Configuring Oozie for Managing Hadoop Jobs

Date published:
Date modified:

https://docs.cloudera.com/

Legal Notice

Contents

Overview of Oozie

Apache Oozie Workflow Scheduler for Hadoop is a workflow and coordination service for managing Apache Hadoop jobs:

- Oozie Workflow jobs are Directed Acyclic Graphs (DAGs) of actions; actions are Hadoop jobs (such as MapReduce, Streaming, Hive, Sqoop and so on) or non-Hadoop actions such as Java, shell, Git, and SSH.
- Oozie Coordinator jobs trigger recurrent Workflow jobs based on time (frequency) and data availability.
- Oozie Bundle jobs are sets of Coordinator jobs managed as a single job.

Oozie is an extensible, scalable and data-aware service that you can use to orchestrate dependencies among jobs running on Hadoop.

Related Information

Apache Oozie Workflow Scheduler for Hadoop

Adding the Oozie service using Cloudera Manager

The Oozie service can be automatically installed and started during your installation of CDP with Cloudera Manager. If required, you can install Oozie manually with the Add Service wizard in Cloudera Manager. The wizard configures and starts Oozie and its dependent services.

Note: If your instance of Cloudera Manager uses an external database, you must also configure Oozie with an external database.

Considerations for Oozie to work with AWS

If you want to access Amazon Web Services (AWS) S3 data through Oozie, then ensure that the required AWS credentials are configured in core-site.xml.

Adding file system credentials to an Oozie workflow

Oozie has to use its Kerberos credentials to obtain delegation tokens on behalf of the user from the services. You must add additional configurations in the Oozie workflow for it to obtain this delegation token.

Credentials for token delegation

A secure cluster requires the Oozie actions to be authenticated; typically using Kerberos. However, due to the way that Oozie runs actions, Kerberos credentials are not available to them. Some actions require communication to an external service like HCatalog, HBase, and Hive or a secure file system like Amazon S3. For these situations, Oozie has to use its Kerberos credentials to obtain delegation tokens which allows Oozie to access the external service or file system.

For information about Action Authentication, see Apache Oozie documentation.

File System Credentials

To allow Oozie to access S3, ABFS, and other filesystems, it has to request a delegation token to obtain credentials. Although HUE supports only the built-in credentials (see Hue limitation)this can be done through properties. There are three options to set file system credentials for an Oozie workflow or action which are resulting the same. You can choose any of them based on your current usage and preference.

1. Set file system credentials for Oozie through hadoop properties using Hue Editor
2. Set default credentials using Cloudera Manager
3. Modify the workflow XML file manually

You can use the advanced settings to override the default configurations.

Setting file system credentials for Oozie through hadoop properties using Hue Editor

This option allows you to configure additional Oozie credentials with configuration properties. The following example shows how to set up an Oozie workflow with a shell action which uses additional file system credentials.

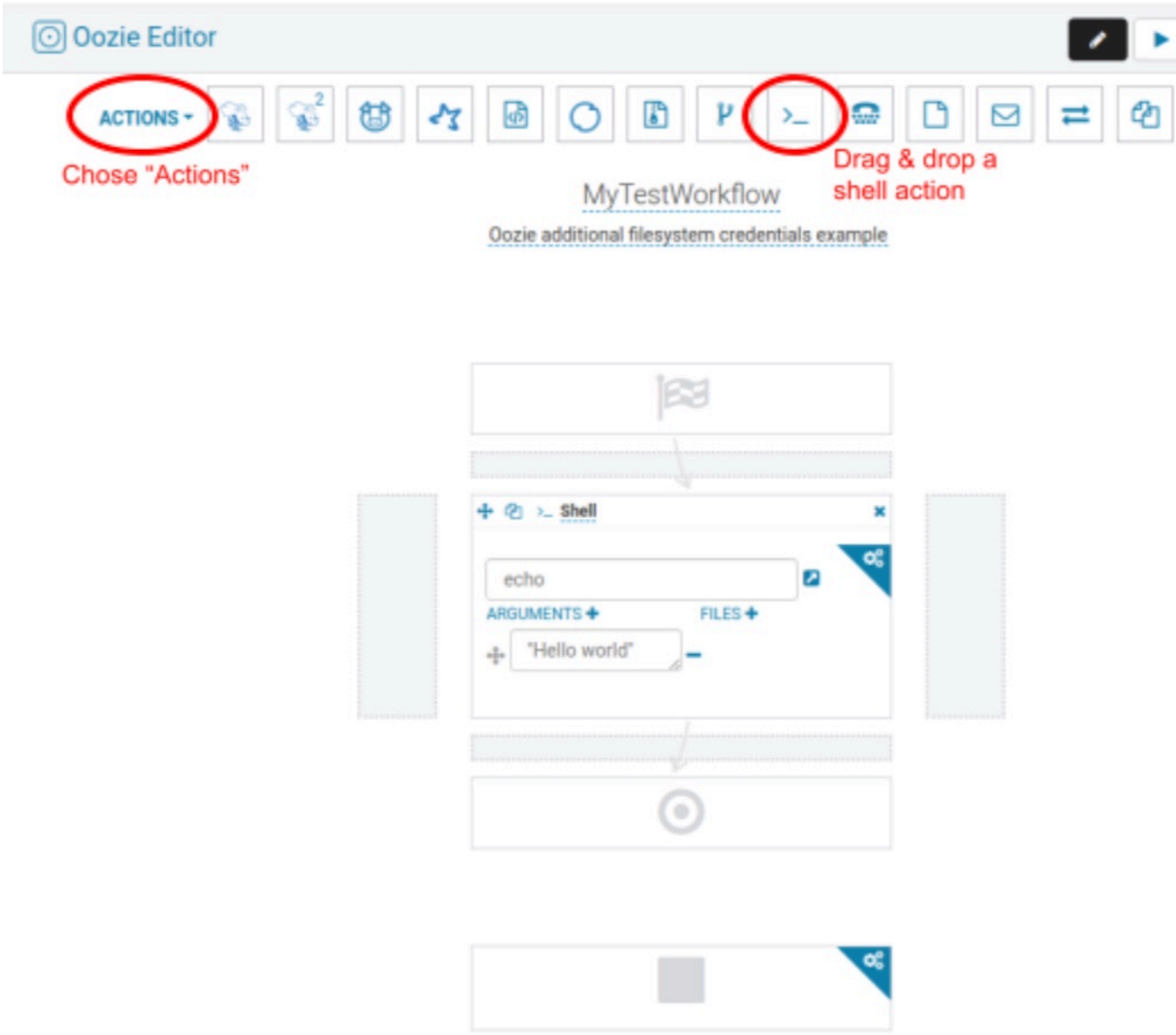
About this task

Note: Hue Limitation - Hue is not capable of adding user defined file system credentials to the Oozie workflow, only built-in credentials can be added. Hence, you must manually define file system credentials for Oozie workflow.

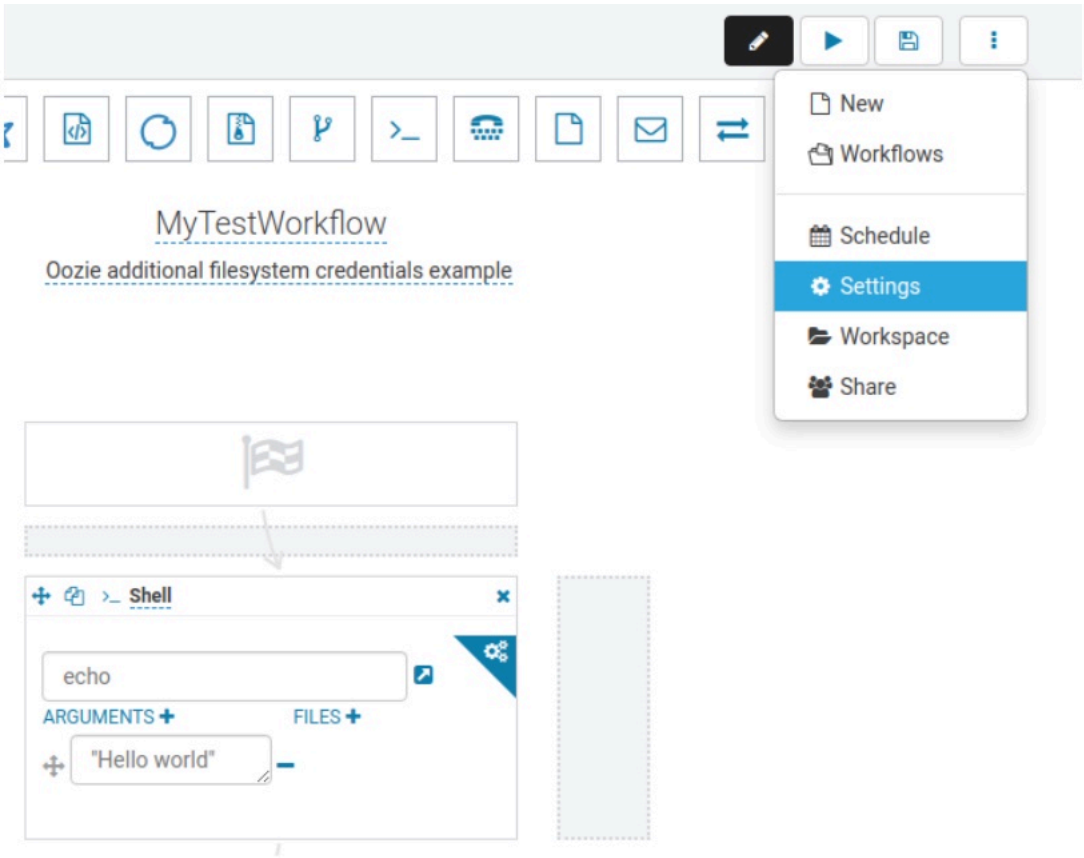
Procedure

1. In Hue web UI, click Scheduler > Workflow to create a new workflow.
2. Click Documents and select Actions in the drop-down list.

3. Select the Shell icon to the workflow to add a shell action. The shell command can be echo and its parameter can be Hello world for this example.

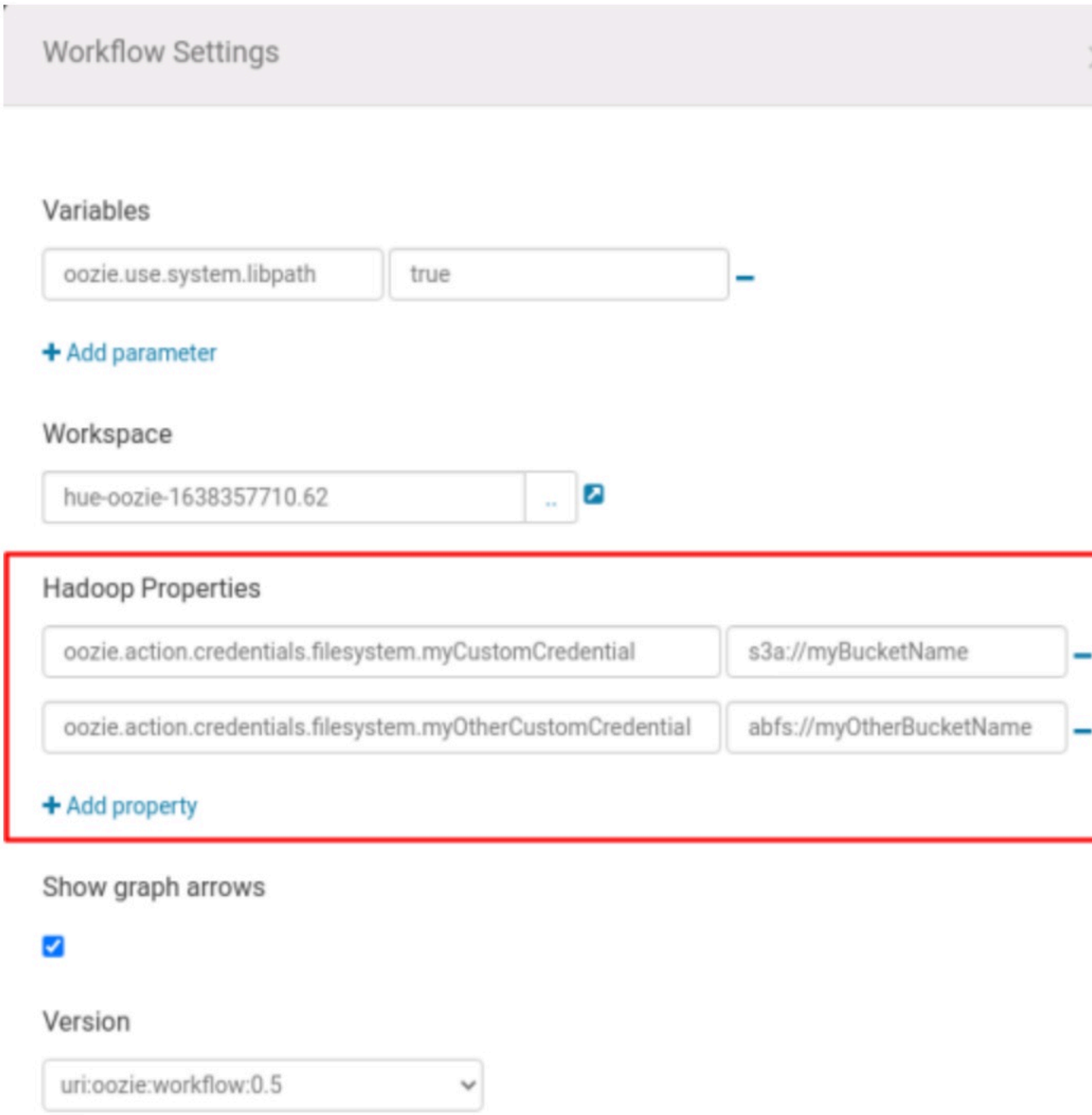


4. Select the settings under the Edit menu to add properties to the workflow. The Workflow Settings dialog box is displayed.

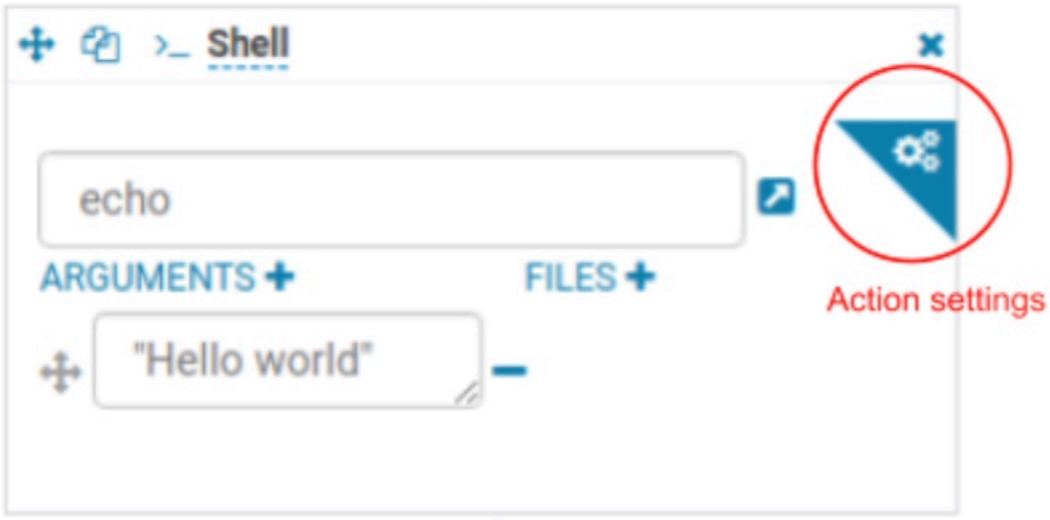


5. In the **Workflow Settings** dialog box, under the **Hadoop Properties** section, click + Add property and add the necessary credentials. The property name must be in the format similar to *oozie.action.credentials.filesystem.<myCustomCredential>* where *myCustomCredential* is the description of the credential. The value of this property must be set to a valid filesystem URI, for example, an S3 bucket URL. Click

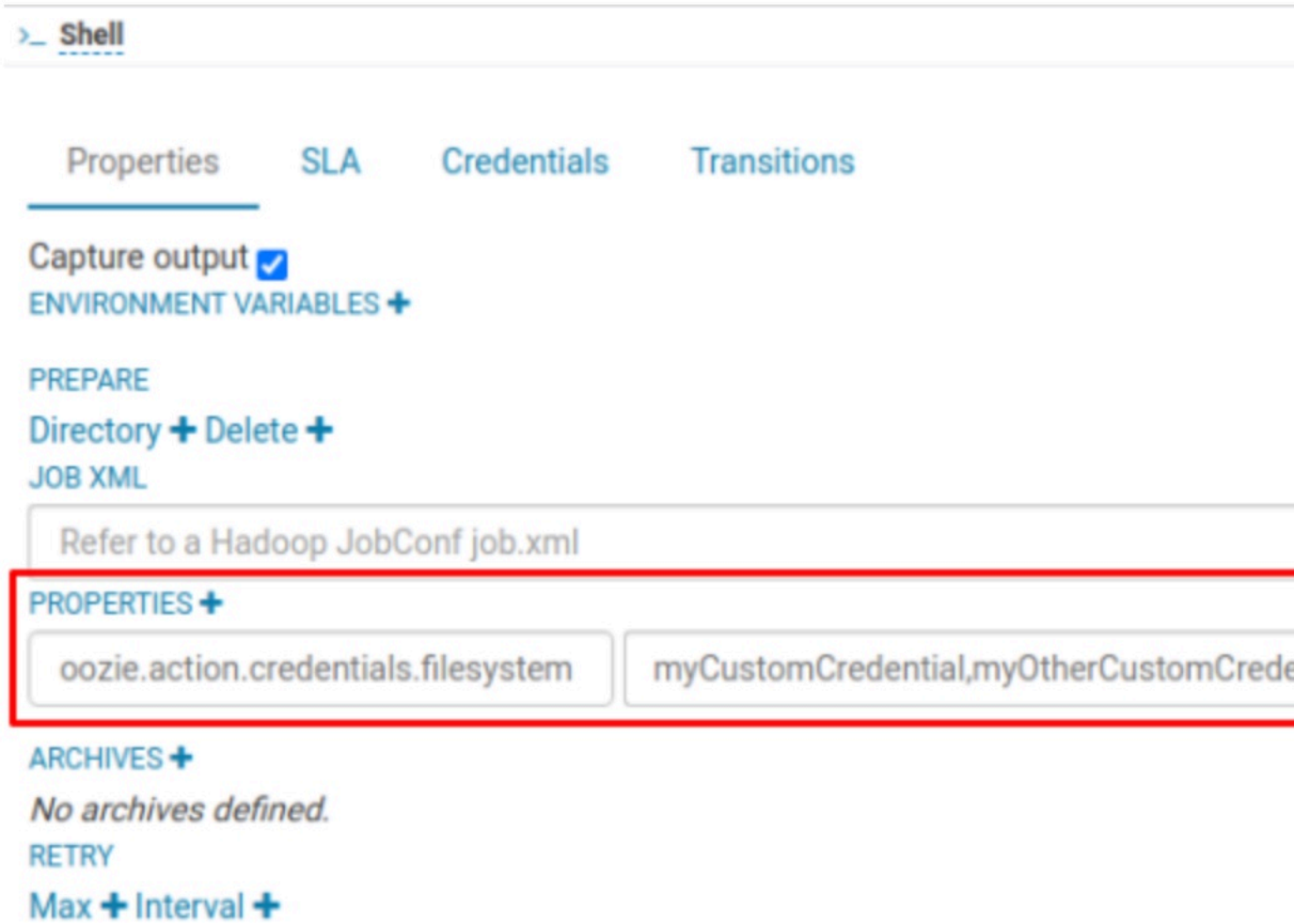
+ Add property to add additional properties if you want to access more than one s3 bucket, and so on, as shown in the below figure.



6. Click Action Settings to set the credential defined in the previous step to use it in the workflow action where it is needed.

