->
List of resultant IMPEX

```
 * <ul>
 * <li>originalFileName: the original ZIP file name before it was exploded</li>
 * <li>unzippedFolder: the directory into which the ZIP was exploded</li>
 * <li>unzippedFiles: a collection of files that were exploded from within the ZIP
 * <li>originalToTransformedMap: a collection which maps an original file name to
 * </ul>
 */
public class ZipBatchHeader extends BatchHeader
{
    private String originalFileName;
    private File unzippedFolder;
    private Collection<File> unzippedFiles;
    private MultiMap originalToTransformedMap;
```

SYNCHRONIZATION
SORTING
FILTERING

Central Azure Blob Store

azureInboundChannelAdapter

AzureBlobSynchronzingMessageSource

AzureBlobInboundSynchrnoizer

hotfolderInboundFileNameHeaderEnricherChannel

ROUTING TO FILE
PROCESSING
CHANNELS

filename lastMod

<router/>
$(filename)

cloudHotFolderUnzipChannel

$(filename) =
^(?:.*)\.(?I)(zip)$

batchFilesCloudHotFolderDefaultProc
$(filename) =
$(cloud.hotfolder.default.mapping.file.name.patten)

unmappedHotFolderChannel

See other figure
For the process
Via these two channels

Central Azure Blob Store

Archive

Error

All Tasks Successful

Error In Any Task

	Zip Header Set Up Task	Zip Header Init Task	Zip Header Transformer Task	Zip Header Execution Task	Zip Header Cleanup Task
Unzipped folder with a raw IMPEX and CSV is handled					
To Import COREDATA via the zip flow we must observe this file structure	As above	As above	Not necessary, only CSVs are transformed to IMPEX	COREDATA is imported in a specific sequence	As above
To Import SAMPLEDATA via the zip flow we must observe this file structure	As above	As above	Not necessary, only CSVs are transformed to IMPEX	SAMPLEDATA is imported in a specific sequence	As above

ZIP FOLDER
PROCESSING

Synchronization, Filtering, and Sorting

These 3 elements are the job of the **azurecloudhotfolder**

- **Synchronization:** Remote Azure > disk
- **Filter:** Only sync those that match a configured regex
- **Sort:** We must preserve some sort of order for impex importing

The Synchronizer Bean

```
<!-- Azure Inbound File Synchronizer and Channel Adapter -->

<alias name="defaultAzureBlobInboundSynchronizer" alias="azureBlobInboundSynchronizer"/>
<bean id="defaultAzureBlobInboundSynchronizer"
      class="de.hybris.platform.cloud.azure.hotfolder.remote.inbound.AzureBlobInboundSynchronizer">
    <constructor-arg name="sessionFactory" ref="azureBlobSessionFactory"/>
    <property name="remoteDirectory" value="#{azureHotfolderRemotePath}"/>
    <property name="moveToRemoteDirectory" value="#{azureHotfolderProcessingRemotePath}"/>
    <property name="deleteRemoteFiles" value="${azure.hotfolder.storage.delete.remote.files}"/>
    <property name="preserveTimestamp" value="true"/>
    <property name="filter" ref="azureHotfolderFileFilter"/>
    <property name="comparator" ref="azureHotFolderFileComparator"/>
</bean>
```

