



BIG DATA WORKFLOW SCHEDULING

Bin Jiang

04/22/2017

Scheduling Strategies

- Timer Driven
- CRON Driven
- Event Driven
- Email Driven
- Message Driven
- HTTP Driven
- Web Socket Driven
- Job Driven
- Dataset Driven
- Files Driven

Scheduling Frameworks

- Apache Oozie
- Apache NiFi
- AirFlow
- UNIX Crontab
- Quartz scheduler



UNIX CRONTAB

Bin Jiang

04/22/2017

What is cron

- Cron is a daemon which runs at the times of system boot from /etc/init.d scripts. If needed it can be stopped/started/restart using init script or with command `service crond start` in Linux systems.
- Crontab (CRON TABLE) is a file which contains the schedule of cron entries to be run and at specified times. File location varies by operating systems, See Crontab file location at the end of this document.

Crontab Restrictions

- Jobs Management
- Monitoring
- Jobs Dependencies
- Only support timer driven

Crontab Commands

- `crontab -e` Edit crontab file, or create one if it doesn't already exist
- `crontab -l` crontab list of cronjobs , display crontab file contents
- `crontab -r` Remove your crontab file
- `crontab -v` Display the last time you edited your crontab file

Crontab syntax

```
* * * * *      command to be executed
- - - - -
| | | | |
| | | | +----- day of week (0 - 6) (Sunday=0)
| | | +----- month (1 - 12)
| | +----- day of          month (1 - 31)
| +----- hour (0 - 23)
+----- min (0 - 59)
```


Crontab examples

min	hour	day/month	month	day/week	Execution time
30	0	1	1,6,12	*	— 00:30 Hrs on 1st of Jan, June & Dec.
0	20	*	10	1-5	—8.00 PM every weekday (Mon-Fri) only in Oct.
0	0	1,10,15	*	*	— midnight on 1st ,10th & 15th of month
5,10	0	10	*	1	— At 12.05,12.10 every Monday & on 10th of every month



APACHE NIFI SCHEDULING STRATEGIES

Bin Jiang

04/22/2017

NiFi – Scheduling Strategies

- Time Driven

Configure Processor

SETTINGS

SCHEDULING

PROPERTIES

COMMENTS

Scheduling Strategy ?

Timer driven

Timer driven

CRON driven

Run Schedule ?

0 sec

Run Duration ?

0ms

25ms

50ms

100ms

250ms

500ms

1s

2s

Lower latency

Higher throughput

CANCEL

APPLY

NiFi – Scheduling Strategies

- CRON Driven

Configure Processor

SETTINGS

SCHEDULING

PROPERTIES

COMMENTS

Scheduling Strategy ?

CRON driven ▼

Concurrent Tasks ?

1

Run Duration ?

0ms 25ms 50ms 100ms 250ms 500ms 1s 2s

Lower latency Higher throughput

Run Schedule ?

*****?

CANCEL

APPLY

NiFi – Scheduling Strategies

- CRON Driven

The CRON driven scheduling value is a string of six required fields and one optional field, each separated by a space. These fields are

NiFi – Scheduling Strategies

- CRON Driven

Field	Valid values
Seconds	0-59
Minutes	0-59
Hours	0-23
Day of Month	1-31
Month	1-12 or JAN-DEC
Day of Week	1-7 or SUN-SAT
Year (optional)	empty, 1970-2099

NiFi – Scheduling Strategies

- CRON Driven

Use one of the following ways to specify values:

- Number: Specify one or more valid value. You can enter more than one value using a comma-separated list.
- Range: Specify a range using the <number>-<number> syntax.
- Increment: Specify an increment using <start value>/<increment> syntax. For example, in the Minutes field, 0/15 indicates the minutes 0, 15, 30, and 45.

NiFi – Scheduling Strategies

- CRON Driven

Quartz documentation

<http://www.quartz-scheduler.org/documentation/quartz-2.x/tutorials/crontrigger.html>

NiFi – Scheduling Strategies

- Event Driven

The Processor will be triggered to run by an event, and that event occurs when FlowFiles enter Connections feeding this Processor

NiFi – Scheduling Strategies

- Email Driven

Displaying 8 of 224

Email

Type ▲	Version	Tags
ConsumeEWS	1.2.0.3.0.2.0-76	EWS, Exchange, Email, Consum...
ConsumeIMAP	1.2.0.3.0.2.0-76	Imap, Email, Consume, Ingest, ...
ConsumePOP3	1.2.0.3.0.2.0-76	Email, Consume, Ingest, Messa...
ExtractEmailAttachments	1.2.0.3.0.2.0-76	split, email
ExtractEmailHeaders	1.2.0.3.0.2.0-76	split, email
ExtractTNEFAttachments	1.2.0.3.0.2.0-76	split, email
ListenSMTP	1.2.0.3.0.2.0-76	smtp, listen, email
PutEmail	1.2.0.3.0.2.0-76	smtp, email, put, notify

NiFi – Scheduling Strategies

- Email Driven

Configure Processor

SETTINGS

SCHEDULING

PROPERTIES

COMMENTS

Required field



Property		Value	
User Name	?	No value set	
Password	?	No value set	
Folder	?	INBOX	
Fetch Size	?	10	
Delete Messages	?	false	
Connection timeout	?	30 sec	
Exchange Version	?	Exchange2010_SP2	
EWS URL	?	No value set	
Auto Discover URL	?	true	
Mark Messages as Read	?	true	
Original Headers to Include	?	Empty string set	
Original Headers to Exclude	?	Empty string set	

NiFi – Scheduling Strategies

- Web Socket Driven

Configure Processor

SETTINGS	SCHEDULING	PROPERTIES	COMMENTS
Name ConnectWebSocket		<input checked="" type="checkbox"/> Enabled	Automatically Terminate Relationships ?
Id 037241dc-11b2-1175-f01f-11e86a23b9c6			<input type="checkbox"/> binary message The WebSocket binary message output
Type ConnectWebSocket 1.2.0.3.0.2.0-76			<input type="checkbox"/> connected The WebSocket session is established
Bundle org.apache.nifi - nifi-websocket-processors-nar			<input type="checkbox"/> text message The WebSocket text message output
Penalty Duration ? 30 sec	Yield Duration ? 1 sec		
Bulletin Level ? WARN ▼			