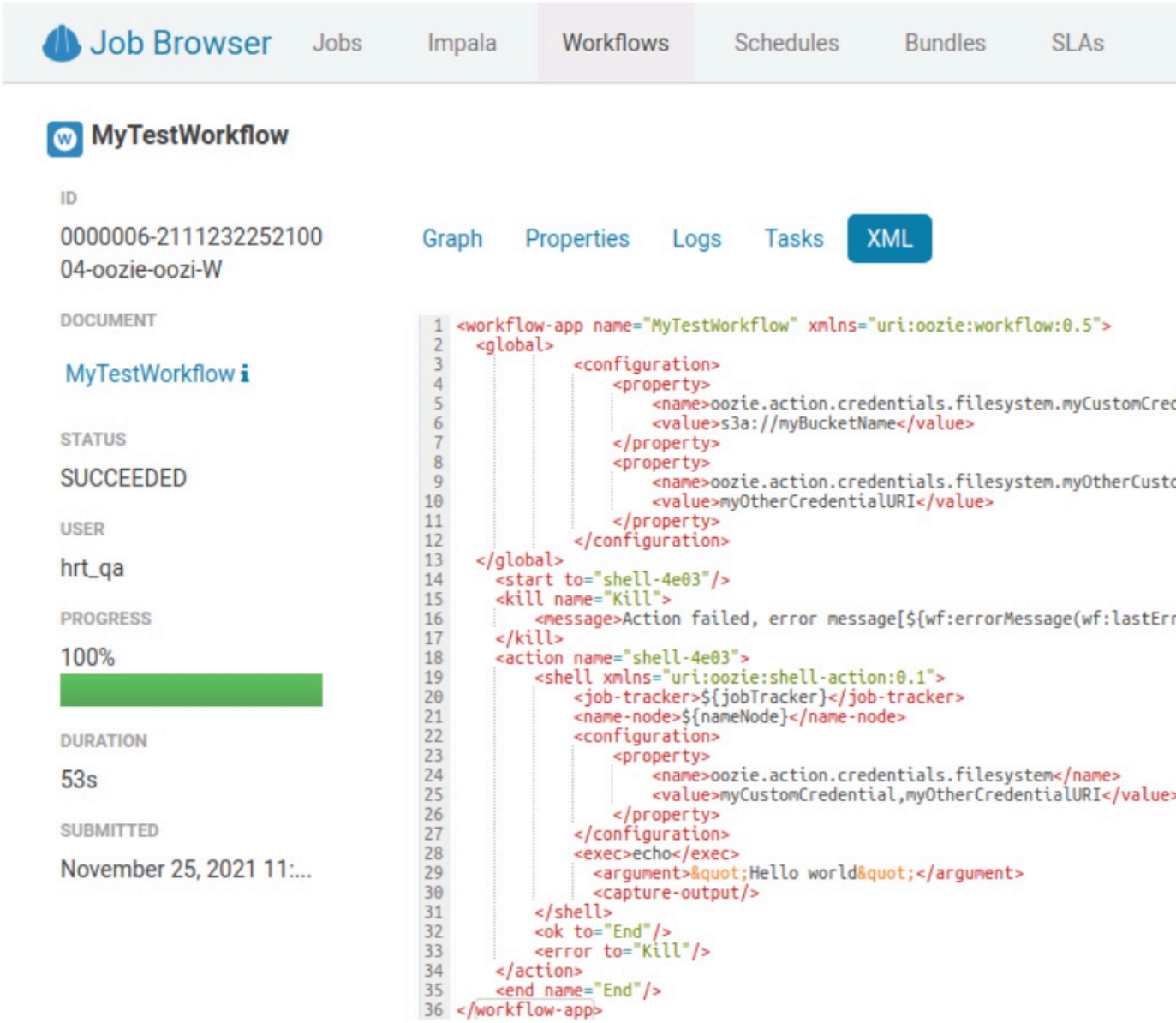


7. Add a new property to the action with the name oozie.action.credentials.filesystem and its value must be a comma-separated list of credential names defined in the hadoop properties. For example “myCustomCredential,myOtherCustomCredential”. Based on this property, Oozie requests delegation tokens for the file-systems defined in the given credentials.



Results

Now the workflow is ready to be submitted and the following is the generated workflow.xml



file.

Setting default credentials using Cloudera Manager

In scenarios where you need to set a credential for every action of every workflow, you can set the credential definitions and its usages using Cloudera Manager. You can also set these as the default configurations.

Procedure

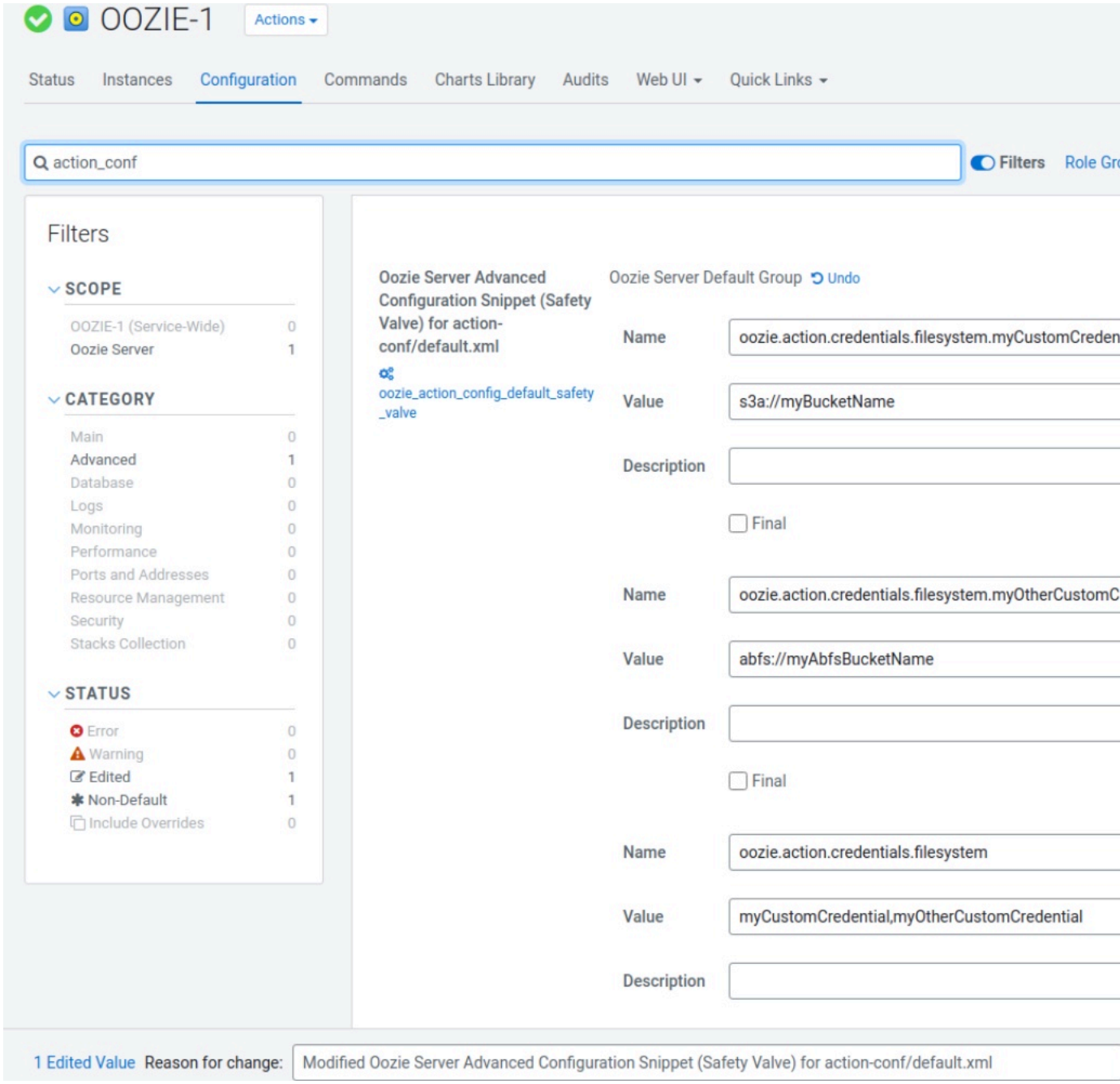
1. In Cloudera Manager, click the Oozie service.
2. Click the Configuration tab.
3. Search for action_conf and add the following in the Oozie Server Advanced Configuration Snippet (Safety Valve) for action-conf/default.xml field:

```
Definition (can be seen by all the actions started by Oozie)
```

```
Name: oozie.action.credentials.filesystem.myCustomCredential
Value: s3a://<bucket-name>

Name: oozie.action.credentials.filesystem.myotherCustomCredential
Value: abfs://<bucket-name>
```

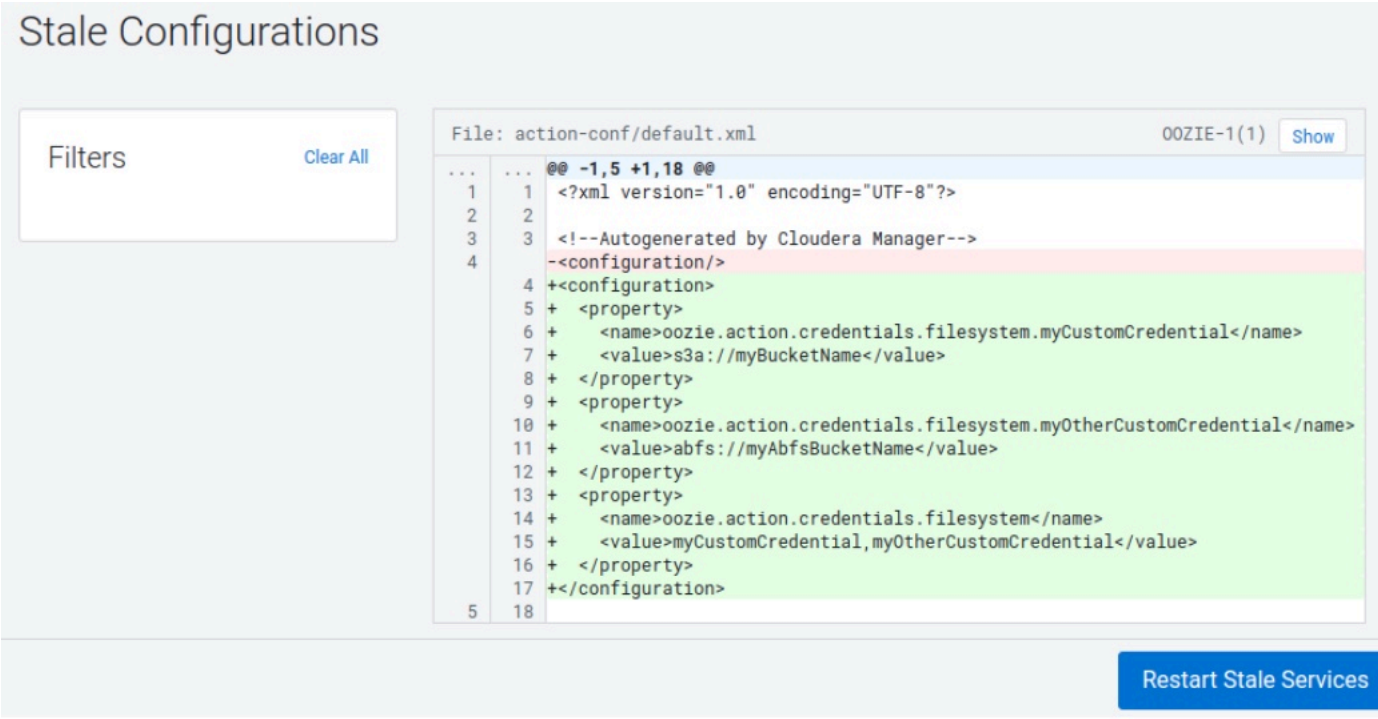
```
Usage (applies to all the actions started by Oozie)

Name: oozie.action.credentials.filesystem
Value: myCustomCredential,myotherCustomCredential
```

4. Click Save Changes on the bottom right corner.



5. Click Stale Service Restart  that is next to the Oozie service name.

6. Review the properties added to the default action configuration. All these properties will be available in every Oozie action.



7. Click Restart Stale Services to make this change happen on the Oozie instances.

Note: After this configuration, you need not add anything in HUE, Oozie obtains a delegation token for those file-system paths for every action.

Advanced settings: Overriding default configurations

You can override the default value set using Cloudera Manager, if required. For example, you have five different actions in the workflow and four of them use S3 buckets and one use ABFS. In this case, the S3 can be set as a default credential and can be overridden in only that one action to have a different value.

Procedure

1. Using Cloudera Manager, set the credentials to use s3 bucket in the Oozie Server Advanced Configuration Snippet (Safety Valve) for action-conf/default.xml field:

```
Name: oozie.action.credentials.filesystem.myCustomCredential
Value: s3a://<bucket-name>
```

2. Using the HUE editor, edit the action for which you want to use a different credential and add the following:

```
Name: oozie.action.credentials.filesystem.myCustomCredential
Value: abfs://myAbfsBucketName
```

Add the following property as described in the Step 4 and 5 of the Option 1: Setting file system credentials for Oozie through hadoop properties using Hue Editor .

Now Oozie actions will use the default S3 credential except the one which has the non-default value. It is also possible to combine the global configuration, where you define the possible file-system credentials, and in HUE you use the pre-defined global credentials.

Modifying the workflow file manually

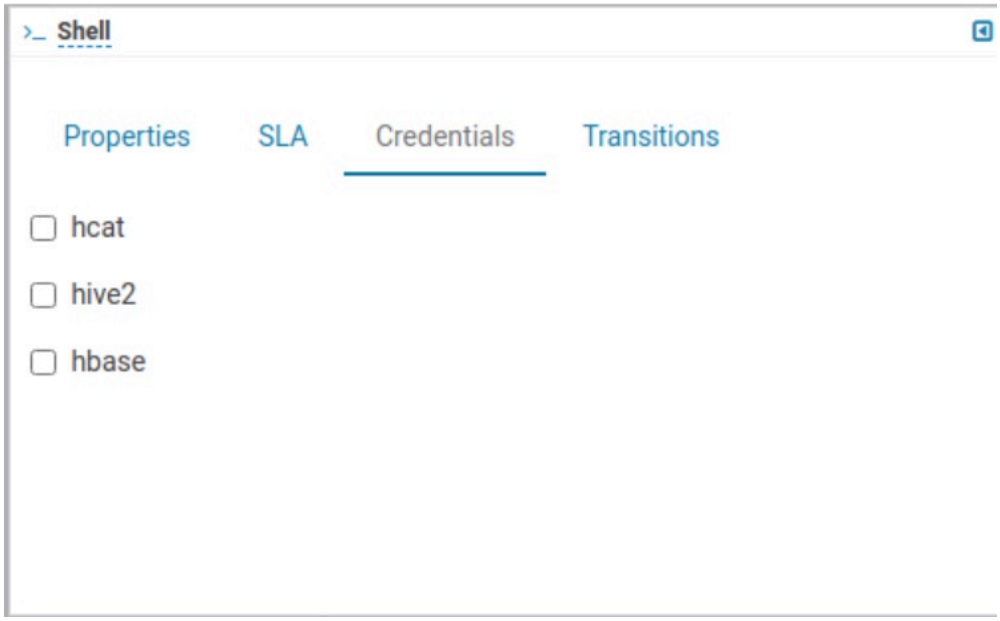
You can modify or add the "<credentials>...</credentials>" block at the beginning of the Oozie workflow.xml file. Do not remove any existing credentials from this block, add the new file system credential. Refer the newly defined credential in the action where it needs to be used.

Example

```
<workflow-app name="MyTestWorkflow" xmlns="uri:oozie:workflow:0.5">
    <credentials>
        <credential name="my-s3-creds" type="filesystem">
           <property>
               <name>filesystem.path</name>
               <value>s3a://{yourBucketName}</value>
           </property>
        </credential>
    </credentials>
    <start to="shell-action"/>
    <kill name="Kill">
        <message>Action failed</message>
    </kill>
    <action name="shell-action" cred="my-s3-creds">
        <shell xmlns="uri:oozie:shell-action:0.1">
            <job-tracker>${jobTracker}</job-tracker>
            <name-node>${nameNode}</name-node>
            <exec>echo</exec>
              <argument>&quot;Hello world&quot;</argument>
              <capture-output/>
        </shell>
        <ok to="End"/>
        <error to="Kill"/>
    </action>
    <end name="End"/>
</workflow-app>
```

Hue Limitation

Hue is not capable of adding user defined file system credentials to the Oozie workflow but can add only built-in credentials as seen in the below figure.



User authorization configuration for Oozie

Learn about user authorization model for Oozie and Access Control List (ACL). Also, learn about how to define admin users for Oozie jobs and ACL.

Oozie has a basic authorization model which is as follows:

- Users have read access to all jobs
- Users have write access to their own jobs
- Users have write access to jobs based on an ACL, which is a list of users and groups
- Users have read access to admin operations
- Admin users have write access to all jobs
- Admin users have write access to admin operations

If security is disabled all users are admin users.

Oozie security is set through the following configuration property, which is false by default:

```
oozie.service.AuthorizationService.security.enabled=false
```

Note: The old ACL model where a group was provided is still supported if the following property is set in the oozie-site.xml file:

```
oozie.service.AuthorizationService.default.group.as.acl=true
```

Defining admin users

Admin users are determined from the list of admin groups, specified in the oozie.service.AuthorizationService.admin .groups property. Use commas to separate multiple groups. Spaces, tabs, and ENTER characters are trimmed.

If the above property for admin groups is not set, then you can define the admin users in the following manner. The list of admin users can be in the conf/adminusers.txt file. The syntax of this file is as follows:

- One user name per line
- Empty lines and lines starting with # are ignored

Admin users can also be defined in the oozie.serviceAuthorizationService.admin.users property. Use commas to separate multiple admin users. Spaces, tabs, and ENTER characters are trimmed.

In case there are admin users defined using both methods, the effective list of admin users will be the union of the admin users found in the adminusers.txt file and those specified with the oozie.serviceAuthorizationService.admin.users property.

Defining access control lists

ACLs are defined in the following ways:

- workflow job submission over CLI

  Configuration property group.name of job.properties.
- workflow job submission over HTTP

  Configuration property group.name of the XML submitted over HTTP.
- workflow job re-run

  Configuration property oozie.job.acl (preferred) or configuration property group.name of job.properties.
- coordinator job submission over CLI

  Configuration property oozie.job.acl (preferred) or configuration property group.name of job.properties.
- bundle job submission over CLI

  Configuration property oozie.job.acl (preferred) or configuration property group.name of job.properties.

For all other workflow, coordinator, or bundle actions, the ACL set in beforehand are used as basis.

Once the ACL for the job is defined, Oozie checks over HDFS whether the user trying to perform a specific action is part of the necessary group(s). For implementation details, check out org.apache.hadoop.security.Groups#getGroups(String user).

Note that it is enough that the submitting user be part of at least one group of the ACL. Note also that the ACL can contain user names as well. If there is an ACL defined and the submitting user is not part of any group or user name present in the ACL, an AuthorizationException is thrown.

Example: A typical ACL setup

Detail of job.properties on workflow job submission:

```
user.name=joe
group.name=marketing,admin,qa,root
```

HDFS group membership of HDFS user joe is qa. That is, the check to org.apache.hadoop.security.Groups#getGroups("joe") returns qa. Hence, ACL check passes inside AuthorizationService, because the user.name provided belongs to at least one of the ACL list elements provided as group.name.

Redeploying the Oozie ShareLib

Some Oozie actions – specifically DistCp, Streaming, Sqoop, and Hive – require external JAR files in order to run. Instead of having to keep these JAR files in each workflow's lib folder, or forcing you to manually manage them using the oozie.libpath property on every workflow using one of these actions, Oozie provides the ShareLib.

The ShareLib behaves very similarly to oozie.libpath, except that it is specific to the aforementioned actions and their required JARs.

Redeploying the Oozie sharelib using Cloudera Manager

When you switch between MapReduce and YARN computation frameworks, you must redeploy the Oozie ShareLib.

About this task

Procedure

1. Go to the Oozie service.
2. Select Actions Install Oozie ShareLib .

Oozie configurations with CDP services

You can configure Oozie to work with different CDP services.

Some of the different services for which you can configure Oozie are as follows:

- Using Sqoop actions with Oozie
- Enabling MapReduce jobs controlled by Oozie to read from or write to Amazon S3 or Microsoft Azure ADLS
- Configuring Oozie to use HDFS HA

Using Sqoop actions with Oozie

There are certain recommendations that you must consider for using Sqoop actions with Oozie.

Note: Sqoop1 does not ship with third party JDBC drivers. You must download them separately and save them to the /var/lib/sqoop/ directory on the Oozie server.

Recommendations for using Sqoop actions with Oozie

- Cloudera recommends that you not use Sqoop CLI commands with an Oozie Shell Action. Such deployments are not reliable and prone to breaking during upgrades and configuration changes.
- To import data into Hive, use a combination of a Sqoop Action with a Hive2 Action.
  - A Sqoop Action to simply ingest data into HDFS.
  - A Hive2 Action that loads the data from HDFS into Hive.

Deploying and configuring Oozie Sqoop1 Action JDBC drivers

You must deploy and configure the Oozie Sqoop1 action JDBC drivers on HDFS.

Before you begin

Confirm that your Sqoop1 JDBC drivers are present in /var/lib/sqoop.

Procedure

- SSH to the Oozie server host and run the following commands to deploy and configure the drivers on HDFS.

```
cd /var/lib/sqoop
sudo -u hdfs hdfs dfs -mkdir /user/oozie/libext
sudo -u hdfs hdfs dfs -chown oozie:oozie /user/oozie/libext
sudo -u hdfs hdfs dfs -put /opt/cloudera/parcels/SQOOP_NETEZZA_CONNECTOR/s
qoop-nz-connector*.jar /user/oozie/libext/
sudo -u hdfs hdfs dfs -put /opt/cloudera/parcels/SQOOP_TERADATA_CONNECTOR/
lib/*.jar /user/oozie/libext/
```

```
sudo -u hdfs hdfs dfs -put /opt/cloudera/parcels/SQOOP_TERADATA_CONNECTOR/
sqoop-connector-teradata*.jar /user/oozie/libext/
sudo -u hdfs hdfs dfs -put /var/lib/sqoop/*.jar /user/oozie/libext/
sudo -u hdfs hdfs dfs -chown oozie:oozie /user/oozie/libext/*.jar
sudo -u hdfs hdfs dfs -chmod 755 /user/oozie/libext/*.jar
sudo -u hdfs hdfs dfs -ls /user/oozie/libext
# [sample contents of /user/oozie/libext]
-rwxr-xr-x   3 oozie oozie     959987 2016-05-29 09:58 /user/oozie/libext/
mysql-connector-java.jar
-rwxr-xr-x   3 oozie oozie     358437 2016-05-29 09:58 /user/oozie/libext/
nzjdbc3.jar
-rwxr-xr-x   3 oozie oozie    2739670 2016-05-29 09:58 /user/oozie/libext/
ojdbc6.jar
-rwxr-xr-x   3 oozie oozie    3973162 2016-05-29 09:58 /user/oozie/libext/
sqoop-connector-teradata-1.5c5.jar
-rwxr-xr-x   3 oozie oozie      41691 2016-05-29 09:58 /user/oozie/libext/
sqoop-nz-connector-1.3c5.jar
-rwxr-xr-x   3 oozie oozie       2405 2016-05-29 09:58 /user/oozie/libext/
tdgssconfig.jar
-rwxr-xr-x   3 oozie oozie     873860 2016-05-29 09:58 /user/oozie/libext/
terajdbc4.jar
```

Configuring Oozie Sqoop1 Action workflow JDBC drivers

You must confirm that the Sqoop1 JDBC drivers are present in HDFS and then configure the required variables.