Filter and comparator chain simply fields on our synchronizer

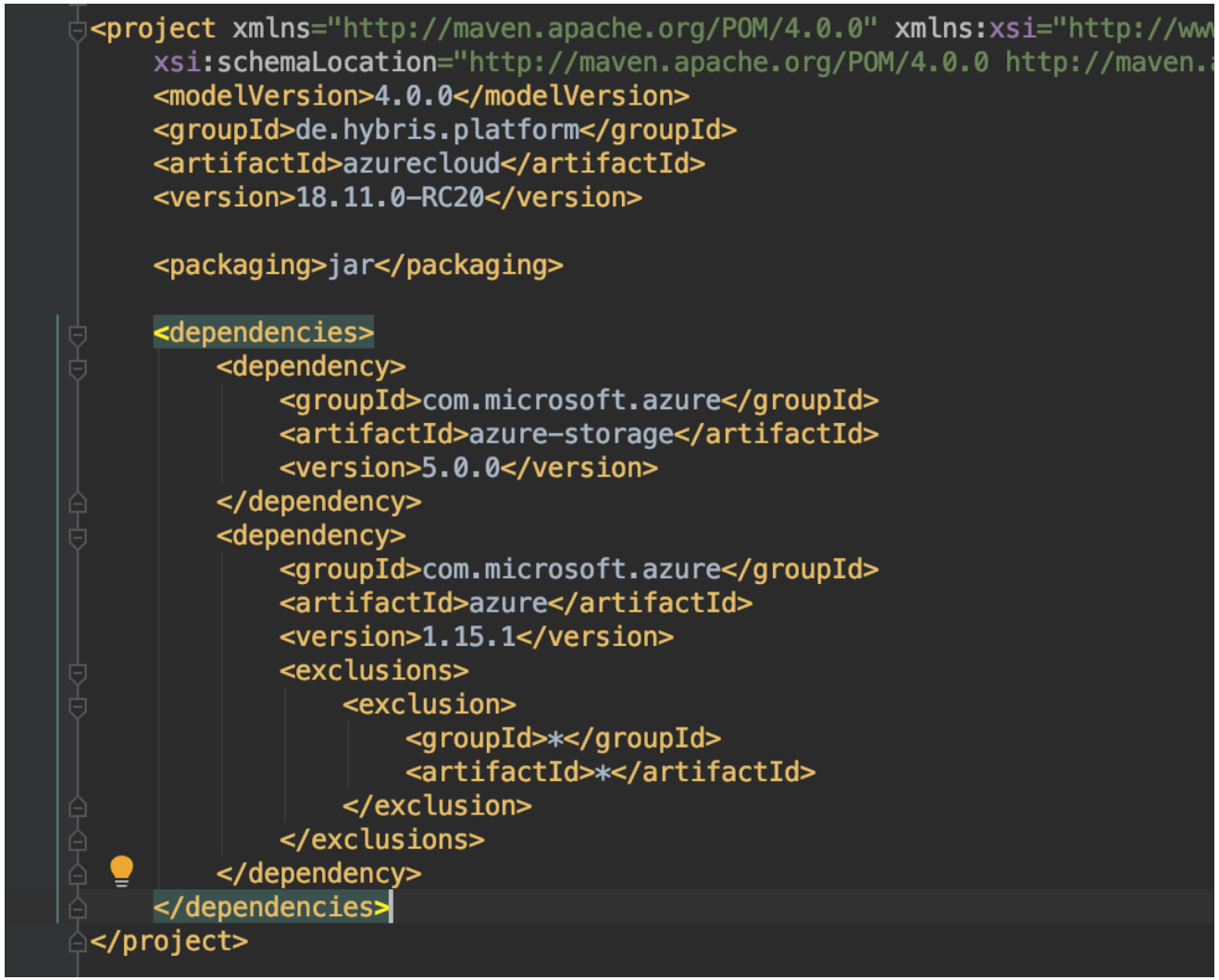
Important configuration properties

azurecloudhotfolder/.../azurecloudhotfolder-spring.xml

```
<!-- Local and Remote Path properties -->

<bean id="azureHotfolderRemotePath" class="java.lang.String">
    <constructor-arg name="value"
                     value="${azure.hotfolder.storage.container.hotfolder}"/>
</bean>

<bean id="azureHotfolderProcessingRemotePath" class="java.lang.String">
    <constructor-arg name="value"
                     value="${azure.hotfolder.storage.container.hotfolder}/processing"/>
</bean>

<bean id="azureHotfolderErrorRemotePath" class="java.lang.String">
    <constructor-arg name="value"
                     value="${azure.hotfolder.storage.container.hotfolder}/error"/>
</bean>

<bean id="azureHotfolderArchiveRemotePath" class="java.lang.String">
    <constructor-arg name="value"
                     value="${azure.hotfolder.storage.container.hotfolder}/archive"/>
</bean>

<bean id="azureHotfolderUnmappedRemotePath" class="java.lang.String">
    <constructor-arg name="value"
                     value="${azure.hotfolder.storage.container.hotfolder}/unmappped"/>
</bean>

<bean id="azureHotfolderLocalDirectoryBase" class="java.lang.String">
    <constructor-arg name="value"
                     value="#{configurationService.configuration.getProperty('azure.hotfolder.local.sync.basefolder')}"/>
</bean>
```

How do we implement synchronization?

We utilize the CloudStorageAccount class as part of the azure-storage library



How do we Filter and Why?

- AzureBlobRegexPatternFileListFilter extends spring AbstractRegexPatternFileListFilter<CloudBlob>
  - Implementation provided as pattern to be customized
  - Prefix != ignore

- AzureBlobPersistentAcceptOnceListFilter extends spring AbstractPersistentAcceptOnceFileListFilter<CloudBlob>
  - By default in memory store used – once only for current node only
  - Persistence store has performance concerns
  - Easily swap between two using Spring