NiFi – Scheduling Strategies

- HTTP Driven

Configure Processor

SETTINGS

SCHEDULING

PROPERTIES

COMMENTS

Name

ListenHTTP

☒ Enabled

Automatically Terminate Relationships [?](#)

☐ success

Relationship for successfully received FlowFiles

Id

037241de-11b2-1175-abe2-ce2a9fea5ca3

Type

ListenHTTP 1.2.0.3.0.2.0-76

Bundle

org.apache.nifi - nifi-standard-nar

Penalty Duration [?](#)

30 sec

Yield Duration [?](#)

1 sec

Bulletin Level [?](#)

WARN



NiFi – Scheduling Strategies

- Message Driven



Displaying 227 of 227

Type ▲	Version	Tags
ConsumeEWS	1.2.0.3.0.2.0-76	EWS, Exchange, Email, Consume...
ConsumeIMAP	1.2.0.3.0.2.0-76	Imap, Email, Consume, Ingest, ...
ConsumeJMS	1.2.0.3.0.2.0-76	jms, receive, get, consume, me...
ConsumeKafka	1.2.0.3.0.2.0-76	PubSub, Consume, Ingest, Get,...
ConsumeKafkaRecord_0_10	1.2.0.3.0.2.0-76	0.10.x, PubSub, Consume, Inge...
ConsumeKafka_0_10	1.2.0.3.0.2.0-76	0.10.x, PubSub, Consume, Inge...
ConsumeMQTT	1.2.0.3.0.2.0-76	MQTT, subscribe, consume, lis...
ConsumePOP3	1.2.0.3.0.2.0-76	Email, Consume, Ingest, Mess...
ConsumeWindowsEventLog	1.2.0.3.0.2.0-76	event, windows, ingest
ControlRate	1.2.0.3.0.2.0-76	throttle, rate, rate control, throu...
ConvertAvroSchema	1.2.0.3.0.2.0-76	convert, kite, avro
ConvertAvroToJSON	1.2.0.3.0.2.0-76	json, convert, avro

NiFi – Scheduling Strategies

- Signal Driven

	 Wait Wait 1.2.0.3.0.2.0-76 org.apache.nifi - nifi-standard-nar	
In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	0 / 00:00:00.000	5 min

	 Notify Notify 1.2.0.3.0.2.0-76 org.apache.nifi - nifi-standard-nar	
In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	0 / 00:00:00.000	5 min



APACHE OOZIE

BIG DATA WORKFLOW SCHEDULER

Bin Jiang

04/22/2017

What's Oozie

- Oozie is a workflow scheduler system that orchestrates Hadoop MapReduce (MR) and other jobs, as well as manages job inter-dependencies
- It is written in Java as a Web-Application running in a Java Web Server Servlet Container (embedded servlet container, Tomcat etc.)
- Oozie is distributed under the Apache License

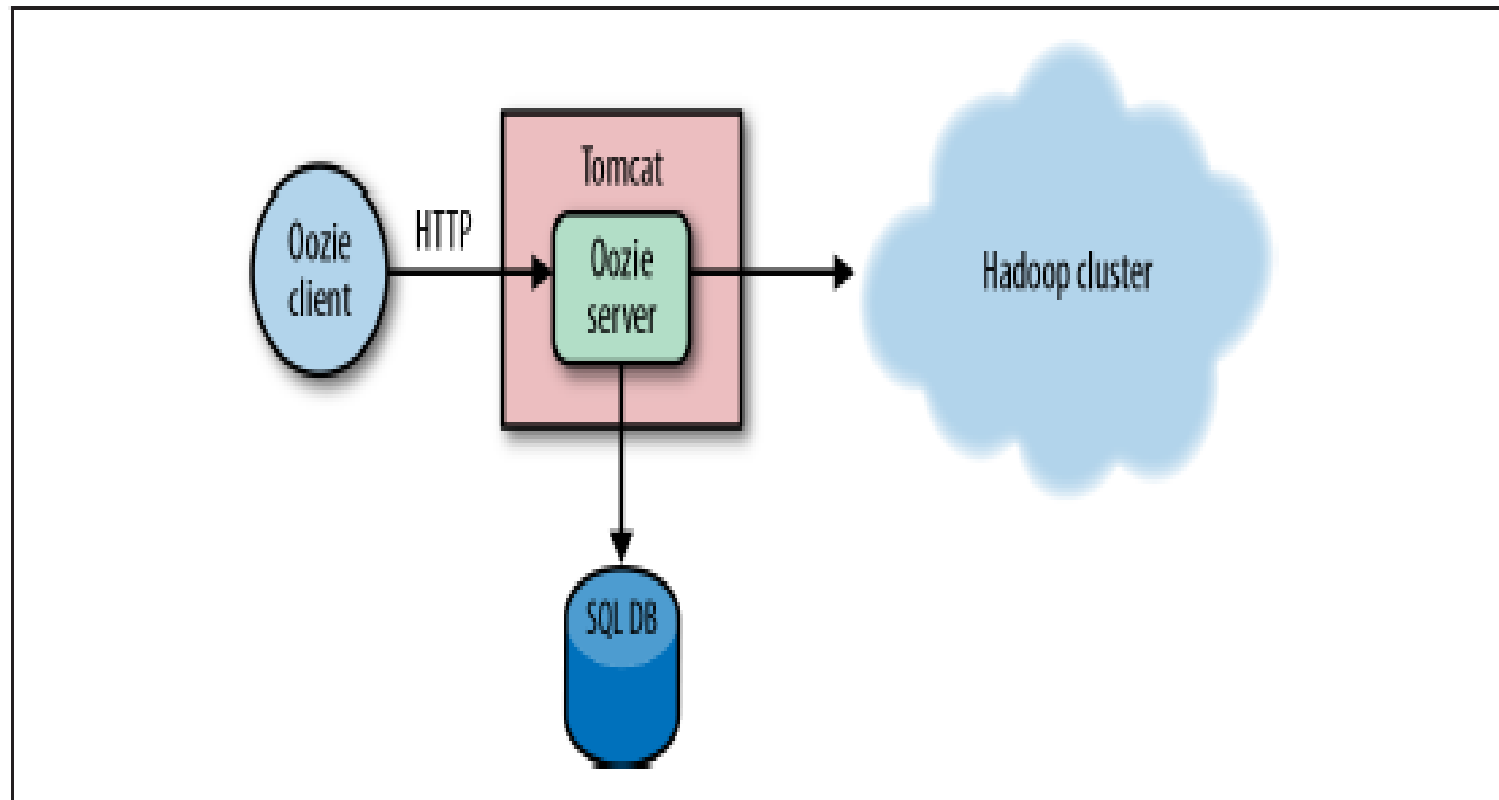
Directed Acyclic Graph

- Oozie sees a workflow as a collection of actions
- Actions are arranged in a Directed Acyclic Graph (DAG)
 - ❑ DAG prevents action execution infinite loops that may happen due to workflow misconfiguration when the same flow of actions gets executed over and over again (the Acyclic part of DAG)
 - ❑ DAG also enforces sequential execution of actions when the next configured action is not run until the previous action has completed (the Directed part of DAG). This arrangement is also referred to as “control dependency”