Procedure

1. Confirm that the Sqoop1 JDBC drivers are present in HDFS. To do this, SSH to the Oozie Server host and run the following command:

   ```
   sudo -u hdfs hdfs dfs -ls /user/oozie/libext
   ```

2. Configure the following Oozie Sqoop1 Action workflow variables in Oozie's job.properties file as follows:

   ```
   oozie.use.system.libpath = true
   oozie.libpath = /user/oozie/libext
   ```

Configuring Oozie to enable MapReduce jobs to read or write from Amazon S3

MapReduce jobs controlled by Oozie as part of a workflow can read from and write to Amazon S3.

Before you begin

You will need your AWS credentials (the appropriate Access key ID and Secret access key obtained from Amazon Web Services for your Amazon S3 bucket). After storing these credentials in the keystore (the JCEKS file), specify the path to this keystore in the Oozie workflow configuration.

About this task

This setup is for use in the context of Oozie workflows only, and does not support running shell scripts on Amazon S3 or other types of scenarios.

Note: In the following steps, replace the *path/to/file* with the HDFS directory where the .jceks file is located, and replace *access_key_ID* and *secret_access_key* with your AWS credentials.

Procedure

1. Create the credential store (.jceks) and add your AWS access key to it as follows:

```
hadoop credential create fs.s3a.access.key -provider \
jceks://hdfs/path/to/file.jceks -value access_key_id
```

For example:

```
hadoop credential create fs.s3a.access.key -provider \
jceks://hdfs/user/root/awskeyfile.jceks -value AKIAIPVYH....
```

2. Add the AWS secret to this same keystore.

```
hadoop credential create fs.s3a.secret.key -provider \
jceks://hdfs/path/to/file.jceks -value secret_access_key
```

3. Set hadoop.security.credential.provider.path to the path of the .jceks file in Oozie's workflow.xml file in the MapReduce Action's <configuration> section so that the MapReduce framework can load the AWS credentials that give access to Amazon S3.

```
<action name="S3job">
    <map-reduce>
        <job-tracker>${jobtracker}</job-tracker>
        <name-node>${namenode}</name-node>
        <configuration>
            <property>
                <name>hadoop.security.credential.provider.path</name>
                <value>jceks://hdfs/path/to/file.jceks</value>
            </property>
            ....
            ....
</action>
```

Configuring Oozie to use HDFS HA

To configure an Oozie workflow to use HDFS HA, use the HDFS nameservice instead of the NameNode URI in the <name-node> element of the workflow.

Example:

```
<action name="mr-node">
  <map-reduce>
    <job-tracker>${jobTracker}</job-tracker>
    <name-node>hdfs://ha-nn</name-node>
```

where *ha-nn* is the value of dfs.nameservices in hdfs-site.xml.

Related Information

Additional considerations when configuring TLS/SSL for Oozie HA

Using Oozie with Ozone

Oozie supports Ozone storage along with HDFS. Learn how to store Oozie workflows in Ozone.

Apache Ozone is a highly scalable next-gen object store available on the CDP Private Cloud Base cluster which enables you to optimize storage for big data workloads.

You can use Ozone storage by performing the following steps:

1. Upload Oozie ShareLib to Ozone.
2. Enable Oozie workflows that access Ozone storage.

Related Information

Apache Ozone

Uploading Oozie ShareLib to Ozone

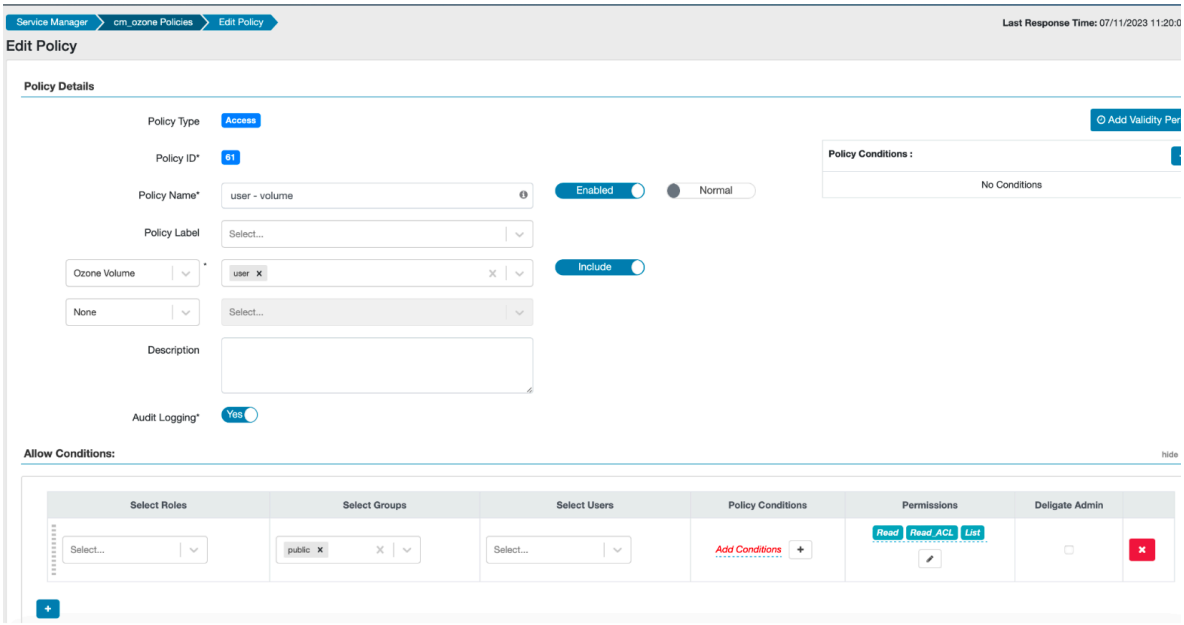
Learn how to create an Ozone volume and a bucket to store Oozie ShareLib, how to set the required permissions on the bucket, how to search and update Oozie ShareLib root directory, and install Oozie ShareLib.

Procedure

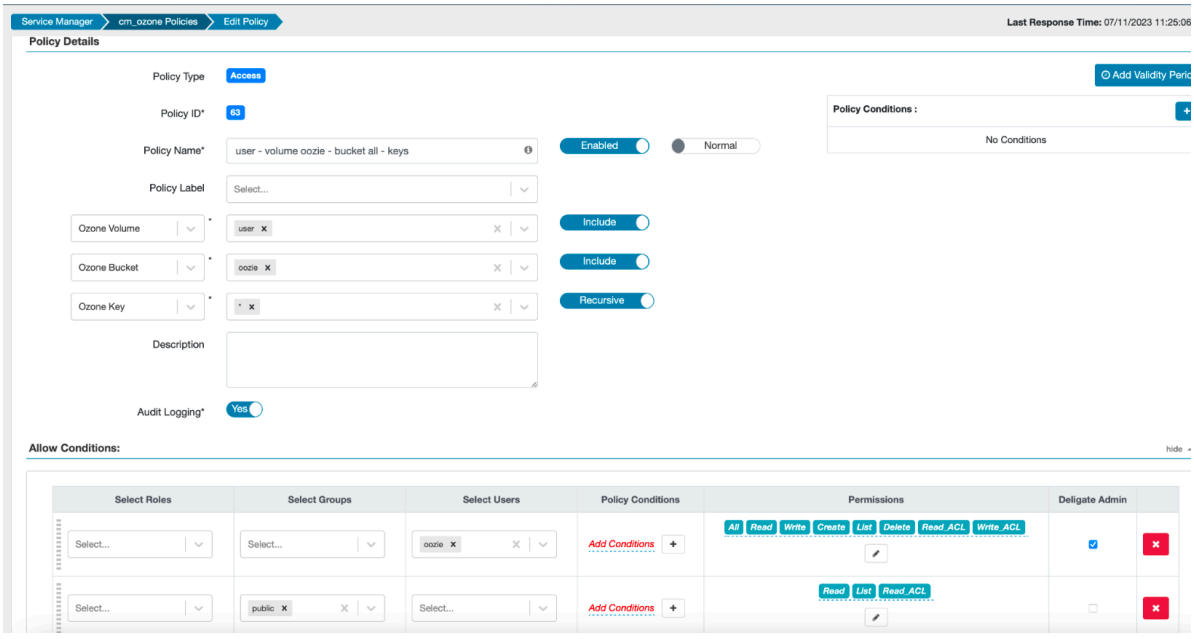
1. Create an Ozone volume and bucket to store Oozie ShareLib as an Ozone admin user:

```
ozone sh volume create /user
ozone sh bucket create /user/oozie
```

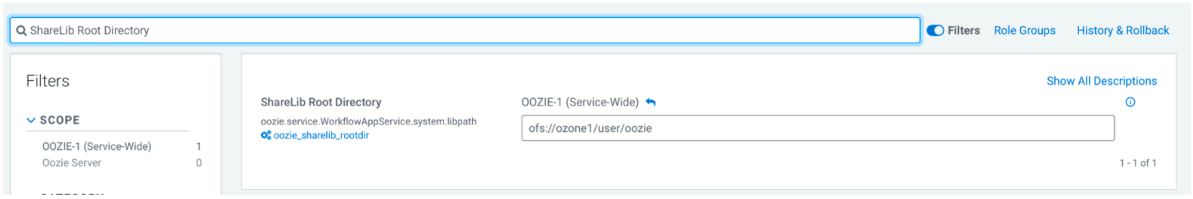
2. Ensure that the oozie user has all permissions on the bucket and everybody else has read permissions on the bucket.
   a) Go to the Ranger Admin UI.
   b) Click the Ozone repo.
   c) Click Add New policy.
   d) Create a policy user - volume, provide read permission to all users on the /user volume, and save the policy.



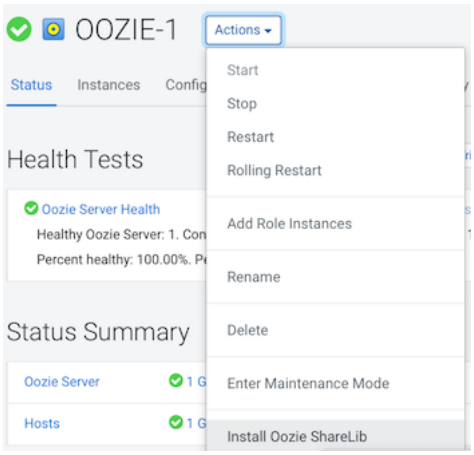
   e) Click Add New policy.
   f) Create a policy user - volume oozie - bucket all - keys, provide all permissions to the oozie user, read permissions to all on the /user/oozie bucket, and save the policy.



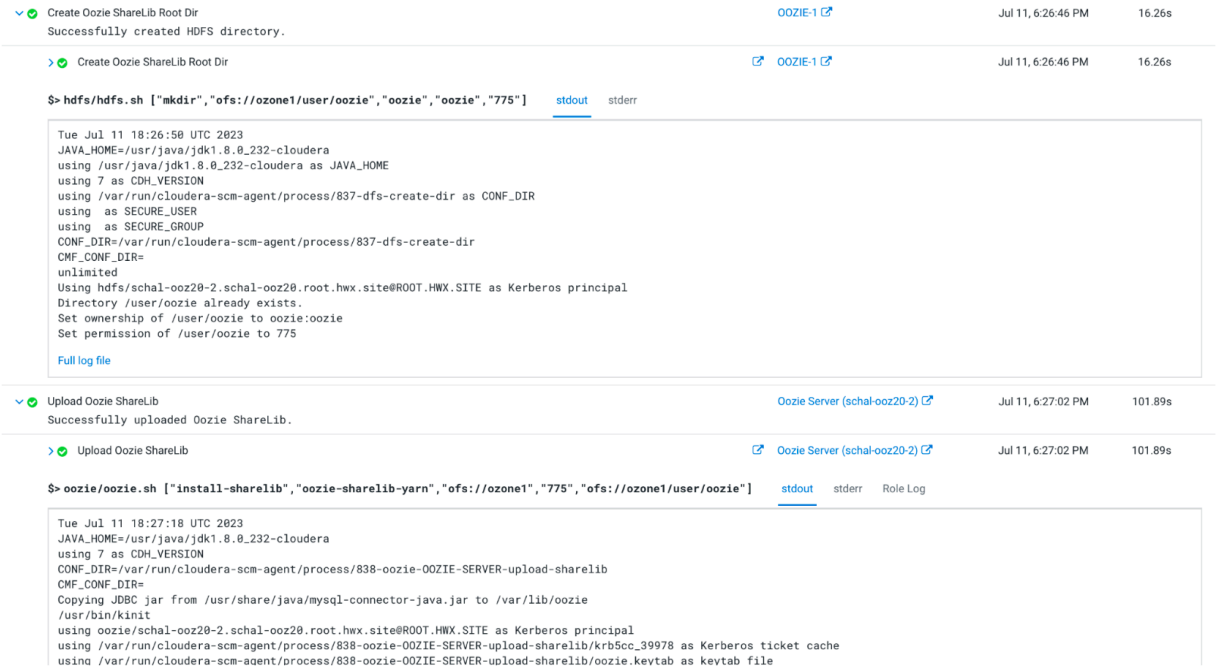
3. Update Oozie ShareLib root directory to the oozie bucket.
   a) Go to Oozie Configuration Search and Update ShareLib Root Directory .



   b) Click Save Changes.
   c) Restart Oozie.
4. After restarting Oozie, go to Oozie Actions Install Oozie ShareLib .



After the installation finishes, you can view the result as follows:



Enabling Oozie workflows that access Ozone storage

This section provides some examples of how to enable some Oozie workflows to use Ozone storage.

First you create Ozone volume and bucket:

```
kinit admin
ozone sh volume create /admin
ozone sh bucket create /admin/oozie
```

After you create volume and bucket, ensure that your user has the appropriate privileges on the buckets or keys in Ranger. For more details, see *Using Ranger with Ozone*.

Related Information

Using Ranger with Ozone

Oozie Fs action

Learn how to enable some Oozie workflows to use Ozone storage through Fs action.

Procedure

1. Create a workflow XML file, to create, move, and delete directories, and add files.

   In this example the XML file name is fs_wf.xml.

```
<workflow-app name="oozie_ozone_wf" xmlns="uri:oozie:workflow:0.5">
    <start to="create-dir"/>
    <kill name="Kill">
        <message>Action failed, error message[${wf:errorMessage(wf:lastE
rrorNode())}]</message>
    </kill>
    <action name="create-dir">
        <fs>
              <mkdir path='ofs://ozone1/admin/oozie/dir1'/>
              <mkdir path='ofs://ozone1/admin/oozie/dir2'/>
              <touchz path='ofs://ozone1/admin/oozie/dir1/file1'/>
        </fs>
        <ok to="move-file"/>
        <error to="Kill"/>
    </action>
    <action name="move-file">
        <fs>
              <move source='ofs://ozone1/admin/oozie/dir1/file1' target='o
fs://ozone1/admin/oozie/dir2/file1'/>
        </fs>
        <ok to="del-dir"/>
        <error to="Kill"/>
    </action>
    <action name="del-dir">
        <fs>
              <delete path='ofs://ozone1/admin/oozie/dir1'/>
        </fs>
        <ok to="End"/>
        <error to="Kill"/>
    </action>
    <end name="End"/>
</workflow-app>
```

2. Upload the workflow file to Ozone.

```
## Create a directory on ozone to store the workflow.xml
ozone fs -mkdir ofs://ozone1/admin/oozie/wf
ozone fs -put fs_wf.xml ofs://ozone1/admin/oozie/wf/workflow.xml
```

3. Create a properties file.

   In this example the properties file name is fs_job.properties.

```
user.name=admin
oozie.wf.application.path=ofs://ozone1/admin/oozie/wf
oozie.use.system.libpath=True
```

4. Run the Oozie workflow.

```
## Run Oozie job
oozie -Djavax.net.ssl.trustStore={trustStoreFile} -Djavax.net.ssl.trust
StorePassword={trustStorePassword} job -oozie https://{oozieHost}:{oozie
Port}/oozie -config fs_job.properties -run
```

Oozie Hive2 action

Learn how to test creating and inserting a Hive table on Ozone.

Procedure

1. Create a workflow file, to run a Hive script, and modify cluster details in the workflow file as necessary.

   In this example the name of the workflow file is hive_ozone_wf.xml.

```
<workflow-app name="ozone_hive_wf" xmlns="uri:oozie:workflow:0.5">
  <credentials>
    <credential name="hive2" type="hive2">
      <property>
        <name>hive2.jdbc.url</name>
        <value>jdbc:hive2://schal-ooz20-2.schal-ooz20.root.hwx.site:2181/
default;serviceDiscoveryMode=zooKeeper;ssl=true;sslTrustStore=/var/lib/c
loudera-scm-agent/agent-cert/cm-auto-global_truststore.jks;trustStorePas
sword=WTtrsQKjCimqLaArf5oe9TUBxvBSDODfDfZ13Tubkfh;zooKeeperNamespace=hiv
eserver2</value>
      </property>
      <property>
        <name>hive2.server.principal</name>
        <value>hive/_HOST@ROOT.HWX.SITE</value>
      </property>
    </credential>
  </credentials>
    <start to="hive2-test"/>
    <kill name="Kill">
        <message>Action failed, error message[${wf:errorMessage(wf:last
ErrorNode())}]</message>
    </kill>
    <action name="hive2-test" cred="hive2">
        <hive2 xmlns="uri:oozie:hive2-action:0.1">
            <job-tracker>${jobTracker}</job-tracker>
            <name-node>${nameNode}</name-node>
            <jdbc-url>jdbc:hive2://schal-ooz20-2.schal-ooz20.root.hwx.si
te:2181/default;principal=hive/_HOST@ROOT.HWX.SITE;serviceDiscoveryMode=
zooKeeper;ssl=true;sslTrustStore=/var/lib/cloudera-scm-agent/agent-cert/
cm-auto-global_truststore.jks;trustStorePassword=WTtrsQKjCimqLaArf5oe9TU
BxvBSDODfDfZ13Tubkfh;zooKeeperNamespace=hiveserver2</jdbc-url>
            <script>ofs://ozone1/admin/oozie/hive_wf/hive_script.sql</scr
ipt>
        </hive2>
        <ok to="End"/>
        <error to="Kill"/>
    </action>
    <end name="End"/>
</workflow-app>
```

2. Create a hive script, to create and insert data into a table on Ozone.

   In this example the hive script is hive_script.sql.

```
CREATE EXTERNAL TABLE `oozie_test`(`code` string, `description` string,
`total_emp` int, `salary` int)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE
LOCATION 'ofs://ozone1/hive/warehouse/default.db/oozie_test';
insert into table default.oozie_test values ('oh-0001','Oozie Hive Insert
 test',1000,110000);
```

3. Create a directory on Ozone to store the workflow.xml file.

   For example:

```
ozone fs -mkdir ofs://ozone1/admin/oozie/hive_wf
ozone fs -put hive_ozone_wf.xml ofs://ozone1/admin/oozie/hive_wf/workfl
ow.xml
ozone fs -put hive_script.sql ofs://ozone1/admin/oozie/hive_wf/

## Run Oozie job
oozie -Djavax.net.ssl.trustStore={trustStoreFile} -Djavax.net.ssl.trust
StorePassword={trustStorePassword} job -oozie https://{oozieHost}:{oozie
Port}/oozie -config hive_ozone_job.properties -run
```

4. Create a properties file, and modify cluster details in the properties file.

   In this example the properties file is hive_ozone_job.properties.

```
nameNode=hdfs://schal-ooz20-2.schal-ooz20.root.hwx.site:8020
jobTracker=schal-ooz20-2.schal-ooz20.root.hwx.site:8032
mapreduce.job.user.name=admin
user.name=admin
oozie.wf.application.path=ofs://ozone1/admin/oozie/hive_wf
oozie.use.system.libpath=True
```

5. Run the Oozie job.

```
oozie -Djavax.net.ssl.trustStore={trustStoreFile} -Djavax.net.ssl.trustS
torePassword={trustStorePassword} job -oozie https://{oozieHost}:{oozieP
ort}/oozie -config hive_ozone_job.properties -run
```

6. Verify that the table is created in Hive.

```
## Open beeline shell and run the following
select * from default.oozie_test where code='oh-0001';
```

Oozie Spark action

Learn how to test inserting and selecting from a table created on Ozone using Spark engine.

Procedure

1. Create a workflow file, to run a PySpark job, and modify cluster details in the workflow file as necessary.

   In this example the workflow file is spark_ozone_wf.xml.

```
<workflow-app name="spark_ozone_wf" xmlns="uri:oozie:workflow:0.5">
   <credentials>
    <credential name="hcat" type="hcat">
      <property>
        <name>hcat.metastore.uri</name>
```

```
        <value>thrift://schal-ooz20-2.schal-ooz20.root.hwx.site:9083</val
ue>
      </property>
      <property>
        <name>hcat.metastore.principal</name>
        <value>hive/_HOST@ROOT.HWX.SITE</value>
      </property>
    </credential>
  </credentials>
    <start to="spark-test"/>
    <kill name="Kill">
        <message>Action failed, error message[${wf:errorMessage(wf:lastErr
orNode())}]</message>
    </kill>
    <action name="spark-test" cred="hcat">
        <spark xmlns="uri:oozie:spark-action:0.1">
            <job-tracker>${jobTracker}</job-tracker>
            <name-node>${nameNode}</name-node>
            <master>yarn</master>
            <mode>cluster</mode>
            <name>Spark Ozone Example</name>
            <jar>spark_ozone_test.py</jar>
            <spark-opts>--num-executors 2 --executor-cores 2 --executor-m
emory 4g --driver-memory 2g </spark-opts>
        </spark>
        <ok to="End"/>
        <error to="Kill"/>
    </action>
    <end name="End"/>
</workflow-app>
```

2. Create a PySpark script to insert data into a table created on Ozone.

   In this example the PySpark script is spark_ozone_test.py.

```
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("Spark Ozone Example").getOrCreate()
spark.sql("select * from default.oozie_test where code='opi-0001'").show()
spark.sql("insert into table default.oozie_test values ('opi-0001','Oozie
 PySpark Insert test',1000,110000)")

spark.sql("select * from default.oozie_test where code='opi-0001'").show()
```

3. Create a directory on Ozone to store the workflow.xml file.

```
ozone fs -mkdir -p ofs://ozone1/admin/oozie/spark_wf/lib
ozone fs -put spark_ozone_wf.xml ofs://ozone1/admin/oozie/spark_wf/work
flow.xml
ozone fs -put spark_ozone_test.py ofs://ozone1/admin/oozie/spark_wf/lib/
```

4. Create a properties file, and modify cluster details in the properties file as necessary.

   In this example the properties file is spark_ozone_job.properties.

```
nameNode=hdfs://schal-ooz20-2.schal-ooz20.root.hwx.site:8020
jobTracker=schal-ooz20-2.schal-ooz20.root.hwx.site:8032
mapreduce.job.user.name=admin
user.name=admin
oozie.wf.application.path=ofs://ozone1/admin/oozie/spark_wf
oozie.use.system.libpath=True
```

5. Run the Oozie job.

```
oozie -Djavax.net.ssl.trustStore={trustStoreFile} -Djavax.net.ssl.trustS
torePassword={trustStorePassword} job -oozie https://{oozieHost}:{oozieP
ort}/oozie -config spark_ozone_job.properties -run
```

6. Verify that the data is inserted into the table.

```
## Open beeline shell and run the following
select * from default.oozie_test where code='opi-0001';
```

Using Hive Warehouse Connector with Oozie Spark Action

You can use Hive Warehouse Connector (HWC) with Oozie Spark action by updating job.properties file or action-level configurations.