NOTE order in filter chain

```
    Use the simple metadata store if you expect a high fregency of many, small files.
    The simple metadata store uses an in-memory, concurrent map as a store. This provides
    considerable performance benefits when compared to the hybris persistent store.
    However, the store is not cluster wide, and is lost on restarts.
-->
<bean id="simpleHotfolderMetadataStore"
      class="de.hybris.platform.cloud.commons.spring.integration.metadata.ExpiringSimpleMetadataStore">
    <constructor-arg name="maxSize" value="${cloud.hotfolder.metadata.max-size}"/>
    <constructor-arg name="ttl" value="${cloud.hotfolder.metadata.ttl}"/>
</bean>

<!--
    Use the persistent metadata store if you expect a low volume of files and you
    need to cluster-wide, persistent and guaranteed once only processing.
-->
<bean id="persistenHotfolderMetadataStore"
    parent="abstractHybrisMetadataStore">
    <constructor-arg name="region" value="${azure.hotfolder.storage.metadatastore.region}"/>
</bean>

<alias name="defaultAzureBlobPersistentAcceptOnceListFilter"
       alias="azureBlobPersistentAcceptOnceListFilter"/>
<bean id="defaultAzureBlobPersistentAcceptOnceListFilter"
      class="de.hybris.platform.cloud.azure.hotfolder.remote.file.filters.AzureBlobPersistentAcceptOnceListFilter">
    <constructor-arg name="store" ref="hotfolderMetadataStore"/>
    <constructor-arg name="prefix" value="${azure.hotfolder.storage.metadatastore.prefix}"/>
</bean>

<alias name="defaultAzureFileFilterList" alias="azureHotfolderFileFilterList"/>
<util:list id="defaultAzureFileFilterList">
    <ref bean="azureBlobRegexPatternFileListFilter"/>
    <ref bean="azureBlobPersistentAcceptOnceListFilter"/>
</util:list>

<alias name="azureChainFileListFilter" alias="azureHotfolderFileFilter"/>
<bean id="azureChainFileListFilter"
      class="de.hybris.platform.cloud.commons.spring.integration.file.filters.ChainFileListFilter">
    <constructor-arg name="fileFilters" ref="azureHotfolderFileFilterList"/>
</bean>
```

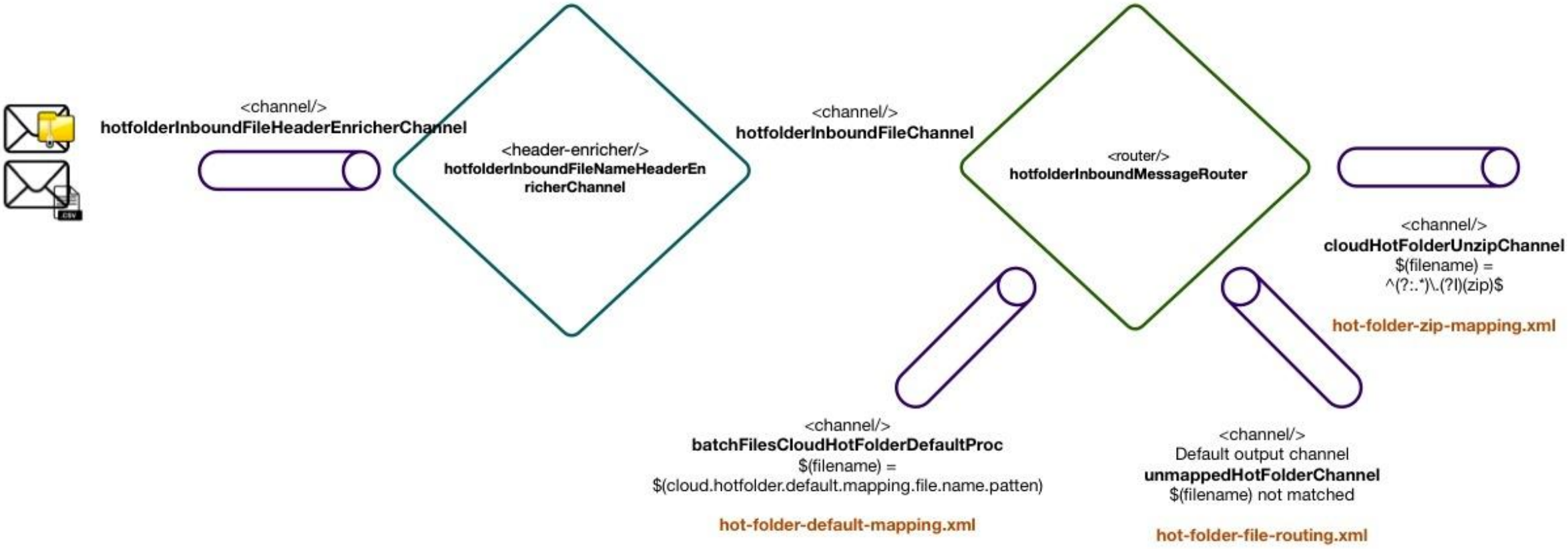
How do we sort, and why?