Oozie Terminology

- **Action** (a.k.a. task or action node)
 - An execution/computation workflow task invoking Direct/Streaming MapReduce jobs, Spark/Sqoop/Pig/Hive scripts, shell commands, sending an email etc.
- **Workflow**
 - A collection of actions arranged in a control dependency Direct Acyclic Graph (DAG)
- **Workflow Definition**
 - XML-based description of a workflow

Oozie Architecture



Oozie Terminology

- **Workflow Definition Language**
 - The domain-specific language used to define a Workflow Definition
- **Workflow Job (a.k.a. application)**
 - A unit of work executed as per the workflow definition
- **Workflow Engine (a.k.a DAG engine)**
 - Workflow run-time system

Notes:

Oozie Workflow definitions are written in hPDL, a domain-specific similar to Jboss Process Definition Language (jPDL).

A workflow job (application) is packaged as a zip file containing all the resources needed to execute the workflow actions, such as the workflow definition XML file, system JARs, MR JAR files (or MR scripts for Streaming MapReduce), Sqoop/Hive/Pig scripts, etc

Oozie Job Types

- Oozie executes three types of jobs:
 - ☐ MapReduce (including Streaming MR), Spark, Sqoop, Pig, Hive, Shell Scripts, Java etc.
 - **Note:** The oozie server can work with either the legacy MapReduce (MRv1) or YARN (but not simultaneously)
 - ☐ Coordinator jobs: Oozie workflow tasks are triggered based on a certain condition, e.g. date/time or data availability
 - ☐ Bundled jobs: a bunch of coordinator jobs managed as a single job unit

Oozie Configuration

- Oozie system can use any of the following databases as its configuration repository:
 - ❖ (Embedded) Apache Derby Java Database (the default option)
 - ❖ MySQL (by default needs port 3306)
 - ❖ Oracle (by default needs port 1521)
 - ❖ PostgreSQL (by default needs port 5432)

Notes:

Oozie's database connectivity configuration is done in `oozie-site.xml`

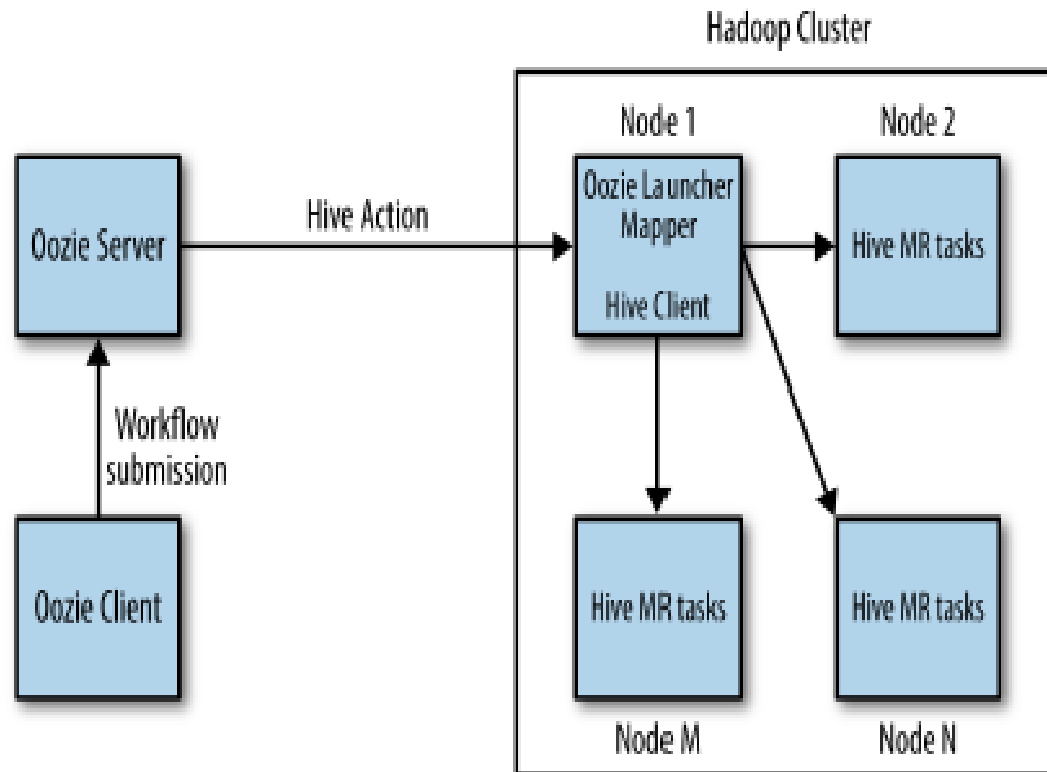
You will need to create the oozie database in the database repository of your choice.