Note: There are known issues related to using Hive Warehouse Connector with Oozie Spark Action. Read the known issues and use the workaround listed in the *Cloudera Runtime Release Notes*.

For Updating job properties file

Steps

1. Create a new ShareLib using a different name, such as hwc.
2. Place the HWC JAR onto the new ShareLib. For information about placing HWC JARs in the new ShareLib, see the **Appendix - Creating a new 'hwc' ShareLib** section below.
3. Execute a ShareLib update.
4. When executing a Spark action using the HWC include the following properties in the job.properties file:

```
oozie.action.sharelib.for.spark=spark,hwc
```

For Updating action-level configuration

You can update the action-level configurations to execute Hive commands using both HWC and non-HWC. If you have a workflow which contains an action where you would like to use HWC and another action where you do not want to use HWC, you can achieve the same by specifying the ShareLib properties at the action level.

Example

```
<spark xmlns="uri:oozie:spark-action:1.0">
            ...
            <configuration>
            <property xmlns="">
            <name>oozie.action.sharelib.for.spark</name>
            <value>spark,hwc</value>
            </property>
            </configuration>
            ...
            </spark>
```

Related Information

Hive Warehouse Connector for accessing Apache Spark data

Appendix - Creating a new 'hwc' ShareLib

The oozie admin commands have to be executed by the oozie user.

1. Kinit as oozie.

2. Check the current available ShareLibs:

```
oozie admin -shareliblist -oozie {url}
```

3. Create the folder for it on HDFS:

```
hdfs dfs -mkdir /user/oozie/share/lib/lib_{latestTimestamp}/hwc
```

4. Add the JAR files to it from the /opt/cloudera/parcels/CDH/jars directory:
   - hive-warehouse-connector-assembly-1.0.0.***VERSION NUMBER***-XXX.jar
   - hive-jdbc-3.1.3000.***VERSION NUMBER***-XXX.jar
   - hive-jdbc-handler-3.1.3000.***VERSION NUMBER***-XXX.jar
   - hive-service-3.1.3000.***VERSION NUMBER***-XXX.jar
   - spark-sql-kafka-0-10_2.11-***VERSION NUMBER***-XXX.jar

   Note: Do not add any standalone JARs (*-standalone.jar) in this directory.

5. Update the ShareLib property:

```
oozie admin -sharelibupdate -oozie {url}
```

6. List the ShareLibs again to check if hwc is present:

```
oozie admin -shareliblist -oozie {url}
```

Example for using HWC with Oozie Spark action

Understand how you can use the Hive Warehouse Connector (HWC) with Oozie Spark actions through an example that creates an application to read tables from Hive using HWC and display its contents. You can do it either by using a JAR application or by using a Python application.

Using application JAR

This example provides detailed information about the job.properties file, workflow.xml file, and application logic required for this task, and lists the necessary information required for using HWC in Oozie Spark action when you build an application JAR.

Example application logic

You can package the following Scala application logic into a JAR by using either Maven or SBT command line utility with the compatible CDP versions. You can call it the org.example package. The following application logic requires only two dependencies - spark-sql and HWC.

```
import com.hortonworks.hwc.HiveWarehouseSession
import org.apache.spark.sql.SparkSession

object ExampleRun {
  def main(args: Array[String]): Unit = {
    println("Using Hive Warehouse connector")
    //Create a Spark session
    val spark = SparkSession.builder().enableHiveSupport().getOrC
reate()
    //Create a HWC session using the Spark Session
    val hive = HiveWarehouseSession.session(spark).build()
    println(args(0)) // Print the input
    //Query the string provided in the arguments using hive.sql()
    hive.sql("SELECT * FROM " + args(0)).show
    //Close the Spark session
    spark.close()
  }
```

```
}
```

- Maven method

  Add the following dependencies when you build an application JAR by using Maven:

  ```
  <dependency>
          <groupId>org.apache.spark</groupId>
          <artifactId>spark-sql_2.11</artifactId>
          <version>${spark.version}</version>
      </dependency>
      <dependency>
          <groupId>com.hortonworks.hive</groupId>
          <artifactId>hive-warehouse-connector_2.11</artifactId>
          <version>${hwc.version}</version>
      </dependency>
  ```

  Create a JAR using the following command:

  ```
  mvn clean package -Dspark.version=<CDP Spark version> -Dhwc.
  version=<CDP HWC Version>
  ```

  For example,

  ```
  mvn clean package -Dspark.version=2.4.7.7.1.7.61-1 -Dhwc.ver
  sion=1.0.0.7.1.7.61-1
  ```

- SBT method

  Add the following library dependencies when you build an application JAR by using SBT:

  ```
  val sparkVersion = sys.props.getOrElse("spark.version", "<CDP
   Spark version>")
  val hwcVersion = sys.props.getOrElse("hwc.version", "<CDP HWC
   Version>")
  libraryDependencies ++= Seq(
    "com.hortonworks.hive" % "hive-warehouse-connector_2.11" % h
  wcVersion % "provided",
    "org.apache.spark" %% "spark-sql" % sparkVersion % "provid
  ed" force()
  )
  ```

  Create a JAR file using the following command:

  ```
  sbt clean compile assembly -Dspark.version=<CDP Spark version>
   -Dhwc.version=<CDP HWC Version>
  ```

  For example,

  ```
  sbt clean compile assembly -Dspark.version=2.4.7.7.1.7.61-1 -
  Dhwc.version=1.0.0.7.1.7.61-1
  ```

Save these JAR files in HDFS in a specific location so that it can be used later in the job.properties file. The class name here is org.example.ExampleRun which you will use later while specifying the job.

Example job.properties file

```
HCAT_METASTORE_URI=thrift://myhost-1.myhost.example.site:9083
ROOT_LOGGER_LEVEL=INFO
HCAT_PRINCIPAL=hive/_HOST@EXAMPLE.COM
oozie.action.sharelib.for.spark=spark,hwc
```

```
MASTER=yarn
JDBC_PRINCIPAL=hive/_HOST@EXAMPLE.COM
JDBC_URL=jdbc:hive2://myhost-1.myhost.example.site:10001/default;
transportMode=http;httpPath=cliservice;ssl=true;sslTrustStore=/v
ar/lib/cloudera-scm-agent/agent-cert/cm-auto-global_truststore.j
ks;trustStorePassword=update_this_password
oozie.wf.application.path=hdfs:///tmp/workdir
HIVE_TABLE_NAME=sampleTable
JDBC_MODE=JDBC_CLUSTER
APP_NAME=MyApp
MODE=cluster
JAR=hdfs:///tmp/workdir/hwc-examples-1.0.jar
CLASSNAME=org.example.ExampleRun
OOZIE_LAUNCHER_OPTS=”-verbose:class”
SPARK_OPTS=--conf spark.driver.extraJavaOptions='-verbose:class'
--conf spark.executor.extraJavaOptions='-verbose:class'
```

All the values are example values and are indicative of what you need to write in the file.

HCAT_METASTORE_URI represents the Hive metastore URI and HCAT_PRINCIPAL is the configuration required for Kerberos authentication for the Hive metastore. oozie.action.shareli b.for.spark=spark,hwc must be set as it is. MASTER specifies running Spark in the YARN mode. JDBC_PRINCIPAL is required for Kerberos authentication for HiveServer2. JDBC_URL is required to create a connection to Hive Server 2.

If you run into any classpath issues while executing the Oozie job, then you can check the details by using OOZIE_LAUNCHER_OPTS and SPARK_OPTS. These configurations show you what classes are loaded from which JAR files in the Spark job by checking the YARN logs of the Spark job.

Note: The sharelib folder contains an additional folder by the name hwc and this folder must not contain a *-standalone.jar (standalone JAR files) in it.

Example workflow.xml file

```
<?xml version="1.0" encoding="utf-8"?>
<workflow-app name="spark-hwc-hive-wf" xmlns="uri:oozie:workflow:
1.0">
    <credentials>
        <credential name="hcatauth" type="hcat">
            <property>
                <name>hcat.metastore.uri</name>
                <value>${HCAT_METASTORE_URI}</value>
            </property>
            <property>
                <name>hcat.metastore.principal</name>
                <value>${HCAT_PRINCIPAL}</value>
            </property>
        </credential>
        <credential name="hs2-creds" type="hive2">
            <property>
                <name>hive2.server.principal</name>
                <value>${JDBC_PRINCIPAL}</value>
            </property>
            <property>
                <name>hive2.jdbc.url</name>
                <value>${JDBC_URL}</value>
            </property>
        </credential>
    </credentials>

    <start to="SPARK_HWC_JDBC_READ"/>
    <action name="SPARK_HWC_JDBC_READ" cred="hs2-creds,hcatauth">
```

```
        <spark xmlns="uri:oozie:spark-action:1.0">
            <configuration>
                <property>
                    <name>mapreduce.job.hdfs-servers</name>
                    <value>${firstNotNull(wf:conf('HDFS_SERVER
S'),' ')}</value>
                </property>
                <property>
                    <name>oozie.launcher.mapreduce.map.java.opts<
/name>
      <value>${firstNotNull(wf:conf('OOZIE_LAUNCHER_OPTS'),' ')}<
/value>
                </property>
                <property>
                    <name>oozie.action.rootlogger.log.level</na
me>
                    <value>${firstNotNull(wf:conf('ROOT_LOGGER_L
EVEL'),'INFO')}</value>
                </property>
            </configuration>
            <master>${MASTER}</master>
            <mode>${MODE}</mode>
     <name>${APP_NAME}</name>
     <class>${CLASSNAME}</class>
     <jar>${JAR}</jar>
            <spark-opts>--conf spark.sql.hive.hiveserver2.jdbc
.url=${JDBC_URL}  --conf spark.sql.extensions="com.hortonworks.s
park.sql.rule.Extensions" --conf spark.datasource.hive.warehouse
.read.mode=${JDBC_MODE} --conf spark.sql.hive.hiveserver2.jdbc.u
rl.principal=${JDBC_PRINCIPAL} ${firstNotNull(wf:conf('SPARK_OPT
S'),' ')}</spark-opts>
            <arg>${HIVE_TABLE_NAME}</arg>
        </spark>
        <ok to="end"/>
        <error to="fail"/>
    </action>

    <kill name="fail">
        <message>Workflow failed, error message[${wf:errorMess
age(wf:lastErrorNode())}</message>
    </kill>
    <end name="end"/>
</workflow-app>
```

Save this workflow.xml file in the directory where you have defined oozie.wf.application.path. The different properties seen in "${}" are properties that are written either in the job.properties file or can be passed in the command line. Notice that the <spark-opts> tag contains the necessary configurations that are required for HWC. The <arg> tag contains the input for the application. The <arg> tag is currently set to a Hive table name which is used by a SELECT statement written in the application code.

Note: Do not include the HWC JAR file anywhere in the workflow.xml file as part of the Spark Job. The file is present in the sharelib folder that is explicitly created for HWC.

Using Python application

This example provides detailed information about the job.properties file, workflow.xml file, and application logic required for this task, and lists the necessary information required for using HWC in Oozie Spark action when a Python application is built.

Example application logic

```
import sys
from pyspark.sql import SparkSession
from pyspark_llap import HiveWarehouseSession

spark = SparkSession.builder.enableHiveSupport().getOrCreate()
hwc = HiveWarehouseSession.session(spark).build()
tableName = sys.argv[1]
print "=======Reading hive table - " + tableName + " via HWC===
===="
# Read via HWC
hwc.sql("select * from " + tableName).show()

hwc.close()
spark.stop()
```

You are using the pyspark module and HWC specific pyspark_llap module for executing the Python program. The pyspark_llap module is derived from the HWC artifacts given in the CDP builds.

Example job.properties file

```
HCAT_METASTORE_URI=thrift://myhos
t-1.myhost.example.site:9083
ROOT_LOGGER_LEVEL=INFO
HCAT_PRINCIPAL=hive/_HOST@EXAMPLE.COM
oozie.action.sharelib.for.spark=spark,hwc
MASTER=yarn
JDBC_PRINCIPAL=hive/_HOST@EXAMPLE.COM
JDBC_URL=jdbc:hive2://myhost-1.myhost.example.site:10001/default
;transportMode=http;httpPath=cliservice;ssl=true;sslTrustStore=/
var/lib/cloudera-scm-agent/agent-cert/cm-auto-global_truststore.
jks;trustStorePassword=update_this_password
oozie.wf.application.path=hdfs:///tmp/workdir
HIVE_TABLE_NAME=sampleTable
JDBC_MODE=JDBC_CLUSTER
APP_NAME=MyApp
MODE=cluster
PY_FILE=hdfs:///tmp/workdir/testhwcread.py
PYSPARK_HWC_ZIP=/opt/cloudera/parcels/CDH/lib/hive_warehouse_con
nector/pyspark_hwc-1.0.0.7.1.7.61-1.zip
OOZIE_LAUNCHER_OPTS=”-verbose:class”
SPARK_OPTS=--conf spark.driver.extraJavaOptions='-verbose:class'
 --conf spark.executor.extraJavaOptions='-verbose:class'
```

All the values are example values and are indicative of what you need to write in the file.

HCAT_METASTORE_URI represents the hive metastore URI and HCAT_PRINCIPAL is the configuration required for Kerberos authentication for the Hive metastore. oozie.action.shareli b.for.spark=spark,hwc must be set as it is. MASTER specifies running Spark in the YARN mode. JDBC_PRINCIPAL is required for Kerberos authentication for HiveServer2. JDBC_URL is required to create a connection to Hive Server 2.

If you run into any classpath issues while executing the Oozie job, then you can check the details by using OOZIE_LAUNCHER_OPTS and SPARK_OPTS. These configurations show you what classes are loaded from which JAR files in the Spark job by checking the YARN logs of the Spark job.

Note: The sharelib folder contains an additional folder by the name hwc and this folder must not contain a *-standalone.jar (standalone JAR files) in it.

Example workflow.xml file

```
<?xml version="1.0" encoding="utf-8"?>
<workflow-app name="spark-hwc-hive-wf" xmlns="uri:oozie:workflow:
1.0">
    <credentials>
        <credential name="hcatauth" type="hcat">
            <property>
                <name>hcat.metastore.uri</name>
                <value>${HCAT_METASTORE_URI}</value>
            </property>
            <property>
                <name>hcat.metastore.principal</name>
                <value>${HCAT_PRINCIPAL}</value>
            </property>
        </credential>
        <credential name="hs2-creds" type="hive2">
            <property>
                <name>hive2.server.principal</name>
                <value>${JDBC_PRINCIPAL}</value>
            </property>
            <property>
                <name>hive2.jdbc.url</name>
                <value>${JDBC_URL}</value>
            </property>
        </credential>
    </credentials>

    <start to="SPARK_HWC_JDBC_READ"/>
    <action name="SPARK_HWC_JDBC_READ" cred="hs2-creds,hcatauth">
        <spark xmlns="uri:oozie:spark-action:1.0">
            <configuration>
                <property>
                    <name>mapreduce.job.hdfs-servers</name>
                    <value>${firstNotNull(wf:conf('HDFS_SERVER
S'),' ')}</value>
                </property>
                <property>
                     <name>oozie.launcher.mapreduce.map.java.opts<
/name>
                     <value>${firstNotNull(wf:conf('OOZIE_LAUNCHER
_OPTS'),' ')}</value>
                </property>
                <property>
                     <name>oozie.action.rootlogger.log.level</name
>
                     <value>${firstNotNull(wf:conf('ROOT_LOGGER
_LEVEL'),'INFO')}</value>
                </property>
            </configuration>
            <master>${MASTER}</master>
            <mode>${MODE}</mode>
     <name>${APP_NAME}</name>
     <jar>${PY_FILE}</jar>
            <spark-opts>--conf spark.sql.hive.hiveserver2.jdbc.ur
l=${JDBC_URL}  --conf spark.sql.extensions="com.hortonworks.spar
k.sql.rule.Extensions" --conf spark.datasource.hive.warehouse.re
ad.mode=${JDBC_MODE} --conf spark.sql.hive.hiveserver2.jdbc.url.
principal=${JDBC_PRINCIPAL} --conf spark.submit.pyFiles=${PYSPAR
K_HWC_ZIP} ${firstNotNull(wf:conf('SPARK_OPTS'),' ')}</spark-opt
s>
            <arg>${HIVE_TABLE_NAME}</arg>
        </spark>
```

```
        <ok to="end"/>
        <error to="fail"/>
    </action>

    <kill name="fail">
        <message>Workflow failed, error message[${wf:errorMess
age(wf:lastErrorNode())}</message>
    </kill>
    <end name="end"/>
</workflow-app>
```

Save this workflow.xml file in the directory where you have defined oozie.wf.application.path. The different properties seen in "${}" are properties that are written either in the job.properties file or can be passed in the command line. The <spark-opts> tag contains the necessary configurations that are required for HWC. The <arg> tag contains the input for the application that needs to be run. It is currently set to a Hive table name which is used for executing a SELECT query on the same table.

Note: Do not include the HWC JAR anywhere in the workflow.xml file as part of the Spark Job. It is already present in the sharelib folder that is explicitly created for HWC.

Oozie and client configurations

Learn about how Oozie handles the client configuration files for different actions that rely on Hadoop, Hive, and so on, clients.

In certain cases, it is necessary to have the required client configurations (site.xml files) available for action executions in YARN. Previously, you manually copied some site.xml files to either your workflow lib folder or the corresponding Oozie ShareLib. This section summarizes how Oozie utilizes the client configuration files.

Hive2 action

The hive-site.xml is not automatically propagated to YARN. The Hive2 action executes the beeline command, which uses the hive-site.xml file from /etc/hive/conf.

Oozie automatically propagates the hbase-site.xml to YARN if the conditions described in the *Conditions* section are met.

MapReduce and Shell actions

The hbase-site.xml, hive-site.xml, and sqoop-site.xml are propagated to the YARN container if the corresponding conditions mentioned in the *Conditions* section are met.

Spark and Spark3 actions

Oozie automatically propagates the sqoop-site.xml to YARN if the conditions described in the *Conditions* section are met.

Oozie automatically propagates the hive-site.xml to YARN if the Hive service dependency is enabled for Oozie in Cloudera Manager.

Oozie automatically propagates the hbase-site.xml to YARN if the HBase service dependency is enabled for Oozie in Cloudera Manager.

Sqoop action

Oozie automatically propagates the hive-site.xml to YARN if the conditions described in the *Conditions* section are met.

Oozie automatically propagates the sqoop-site.xml to YARN if the Sqoop client dependency is enabled for Oozie in Cloudera Manager.

Oozie automatically propagates the hbase-site.xml to YARN if the HBase service dependency is enabled for Oozie in Cloudera Manager.

Conditions

All the client configurations are only automatically saved to the YARN container if no existing file is already present. This allows you to manually make a client configuration file available in the Workflow's lib folder or the corresponding Oozie ShareLib.

The following applies to the client configurations mentioned in the sections above, where the *Conditions* section is referenced:

To enable Oozie to automatically copy the client configuration files, the following dependencies must be enabled in Cloudera Manager for Oozie:

- hbase-site.xml

  HBase service dependency.
- hive-site.xml

  Hive service dependency.
- sqoop-site.xml

  Sqoop client dependency.

Automatic propagation of these configurations can be disabled by setting the following settings to true:

- hbase-site.xml: oozie.action.propagate.hbase-site.xml.disabled
- hive-site.xml: oozie.action.propagate.hive-site.xml.disabled
- sqoop-site.xml: oozie.action.propagate.sqoop-site.xml.disabled

You can configure these settings globally using a safety-valve, or in the Workflow's global configuration, or in the job.properties file, or at the action-level through the action's configuration in the workflow definition. The order of precedence is as follows:

1. Action-level configuration
2. Workflow global configuration
3. job.properties configuration
4. Global safety-valve configuration

Spark 3 support in Oozie

This section describes how to re-enable the Spark actions, provides guidance on migrating your workflows from Spark actions to Spark 3 actions, highlights the differences between the two, and outlines the enhancements made to Spark actions to simplify the migration process.

Due to the discontinuation and deprecation of Spark 2, Oozie's Spark actions are deprecated, which are based on Spark 2. Consequently, Oozie's Spark actions are disabled by default, and if you attempt to execute a Spark action, an error appears. Oozie supports the new Spark 3 based Spark 3 actions.