cloudhotfolder/integration/hot-folder-file-sorting.xml

- Classes implement Comparator<CloudBlob>
- Reserved prefixes (coredata, sampledata…) are first
- Then sort by filename
- Identical filename prefixes then sorted by sequence ID (optional)
- …then by last modified of Blob item (NOT timestamp in filename)
- Regex expects <filename>-<?sequenceid>-<?timestamp>.csv

```
<!-- Comparator chain for comparing remote files before synchronisation -->

<bean id="azureHotFolderFileNamePrefixComparator"
      class="de.hybris.platform.cloud.azure.hotfolder.remote.file.comparators.AzureBlobNameComparatorAdapter">
    <constructor-arg name="comparator" ref="cloudHotFolderNamePrefixComparator"/>
</bean>

<bean id="azureHotFolderFileNameComparator"
      class="de.hybris.platform.cloud.azure.hotfolder.remote.file.comparators.AzureBlobNameComparatorAdapter">
    <constructor-arg name="comparator" ref="cloudHotFolderNameComparator"/>
</bean>

<bean id="azureHotFolderFileNameSequenceComparator"
      class="de.hybris.platform.cloud.azure.hotfolder.remote.file.comparators.AzureBlobNameComparatorAdapter">
    <constructor-arg name="comparator" ref="cloudHotFolderNameSequenceComparator"/>
</bean>

<bean id="azureHotFolderFileModifiedComparator"
      class="de.hybris.platform.cloud.azure.hotfolder.remote.file.comparators.AzureBlobTimestampComparatorAdapter"
    <constructor-arg name="comparator" ref="cloudHotFolderModifiedComparator"/>
</bean>

<alias name="defaultAzureHotFolderFileComparatorList" alias="azureHotFolderFileComparatorList"/>
<util:list id="defaultAzureHotFolderFileComparatorList">
    <ref bean="azureHotFolderFileNamePrefixComparator"/>
    <ref bean="azureHotFolderFileNameComparator"/>
    <ref bean="azureHotFolderFileNameSequenceComparator"/>
    <ref bean="azureHotFolderFileModifiedComparator"/>
</util:list>

<alias name="azureHotFolderFileComparatorChain" alias="azureHotFolderFileComparator"/>
<bean id="azureHotFolderFileComparatorChain" class="org.apache.commons.collections4.comparators.ComparatorChain">
    <constructor-arg ref="azureHotFolderFileComparatorList"/>
</bean>
```

File Routing for ZIP Folders



What can I import in a ZIP and how?

- **ZIP Flow - Import CSV file within a ZIP Folder**
- To import CSV files, simply include them in the root of the ZIP file

- **ZIP Flow - Uploading raw ImpEx within a ZIP Folder**
- To import raw impex, include it in the root of the ZIP file. You can use relative file references within your ImpEx, using a suitable folder structure inside the ZIP.

- **ZIP Flow - Using a ZIP folder to Import Core and Sample data**
- The ZIP file flow is able to import core and sample data - Add the files to the ZIP archive following the conventions and rules for core and sample data. The ZIP file name must match the pattern '**coredata**.* ' for core data and '**sampledata**.*' for sample data.
  Inside the ZIP, the data files should be in a folder called '**import/coredata**' for core data and '**import/sampledata**' for sample data.