You will also need to have the appropriate JDBC driver for databases (other than the default embedded Derby option). In the Oozie standard deployment, that driver (or its symbolic link) must be placed in the `/var/lib/oozie/` directory (it is not there already)

Basics of Oozie Action

- Action nodes define the jobs, which are the individual units of work that are chained together to make up the Oozie workflow
- Actions do the actual processing in the workflow
- An action node can run a variety of jobs: MapReduce, Pig, Hive, and more
- Workflow is composed of nodes
- Each node does a specified work and on success moves to one node or moves to another node on failure
- Nodes in the Oozie Workflow are of control flow nodes and action nodes

Oozie Actions and Execution Model



Action Types

- MapReduce Action
- Java Action
- Pig Action
- FS Action (**Sync**)
- Sub-Workflow Action (**Sync**)
- Hive Action
- Hive 2 Action
- DistCp Action
- Email Action (**Sync**)
- Shell Action
- SSH Action (**Sync**)
- Sqoop Action
- Spark Action

Action Sync vs Async

- All Hadoop actions and the <shell> action follow the Action Execution Model. These are called asynchronous actions because they are launched via a launcher as Hadoop jobs
- The filesystem action, email action, SSH action, and sub-workflow action are executed by the Oozie server itself and are called synchronous actions
- The execution of these synchronous actions do not require running any user code—just access to some libraries

Oozie Sqoop action

Oozie's sqoop action helps users run Sqoop jobs as part of the workflow

The following elements are part of the Sqoop action:

- job-tracker (required)
- name-node (required)
- prepare
- job-xml
- configuration
- command (required if arg is not used)
- arg (required if command is not used)
- file
- archive

One Sqoop Action Example

```
<action name="sqoop-import">
  <sqoop xmlns="uri:oozie:sqoop-action:0.2">
    <job-tracker>localhost:8032</job-tracker>
    <name-node>hdfs://localhost:8020</name-node>
    <prepare>
      <delete path=" hdfs://localhost:8020/hdfs/root/sqoop/output-data"/>
    </prepare>
    <configuration>
      <property>
        <name>mapred.job.queue.name</name>
        <value>default</value>
      </property>
    </configuration>
    <command>import --connect jdbc:mysql://localhost:3306/default --table test_table
      -username mytestsqoop -password password
      --target-dir /user/alti-test-01/ara/output-data/sqoop -m 1</command>
  </sqoop>
  <ok to="end"/>
  <error to="fail"/>
</action>
```

More Sqoop Action Example

```
<action name="sqoopAction">
  <sqoop xmlns="uri:oozie:sqoop-action:0.2">
    <job-tracker>${jobTracker}</job-tracker>
    <name-node>${nameNode}</name-node>
    <command>export --connect jdbc:mysql://cdh-dev01/airawat -
-username devUser --password sun123123 --table eventsgranularreport --
direct --enclosed-by " " --export-dir
/user/hive/warehouse/eventsgranularreport</command>
  </sqoop>
  <ok to="end"/>
  <error to="killJob"/>
</action>
```

More Sqoop Action Example

```
<action name="sqoop_import">
  <sqoop xmlns="uri:oozie:sqoop-action:0.4">
    <prepare>
      <delete path="${pig_base_input}/${year}/${month}"/>
    </prepare>
    <arg>import</arg>
    <arg>--connect</arg>
    <arg>jdbc:mysql://${mysqlurl}</arg>
    <arg>--username</arg>
    <arg>root</arg>
    <arg>--password</arg>
    <arg>" "</arg>
    <arg>--query</arg>
    <arg>${query}</arg>
    <arg>--target-dir</arg>
    <arg>${pig_base_input}/${year}/${month}</arg>
    <arg>-m</arg>
    <arg>1</arg>
    <arg>--direct</arg>
  </sqoop>
  <ok to="max_rainfall"/>
  <error to="Kill"/>
</action>
```

One Hive Action

- Hive actions run a Hive query on the cluster
- The Hive query and the required configuration, libraries, and code for user-defined functions have to be packaged as part of the workflow bundle and deployed to HDFS
 - job-tracker (required)
 - name-node (required)
 - prepare
 - job-xml
 - configuration
 - script (required)
 - param
 - argument
 - file
 - archive

One Hive Action Example

```
<action name="hiveAction">  
    <hive xmlns="uri:oozie:hive-action:0.2">  
        <job-tracker>${jobTracker}</job-tracker>  
        <name-node>${nameNode}</name-node>  
        <job-xml>${appPath}/hive-site.xml</job-xml>  
  
        <script>${appPath}/runHiveLogReport.hql</script>  
    </hive>  
    <ok to="sendEmailSuccess"/>  
    <error to="sendEmailKillHive"/>  
</action>
```


One Hive 2 Action

- Oozie also has Hive 2 action, where we can use Hive Server 2 and execute our Hive queries
- Hive 2 action uses Beeline to execute queries via the Hive Server 2

<action>

<job-tracker> // Job tracker details

<name-node> // Name node details

<prepare> // Create or Delete directory

<job-xml> // Any job xml properties

<configuration> // Hadoop job configuration

<jdbc-url> // HiveServer2 JDBC URL

<password> // Password (if any for Hiveserver2)

<script> // Hive script to run

<param> // Parameters to hive script

<argument> // Arguments

<file> // Any files needed to run job

<archive> // Any job dependencies (jar etc)

</action>

One Hive 2 Action Example

```
<action name="hive2">  
  <hive2 xmlns="uri:oozie:hive2-action:0.1">  
    <jdbc-url>${hivejdbcurl}/${outputHiveDatabase}</jdbc-  
url>  
    <script>${hivescript}</script>  
    <param>outputHive=${outputHive}</param>  
  </hive2>  
  <ok to="End"/>  
  <error to="Kill"/>  
</action>
```


One Shell Action

- Oozie provides a convenient way to run any shell command. This could be linux commands, Perl/Python scripts, or even Java programs invoked through the linux shell
- The shell command runs on an arbitrary Hadoop cluster node and the commands being run have to be available locally on that node
- limitations and characteristics:
 - Interactive commands are not allowed
 - Can't run sudo or run as another user
 - The executable has to be either available on the node or copied by the action via the distributed cache using the <file> tag
 - On a nonsecure Hadoop cluster, the shell command will execute as the linux user runs on the YARN container
 - On secure Hadoop clusters running Kerberos, the shell commands will run as the linux user who submitted the workflow containing the <shell> action

Understand Shell Action Concept

The elements that make up this action are as follows:

- job-tracker (required)
- name-node (required)
- prepare
- job-xml
- configuration
- exec (required)
- argument
- env-var
- file
- archive
- capture-output

One Shell Action Example

```
<shell xmlns="uri:oozie:shell-action:0.1">
  <job-tracker>${jobTracker}</job-tracker>
  <name-node>${nameNode}</name-node>
  <configuration>
    <property>
      <name>mapred.job.queue.name</name>
      <value>${queueName}</value>
    </property>
  </configuration>
  <exec>${lineCountShellScript}</exec>
    <argument>${inputDir}</argument>
  <file>${lineCountShScriptPath}#${lineCountShellScript}</file>
  <capture-output/>
</shell>
```