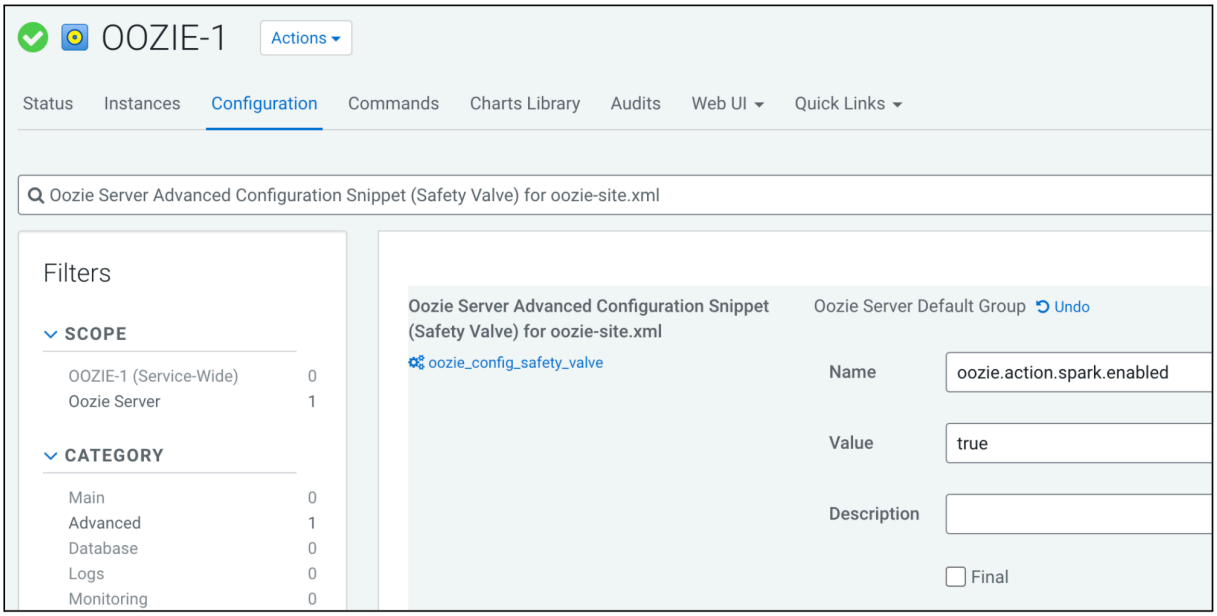
Enable Spark actions

While Spark actions are initially disabled, you have the flexibility to enable them at your discretion if you still want to utilize them.

Enabling Spark actions globally

Perform the following steps to enable Spark action globally through Cloudera Manager:

1. Navigate to Oozie's configuration page in Cloudera Manager.
2. Search for Oozie Server Advanced Configuration Snippet (Safety Valve) for oozie-site.xml.
3. Add a new property named oozie.action.spark.enabled with the value true.



4. Save the modifications.
5. Allow Cloudera Manager some time to recognize the changes.
6. Redeploy Oozie.

Note: You can also enable the above setting through the same safety-valve property before upgrading to Private Cloud Base 7.1.9 or higher. By doing so, you can ensure that your scheduled Spark 2 applications with Oozie continue to run successfully after the upgrade, avoiding any potential failures.

Enabling Spark actions per workflows

If you do not want to enable Spark actions globally, you can enable them on a per workflow basis. To do so, set the value of the oozie.action.spark.enabled property to true in the global configuration of the specific workflow. For example:

```
<workflow-app name="spark_workflow" xmlns="uri:oozie:workflow:1.0">
    <global>
        <configuration>
            <property>
                <name>oozie.action.spark.enabled</name>
                <value>true</value>
            </property>
        </configuration>
    </global>
    <start to="spark_action"/>
    <action name="spark_action">
        ...
```

By following this approach, if your workflow contains multiple Spark actions, all of them will be enabled and operational.

You can also enable Spark actions for a given workflow by specifying the oozie.action.spark.enabled property in your job.properties file.

Enabling a given Spark action only

If you want to further restrict the enablement of a Spark action, you have the option to enable it only for a specific Spark action within a workflow. This can be achieved by utilizing the oozie.action.spark.enabled property in the configuration of that particular action. For example:

```
<workflow-app name="spark_workflow" xmlns="uri:oozie:workflow:1.0">
    <start to="spark_action"/>
    <action name="spark_action">
        <spark xmlns="uri:oozie:spark-action:1.0">
            <resource-manager>${resourceManager}</resource-manager>
            <name-node>${nameNode}</name-node>
            <configuration>
                <property>
                    <name>oozie.action.spark.enabled</name>
                    <value>true</value>
                </property>
            </configuration>
            ...
```

The order of precedence for the above options is as follows:

1. If you have configured the property at the action level, it takes precedence over all other settings, and the remaining configurations are disregarded.
2. If you have configured the property in the global configuration of the workflow, the value from there is used.
3. If the setting is not available in either of the previous locations, the value configured in your job.properties file is used instead.
4. Lastly, the global safety-valve setting comes into effect.

Use Spark actions with a custom Python executable

Spark 2 supports both PySpark and JavaSpark applications. Learn how to use a custom Python executable in a given Spark action.

In case of PySpark, in Spark 2, you can designate a custom Python executable for your Spark application by utilizing the spark.pyspark.python Spark conf argument. For more details, see Spark 2.4 documentation. Consequently, if you include the spark.pyspark.python Spark conf argument in your Oozie Spark action, the Python executable you specify is used when executing the Spark action through Oozie.

To simplify the usage of a customized Python executable with Oozie's Spark action, you can use the oozie.service.SparkConfigurationService.spark.pyspark.python property. This property functions similarly to Spark's spark.pyspark.python conf argument, allowing you to specify a custom Python executable. Oozie then passes this executable to the underlying Spark application executed through Oozie.

You can specify the oozie.service.SparkConfigurationService.spark.pyspark.python property in different ways.

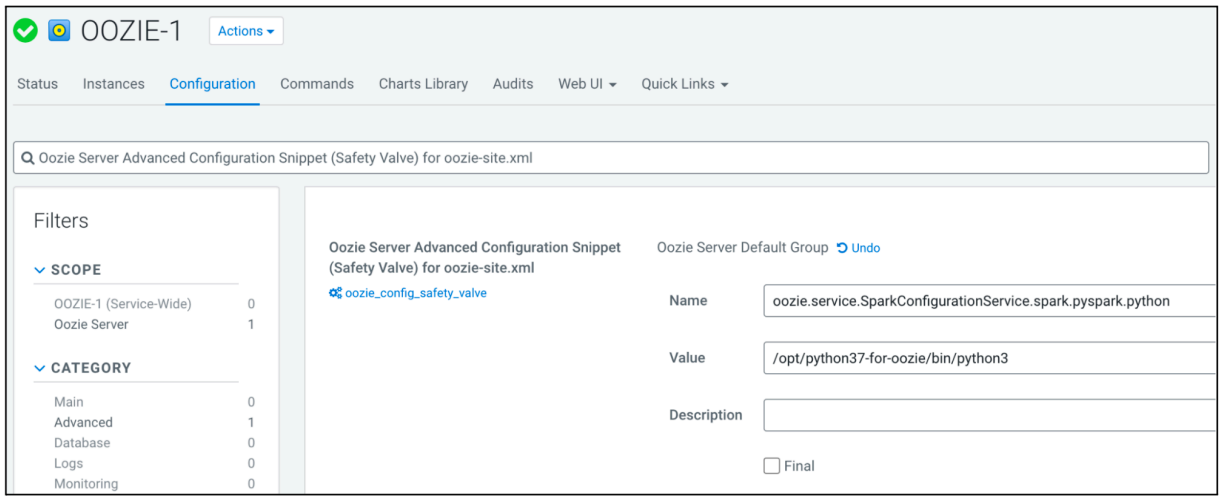
Setting Spark actions with a custom Python executable globally

You can set it globally in Cloudera Manager through a safety-valve. To do that, perform the following steps:

1. Navigate to Oozie's configuration page in Cloudera Manager.
2. Search for Oozie Server Advanced Configuration Snippet (Safety Valve) for oozie-site.xml.
3. Add a new property named oozie.service.SparkConfigurationService.spark.pyspark.python.

4. Specify its value to point to your custom Python executable.

   For example, if you installed Python 3.7 to /opt/python37-for-oozie, then specify the value as /opt/python37-for-oozie/bin/python3.



5. Save the modifications.
6. Allow Cloudera Manager some time to recognize the changes.
7. Redeploy Oozie.

Setting Spark actions with a custom Python executable per workflows

You can also specify a custom Python executable for a given workflow using the same property:

```
<workflow-app name="spark_workflow" xmlns="uri:oozie:workflow:1.0">
    <global>
        <configuration>
            <property>
                <name>oozie.service.SparkConfigurationService.spark.pyspark.
python</name>
                <value>/opt/python37-for-oozie/bin/python3</value>
            </property>
        </configuration>
    </global>
    <start to="spark_action"/>
    <action name="spark_action">
        ...
```

The same workflow-level Python executable can be achieved if you set the property in your job.properties file.

Setting Spark actions with a custom Python executable for a given Spark action only

Finally, you can only change the Python executable for a given Spark action. For example:

```
<workflow-app name="spark_workflow" xmlns="uri:oozie:workflow:1.0">
    <start to="spark_action"/>
    <action name="spark_action">
        <spark xmlns="uri:oozie:spark-action:1.0">
            <resource-manager>${resourceManager}</resource-manager>
            <name-node>${nameNode}</name-node>
            <configuration>
                <property>
                    <name>oozie.service.SparkConfigurationService.spark.pys
park.python</name>
                    <value>/opt/python37-for-oozie/bin/python3</value>
```

```
                </property>
            </configuration>
            ...
```

The following order of precedence is applied for this configuration:

1. Oozie does not override the configuration of spark.pyspark.python in the <spark-opts> tag of your action definition if you have already set it.
2. If you have configured the property at the action level, it takes precedence over all other settings, and the remaining configurations are disregarded.
3. If you have configured the property in the global configuration of the workflow, the value from there is used.
4. If the setting is not available in either of the previous locations, the value configured in your job.properties file is used.
5. Lastly, the global safety-valve setting comes into effect.

It is also possible to inform Oozie that you do not want to use a custom Python executable in a given Spark action, but you want to use the default one configured for Spark 2, even if you already configured at a lower level of precedence. For instance, if the oozie.service.SparkConfigurationService.spark.pyspark.python is set as a safety-valve to /opt/python37-for-oozie/bin/python3, but in a workflow or in a specific action you want to use the default Python executable configured for Spark 2, you can set the value of the property to default. For example:

```
<workflow-app name="spark_workflow" xmlns="uri:oozie:workflow:1.0">
    <global>
        <configuration>
            <property>
                <name>oozie.service.SparkConfigurationService.spark.pyspark.
python</name>
                <value>default</value>
            </property>
        </configuration>
    </global>
    <start to="spark_action"/>
    <action name="spark_action">
        ...
```

In this scenario, the value set in Cloudera Manager in Oozie's safety-valve configuration, is ignored, and the spark.pyspark.python Spark conf is not set at all.

Important: If you choose to use a custom Python executable in Spark actions, then the given executable must be available on all nodes where YARN Node Manager is also available.

Note: Since Spark 2 supports Python up to Python 3.7, you can use the above configuration option to specify any custom Python installation from 2.7 to 3.7.

Spark 3 Oozie action schema

Refer to the following for the schema of a Spark 3 Oozie action.

```
<xs:schema elementFormDefault="qualified"
           targetNamespace="uri:oozie:spark3-action:1.0"
           xmlns:spark3="uri:oozie:spark3-action:1.0"
           xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:include schemaLocation="oozie-common-1.0.xsd"/>
    <xs:element name="spark3" type="spark3:ACTION"/>
    <xs:complexType name="ACTION">
        <xs:sequence>
            <xs:element maxOccurs="1" minOccurs="1" name="resource-manager"
 type="xs:string"/>
```

```
            <xs:element maxOccurs="1" minOccurs="0" name="name-node" type=
"xs:string"/>
            <xs:element maxOccurs="1" minOccurs="0" name="prepare" type="s
park3:PREPARE"/>
            <xs:element maxOccurs="1" minOccurs="0" name="launcher" type="s
park3:LAUNCHER"/>
            <xs:element maxOccurs="unbounded" minOccurs="0" name="job-xml"
 type="xs:string"/>
            <xs:element maxOccurs="1" minOccurs="0" name="configuration" typ
e="spark3:CONFIGURATION"/>
            <xs:element maxOccurs="1" minOccurs="1" name="master" type="x
s:string"/>
            <xs:element maxOccurs="1" minOccurs="0" name="mode" type="xs:str
ing"/>
            <xs:element maxOccurs="1" minOccurs="1" name="name" type="xs:
string"/>
            <xs:element maxOccurs="1" minOccurs="0" name="class" type="xs:
string"/>
            <xs:element maxOccurs="1" minOccurs="1" name="jar" type="xs:st
ring"/>
            <xs:element maxOccurs="1" minOccurs="0" name="spark-opts" type="
xs:string"/>
            <xs:element maxOccurs="unbounded" minOccurs="0" name="arg" type
="xs:string"/>
            <xs:element maxOccurs="unbounded" minOccurs="0" name="file" t
ype="xs:string"/>
            <xs:element maxOccurs="unbounded" minOccurs="0" name="archive"
 type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```

Differences between Spark and Spark 3 actions

Learn about the differences between a Spark action and a Spark 3 action definition, the difference in the logging frameworks used in Spark and Spark 3, and the action credentials.

There are several notable distinctions between a Spark action and a Spark 3 action definition. Firstly, the XML element representing the action differs between the two. For Spark actions, it is denoted as spark, whereas for Spark 3 actions, it is labeled as spark3.

Additionally, the support for certain tags also varies. Specifically, the Spark 3 action definition does not include support for the job-tracker tag, but instead exclusively employs the resource-manager tag.

In contrast to Oozie's Spark action, where omitting the <mode> tag in the workflow definition allows setting the <master> tag to either yarn-cluster or yarn-client, the same flexibility does not apply to Spark 3 actions. With Spark 3 actions, if you previously set the <master> tag to yarn-cluster in Spark actions, you must set the <master> tag to yarn and the <mode> tag to cluster in Spark 3 actions. Similarly, if you previously set the <master> tag to yarn-client in Spark actions, for Spark 3 actions, you must set the <master> tag to yarn and the <mode> tag to client.

Differences in the actions' configuration

To configure Spark actions, there are multiple configuration options available. These options are prefixed with oozie.service.SparkConfigurationService. For example:

- oozie.service.SparkConfigurationService.spark.configurations.blacklist
- oozie.service.SparkConfigurationService.hive2.configurations

In order to differentiate between configurations for Spark actions and Spark 3 actions, the prefix for properties related to Spark 3 actions is modified to oozie.service.Spark3ConfigurationService.

During the migration from Spark actions to Spark 3 actions, it is necessary to adjust the prefix for any Spark action configuration that is configured in Cloudera Manager using a safety-valve.

Log4j vs. Log4j 2

When comparing Spark 2 and Spark 3, there is a difference in the logging frameworks used. While Spark 2 relies on log4j or reload4j, Spark 3 has transitioned to log4j2. As a result, Oozie's Spark 3 action also utilizes log4j2.

Oozie provides a logging property file for both Spark and Spark 3 actions. However, there is a distinction in the naming conventions of these files. In the case of Spark actions, the file is named spark-log4j.properties, whereas for Spark 3 actions, it is named spark3-log4j2.properties.

During the migration process from Spark to Spark 3 actions, if you have a custom spark-log4j.properties file located in the lib folder of your workflow or within Oozie's ShareLib, you need to rename this file. Since Spark 3 uses log4j2, you might also need to modify your custom Spark logging configuration file to ensure compatibility with log4j2.

Action credentials

The Spark 3 action in Oozie provides support for the same credentials as Spark actions.

Use Spark 3 actions with a custom Python executable

Learn how to use a custom Python executable in a given Spark 3 action.

Similar to Spark 2, Spark 3 also provides the capability to define a custom Python executable for use with spark3-submit through the spark.pyspark.python Spark 3 conf argument. For more details, please see the Latest Spark3 documentation. Consequently, if you include the spark.pyspark.python Spark 3 conf in your Oozie Spark 3 action, the Python executable you specify is used when executing the Spark 3 action through Oozie.

To simplify the usage of a customized Python executable with Oozie's Spark 3 action, you can use the oozie.servic e.Spark3ConfigurationService.spark.pyspark.python property. This property functions similar to Spark 3's spark.py spark.python conf argument, allowing you to specify a custom Python executable. Oozie then passes this executable to the underlying Spark 3 application executed through Oozie.

You can specify this configuration in different ways.

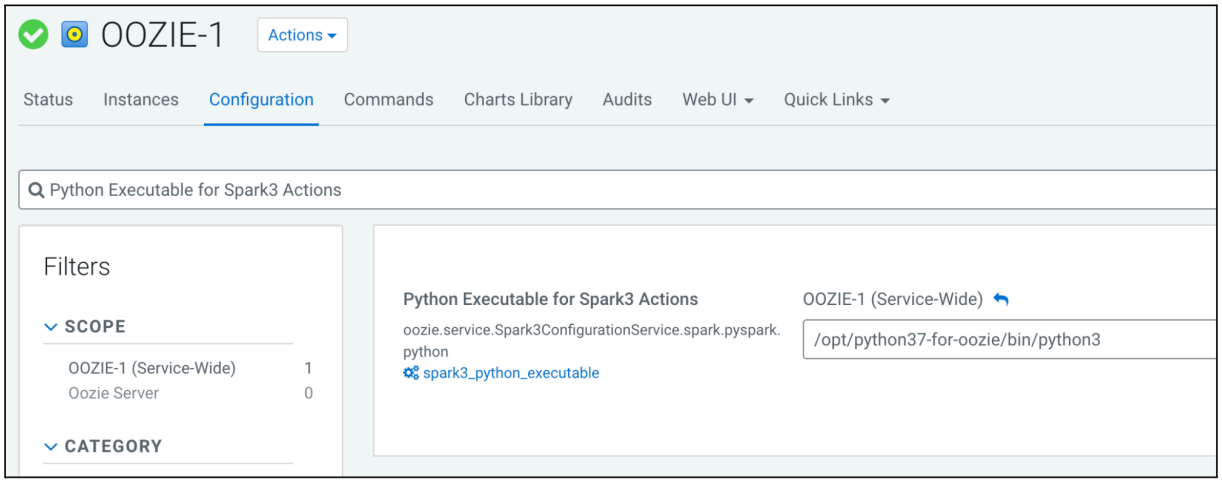
Setting Spark 3 actions with a custom Python executable globally

You can set it globally in Cloudera Manager. To do that, perform the following steps:

1. Navigate to Oozie's configuration page in Cloudera Manager.
2. Search for Python Executable for Spark3 Actions.

3. Specify its value to point to your custom Python executable.

   For example, if you installed Python 3.7 to /opt/python37-for-oozie, then specify the value as /opt/python37-for-oozie/bin/python3.



4. Save the modifications.
5. Allow Cloudera Manager some time to recognize the changes.
6. Redeploy Oozie.

Setting Spark 3 actions with a custom Python executable per workflows

You can also specify a custom Python executable for a given workflow using the same property:

```
<workflow-app name="spark_workflow" xmlns="uri:oozie:workflow:1.0">
    <global>
        <configuration>
            <property>
                <name>oozie.service.Spark3ConfigurationService.spark.pyspark
.python</name>
                <value>/opt/python37-for-oozie/bin/python3</value>
            </property>
        </configuration>
    </global>
    <start to="spark_action"/>
    <action name="spark_action">
        ...
```

The same workflow-level Python executable can be achieved if you set the property in your job.properties file.

Setting Spark 3 actions with a custom Python executable for a given Spark action only

Finally, you can only change the Python executable for a given Spark 3 action. For example:

```
<workflow-app name="spark_workflow" xmlns="uri:oozie:workflow:1.0">
    <start to="spark_action"/>
    <action name="spark_action">
        <spark3 xmlns="uri:oozie:spark3-action:1.0">
            <resource-manager>${resourceManager}</resource-manager>
            <name-node>${nameNode}</name-node>
            <configuration>
                <property>
                    <name>oozie.service.Spark3ConfigurationService.spark.
pyspark.python</name>
                    <value>/opt/python37-for-oozie/bin/python3</value>
                </property>
```

```
            </configuration>
            ...
```

The following order of precedence is applied for this configuration:

1. Oozie does not override the configuration of spark.pyspark.python in the <spark-opts> tag of your action definition if you have already set it.
2. If you have configured the property at the action level, it takes precedence over all other settings, and the remaining configurations are disregarded.
3. If you have configured the property in the global configuration of the workflow, the value from there is used.
4. If the setting is not available in either of the previous locations, the value configured in your job.properties file is used.
5. Lastly, the global setting in Cloudera Manager comes into effect.

It is also possible to inform Oozie that you do not want to use a custom Python executable in a given Spark 3 action, but you want to use the default one configured for Spark 3, even if you already configured at a lower level of precedence. For instance, if the Python Executable for Spark3 Actions property is set in Cloudera Manager to /opt/python37-for-oozie/bin/python3, but in a workflow or in a specific action you want to use the default Python executable configured for Spark 3, you can set the value of the property to default. For example:

```
<workflow-app name="spark_workflow" xmlns="uri:oozie:workflow:1.0">
    <global>
        <configuration>
            <property>
                <name>oozie.service.Spark3ConfigurationService.spark.pyspark
.python</name>
                <value>default</value>
            </property>
        </configuration>
    </global>
    <start to="spark_action"/>
    <action name="spark_action">
        ...
```

In this scenario, the value set in Cloudera Manager is ignored, and the spark.pyspark.python Spark 3 conf is not set at all.

Important: If you choose to use a custom Python executable in Spark 3 actions, then the given executable must be available on all nodes where YARN Node Manager is also available.

Spark 3 compatibility action executor

To facilitate smooth transitions from Oozie's Spark actions to Spark 3 actions, you can use the Spark 3 compatibility action executor. The purpose of this executor is to allow you to retain your existing Spark action definitions in your workflows while executing them with Spark 3 instead of Spark 2.

When you use this executor, Oozie automatically converts a Spark action definition to a Spark 3 action definition before executing it. To configure the Spark 3 action, Oozie utilizes the Spark action configurations (prefixed with oozie.service.SparkConfigurationService) and converts them to Spark 3 configurations.

This feature enables you to run Spark 3 actions without making any modifications to your workflow definitions.

Important: However, it is important to note that this compatibility executor should be considered as a temporary solution while you transition your Spark action definitions to Spark 3 action definitions. It might not support all the features provided by the actual Spark 3 action executor.

Additionally, remember that your Python or Java Spark actions must be runtime and binary compatible with Spark 3. This is necessary to execute them using the compatibility action executor. For more detailed information on this topic, please refer to the *Migration of Spark 2 applications* in this document.

To enable Spark 3 compatibility mode for Spark 2 action workflow definitions, you use the oozie.action.spark.compatibility property.

You can configure this property in the following ways:

- Global configuration:
  1. Go to the Oozie configuration page in Cloudera Manager and search for Oozie Server Advanced Configuration Snippet (Safety Valve) for oozie-site.xml.
  2. Add a new key with the name oozie.action.spark.compatibility and set the value to true.
  3. Redeploy Oozie.
- Workflow-level configuration

  Add the property to the global configuration section of your workflow.xml file. Alternatively, you can add it to your job.properties file.
- Action-level Configuration

  If you only want to enable compatibility mode for a specific Spark action in your workflow definition, add an action-level property with the name oozie.action.spark.compatibility and set the value to true.

Limitations

Due to the high customizability of Oozie's Spark and Spark 3 actions, certain restrictions and limitations have been introduced. The current known limitations are outlined as follows:

- The following properties are not permitted in your action configuration, which includes configurations at the global level, workflow level, and action level:
  - oozie.action.sharelib.for.spark
  - oozie.action.sharelib.for.spark3
  - oozie.action.sharelib.for.spark.exclude
  - oozie.action.sharelib.for.spark3.exclude
- As the Spark 3 action utilizes log4j2 instead of log4j (used by the Spark action), it is crucial to avoid mixing the two in compatibility mode. Therefore, having a spark-log4j.properties file in the lib folder of your workflow or in ShareLib is not allowed.

Related Information

Migration of Spark 2 applications

Spark 3 examples with Python or Java application

This section provides you some examples of Spark 3 with Python and Java applications.

Spark3 with a Java application with Hive2 credentials

```
<workflow-app name="JavaSpark" xmlns="uri:oozie:workflow:1.0">
    <credentials>
        <credential name="hive-credential" type="hive2">
            <property>
                <name>hive2.jdbc.url</name>
                <value>jdbc:hive2://...</value>
            </property>
            <property>
                <name>hive2.server.principal</name>
                <value>...</value>
            </property>
        </credential>
    </credentials>
    <start to="spark-node-javaspark"/>
    <action name="spark-node-javaspark" cred="hive-credential">
```

```
        <spark3 xmlns="uri:oozie:spark3-action:1.0">
            <resource-manager>${resourceManager}</resource-manager>
            <name-node>${nameNode}</name-node>
            <master>${master}</master>
            <mode>${mode}</mode>
            <name>JavaSpark-Example</name>
            <class>com.company.spark.JavaSpark</class>
            <jar>${nameNode}/user/${wf:user()}/javaspark_lib/JavaSparkForOo
zie.jar</jar>
            <arg>${inputFile}</arg>
        </spark3>
        <ok to="end"/>
        <error to="fail"/>
    </action>
    <kill name="fail">
        <message>Workflow failed, error
            message[${wf:errorMessage(wf:lastErrorNode())}]
        </message>
    </kill>
    <end name="end"/>
</workflow-app>
```

Spark3 with a Python application with Hive2 credentials

```
<workflow-app name="PySpark" xmlns="uri:oozie:workflow:1.0">
    <credentials>
        <credential name="hive-credential" type="hive2">
            <property>
                <name>hive2.jdbc.url</name>
                <value>jdbc:hive2://...</value>
            </property>
            <property>
                <name>hive2.server.principal</name>
                <value>...</value>
            </property>
        </credential>
    </credentials>
    <start to="spark-node-pyspark"/>
    <action name="spark-node-pyspark" cred="hive-credential">
        <spark3 xmlns="uri:oozie:spark3-action:1.0">
            <resource-manager>${resourceManager}</resource-manager>
            <name-node>${nameNode}</name-node>
            <master>${master}</master>
            <mode>${mode}</mode>
            <name>PySpark-Example</name>
            <jar>${pythonScript}</jar>
            <arg>${inputFile}</arg>
        </spark3>
        <ok to="end"/>
        <error to="fail"/>
    </action>
    <kill name="fail">
        <message>Workflow failed, error
            message[${wf:errorMessage(wf:lastErrorNode())}]
        </message>
    </kill>
    <end name="end"/>
</workflow-app>
```

Shell action for Spark 3

Learn about how to execute Spark 3's spark3-submit through Oozie's Shell action.

Similar to the support for executing Spark 2's spark-submit through Oozie's Shell action, Cloudera also provides full support for executing Spark 3's spark3-submit through Oozie's Shell action.

As Oozie utilizes delegation tokens instead of Kerberos tickets in its YARN applications, it is recommended to unset the HADOOP_TOKEN_FILE_LOCATION environment variable in your Shell script before executing spark3-submit, if you intend to use spark3-submit without relying on Oozie's default delegation tokens. This is because spark3-submit might not function properly with both delegation tokens and Kerberos tickets. However, to ensure the successful completion of your Shell action, please ensure that you reset the HADOOP_TOKEN_FILE_LOCATION environment variable after the execution of your custom Shell script segment. The following example illustrates how you can accomplish this:

```
#!/usr/bin/env bash
# By executing the commands within brackets,
# we can ensure that the parent environment remains untouched
(
    unset HADOOP_TOKEN_FILE_LOCATION

    kinit -kt /var/keytabs/user.keytab user
    /usr/bin/spark3-submit --master yarn --deploy-mode cluster \
        create_table_with_data_spark3.py tableUsingSpark3FromShellAction
    /usr/bin/spark3-submit --master yarn --deploy-mode cluster \
        read_created_table.py tableUsingSpark3FromShellAction
)
```

Migration of Spark 2 applications

To ensure a smooth migration of your underlying Spark 2 application when using Oozie's Spark 3 action, it is highly recommended to follow the official Spark 2 to Spark 3 migration guide. Specifically, refer to the Cloudera runtime documentation's comprehensive resource titled *Updating Spark 2 applications for Spark3*.

This guide provides you with detailed instructions and best practices specifically tailored to the migration process. By adhering to this guide, you can effectively transition your Spark 2 application and leverage the capabilities of Spark 3 seamlessly within Oozie.

It is highly advised to first test your migrated Spark 3 applications directly using the official Spark 3 runtime. By executing your applications with spark3-submit before running them as Oozie actions, you can identify any potential issues and find the root cause. This approach enables you to determine whether any problems arise from Oozie, Spark 3 itself, or the compatibility of your migrated application. Taking this proactive step assists you in troubleshooting and resolving any potential obstacles during the migration process.

Migrating Java applications

If you are currently running Java applications with Oozie's Spark action, there are several important considerations and steps to follow:

- Recompile your application using Spark 3 dependencies instead of Spark 2 dependencies.
- If you are using Scala, it may be necessary to migrate from Scala 2.11 to Scala 2.12 according to the official Spark migration guide.
- If your application relies on the Scala module of the Jackson library, you might need to replace the Scala 2.11 flavor with the Scala 2.12 flavor.
- Ensure that your 3rd-party runtime dependencies align with the versions used by Spark 3 and Oozie's Spark 3 action.
- Since Spark 3 has transitioned from log4j to log4j2, you might need to adjust the logging library and/or logging configuration used in your application. Additionally, ensure that all necessary logging frameworks are present

in the classpath. Note that Oozie's Spark 3 action executor configures log4j2 instead of log4j, meaning that log4j runtime libraries are no longer included in the classpath by default, only log4j2 libraries are.

By following these guidelines, you can successfully migrate your Java applications to work seamlessly with Oozie's Spark 3 action.

Migrating Python applications

When upgrading PySpark applications from Spark 2 to Spark 3, it is important to consider not only the framework migration but also the compatibility of your Python application with Python 3. If your Python application is not compatible, you need to perform a migration from Python 2 to Python 3 as well.

Given that Spark 2 is now deprecated and Python 2 has reached its end of life, Cloudera strongly advise migrating your PySpark applications from Python 2 and Spark2 simultaneously. This approach ensures that your application runs on up-to-date frameworks and avoids potential security vulnerabilities.

However, depending on the nature of your applications (such as their size or quantity), you might want to take the following steps:

1. Follow the instructions in the *Enable Spark actions* section to re-enable Spark actions.
2. Install Python 2 to a custom location to prevent interference with other services and avoid its usage by other services.
3. Configure Oozie's Spark action globally to use this custom Python 2 installation. Refer to the *Using Spark actions with a custom Python executable* section for guidance.
4. Additionally, install Python 3.7 to a custom location.
5. Begin migrating your PySpark applications to ensure compatibility with Python 3.
6. For workflows where you have already made the underlying PySpark application Python 3 compatible, enable them to run with the custom Python 3 installation instead of Python 2. Refer to the *Using Spark actions with a custom Python executable* section for detailed instructions.
7. After all the Spark actions in a workflow are compatible with Python 3, start migrating the Spark actions to Spark 3 actions.

By following these steps, you can successfully migrate your PySpark applications from Python 2 and Spark 2 to Python 3 and Spark 3.

Related Information

Enable Spark actions

Use Spark actions with a custom Python executable

Updating Spark 2 applications for Spark3

Hue support for Oozie

As Hue's Oozie designer feature is not currently being enhanced, there are no plans to provide support for Oozie Spark 3 actions within the Hue platform at this time.

Oozie High Availability

Oozie High Availability is "active-active" so that both Oozie servers are active at the same time, with no failover. High availability for Oozie is supported in both MRv1 and MRv2 (YARN).

Requirements for Oozie High Availability

You must ensure your cluster meets all the requirements for configuring Oozie High Availability (HA).

- Multiple active Oozie servers, preferably identically configured.

- JDBC JAR in the same location across all Oozie hosts (for example, /var/lib/oozie/).
- External database that supports multiple concurrent connections, preferably with HA support.
- ZooKeeper ensemble with distributed locks to control database access, and service discovery for log aggregation.
- Load balancer (preferably with HA support, for example HAProxy), virtual IP, or round-robin DNS to provide a single entry point (of the multiple active servers), and for callbacks from the Application Master or JobTracker.

Configuring Oozie High Availability using Cloudera Manager

You can use Cloudera Manager to enable or disable Oozie High Availability (HA).

Important: Enabling or disabling HA makes the previous monitoring history unavailable.

Oozie Load Balancer configuration

To enable Oozie High Availability, you must manually configure a Load Balancer.

About this task

Cloudera recommends using the HAProxy Load Balancer. These steps explain how to configure the HAProxy load balancer. However, you can choose to configure a different Load Balancer.

Procedure

1. Install HAProxy on the host where you are setting up and configuring the Oozie load balancer. For more information, see the HAProxy documentation.
2. You must configure the Oozie load balancer for both HTTP and HTTPS ports.

```
This is an example:
global
    log         127.0.0.1 local2
    pidfile     /var/run/haproxy.pid
    maxconn     4000
    user        haproxy
    group       haproxy
    daemon
    stats       socket /tmp/haproxy

defaults
    mode                    http
    log                     global
    option                  httplog
    option                  dontlognull
    option forwardfor       except 127.0.0.0/8
    option                  redispatch
    retries                 3
    timeout http-request    10s
    timeout queue           1m
    timeout connect         10s
    timeout client          10m
    timeout server          10m
    timeout check           10s
    maxconn                 3000

listen admin
        bind *:8000
        stats enable

#---------------------------------------------------------------------
# main frontend which proxys to the backends
```

```
#---------------------------------------------------------------------
frontend                        oozie_front
    bind                        *:5000 ssl crt /var/lib/cloudera-scm-agent
/agent-cert/cdep-host_key_cert_chain_decrypted.pem
    default_backend             oozie

#---------------------------------------------------------------------
# round robin balancing between the various backends
#---------------------------------------------------------------------
backend oozie
    balance                     roundrobin
    server  oozie1 my-oozie-host-1:11443/oozie check ssl ca-file /var/lib/
cloudera-scm-agent/agent-cert/cm-auto-global_cacerts.pem
    server  oozie2 my-oozie-host-2:11443/oozie check ssl ca-file /var/lib/
cloudera-scm-agent/agent-cert/cm-auto-global_cacerts.pem
    server  oozie3 my-oozie-host-3:11443/oozie check ssl ca-file /var/lib/
cloudera-scm-agent/agent-cert/cm-auto-global_cacerts.pem

#---------------------------------------------------------------------
# main frontend which proxys to the http backends
#---------------------------------------------------------------------
frontend                        oozie_front_http
    bind                        *:5002
    default_backend             oozie_http

#---------------------------------------------------------------------
# round robin balancing between the various http backends
#---------------------------------------------------------------------
backend oozie_http
    balance                     roundrobin
    server  oozie_http1 my-oozie-host-1:11000/oozie check
    server  oozie_http2 my-oozie-host-2:11000/oozie check
    server  oozie_http3 my-oozie-host-3:11000/oozie check
```

Using the example, the load balancer is setup for three Oozie instances. The load balancer listens on port 5002 for HTTP connections and forwards it to Oozie's port 11000. The load balancer listens on port 5000 for HTTPS connections and forwards it to Oozie's port 11443.

If you not enabled SSL in Oozie, then you do not need the HTTPS load balancer. For HTTPS load balancing, ensure that you set up the certificate.

3. Continue to configure the load balancer by enabling Oozie High Availability. For information about enabling Oozie High Availability, see Enabling Oozie High Availability .

Enabling Oozie High Availability

You must select the host on which to install the additional Oozie server and specify the required property values to install Oozie High Availability (HA).

Before you begin

Ensure that the requirements are satisfied.

Procedure

1. In the Cloudera Manager Admin Console, go to the Oozie service.
2. Select Actions Enable High Availability to see eligible Oozie server hosts. The host running the current Oozie server is not eligible.
3. Select the host on which to install an additional Oozie server and click Continue.

4. Update the following fields for the Oozie load balancer:
   - Hostname

     For example:

     ```
     nightly6x-1.vpc.cloudera.com
     ```
   - HTTP Port

     For example:

     ```
     5002
     ```
   - HTTPS Port

     For example:

     ```
     5000
     ```
5. Click Continue.

   Cloudera Manager stops the Oozie servers, adds another Oozie server, initializes the Oozie server High Availability state in ZooKeeper, configures Hue to reference the Oozie load balancer, and restarts the Oozie servers and dependent services. In addition, Cloudera Manager generates Kerberos credentials for the new Oozie server and regenerates credentials for existing servers.

Disabling Oozie High Availability

Based on your requirements, you can disable Oozie High Availability (HA) using Cloudera Manager.

Procedure

1. In the Cloudera Manager Admin Console, go to the Oozie service.
2. Select Actions Disable High Availability to see all hosts currently running Oozie servers.
3. Select the one host to run the Oozie server and click Continue.

   Cloudera Manager stops the Oozie service, removes the additional Oozie servers, configures Hue to reference the Oozie service, and restarts the Oozie service and dependent services.

Scheduling in Oozie using cron-like syntax

Most Linux distributions include the cron utility, which is used for scheduling time-based jobs. You can schedule Oozie using Cron-like syntax.

Location

Set the scheduling information in the frequency attribute of the coordinator.xml file. A simple file looks like the following example. The frequency attribute and scheduling information appear in bold.

```
<coordinator-app name="MY_APP" frequency="30 14 * * *"
start="2009-01-01T05:00Z" end="2009-01-01T06:00Z" timezone="UTC" xmlns="ur
i:oozie:coordinator:0.5">
   <action>
      <workflow>
         <app-path>hdfs://localhost:8020/tmp/workflows</app-path>
      </workflow>
   </action>
</coordinator-app>
```

Syntax and structure

The cron-like syntax used by Oozie is a string with five space-separated fields:

- minute
- hour
- day-of-month
- month
- day-of-week

The structure takes the form of * * * * *. For example, 30 14 * * * means that the job runs at at 2:30 p.m. everyday. The minute field is set to 30, the hour field is set to 14, and the remaining fields are set to *.

Allowed values and special characters

The following table describes special characters allowed and indicates in which fields they can be used.

Table 1: Special characters

Character	Fields Allowed	Description
* (asterisk)	All	Match all values.
, (comma)	All	Specify multiple values.
- (dash)	All	Specify a range.
/ (forward slash)	All	Specify an increment.
? (question mark)	Day-of-month, day-of-week	Indicate no specific value (for example, if you want to specify one but not the other).
L	Day-of-month, day-of-week	Indicate the last day of the month or the last day of the week (Saturday). In the day-of-week field, 6L indicates the last Friday of the month.
W	Day-of-month	Indicate the nearest weekday to the given day.
# (pound sign)	Day-of-week	Indicate the nth day of the month

The following table summarizes the valid values for each field.

Field	Allowed Values	Allowed Special Characters
Minute	0-59	, - * /
Hour	0-23	, - * /
Day-of-month	0-31	, - * ? / L W
Month	1-12 or JAN-DEC	, - * /
Day-of-week	1-7 or SUN-SAT	, - * ? / L #

Important: Some cron implementations accept 0-6 as the range for days of the week. Oozie accepts 1-7 instead.

Oozie scheduling examples

You can use cron scheduling in Oozie to ensure that the jobs run according to the criteria that you specify.

The following examples show cron scheduling in Oozie. Oozie's processing time zone is UTC. If you are in a different time zone, add to or subtract from the appropriate offset in these examples.

Run at the 30th minute of every hour

Set the minute field to 30 and the remaining fields to * so they match every value.

```
frequency="30 * * * *"
```

Run at 2:30 p.m. every day

Set the minute field to 30, the hour field to 14, and the remaining fields to *.

```
frequency="30 14 * * *"
```

Run at 2:30 p.m. every day in February

Set the minute field to 30, the hour field to 14, the day-of-month field to *, the month field to 2 (February), and the day-of-week field to *.

```
frequency="30 14 * 2 *"
```

Run every 20 minutes between 5:00-10:00 a.m. and between 12:00-2:00 p.m. on the fifth day of each month

Set the minute field to 0/20, the hour field to 5-9,12-13, the day-of-month field to 0/5, and the remaining fields to *.

```
frequency="0/20 5-9,12-13 0/5 * *"
```

Run every Monday at 5:00 a.m.

Set the minute field to 0, the hour field to 5, the day-of-month field to ?, the month field to *, and the day-of-week field to MON.

```
frequency="0 5 ? * MON"
```

Note: If the ? was set to *, this expression would run the job every day at 5:00 a.m., not just Mondays.

Run on the last day of every month at 5:00 a.m.

Set the minute field to 0, the hour field to 5, the day-of-month field to L, the month field to *, and the day-of-week field to ?.

```
frequency="0 5 L * ?"
```

Run at 5:00 a.m. on the weekday closest to the 15th day of each month

Set the minute field to 0, the hour field to 5, the day-of-month field to 15W, the month field to *, and the day-of-week field to ?.

```
frequency="0 5 15W * ?"
```

Run every 33 minutes from 9:00-3:00 p.m. on the first Monday of every month

Set the minute field to 0/33, the hour field to 9-14, the day-of-week field to 2#1 (the first Monday), and the remaining fields to *.

```
frequency="0/33 9-14 ? * 2#1"
```

Run every hour from 9:00 a.m.-5:00 p.m. on weekdays

Set the minute field to 0, the hour field to 9-17, the day-of-month field to ?, the month field to *, and the day-of-week field to 2-6.

```
frequency="0 9-17 ? * 2-6"
```

Run on the second-to-last day of every month

Set the minute field to 0, the hour field to 0, the day-of-month field to L-1, the month field to *, and the day-of-week field to ?.

```
frequency="0 0 L-1 * ?"
```

Note: “L-1# means the second-to-last day of the month.

Oozie uses Quartz, a job scheduler library, to parse the cron syntax. For more examples, go to the CronTrigger Tutorial on the Quartz website. Quartz has two fields (second and year) that Oozie does not support.

Configuring an external database for Oozie

Oozie is a stateless web application by design. All information about running and completed workflows, coordinators, and bundle jobs are stored in a relational database. Oozie supports an embedded PostgreSQL database; however, Cloudera strongly recommends that you use an external database for production systems.

Related Information

Oozie database configurations

Configuring PostgreSQL for Oozie

You must install PostgreSQL, create the Oozie user and database, and configure PostgreSQL to accept network connections for the Oozie user.

Procedure

1. Install PostgreSQL

   See the PostgreSQL documentation to install it.
2. Create the Oozie User and Oozie Database.

   For example, using the PostgreSQL psql command-line tool:

```
$ psql -U postgres
Password for user postgres: *****

postgres=# CREATE ROLE oozie LOGIN ENCRYPTED PASSWORD 'oozie'
 NOSUPERUSER INHERIT CREATEDB NOCREATEROLE;
CREATE ROLE

postgres=# CREATE DATABASE "oozie" WITH OWNER = oozie
 ENCODING = 'UTF8'
 TABLESPACE = pg_default
 LC_COLLATE = 'en_US.UTF-8'
 LC_CTYPE = 'en_US.UTF-8'
 CONNECTION LIMIT = -1;
CREATE DATABASE

postgres=# \q
```

3. Configure PostgreSQL to Accept Network Connections for the Oozie User.
   a) Edit the postgresql.conf file and set the listen_addresses property to *, to make sure that the PostgreSQL server starts listening on all your network interfaces. Also make sure that the standard_conforming_strings property is set to off.
   b) Edit the PostgreSQL data/pg_hba.conf file as follows:

```
host    oozie         oozie         0.0.0.0/0             md5
```

4. Reload the PostgreSQL Configuration.