The following properties are available to customize the behavior of the ZIP flow in <*HYBRIS_BIN_DIR*>/ext-cloud/cloudhotfolder/project.properties

```
<!-- 1) Map zip files to channel based on name pattern -->
<bean class="org.springframework.beans.factory.config.MethodInvokingFactoryBean">
    <property name="targetObject" ref="hotfolderInboundFileChannelMappings"/>
    <property name="targetMethod" value="put"/>
    <property name="arguments">
        <list>
            <bean class="java.util.regex.Pattern" factory-method="compile">
                <constructor-arg value="^(?:.*)\.(?i)(zip)$"/>
            </bean>
            <ref bean="cloudHotFolderUnzipChannel"/>
        </list>
    </property>
</bean>

<int:channel id="cloudHotFolderUnzipChannel"/>

<!-- 2) extract the zip -->
<int:transformer input-channel="cloudHotFolderUnzipChannel"
                 output-channel="setupUnzippedFilesHeader"
                 ref="unZipTransformer"/>

<bean id="unZipTransformer"
      class="de.hybris.platform.cloud.hotfolder.spring.integration.zip.CloudUnZipTransformer">
    <property name="zipResultType" value="FILE"/>
    <property name="deleteFiles" value="false"/>
    <property name="expectSingleResult" value="false"/>
    <property name="workDirectory" ref="zipProcessingDir"/>
</bean>
```

```
    <!-- 3) prepare the header for the contents of the extracted zip -->
    <int:service-activator input-channel="setupUnzippedFilesHeader"
                           output-channel="initUnzippedFilesHeader"
                           ref="zipHeaderSetupTask"
                           method="execute"/>

    <bean id="zipHeaderSetupTask"
          class="de.hybris.platform.cloud.hotfolder.dataimport.batch.zip.task.ZipHeaderSetupTask">
        <property name="fileNameHeaderKey" ref="fileNameHeaderKey"/>
        <property name="unZipDirectoryHeaderKey" ref="unzipDirectoryHeaderKey"/>
        <property name="catalog" value="electronicsProductCatalog"/>
        <property name="net" value="false"/>
    </bean>

    <!-- 4) init default values required -->
    <int:service-activator input-channel="initUnzippedFilesHeader"
                           output-channel="convertUnzippedCsvFilesHeader"
                           ref="zipHeaderInitTask"
                           method="executeZip"/>

    <bean id="zipHeaderInitTask" class="de.hybris.platform.cloud.hotfolder.dataimport.batch.zip.task.ZipHeaderInitTask"
          parent="headerInitTask">
        <property name="sequenceIdParser" ref="zipBatchSequenceIdParser"/>
        <property name="languageParser" ref="zipBatchLanguageParser"/>
        <property name="fallbackLanguage" value="en"/>
    </bean>

    <bean id="zipBatchSequenceIdParser" class="de.hybris.platform.acceleratorservices.dataimport.batch.util.SequenceIdParser">
        <property name="parser">
            <bean class="de.hybris.platform.acceleratorservices.util.RegexParser">
                <property name="regex" value="-(\d+)\.zip"/>
            </bean>
        </property>
    </bean>
    <bean id="zipBatchLanguageParser" class="de.hybris.platform.acceleratorservices.util.RegexParser">
        <property name="regex" value="-(\w{2})-(\d+)\.zip"/>
    </bean>
```

```
<!-- 5) convert any CSVs contained in the extracted zip -->
<int:service-activator input-channel="convertUnzippedCsvFilesHeader"
                       output-channel="executeUnzippedFilesHeader"
                       ref="zipHeaderTransformerTask"
                       method="executeZip"/>

<bean id="zipHeaderTransformerTask"
      class="de.hybris.platform.cloud.hotfolder.dataimport.batch.zip.task.ZipHeaderTransformerTask"
      parent="batchTransformerTask"/>

<bean id="cloudHotFolderZipMediaConverterMapping"
      class="de.hybris.platform.acceleratorservices.dataimport.batch.converter.mapping.impl.DefaultConverterMapping"
      p:mapping="zip_media"
      p:converter-ref="batchMediaConverter"/>

<bean id="cloudHotFolderZipMediaContainerConverterMapping"
      class="de.hybris.platform.acceleratorservices.dataimport.batch.converter.mapping.impl.DefaultConverterMapping"
      p:mapping="zip_media"
      p:converter-ref="batchMediaContainerConverter"/>

<bean id="cloudHotFolderZipMediaProductConverterMapping"
      class="de.hybris.platform.acceleratorservices.dataimport.batch.converter.mapping.impl.DefaultConverterMapping"
      p:mapping="zip_media"
      p:converter-ref="batchMediaProductConverter"/>
```

class ZipHeaderTransformerTask extends ImpexTransformerTask

```
protected void convertCsv(final ZipBatchHeader header, final File file) throws IOException
{
    final List<ImpexConverter> converters = getConverters(file);
    int position = 1;
    for (final ImpexConverter converter : converters)
    {
        final File impexFile = getImpexFile(file, position++);
        if (convertFile(header, file, impexFile, converter))
        {
            header.addTransformedFile(impexFile);
            header.addOriginalToTransformedEntry(file.getName(), impexFile.getName());
        }
        else
        {
            getCleanupHelper().cleanupFile(impexFile);
        }
    }
}
```