One Shell Action Example

```
<action name="sshAction">  
  <ssh xmlns="uri:oozie:ssh-action:0.1">  
    <host>${focusNodeLogin}</host>  
    <command>${shellScriptPath}</command>  
    <capture-output/>  
  </ssh>  
  <ok to="sendEmail"/>  
  <error to="killAction"/>  
</action>
```

One Shell Action Example

```
<action name="sshAction">  
  <ssh xmlns="uri:oozie:ssh-action:0.1">  
    <host>${focusNodeLogin}</host>  
    <command>${shellScriptPath}</command>  
    <capture-output/>  
  </ssh>  
  <ok to="sendEmail"/>  
  <error to="killAction"/>  
</action>
```

One Spark Action

- The Spark action has been recently added in Oozie
- The general schema is as follows:

<action>

<job-tracker> // Job tracker details

<name-node> // Name node details

<prepare> // Create or Delete directory

<job-xml> // Any job xml properties

<configuration> // Hadoop job configuration

<master> // Spark master details

<mode> // Spark driver mode

<name> // Spark Job name

<class> // Spark main class

<spark-opts> // Spark Job options

<arg> // Arguments for the job

</action>

One Spark Action Example

```
<spark xmlns="uri:oozie:spark-action:0.1">  
  <job-tracker>${jobTracker}</job-tracker>  
  <name-node>${nameNode}</name-node>  
  <master>yarn-cluster</master>  
  <mode>cluster</mode>  
  <name>Spark Calculator</name>  
  <class>ca.spark.Main</class>  
  <jar>/path/my-jar-withdependencies.jar</jar>  
  <arg>${input}</arg>  
  <arg>${output}</arg>  
</spark>
```


One FileSystem Action

- Users can run HDFS commands using Oozie's FS action
- Not all HDFS commands are supported, but the following common operations are allowed:
 - delete, mkdir, move, chmod, <touchz>, chgrp
- The elements that make up the FS action are as follows:
 - name-node (required)
 - job-xml
 - configuration
 - delete
 - mkdir
 - move
 - chmod
 - touchz
 - chgrp

One FileSystem Action Example

```
<action name="myFSAction">
  <fs>
    <delete path='hdfs://localhost:8020/usr/joe/temp-data'/>
    <mkdir path='myDir/${wf:id()}'/>
    <move source='${jobInput}'
target='myDir/${wf:id()}/input'/>
    <chmod path='${jobOutput}' permissions='-rwxrw-rw-'
dir-files='true'/>
  </fs>
</action>
```

One SSH Action

- The `<ssh>` action runs a shell command on a specific remote host using a secure shell
- The command should be available in the path on the remote machine and it is executed in the user's home directory on the remote machine
- The shell command can be run as another user on the remote host from the one running the workflow
- The `oozie.action.ssh.allow.user.at.host` should be set to true in `oozie-site.xml`
 - Here are the elements of an `<ssh>` action:
 - `host` (required)
 - `command` (required)
 - `args`
 - `arg`
 - `capture-output`

One SSH Action Example

```
<action name="mySSHAction">
```

```
  <ssh>
```

```
    <host>foo@bar.com</host>
```

```
    <command>uploaddata</command>
```

```
    <args>jdbc:derby://bar.com:1527/myDB</args>
```

```
    <args>hdfs://foobar.com:8020/usr/joe/myData</args>
```

```
  </ssh>
```

```
</action>
```

One Subworkflow Action

- The sub-workflow action runs a child workflow as part of the parent workflow
- It acts as an embedded workflow
- From a parent's perspective, this is a single action and it will proceed to the next action in its workflow if and only if the subworkflow is done in its entirety
- The child and the parent have to run in the same Oozie system and the child workflow application has to be deployed in that Oozie system
 - app-path (required)
 - propagate-configuration
 - configuration

One Subworkflow Action Example

```
<action name='dataExporterSubWorkflow'>  
  <sub-workflow>  
    <app-path>${subWorkflowCodeDir}</app-path>  
    <propagate-configuration/>  
  </sub-workflow>  
  <ok to="end"/>  
  <error to="killJob" />  
</action>
```


One DistCp Action

- DistCp action supports the Hadoop distributed copy tool, which is typically used to copy data across Hadoop clusters
- Users can use it to copy data within the same cluster as well, and to move data between Amazon S3 and Hadoop clusters
- Here are the elements required to define this action:
 - job-tracker (required)
 - name-node (required)
 - prepare
 - configuration
 - java-opts
 - arg

One DistCp Action Example

```
<action name="myDistcpAction">
  <distcp xmlns="uri:oozie:distcp-action:0.1">
    <job-tracker>localhost:8032</job-tracker>
    <name-node>hdfs://localhost:8020</name-node>
    <prepare>
      <delete
path="hdfs://localhost:8020/hdfs/user/root/logs/2017-04-15/">
    </prepare>
    <arg>-Dfs.s3n.awsAccessKeyId=XXXX</arg>
    <arg>-Dfs.s3n.awsSecretAccessKey=YYYY</arg>
    <arg>-m</arg>
    <arg>100</arg>
    <arg>s3n://my-logfiles/2017-04-15/*</arg>
    <arg>/hdfs/user/root/logs/2017-04-15/</arg>
  </distcp>
  <ok to="success"/>
  <error to="fail"/>
</action>
```

One Email Action

- Oozie's email action provides an easy way to integrate this feature into the workflow
- It takes the usual email parameters: to, cc, subject, and body
- Email IDs of multiple recipients can be comma separated
- The following elements are part of this action:
 - to (required)
 - cc
 - subject (required)
 - body (required)

One Email Action

- Oozie server node has the necessary SMTP email client installed and configured
- The following SMTP server configuration has to be defined in the oozie-site.xml file for this action to work:
 - oozie.email.smtp.host (default: localhost)
 - oozie.email.smtp.port (default: 25)
 - oozie.email.from.address (default: oozie@localhost)
 - oozie.email.smtp.auth (default: false)
 - oozie.email.smtp.username (default: empty)
 - oozie.email.smtp.password (default: empty)