```
sudo -u postgres pg_ctl reload -s -D /opt/PostgreSQL/8.4/data
```

Configuring MariaDB for Oozie

You must install MariaDB, create the Oozie database and MariaDB user, and add the MariaDB JDBC driver jar file to Oozie.

Procedure

1. Install and Start MariaDB.
2. Create the Oozie Database and Oozie MariaDB User.

   For example, using the MariaDB mysql command-line tool:

```
$ mysql -u root -p
Enter password:

MariaDB [(none)]> create database oozie default character set utf8;
Query OK, 1 row affected (0.00 sec)
MariaDB [(none)]>  grant all privileges on oozie.* to 'oozie'@'localhost'
 identified by 'oozie';
Query OK, 0 rows affected (0.00 sec)

MariaDB [(none)]>  grant all privileges on oozie.* to 'oozie'@'%' identi
fied by 'oozie';
Query OK, 0 rows affected (0.00 sec)

MariaDB [(none)]> exit
Bye
```

3. Add the MariaDB JDBC Driver JAR to Oozie.

   Cloudera recommends that you use the MySQL JDBC driver for MariaDB. Copy or symbolically link the MySQL JDBC driver JAR to the /var/lib/oozie/ directory.

Note: You must manually download the MySQL JDBC driver JAR file.

Configuring MySQL 5 for Oozie

You must install MySQL 5, create the Oozie database and MySQL user, and add the MySQL JDBC driver jar file to Oozie.

Procedure

1. Install and Start MySQL 5.

2. Create the Oozie Database and Oozie MySQL User.

   For example, using the MySQL mysql command-line tool:

```
$ mysql -u root -p
Enter password:

mysql> create database oozie default character set utf8;
Query OK, 1 row affected (0.00 sec)

mysql>  grant all privileges on oozie.* to 'oozie'@'localhost' identified
by 'oozie';
Query OK, 0 rows affected (0.00 sec)

mysql>  grant all privileges on oozie.* to 'oozie'@'%' identified by 'oozi
e';
Query OK, 0 rows affected (0.00 sec)

mysql> exit
```

3. Add the MySQL JDBC Driver JAR to Oozie.

   Copy or symbolically link the MySQL JDBC driver JAR into one of the following directories:

   - For installations that use packages: /var/lib/oozie/
   - For installations that use parcels: /opt/cloudera/parcels/CDH/lib/oozie/lib/

   Note: You must manually download the MySQL JDBC driver JAR file.

Configuring MySQL 8 for Oozie

You must install MySQL 8, create the Oozie database and MySQL user, and add the MySQL JDBC driver jar file to Oozie.

Procedure

1. Install and Start MySQL 8.
2. Create the Oozie Database and Oozie MySQL User.

   For example, using the MySQL mysql command-line tool:

```
$ mysql -u root -p
 Enter password:
mysql> create database oozie default character set utf8;
 Query OK, 1 row affected (0.00 sec)
mysql> CREATE USER 'oozie'@'localhost' IDENTIFIED BY 'oozie';
 Query OK, 0 rows affected (0.00 sec)
mysql> GRANT ALL PRIVILEGES ON oozie.* TO 'oozie'@'localhost';
 Query OK, 0 rows affected (0.00 sec)
mysql> CREATE USER 'oozie'@'%' IDENTIFIED BY 'oozie';
 Query OK, 0 rows affected (0.01 sec)
mysql> GRANT ALL PRIVILEGES ON oozie.* TO 'oozie'@'%';
 Query OK, 0 rows affected (0.00 sec)
mysql> exit
```

3. Add the MySQL JDBC Driver JAR to Oozie.

   Copy or symbolically link the MySQL JDBC driver JAR into one of the following directories:

   - For installations that use packages: /var/lib/oozie/
   - For installations that use parcels: /opt/cloudera/parcels/CDH/lib/oozie/lib/

   Note: You must manually download the MySQL JDBC driver JAR file.

Configuring Oracle for Oozie

You must install Oracle 12.2, create the Oozie Oracle user and grant privileges, and add the Oracle JDBC driver jar file to Oozie.

Procedure

1. Install and Start Oracle 12.2

   Use Oracle's instructions.

2. Create the Oozie Oracle User and Grant Privileges.

   The following example uses the Oracle sqlplus command-line tool, and shows the privileges Cloudera recommends. Oozie needs CREATE SESSION to start and manage workflows. The additional roles are needed for creating and upgrading the Oozie database.

   ```
   sqlplus system@localhost/<SERVICE_NAME>

   SQL> create user <user> identified by <password> default tablespace <ta
   blespace> temporary tablespace temp;
       User created.
   SQL> grant create sequence to <user>;
       Grant succeeded.
   SQL> grant create session to <user>;
       Grant succeeded.
   SQL> grant create table to <user>;
       Grant succeeded.
   SQL> alter user <user> quota unlimited on <tablespace>;
       User altered.
   SQL> exit
   ```

   Important:

   For security reasons, do not make the following grant:

   ```
   grant select any table to oozie;
   ```

3. Add the Oracle JDBC Driver JAR to Oozie.

   Copy or symbolically link the Oracle JDBC driver JAR into the /var/lib/oozie/ directory.

   Note: You must manually download the Oracle JDBC driver JAR file.

Working with the Oozie server

You can use the command-line interface to start or stop the Oozie server. In addition, you can access the Oozie server with the Oozie client or with a browser.

Starting the Oozie server

You can use the service oozie start command to start Oozie.

Before you begin

Ensure that you have performed *all* the required configuration steps.

Procedure

- Use the service oozie start command to start Oozie.

  If you see the message Oozie System ID [oozie-oozie] started in the oozie.log log file, the system has started successfully.

  Note: By default, Oozie server runs on port 11000 and its URL is http://<OOZIE_HOSTNAME>:11000/oozie. If SSL is enabled, then Oozie server runs on port 11443 by default.

Stopping the Oozie server

Use the sudo service oozie stop command to stop a running Oozie server.

Accessing the Oozie server with the Oozie Client

The Oozie client is a command-line utility that interacts with the Oozie server using the Oozie web-services API.

Procedure

- Use the /usr/bin/oozie script to run the Oozie client.

  For example, if you want to invoke the client on the same machine where the Oozie server is running:

  ```
  $ oozie admin -oozie https://<oozie_server>:11443/oozie -status
  System mode: NORMAL
  ```

- To make it convenient to use this utility, set the environment variable OOZIE_URL to point to the URL of the Oozie server. Then you can skip the -oozie option.

  For example, if you want to invoke the client on the same machine where the Oozie server is running, set the OOZIE_URL to https://<oozie_server>:11443/oozie.

  ```
  $ export OOZIE_URL=https://<oozie_server>:11443/oozie
  $ oozie admin -version
  Oozie server build version: 4.0.0-cdh5.0.0
  ```

  Important: If Oozie is configured with Kerberos Security enabled:

  - You must have a Kerberos session running. For example, you can start a session by running the kinit command.
  - Do not use localhost.

  As with every service that uses Kerberos, Oozie has a Kerberos principal in the form <SERVICE>/<HOSTNAME>@<REALM>. In a Kerberos configuration, you must use the <HOSTNAME> value in the Kerberos principal to specify the Oozie server; for example, if the <HOSTNAME> in the principal is myoozieserver.mydomain.com, set OOZIE_URL as follows:

  ```
  export OOZIE_URL=https://myoozieserver.mydomain.com:11443/oozie
  ```

  If you use an alternate hostname or the IP address of the service, Oozie will not work properly.

- If you want to access Oozie client through Knox:

```
export OOZIE_URL=https://<knox_host>:<knox_port>/gateway/cdp-proxy-api/o
ozie
```

When you access Oozie client through Knox, you need to specify a username and password in the command line as Knox needs it:

```
export OOZIE_URL=https://<knox_host>:<knox_port>/gateway/cdp-proxy-api/o
ozie
oozie admin -version -auth BASIC -username <username> -password <password>
```

- When the Oozie server has SSL enabled, the Oozie client does not automatically set the necessary trust-store properties to form a connection. You can set these properties using the following methods:
  - Add them as system properties immediately after the oozie command. For instance:

```
oozie \
    "-Djavax.net.ssl.trustStore={trustStorePath}" \
    "-Djavax.net.ssl.trustStorePassword={trustStorePassword}" \
    "-Djavax.net.ssl.trustStoreType={trustStoreType}" \
    {oozieCommand} \
    -oozie "{oozieUrl}" \
    ...
```

  - You can also set these properties by defining the OOZIE_CLIENT_OPTS environment variable before running the Oozie command. For instance:

```
export OOZIE_CLIENT_OPTS="-Djavax.net.ssl.trustStore={trustStorePath} -D
javax.net.ssl.trustStorePassword={trustStorePassword} -Djavax.net.ssl.tr
ustStoreType={trustStoreType}"
```

  - If you prefer, you can also utilize the -insecure argument with the Oozie command line to prevent the client from validating the certificates:

```
oozie \
    {oozieCommand} \
    -oozie "{oozieUrl}" \
    -insecure \
    ...
```

Accessing the Oozie server with a browser

If you have enabled the Oozie web console by adding the ExtJS library, you can connect to the console at http://<OOZIE_HOSTNAME>:11000/oozie.

Note: If the Oozie server is configured to use Kerberos HTTP SPNEGO Authentication, you must use a web browser that supports Kerberos HTTP SPNEGO (for example, Firefox or Internet Explorer).

For information on how to enable the Oozie web console on managed clusters by adding the ExtJS library, see *Enabling the Oozie web console on managed clusters*.

Related Information

Enabling the Oozie web console on managed clusters

Adding schema to Oozie using Cloudera Manager

Cloudera Manager automatically configures Oozie with all available official schemas, and corresponding tables. You can manually add a schema (official or custom) with Cloudera Manager.

Procedure

1. In the Cloudera Manager Admin Console, go to the Oozie service.
2. Click the Configuration tab.
3. Select Scope Oozie Server .
4. Select Category Advanced .
5. Locate the Oozie SchemaService Workflow Extension Schemas property or search for it by typing its name in the Search box.
6. Enter the desired schema from the following schema list appending .xsd to each entry.

   To apply this configuration property to other role groups as needed, edit the value for the appropriate role group.
7. Enter a Reason for change, and then click Save Change to commit the changes.
8. Restart the Oozie service.

Table 2: Oozie schema

Schema	CDP
distcp	distcp-action-0.1 distcp-action-0.2 distcp-action-1.0
email	email-action-0.1 email-action-0.2
git	git-action-1.0
hive	hive-action-0.2 hive-action-0.3 hive-action-0.4 hive-action-0.5 hive-action-0.6 hive-action-1.0
HiveServer2	hive2-action-0.1 hive2-action-0.2 hive2-action-1.0
oozie-bundle	oozie-bundle-0.1 oozie-bundle-0.2
oozie-coordinator	oozie-coordinator-0.1 oozie-coordinator-0.2 oozie-coordinator-0.3 oozie-coordinator-0.4 oozie-coordinator-0.5

Schema	CDP
oozie-sla	oozie-sla-0.1 oozie-sla-0.2
oozie-common	oozie-common-1.0
oozie-workflow	oozie-workflow-0.1 oozie-workflow-0.2 oozie-workflow-0.2.5 oozie-workflow-0.3 oozie-workflow-0.4 oozie-workflow-0.4.5 oozie-workflow-0.5 oozie-workflow-1.0
shell	shell-action-0.1 shell-action-0.2 shell-action-0.3 shell-action-1.0
spark	spark-action-0.1 spark-action-0.2 spark-action-1.0
sqoop	sqoop-action-0.2 sqoop-action-0.3 sqoop-action-0.4 sqoop-action-1.0
ssh	ssh-action-0.1 ssh-action-0.2

Enabling the Oozie web console on managed clusters

You must extract the ext-2.2 libraries to your Oozie server host and enable the Oozie web console.

Procedure

1. Download ext-2.2.
2. Extract the contents of the file to /var/lib/oozie on the same host as the Oozie Server.
   After extraction, the content of the directories is as follows:

```
ls -ltr /var/lib/oozie/
```

```
total 984
drwxr-xr-x 9 oozie   oozie   4096 Aug  4  2008 ext-2.2
-rw-r--r-- 1 systest root  999635 Jan 23 23:24 mysql-connector-java.jar
```

```
ls -ltr /var/lib/oozie/ext-2.2/
```

```
total 1752
```

```
-rw-r--r--  1 oozie oozie    893 Feb 24  2008 INCLUDE_ORDER.txt
drwxr-xr-x 33 oozie oozie   4096 Aug  4  2008 examples
drwxr-xr-x  4 oozie oozie     49 Aug  4  2008 resources
drwxr-xr-x 10 oozie oozie    148 Aug  4  2008 source
drwxr-xr-x 10 oozie oozie    120 Aug  4  2008 build
-rw-r--r--  1 oozie oozie  87524 Aug  4  2008 ext-core.js
-rw-r--r--  1 oozie oozie 163794 Aug  4  2008 ext-core-debug.js
-rw-r--r--  1 oozie oozie 974145 Aug  4  2008 ext-all-debug.js
drwxr-xr-x  6 oozie oozie     55 Aug  4  2008 adapter
-rw-r--r--  1 oozie oozie  11548 Aug  4  2008 CHANGES.html
-rw-r--r--  1 oozie oozie 538956 Aug  4  2008 ext-all.js
-rw-r--r--  1 oozie oozie   1513 Aug  4  2008 license.txt
drwxr-xr-x  4 oozie oozie    108 Aug  4  2008 docs
drwxr-xr-x  5 oozie oozie     94 Jan 24 15:49 air
```

For example:

```
unzip ext-2.2.zip -d /var/lib/oozie
chown -R oozie:oozie /var/lib/oozie/ext-2.2
```

3. In Cloudera Manager Admin Console, go to the Oozie service.
4. Restart the Oozie service.

Enabling Oozie SLA with Cloudera Manager

You can use Oozie to define SLA limits for critical applications and actively monitor these jobs.

Procedure

1. In the Cloudera Manager Admin Console, go to the Oozie service.
2. Click the Configuration tab.
3. Locate the Enable SLA Integration property or search for it by typing its name in the Search box.
4. Select Enable SLA Integration. This sets the required values for oozie.services.ext and oozie.service.EventHandlerService.event.listeners in oozie-site.xml.
5. Enter a Reason for change, and then click Save Changes to commit the changes.
6. Restart the Oozie service.

What to do next

The following properties are set by default when you enable Oozie SLA in Cloudera Manager. You do not have to explicitly define them, unless you want to modify any of these parameters:

```
    oozie.service.SchemaService.wf.schemas
    oozie.service.SchemaService.coord.schemas
    oozie.service.SchemaService.sla.schemas
    oozie.service.ELService.groups
    oozie.service.ELService.constants.wf-sla-submit
    oozie.service.ELService.ext.constants.coord-sla-create
    oozie.service.ELService.functions.coord-sla-create
    oozie.service.ELService.constants.coord-sla-submit
    oozie.service.ELService.functions.coord-sla-submit
    oozie.service.EventHandlerService.filter.app.types
    oozie.service.EventHandlerService.event.queue
    oozie.service.EventHandlerService.queue.size
    oozie.service.EventHandlerService.worker.interval
    oozie.service.EventHandlerService.batch.size
    oozie.service.EventHandlerService.worker.threads
```