getConverters -> Beans of type Converter wired in standard OOTB HF (We plugin back into the system!)

```
<bean id="batchTransformerTask"
      class="de.hybris.platform.acceleratorservices.dataimport.batch.task.ImpexTransformerTask"
      init-method="initConvertersMap">
    <property name="fieldSeparator" value="," />
    <property name="encoding" value="UTF-8" />
    <property name="linesToSkip" value="0"/>
    <property name="cleanupHelper" ref="cleanupHelper" />
</bean>

<!-- Transformer converters mappings -->
<bean id="batchProductConverterMapping"
      class="de.hybris.platform.acceleratorservices.dataimport.batch.converter.mapping.impl.DefaultConverterMapping"
      p:mapping="base_product"
      p:converter-ref="batchProductConverter"/>

<bean id="batchTaxConverterMapping"
      class="de.hybris.platform.acceleratorservices.dataimport.batch.converter.mapping.impl.DefaultConverterMapping"
      p:mapping="tax"
      p:converter-ref="batchTaxConverter"/>

<bean id="batchExternalTaxConverterMapping"
      class="de.hybris.platform.acceleratorservices.dataimport.batch.converter.mapping.impl.DefaultConverterMapping"
      p:mapping="external_tax"
      p:converter-ref="batchExternalTaxConverter"/>

<bean id="batchPriceConverterMapping"
      class="de.hybris.platform.acceleratorservices.dataimport.batch.converter.mapping.impl.DefaultConverterMapping"
      p:mapping="price"
      p:converter-ref="batchPriceConverter"/>

<bean id="batchStockConverterMapping"
      class="de.hybris.platform.acceleratorservices.dataimport.batch.converter.mapping.impl.DefaultConverterMapping"
      p:mapping="stock"
      p:converter-ref="batchStockConverter"/>

<bean id="batchStockIncrementalConverterMapping"
      class="de.hybris.platform.acceleratorservices.dataimport.batch.converter.mapping.impl.DefaultConverterMapping"
      p:mapping="inc-stock"
      p:converter-ref="batchStockIncrementalConverter"/>

<bean id="batchMerchandiseConverterMapping"
      class="de.hybris.platform.acceleratorservices.dataimport.batch.converter.mapping.impl.DefaultConverterMapping"
      p:mapping="merchandise"
      p:converter-ref="batchMerchandiseConverter"/>
```

After transformation, pass to execute as shown in slide 28

```
<!-- 6) import the contents of the extracted zip -->
<int:service-activator input-channel="executeUnzippedFilesHeader"
                       output-channel="cleanUpUnzippedFilesHeader"
                       ref="zipHeaderExecutionTask"
                       method="executeZip"/>

<bean id="zipHeaderExecutionTask" class="de.hybris.platform.cloud.hotfolder.dataimport.batch.zip.task.ZipHeaderExecutionTask">
    <property name="unzippedFolderImportServices" ref="unzippedFolderImportServices"/>
</bean>
```

```
<alias name="defaultUnzippedFolderImportServices" alias="unzippedFolderImportServices"/>
<util:map id="defaultUnzippedFolderImportServices">
    <entry value-ref="coreDataImportUnzippedFolderImportService">
        <key>
            <bean class="java.util.regex.Pattern" factory-method="compile">
                <constructor-arg value="^coredata.*"/>
            </bean>
        </key>
    </entry>
    <entry value-ref="sampleDataImportUnzippedFolderImportService">
        <key>
            <bean class="java.util.regex.Pattern" factory-method="compile">
                <constructor-arg value="^sampledata.*"/>
            </bean>
        </key>
    </entry>
    <entry value-ref="rawImpexUnzippedFolderImportService">
        <key>
            <bean class="java.util.regex.Pattern" factory-method="compile">
                <constructor-arg value="(?!coredata|sampledata).*"/>
            </bean>
        </key>
    </entry>
</util:map>
```

Adding Your Own Mappings

```
<!-- 1) add you mapping so that the file is passed onto your channel when it is downloaded -->
<bean class="org.springframework.beans.factory.config.MethodInvokingFactoryBean">
    <property name="targetObject" ref="hotfolderInboundFileChannelMappings"/>
    <property name="targetMethod" value="put"/>
    <property name="arguments">
        <list>
            <bean class="java.util.regex.Pattern" factory-method="compile">
                <constructor-arg value="^yourHotFolderPattern.*" />
            </bean>
            <ref bean="yourHotFolderChannel"/>
        </list>
    </property>
</bean>

<int:channel id="yourHotFolderChannel"/>

<!-- 2) move the file to processing and setup header -->
<file:outbound-gateway id="yourHotFolderOutboundChannel"
                       request-channel="yourHotFolderChannel"
                       reply-channel="yourHotFolderBatchFilesProc"
                       directory="#{baseLocalDirectory}/yourHotFolder/processing"
                       delete-source-files="true" />


<!-- normal hot folder spring setup below -->
<int:service-activator input-channel="yourHotFolderBatchFilesProc">
    .........
    .........
```