One Email Action Example

```
<action name="myEmailAction">
```

```
  <email>
```

```
    <to>joe@wecloudata.com,the_other_joe@wecloudata.com</to>
```

```
      <cc>john@wecloudata.com</cc>
```

```
      <subject>Email notifications for ${wf:id()}</subject>
```

```
      <body>The wf ${wf:id()} successfully completed.</body>
```

```
    </email>
```

```
</action>
```

One Java Action

- Oozie's Java action is a great way to run custom Java code on the Hadoop cluster
- The Java action will execute the public static void main(String[] args) method of the specified Java main class
- It is technically considered a non-Hadoop action
- This action runs as a single mapper job, which means it will run on an arbitrary Hadoop worker node

One Java Action

- The Java action is made up of the following elements:
 - job-tracker (required)
 - name-node (required)
 - prepare
 - configuration
 - main-class (required)
 - java-opts
 - arg
 - File
 - archive
 - capture-output

One Java Action Example

```
<action>
  <java>
    <job-tracker>localhost:8032</job-tracker>
    <name-node>hdfs://localhost:8020</name-node>
    <prepare>
      <delete path="{myJavaActionOutput}"/>
    </prepare>
    <configuration>
      <property>
        <name>mapred.queue.name</name>
        <value>default</value>
      </property>
    </configuration>
    <main-class>org.apache.oozie.MyJavaMainClass</main-class>
    <java-opts>-DmyOpts</java-opts>
    <arg>argument1</arg>
    <arg>argument2</arg>
    <capture-output/>
  </java>
</action>
```


One MapReduce Action

This action type supports all three variations of a Hadoop MapReduce job: Java, streaming, and pipes

- job-tracker (required)
- name-node (required)
- prepare
- streaming or pipes
- job-xml
- configuration
- file
- archive

One MapReduce Action Example

```
<action name="mapReduceAction">
  <map-reduce>
    <job-tracker>${jobTracker}</job-tracker>
    <name-node>${nameNode}</name-node>
    <prepare>
      <delete path="${outputDir}"/>
    </prepare>
    <configuration>
      <property>
        <name>mapred.mapper.new-api</name>
        <value>true</value>
      </property>
      <property>
        <name>oozie.use.system.libpath</name>
        <value>>false</value>
      </property>
      <property>
        <name>oozie.libpath</name>
        <value>${appPath}/lib</value>
      </property>
    </configuration>
  </map-reduce>
```

One Pig Action

- Although Pig jobs can be launched independently via Pig scripts, Oozie offers the following added-value services:
 - Complex workflow dependencies management
 - Frequency-based execution
 - Visual monitoring of the job progress
- Oozie manages Pig jobs via the Pig Action Template

Notes:

A Pig script contains Pig Latin statement and Pig commands in a single file that can be launched from command-line

One Pig Action Example

```
<action name="pigAction">  
  <pig>  
    <job-tracker>${jobTracker}</job-tracker>  
    <name-node>${nameNode}</name-node>  
    <prepare>  
      <delete path="${outputDir}"/>  
    </prepare>  
    <script>reportScript.pig</script>  
  </pig>  
  <ok to="end"/>  
  <error to="killJobAction"/>  
</action>
```

More Pig Action Examples

```
rmf oozieProject/workflowPigAction/output
```

```
raw_log_DS =
```

```
-- load the logs into a sequence of one element tuples
```

```
LOAD 'oozieProject/data/*/ */ */ */ */' AS line;
```

```
parsed_log_DS =
```

```
-- for each line/log parse the same into a
```

```
-- structure with named fields
```

```
FOREACH raw_log_DS
```

```
    GENERATE
```

```
        FLATTEN (
```

```
            REGEX_EXTRACT_ALL(
```

```
                line,
```

```
'(\\w+)\\s+(\\d+)\\s+(\\d+:\\d+:\\d+)\\s+(\\w+\\W*\\w*)\\s+(.*?\\:|)\\s+(.*$)'
```

```
            )
```

```
        )
```

Oozie and HCatalog

- HCatalog provides the table and storage management layer for Hadoop
- It brings various tools in the Hadoop ecosystem together
- Using HCatalog interface, different tools like Hive, Pig, and MapReduce can read and write data on Hadoop
- All of them can use the shared schema and datatypes provided by HCatalog

HCatalog datasets

- HCatalog uses Hive metastore to provide table-layer abstraction to different tools such as Pig, MapReduce, and so on
- HCatalog allows access to data stored in Hive
- The Datasets can be defined as HCatalog by using the following general syntax:

```
hcat://HiveMetastoreURL/hiveDatabaseName/hiveTableName/HiveTablePARTitionInformation
```

```
<datasets>
```

```
  <dataset name="rainfall_partitioned" frequency="{coord:months(1)}"  
    initial-instance="2015-01-01T00:00Z" timezone="Australia/Sydney">
```

```
    <uritemplate>hcat://{hcaturl}/default/ch07_v1_max_rainfall_trend/yearmonth={YEAR}  
    {MONTH}</uri-template>
```

```
    <done-flag></done-flag>
```

```
  </dataset>
```

```
</datasets>
```


HCatalog EL functions

- `boolean hcat:exists(String uri)`
- `hcat:exists("hcat://${hcaturl}/default/ch07_v1_max_rainfall_trend/year month=${YEAR}${MONTH}")`

HCatalog Coordinator functions

Function	Use
<code>databaseIn(String name)</code>	Returns input database name for the HCatalog Dataset
<code>databaseOut(String name)</code>	Returns output database name for the HCatalog Dataset
<code>tableIn(String name)</code>	Returns input table name for the HCatalog Dataset
<code>tableOut(String name)</code>	Returns output database name for the HCatalog Dataset
<code>dataInPartitionFilter(String name, String type)</code>	Filters the Hive partitions and returns the required Dataset to be consumed by Pig, Hive, or Java actions
<code>dataOutPartitions(String name)</code>	Comma-separated list of output partitions
<code>dataInPartitionMin(String name, String partition)</code>	Minimum value for partition in given input event instances
<code>dataInPartitionMax(String name, String partition)</code>	Maximum value for partition in given input event instances
<code>dataOutPartitionValue(String name, String partition)</code>	Returns the value of partition for output event
<code>dataInPartitions(String name, String type)</code>	List of key-value pairs for the input event Dataset