```
oozie.sla.service.SLAService.alert.events
oozie.sla.service.SLAService.capacity
oozie.sla.service.SLAService.calculator.impl
oozie.sla.service.SLAService.job.event.latency
oozie.sla.service.SLAService.check.interval
```

For oozie.sla.service.SLAService.alert.events, only END_MISS is configured by default. To change the alert events, explicitly set END_MISS, START_MISS, or DURATION_MISS, in Oozie Server Advanced Configuration Snippet (Safety Valve) for oozie-site.xml.

Disabling Oozie UI using Cloudera Manager

From the Cloudera 7.1.7 SP1 release onwards, the Oozie UI is enabled by default. You can disable it by setting a new property in the Oozie site configuration.

Procedure

1. In the Cloudera Manager Admin Console, go to the Oozie service.
2. Click the Configuration tab.
3. Locate the Oozie Server Advanced Configuration Snippet (Safety Valve) for oozie-site.xml property or search for it by typing its name in the Search box.
4. Add the following property:

   ```
   Name: oozie.ui.enabled
   Value: false
   ```

5. Enter a Reason for change, and then click Save Changes to commit the changes.
6. Restart the Oozie service.

   To enable the Oozie UI again, delete the oozie.ui.enabled property set using the safety valve.

Moving the Oozie service to a different host

To move the Oozie service to a new host, do the following:

Procedure

1. Stop the current Oozie service.
2. Add the Oozie service to the desired new host using Add Service wizard in Cloudera Manager. The wizard configures and starts Oozie and its dependent services.
3. Add the role specific configurations for the previous host to the new host, if any.
4. Restart the Oozie service.

Oozie database configurations

You can use Cloudera Manager to configure data purge settings, loading, and dumping the Oozie database. Depending on the database that you are using with Oozie, you can set the timezone for the database.

Related Information

Configuring an external database for Oozie

Configuring Oozie data purge settings using Cloudera Manager

You can change your Oozie configuration to control when data is purged to improve performance, reduce database disk usage, or keep the history for a longer period of time. Limiting the size of the Oozie database can also improve performance during upgrades.

About this task

All Oozie workflows older than 30 days are purged from the database by default. However, actions associated with long-running coordinators do not purge until the coordinators complete. If, for example, you schedule a coordinator to run for a year, all those actions remain in the database for the year.

Procedure

1. In the Cloudera Manager Admin Console, go to the Oozie service.
2. Click the Configuration tab.
3. Type purge in the Search box.
4. Set the following properties as required for your environment:
   - Enable Purge for Long-Running Coordinator Jobs

     Select this property to enable purging of long-running coordinator jobs for which the workflow jobs are older than the value you set for the Days to Keep Completed Workflow Jobs property.
   - Days to Keep Completed Workflow Jobs
   - Days to Keep Completed Coordinator Jobs
   - Days to Keep Completed Bundle Jobs
5. Enter a Reason for change, and then click Save Changes to commit the changes.
6. Select Actions Restart to restart the Oozie Service.

Loading the Oozie database

You must configure the database in which to load your Oozie data, create the required database tables, and then load the Oozie database.

Procedure

1. Stop the Oozie server (in HA mode, stop all Oozie servers).
2. Install and configure the empty database in which to load your Oozie data.

   The db.version of the database must match the db.version of the dump file.
3. Select Actions Create Oozie Database Tables .

   Confirm you want to create the database tables by clicking Create Oozie Database Tables.
4. Verify *Database Dump File* is set correctly.
   a) In the Cloudera Manager Admin Console, click the Oozie service.
   b) Go to the Configuration page.
   c) Select Scope Oozie Server .
   d) Select Category Database .
5. Select Actions Load Database .

   Confirm you want to dump the database to the specified location by clicking Load Database.
6. Select Actions Start .

   Confirm you want to start the service by clicking Start.

Dumping the Oozie database

You must stop the Oozie server, specify the location to which you want to dump the Oozie database, and then perform the dumping operation.

Procedure

1. Stop the Oozie server (in HA mode, stop all Oozie servers).
2. In the Cloudera Manager Admin Console, go to the Oozie service status page.
3. Select Actions Stop .

   Confirm you want to stop the service by clicking Stop.
4. Specify *Database Dump File*.
   a) Go to the Configuration page.
   b) Select Scope Oozie Server .
   c) Select Category Database .
   d) Set a file location for the Database Dump File.
5. Select Actions Dump Database .

   Confirm that you want to dump the database to the specified location by clicking Dump Database.

   During the export process, Cloudera Manager fetches and writes the database content a compressed zip specified by the *Database Dump File* property.

Setting the Oozie database timezone

Depending on the type of database you are using with Oozie, you must configure specific properties for setting the database timezone.

Cloudera recommends that you set the timezone in the Oozie database to GMT. Databases do not handle Daylight Saving Time (DST) shifts correctly. There might be problems if you run any Coordinators with actions scheduled to materialize during the one-hour period that gets lost in DST.

Important: Changing the timezone on an existing Oozie database while Coordinators are already running might cause Coordinators to shift by the offset of their timezone from GMT one time after you make this change.

For more information about how to set your database's timezone, see your database's documentation.

Fine-tuning Oozie's database connection

Learn how you can configure Oozie to use its database well.

About this task

When it comes to configuring database connections, simply providing a hostname, port, username, and password may not be sufficient. In order to optimize Oozie's database connection, you might need to manually construct lengthy connection and configuration strings using safety-valve settings. To simplify this process and enable finer control over Oozie's database connection, you can use several enhancements, as described in this section.

The following properties allow you to configure how Oozie uses its database:

- oozie_database_connection_properties

  You can use this property to directly configure the database connection. For example, if you need to pass a trustStore path to the connection, you can add a property named javax.net.ssl.trustStore and with a value to your trustStore file.

- oozie_datasource_properties

  You can use this property to configure the datasource object created by OpenJPA. For example, if you would like to finetune how many idle connections the datasource instance should keep, you can set the maxIdle property.

  Note: These configurations do not configure the underlying database connection, but the commons-dbcp datasource instance. For more details, see BasicDataSource Configuration Parameters.

With the help of the values set for both the properties, Cloudera Manager assembles one single configuration which will be set in oozie-site.xml. The name of this single configuration is, oozie.service.JPAService.connection.properties, and it is assembled in the following way:

- Whatever you specify in oozie_database_connection_properties, is concatenated with a semicolon, and available under the ConnectionProperties property in the final configuration.
- Whatever you specify in oozie_datasource_properties, is directly set in the final configuration.

For example, if you set the following properties in oozie_database_connection_properties, because your database JDBC driver accepts them,

- javax.net.ssl.trustStore=/path/to/trustStore.jks
- ssl.enabled=true

And you set the following properties in oozie_datasource_properties:

- MaxIdle=10
- DefaultQueryTimeout=120

Then the final value of oozie.service.JPAService.connection.properties will be:

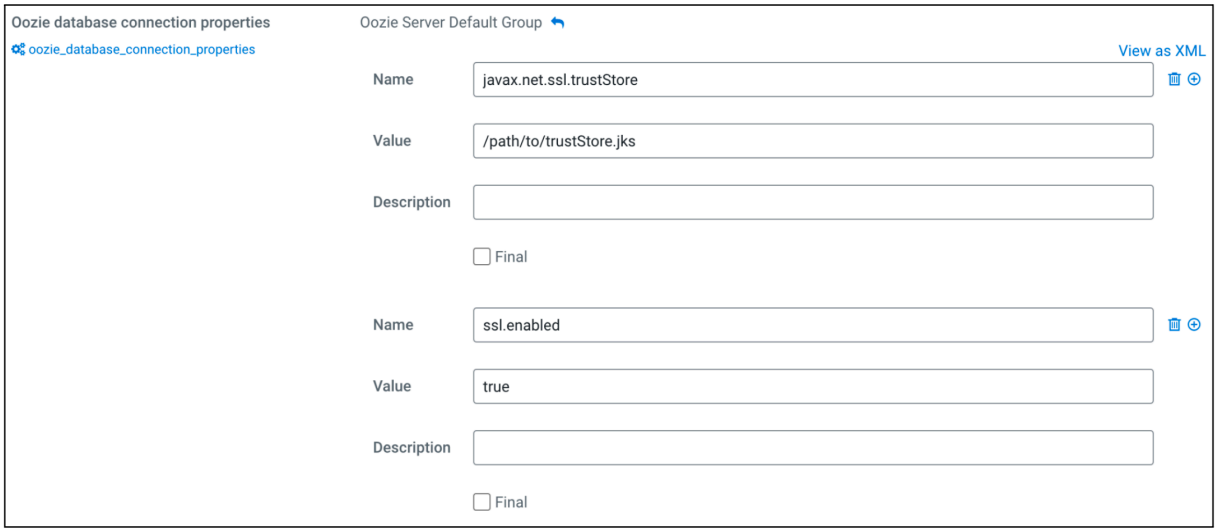
```
ConnectionProperties="javax.net.ssl.trustStore=/path/to/trustStore.jks;ssl.e
nabled=true", MaxIdle=10, DefaultQueryTimeout=120
```

Important: Kindly be aware that the properties mentioned above (such as javax.net.ssl.trustStore, ssl.enabled, and so on) are provided merely as an example. The specific properties that need configuration should align with the JDBC driver you use to connect to the underlying Oozie database.

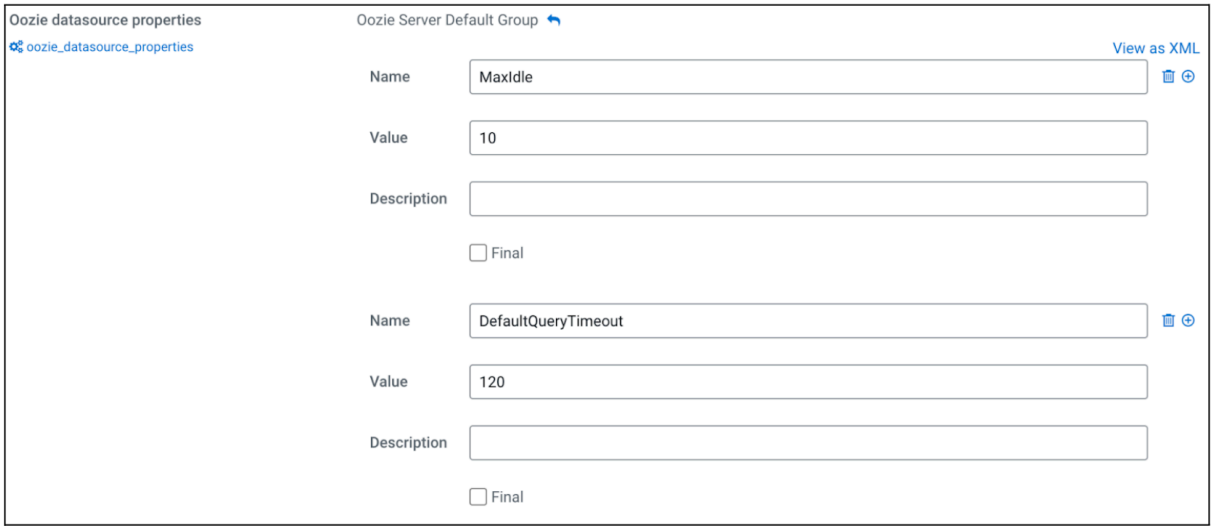
Perform the following steps after you configure the correct properties:

Procedure

1. In Cloudera Manager, click the Oozie service.
2. Click the Configuration tab.
3. Search for and set the oozie_database_connection_properties property.



4. Search for and set the oozie_datasource_properties property.



5. Click Save Changes, and allow Cloudera Manager some time to recognize the changes.
6. Follow the instructions provided by Cloudera Manager to redeploy Oozie.

Assembling a secure JDBC URL for Oozie

Learn how to assemble a secure JDBC URL for Oozie.

Procedure

1. In Cloudera Manager, click the Oozie service.
2. Click the Configuration tab.
3. Search and enable the oozie_database_is_secure property.

   If you enable this property, then Cloudera Manager assembles a secure JDBC URL for Oozie based on the selected database types.
4. Click Save Changes, and allow Cloudera Manager some time to recognize the changes.
5. Follow the instructions provided by Cloudera Manager to redeploy Oozie.

Oracle TCPS

Learn how to establish a connection to a secure Oracle database.

If the oozie_database_is_secure property is enabled, then the JDBC URL generated for Oracle looks like:

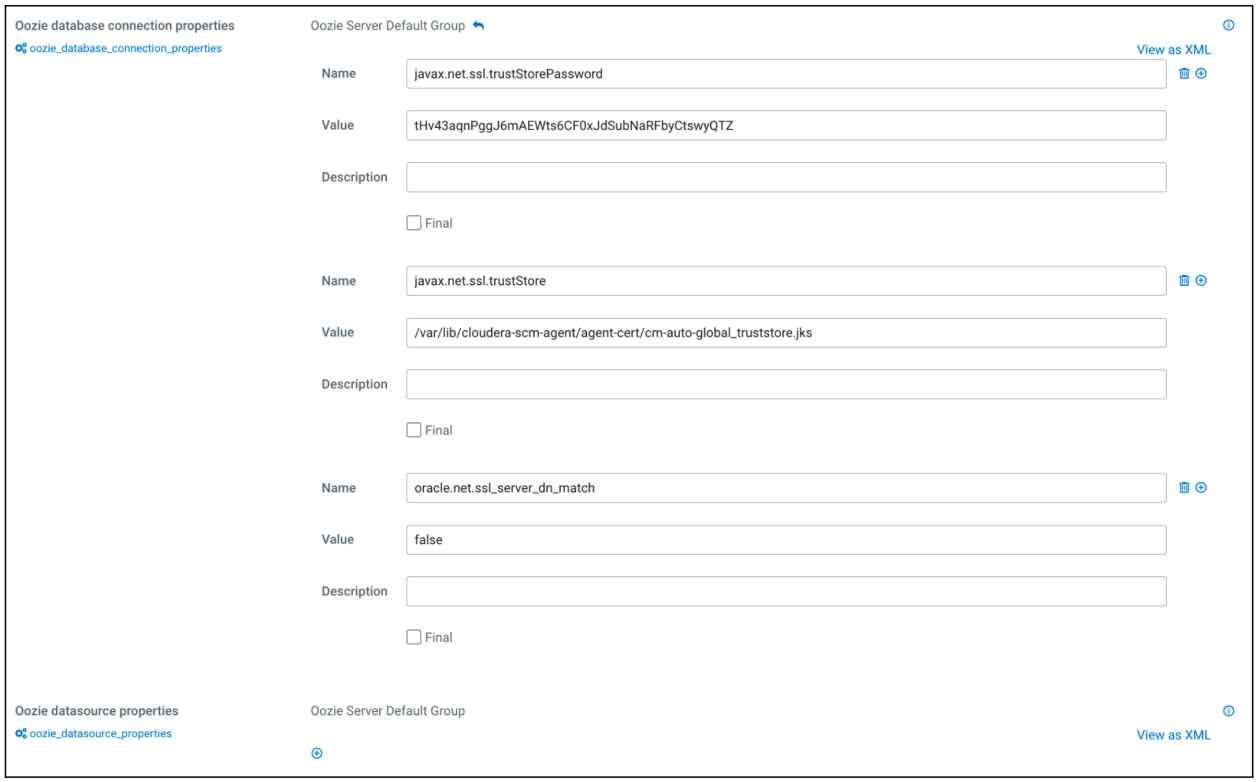
```
jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS=(PROTOCOL=tcps)(HOST=<HOST>)(PORT=<P
ORT>))(CONNECT_DATA=(SERVICE_NAME=<DB_NAME>)))
```

The values of <HOST>, <PORT>, and <DB_NAME> come from the following Oozie properties set in Cloudera Manager:

- <HOST> and <PORT> – oozie_database_host
- <DB_NAME> – oozie_database_name

While the above specifies the protocol as TCPS, this JDBC URL on its own might not be enough to establish a connection to a secure Oracle database. If you need to specify a trustStore, keyStore, or Oracle wallet properties for the connection, please use the oozie_database_connection_properties and/or oozie_datasource_properties properties.

For example,



Prerequisites for configuring TLS/SSL for Oozie

There are certain prerequisites that must be fulfilled for configuring TLS/SSL for Oozie.

- Keystores for Oozie must be readable by the oozie user. This can be a copy of the Hadoop services' keystore with permissions set to 0440 and owned by the oozie group.
- Truststores must have permissions set to 0444, which means that all users can read them.
- Specify absolute paths to the keystore and truststore files. These settings apply to all hosts on which daemon roles of the Oozie service run so the paths you choose must be valid on all hosts.
- If there is a DataNode and an Oozie server running on the same host, they can use the same certificate.

Configure TLS/SSL for Oozie

You can edit properties to enable TLS/SSL for Oozie, specify the keystore file location on the local file system, and set the password for the keystore.

Procedure

1. In Cloudera Manager, select the Oozie service.
2. Click the Configuration tab.
3. In the Search field, type TLS/SSL to show the Oozie TLS/SSL properties.

4. Edit the following TLS/SSL properties according to your cluster configuration.

Property	Description
Enable TLS/SSL for Oozie	Check this field to enable TLS/SSL for Oozie.
Oozie TLS/SSL Server JKS Keystore File Location	Path to the keystore file on the local file system.
Oozie TLS/SSL Server JKS Keystore File Password	Password for the keystore file.
Oozie TLS/SSL Client Trust Store File	Path to the client truststore file.
Oozie TLS/SSL Client Trust Store Password	Password for the truststore file.

5. If SSL is enabled for ZooKeeper, edit the following SSL properties:

Property	Description
Oozie ZooKeeper TLS/SSL Server JKS Keystore File Location	Path to the keystore file.
Oozie ZooKeeper TLS/SSL Server JKS Keystore File Password	Password for the keystore file.
Oozie ZooKeeper TLS/SSL Client Trust Store File	Path to the client truststore file.
Oozie ZooKeeper TLS/SSL Client Trust Store Password	Password for the truststore file.

6. Optionally, you can modify the values of the following properties:
   - Enabled TLS Protocols - List of Cipher Suite names that should be excluded.
   - Excluded Cipher Suites - TLS protocols accepted by the Oozie Server.
7. Click Save Changes.
8. Restart the Oozie service.

Related Information

Oozie security enhancements

Oozie Java-based actions with Java 17

In relation to Java 17, certain applications might require reflective access to internal Java classes, packages, or modules. This section describes how to gain reflective access to these with Oozie.

To enable reflective access, use the --add-opens Java parameter. If your applications executed with Oozie need reflective access, you can use the following properties to specify the required --add-opens arguments:

- oozie.launcher.yarn.app.mapreduce.am.command-opts
- oozie.launcher.mapreduce.map.java.opts
- oozie.launcher.mapred.child.java.opts

For example, in your workflow's configuration, you can set the property as follows:

```
<property>
    <name>oozie.launcher.yarn.app.mapreduce.am.command-opts</name>
    <value>--add-opens=java.base/java.lang=ALL-UNNAMED</value>
</property>
```

The Spark and Spark 3 applications inherently require reflective access. Consequently, Oozie automatically adds the following --add-opens properties by default for Spark and Spark 3 actions:

```
--add-opens=java.base/java.io=ALL-UNNAMED
--add-opens=java.base/java.lang.invoke=ALL-UNNAMED
```

```
--add-opens=java.base/java.lang=ALL-UNNAMED
--add-opens=java.base/java.net=ALL-UNNAMED
--add-opens=java.base/java.nio=ALL-UNNAMED
--add-opens=java.base/java.util.concurrent=ALL-UNNAMED
--add-opens=java.base/java.util=ALL-UNNAMED
--add-opens=java.base/sun.nio.ch=ALL-UNNAMED
--add-opens=java.base/sun.nio.cs=ALL-UNNAMED
--add-opens=java.base/sun.security.action=ALL-UNNAMED
```

It is important to note that this behavior can be overridden for Spark actions by using the oozie.action.spark.launcher.jdk17.opens property, and for Spark3 actions by using the oozie.action.spark3.launcher.jdk17.opens property. By specifying Java modules, packages, or classes in a comma-separated list with these properties, Oozie can configure the appropriate access permissions when launching a Spark or Spark 3 application. Each item in the list should be in a format compatible with the Java --add-opens parameter, such as java.base/java.lang=ALL-UNNAMED.

The following comma-separated list shows the default values of these properties. These are derived from the default values in Spark and Spark 3:

```
java.base/java.io=ALL-UNNAMED,java.base/java.lang.invoke=ALL-UNNAMED,java.base/java.lang=ALL-UNNAMED,java.base/java.net=ALL-UNNAMED,java.base/java.nio=ALL-UNNAMED,java.base/java.util.concurrent=ALL-UNNAMED,java.base/java.util=ALL-UNNAMED,java.base/sun.nio.ch=ALL-UNNAMED,java.base/sun.nio.cs=ALL-UNNAMED,java.base/sun.security.action=ALL-UNNAMED
```

However, you have the flexibility to customize the values of these properties by adding a safety-valve setting for the corresponding property on the Oozie configuration page in Cloudera Manager by using the oozie_config_safety_valve setting.

For other Oozie Java-based actions (simple Java action, Distcp, Git, Map-Reduce, Hive2, Shell, Sqoop), Oozie does not add any --add-opens values by default. Nevertheless, you have the option to specify custom --add-opens overrides for each action using the corresponding properties:

- oozie.action.distcp.launcher.jdk17.opens
- oozie.action.git.launcher.jdk17.opens
- oozie.action.hive2.launcher.jdk17.opens
- oozie.action.java.launcher.jdk17.opens
- oozie.action.map-reduce.launcher.jdk17.opens
- oozie.action.shell.launcher.jdk17.opens
- oozie.action.sqoop.launcher.jdk17.opens

These properties can be specified globally through a safety-valve configuration, in the job.properties file, in the workflow's global configuration, or inside the workflow as a specific action's configuration. The order of precedence for these configurations is as follows:

1. If you have configured the property at the action level, it takes precedence over all other settings, and the remaining configurations are disregarded.
2. If you have configured the property in the global configuration of the workflow, the value from there is used.
3. If the setting is not available in either of the previous locations, the value configured in your job.properties file is used instead.
4. Lastly, the global setting in Cloudera Manager comes into effect.

Note: These properties only have an effect on the Launcher application created by Oozie.

Oozie security enhancements

Learn about Oozie security enhancements related to callback, callback endpoint, FIPS compliance, and SMTP (Simple Mail Transfer Protocol). Oozie will be notified about completion of tasks through HTTPS.

Callback

- Prior to this enhancement, even though SSL was enabled for Oozie, the callback mechanism – which notifies the Oozie server after an action finished (success/failed) – was going through HTTP. With the enhanced callback feature if TLS/SSL is enabled for Oozie, the callback invocation goes through HTTPS. This applies to all Oozie actions, including map-reduce actions. For map-reduce actions, as always, the Oozie Application Master (AM) container does not wait for the map-reduce Job to complete, but YARN makes a callback to Oozie when the map-reduce Job completes. This callback goes through HTTPS as well when TLS/SSL is enabled for Oozie. When TLS/SSL is enabled for Oozie, Oozie listens only on the HTTPS port and not on the HTTP port as the HTTP port was only needed for the callback mechanism. Oozie will `not` explicitly upload the truststore file required for the HTTPS connection to the YARN applications launched by Oozie and neither should you, but Oozie will pass the location of the file used by Oozie itself to the callback mechanism running inside the YARN container. Hence, the truststore file used by Oozie needs to be available on all NodeManager Hosts and accessible by YARN containers.

  Note: Until now, the default callback command for SSH actions was `curl`. If you have enabled TLS/SSL for Oozie, Cloudera Manager will change this to `curl -k`. If you have added a custom callback command setup for SSH actions through a safety valve, that setup will not be overridden by Cloudera Manager. You must make sure that your command supports TLS/SSL.

Callback Endpoint

- Along with the callback mechanism, you can also enable authentication for the callback endpoint. If you have Kerberos configured on your cluster, authentication is enabled for all endpoints of Oozie by default except for the callback endpoint. You can enable authentication for the callback endpoint by setting the oozie.servlet.CallbackServlet.authentication.required property to `true` as a safety-valve in Cloudera Manager.

  Note: After the release of Cloudera Manager 7.3.0, you do not need to configure the callback endpoint authentication through a safety-valve because Cloudera is introducing the Oozie Callback Servlet Authentication property. After the release of Cloudera Manager 7.3.0, upgrade Cloudera Manager, and search and select the new Oozie Callback Servlet Authentication option. If you have set the above property using safety-valve, you can remove it and instead enable it through the new checkbox. No new configuration is required in Cloudera Runtime. When callback authentication is enabled, Oozie does not allow an unauthenticated invocation to the endpoint. Before starting the AM container, Oozie generates a new type of delegation token and when the Job finishes and the AM container notifies the Oozie server. This new Oozie delegation token is used to make the callback.

  Note: If you enable authentication on the callback endpoint, when you are executing an SSH action, make sure your SSH command will create a Kerberized environment. Otherwise, the callback will fail.

FIPS Compliance

To make Oozie FIPS compliant, the following changes are introduced:

- When TLS/SSL is enabled for Oozie, apart from setting the trustStore, trustStorePassword, keyStore, and keyStorePassword properties, Cloudera Manager adds two new properties `oozie.https.truststore.type` and `oozie.https.keystore.type` in the oozie-site.xml file. These properties will contain the value of the globally configured keyStore type in Cloudera Manager.
- When TLS/SSL is enabled for ZooKeeper and Oozie runs with High-Availability, Cloudera Manager sets the `oozie.zookeeper.https.truststore.type` and `oozie.zookeeper.https.keystore.type` properties along with the existing `oozie.zookeeper.https.truststore/keystore.file/password` property in the oozie-site.xml file.

SMTP

To configure custom TLS/SSL protocols when executing an email action, add the new `oozie.email.smtp.ssl.protocols` property using a safety valve in Cloudera Manager.

Related Information

Configure TLS/SSL for Oozie
Installing and Configuring CDP with FIPS

Additional considerations when configuring TLS/SSL for Oozie HA

To enable clients to connect to Oozie servers (the target servers) through the load balancer using TLS/SSL, configure the load balancer for TLS/SSL pass-through.

This means that the load balancer does not perform encryption or decryption but instead passes traffic from clients and servers to the appropriate target host. See the documentation for your load balancer for details.

Related Information

Configuring Oozie to use HDFS HA

Configure Oozie client when TLS/SSL is enabled

You must configure the Oozie client if TLS/SSL is enabled in your cluster. You can configure the Oozie command line client using either the JDK certificate store or using the trust-store file.

Procedure

Using JDK Certificate Store

- Import the certificate into the JDK certificate store. For example,

```
keytool -keystore </usr/java/default/lib/security/cacerts> -import -trus
tcacerts -alias autotls -file </opt/cloudera/CMCA/trust-store/cm-auto-gl
obal_cacerts.pem>  --storepass changeit -noprompt
```

  You must specify the JDK/JRE certificate file location with the -keystore parameter and the certificate you want to import with the -file parameter.

Using Trust Store

- Manually specify the trust-store and trust-store password for the Oozie command line client. For example,

```
oozie -Djavax.net.ssl.trustStore={trustStoreFile} -Djavax.net.ssl.trustS
torePassword={trustStorePassword} jobs -oozie https://{oozieHost}:{oozie
Port}/oozie
```

Using insecure SSL connnection

- From the Cloudera Runtime 7.1.7 SP1 release onwards, you can manually set the SSL connection to insecure. For example,

```
oozie jobs -oozie https://{oozieHost}:{ooziePort}/oozie -insecure
```

  This causes Oozie to allow certificate errors while the data remains encrypted. With this, there is no need to import the certificate into the JDK certificate store or specify the trust-store and trust-store password manually for the Oozie command line client.

Configuring custom Kerberos principal for Oozie

The Kerberos principal for Ozone is configured by default to use the same service principal as the default process user. However, you can change the default setting by providing a custom principal in Cloudera Manager.

Procedure

1. In Cloudera Manager, click Clusters > Oozie .
2. Go to the Configuration tab
3. Search for the Kerberos Principal by entering "kerberos" in the search field.
4. For Kerberos Principal, enter your custom principal value.
5. Click Save Changes.
6. Click Actions and select Restart to restart the service.