What are the Archive and Error folders?

```
<alias name="defaultAzureBlobSynchronizingMessageSource" alias="azureBlobSynchronizingMessageSource"/>
<bean id="defaultAzureBlobSynchronizingMessageSource"
      class="de.hybris.platform.cloud.azure.hotfolder.remote.inbound.AzureBlobSynchronizingMessageSource">
    <constructor-arg name="synchronizer" ref="azureBlobInboundSynchronizer"/>
    <property name="autoCreateLocalDirectory" value="true"/>
    <property name="localDirectory"
              value="#{azureHotfolderLocalDirectoryBase}/#{azureHotfolderRemotePath}"/>
    <property name="maxFetchSize" value="1"/>
</bean>

<int:inbound-channel-adapter id="azureInboundChannelAdapter"
                             auto-startup="false"
                             role="${cloud.hotfolder.storage.services.role}"
                             phase="50"
                             ref="azureBlobSynchronizingMessageSource"
                             channel="hotfolderInboundFileHeaderEnricherChannel">
    <int:poller fixed-rate="15000"
                task-executor="azureChannelAdapterTaskExecutor"
                max-messages-per-poll="1">
        <int:transactional synchronization-factory="defaultAzureSynchronizationFactory"
                           transaction-manager="azurePsuedoTxManager"/>
    </int:poller>
</int:inbound-channel-adapter>

<bean id="azureChannelAdapterTaskExecutor"
      class="de.hybris.platform.cloud.commons.scheduling.HybrisAwareThreadPoolTaskExecutor">
    <property name="waitForTasksToCompleteOnShutdown" value="true"/>
    <property name="threadNamePrefix" value="AzureIntegrationTaskExecutorThread-${tenantId}-"/>
    <property name="threadGroupName" value="AzureIntegrationTaskExecutor-${tenantId}"/>
    <property name="corePoolSize" value="1"/>
    <property name="maxPoolSize" value="1"/>
    <property name="queueCapacity" value="-1"/>
    <property name="keepAliveSeconds" value="60"/>
    <property name="rejectedExecutionHandler">
        <bean class="java.util.concurrent.ThreadPoolExecutor$CallerRunsPolicy"/>
    </property>
    <property name="role" value="integration"/>
    <property name="autoStartup" value="false"/>
    <property name="phase" value="10"/>
    <property name="awaitTerminationSeconds" value="60"/>
</bean>

<int:transaction-synchronization-factory id="defaultAzureSynchronizationFactory">
    <int:after-commit channel="azureArchiveOutboundChannelAdapter"/>
    <int:after-rollback channel="azureErrorOutboundChannelAdapter"/>
</int:transaction-synchronization-factory>
```

Controlling the Services – Once-only Processing in a Clustered Environment

- **Very techy slide! Is NOT important to grasp all of this but useful if you want to make huge changes to the code**
- Hot Folders requires that **only one node acts as a "Leader",** and only it can pull remote files and process them - otherwise files could be run multiple times and/or out of sequence. It also requires that in the event of a Leader server going down, for whatever reason, **another is elected** in its place. To enable this, the functionality described in Platform Node Roles is utilized and a **RoleAwareLockRegistryLeaderInitiator** Spring bean has been defined with the Role **yHotfolderCandidate**.
- **NOTE**: **Any mechanism to synchronize remote files, e.g. Inbound Channel Adapter, should have the Role 'yHotfolderServices'.**
- As previously documented, the ***cluster.node.groups*** in the **builder manifest** should be '**integration,yHotfolderCandidate'** :
- **integration** controls the start-up of the Spring Integration thread pools
- **yHotFolderCandidate** will start the leadership contention. If the node is elected, it will then start **SmartLifecycles** with a role of '**yHotfolderServices**'

<HYBRIS_BIN_DIR>/ext-cloud/cloudhotfolder/resource/cloudhotfolder-spring.xml

```
<!-- define our hybris db lock repository with the role 'integration' to control concurrent access to hotfolder files -->
<bean id="cloudHotfoldersLeaderLockRepository"
      class="de.hybris.platform.cloud.commons.spring.integration.support.locks.database.HybrisLockRepository">
    <property name="region" value="${cloud.hotfolder.locking.dblock.region}"/>
    <property name="ttl" value="${cloud.hotfolder.locking.dblock.ttl}"/>
    <property name="modelService" ref="modelService"/>
    <property name="applicationResourceLockDao" ref="applicationResourceLockDao"/>
    <property name="role" value="integration"/>
    <property name="autoStartup" value="false"/>
    <property name="phase" value="20"/>
</bean>


<!-- define an instance of springs JdbcLockRegistry and provide it with our HybrisLockRepository -->
<bean id="cloudHotfoldersLeaderLockRegistry"=
      class="org.springframework.integration.jdbc.lock.JdbcLockRegistry">
    <constructor-arg name="client" ref="cloudHotfoldersLeaderLockRepository"/>
</bean>


<-- define our role aware lock registry leader elector and give it our lock registry, give it the Role 'yHotfolderCandidate'
There should be one of these per Role that requires to elect a single Leader -->
<alias name="defaultHotfolderLeaderInitiator" alias="hotfolderLeaderInitiator"/>
<bean id="defaultHotfolderLeaderInitiator"
      class="de.hybris.platform.cloud.commons.spring.integration.support.leader.RoleAwareLockRegistryLeaderInitiator">
    <constructor-arg name="locks" ref="cloudHotfoldersLeaderLockRegistry"/>
    <constructor-arg name="childRole" value="${cloud.hotfolder.storage.services.role}"/>
    <property name="autoStartup" value="false"/>
    <property name="role" value="yHotfolderCandidate"/>
    ...
    ...
</bean>
```