Oozie Execution Model

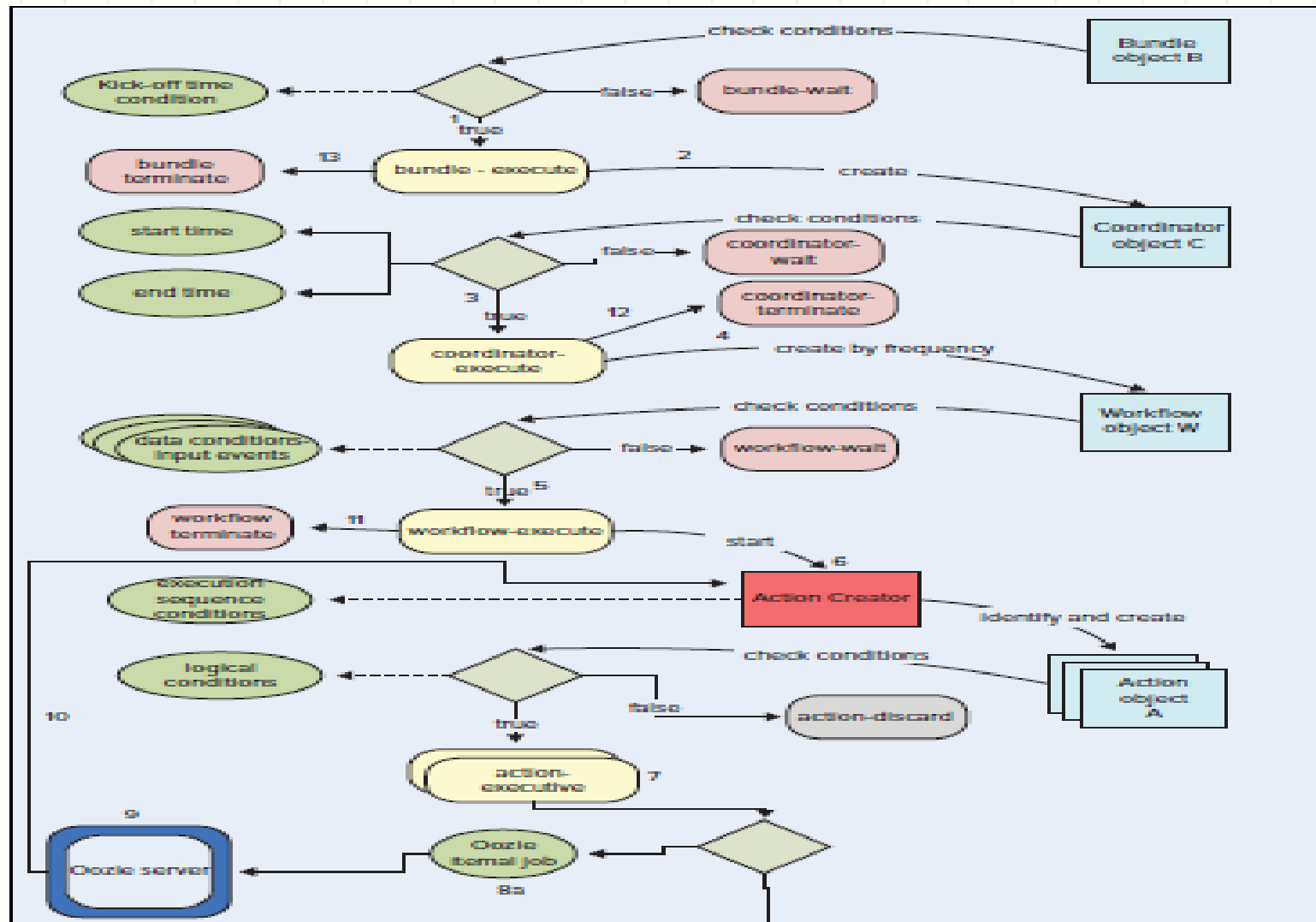
The execution model contains the following types of objects:

- Executable objects representing Oozie jobs (actions):
 - Bundle objects
 - Coordinator objects
 - Workflow objects (the complete Workflow DAG)
 - Workflow action objects

Oozie Execution Model

- Conditional objects defined in Oozie applications:
 - Data conditions for Coordinator objects (expressed via data sets and input events)
 - Time conditions for Coordinator objects (expressed via start, end, and frequency)
 - Logical conditions for action objects (expressed with data-flow nodes)
 - Execution sequence conditions for action objects (expressed via graph of nodes in Workflow)
 - Concurrency conditions for Coordinator objects

Oozie Execution Model



Oozie Launcher

- Oozie's execution model is different from the default approach users take to run Hadoop jobs
- Submitting a workflow job using the Oozie CLI on the edge node. The Oozie client actually submits the workflow to the Oozie server, which typically runs on a different node. Regardless of where it runs, it's the Oozie server's responsibility to submit and run the underlying MapReduce jobs on the Hadoop cluster.
- Oozie first submits a MapReduce job called the "launcher job," which in turn runs the Hadoop, Hive, or Pig job using the appropriate client APIs
- The Oozie launcher is basically a map-only job running a single mapper on the Hadoop cluster
- This will result in other MapReduce jobs being spun up as required. These Oozie jobs are called "asynchronous actions"

Oozie Workflows

- An Oozie workflow specifies a graph (DAG) of actions and flow controls (a.k.a. control nodes) to manage and coordinate the execution path of actions
 - ✓ Flow controls include *start*, *fork*, *join*, *decision* and *end* nodes.
- An oozie workflow can contain one to many actions
- Oozie workflow can be triggered by such external events as time and when data becomes available

The Flow Of Oozie Workflows

- An Oozie workflow gets executed as follows:
 - An Oozie workflow action initiates a job on a remote system
 - When the target job gets completed on the remote system, a notification is sent back to Oozie
 - Oozie proceeds to the next action in the workflow

More on Oozie Workflows

- Identical workflow jobs can run concurrently
 - This required parameterization of job properties, directories, etc. (using the `${xxx}` notation) specified in the workflow definition in order to avoid conflicts at run-time
 - Parameterization is used by the Oozie Coordinator

Oozie Workflow Control Nodes

- **Start control node**

- The entry point for a workflow job. A workflow automatically transitions to this node when started. Only one start node is allowed in the workflow definition XML file