Monitoring – Use of AOP

- We have utilized the Spring Aspect-Oriented-Programming framework within Cloud Hot Folders.
- This adds additional monitoring and metrics, over and above that provided by the standard implementation.
- Spring integration and AOP interceptors are used to add logging around the default hot folder tasks and converters.

How are aspects configured?

They are spring beans which perform logic in or around the method they are declared to intercept. You will find most of our monitoring aspects configured in ***<HYBRIS_BIN_DIR>*/ext-cloud/cloudhotfolder/resources/integration/hotfolder-aop-spring.xml**

```
<aop:config>
    <aop:aspect ref="fileUnzippedAspectBean">
        <aop:around method="aroundUnzipped"
                    pointcut="bean(unZipTransformer) and execution(* doTransform(..)) and args(message)" />
    </aop:aspect>
    <aop:aspect ref="batchHeaderAspectBean" id="zipBatchHeaderAspect">
        <aop:around method="aroundMessageSetup"
                    pointcut="execution(* de.hybris.platform.cloud.hotfolder.dataimport.batch.zip.task.ZipHeaderSetupTask.execute(..)) and arg
        ...
        ...
    </aop:aspect>
    <aop:aspect ref="batchHeaderAspectBean" id="fileBatchHeaderAspect">
        <aop:around method="aroundFileSetup"
                    pointcut="execution(* de.hybris.platform.acceleratorservices.dataimport.batch.task.HeaderSetupTask.execute(..)) and args(
        ...
        ...
    </aop:aspect>
</aop:config>
```

Additional Monitoring Steps

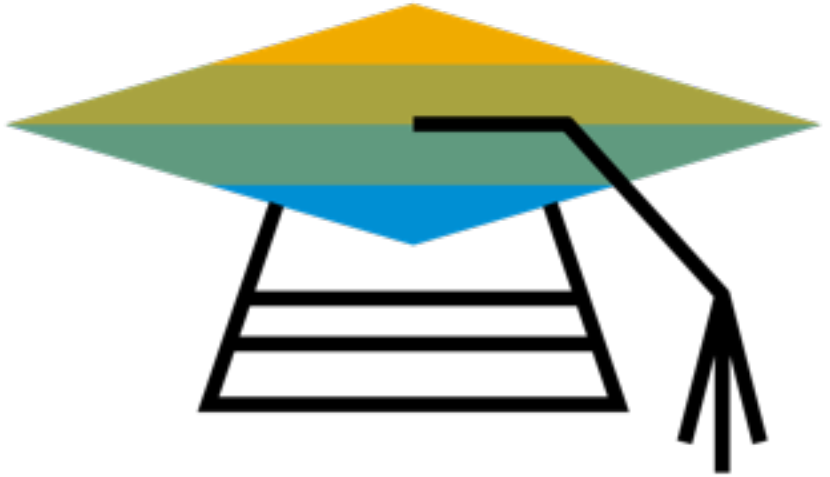
Step	Notes
Downloaded	The Azure Blob integration currently provides information on how long the file took to download along with the size of the file.
File Routed	This step is not recorded on files successfully routed, but only as a warning for files NOT routed.
File Unzipped	This step is used to record how long it took to unzip the file ready to be processed
Header Setup	This step is used to record how long it took to setup the BatchHeader/ZipBatchHeader
Header Initialised	This step is used to record how long it took to initialise the BatchHeader/ZipBatchHeader
Header Transformed	This step is used to record how long it took to transform any raw CSVs to Impex in the BatchHeader/ZipBatchHeader
Header Executed	This step is used to record how long it took to execute all the Impex files in the BatchHeader/ZipBatchHeader It also records the time taken and result of each Impex run as a separate Action
Header Cleanup	This step is used to record how long it took to clean up any converted files and also move the original files into the archive directory

Local Set-up

- IDE IntelliJ (I just prefer it) – with Commerce plugin
- CHF code with SAP Commerce v1811
- Azurite – Docker Azure emulator
- MSSQL – Docker SQL DB (as our persistence layer)
- Kitematic – GUI for Docker container management (just easier)
- Azure Storage Explorer

- Setting these parts up is covered in the SAP Wiki documentation and also in another video I recorded, so I won’t go through this here.

Thank you.



THE BEST RUN SAP