- **End control node**

- Designates the successful end for a workflow job, which indicates that the workflow job has completed successfully. When a workflow job reaches the end control node, the job is regarded as finished successfully. Only one end node is allowed in the workflow definition XML file

- **Kill control node**

- When a workflow job reaches the kill control node, it gets “killed” or terminated. A workflow definition file may have zero to many kill nodes

Notes:

According to Apache Oozie documentation: “If one or more actions started by the workflow job are executing when the end node is reached, the actions will be killed. In this scenario the workflow job is still considered as successfully run... if one or more actions started by the workflow job are executing when the kill node is reached, the actions will be killed”

More Oozie Workflow Control Nodes

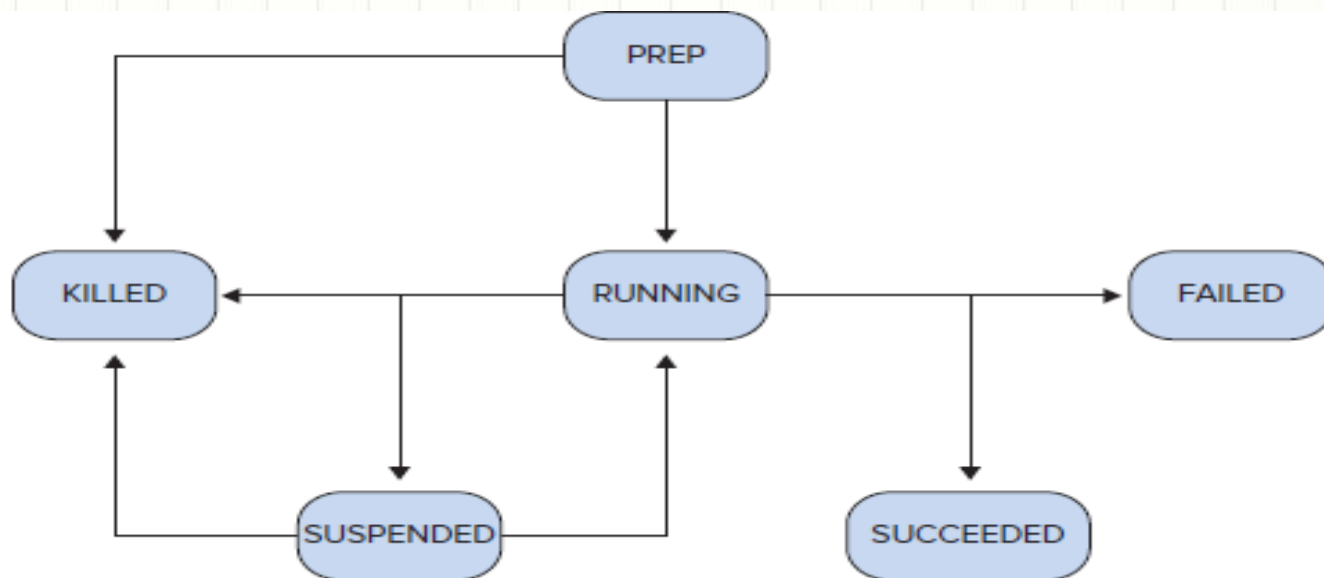
- **Decision node**
 - Designates a decision point in the execution path that uses an expression language based on the JSP (JavaServer Pages) Expression Language expressions that resolve to a Boolean value: true or false
- **Fork/Join control nodes**
 - A **fork** node splits one path execution into multiple paths of concurrent execution of a workflow job
 - A **join** control node waits until every concurrently executing forked tasks arrive at the join node. A join node complements the fork node and they are used in pairs.

One Decision Control Example

```
<decision name="inputAvailableCheckDecision">  
  <switch>  
    <case to="startTaskFork">  
      ${inputDirRecordCount gt minRequiredRecordCount}  
    </case>  
    <default to="end"/>  
  </switch>  
</decision>
```

Oozie Workflow Lifecycles

Once the Workflow application is loaded to the Workflow server, it becomes a Workflow job, possible states for a Workflow job are PREP, RUNNING, SUSPENDED, SUCCEEDED, KILLED, and FAILED.



Oozie Workflow Job Configuration

- Global configuration
- Job XML file(s) specified in <job-xml>
- Inline <configuration> section in workflow.xml

The job.properties File

```
oozie job -oozie http://localhost:4080/oozie/ -config ~/job.properties -run
```

- Example:

```
nameNode= hdfs://localhost:8020
```

```
jobTracker=localhost:8032
```

```
queueName=research
```

```
oozie.use.system.libpath=true
```

```
oozie.wf.application.path=${nameNode}/user/joe/oozie/mrJob/firstWorkflow.xml
```

Configuration and Parameterization Examples

workflow.xml file:

```
...  
<job-xml>my-job.xml</job-xml>  
...  
<configuration>  
  <property>  
    <name>mapred.job.queue.name</name>  
    <value>integration</value>  
  </property>  
</configuration>
```

my-job.xml file:

```
...  
<property>  
  <name>mapred.job.queue.name</name>  
  <value>staging</value>  
</property>  
...
```

Configuration and Parameterization Examples

config-default.xml:

```
<property>  
<name>queue_var</name>  
<value>default</value>  
</property>
```

job.properties:

```
queue_var=research
```

Configuration and Parameterization Examples

workflow.xml:

```
<job-xml>my-job.xml</job-xml>
```

...

```
<property>
```

```
<name>queue_var</name>
```

```
<value>production</value>
```

```
</property>
```

my-job.xml

```
<property>
```

```
<name>mapred.job.queue.name</name>
```

```
<value>${queue_var}</value>
```

```
</property>
```

...

Capture Output to Variables

- Writing to a local file with absolute path
- Writing to a local file with relative path
- Writing to a HDFS file with absolute path
- Stores the output of an action in its launcher logs
- Capture output tag, output a key value pair and oozie will read them and add them to the flow as variables

Capture Output to Variables

```
<action name="load-files">
  <shell xmlns="uri:oozie:ssh-action:0.1">
    ...
    <command>load.sh</command>
    <capture-output/>
  </shell>
  <ok to="check-if-data"/>
  <error to="kill"/>
</action>

<decision name="check-if-data">
  <switch>
    <case to="end">${ wf:actionData('load-files')['output'] eq 'success'}</case>
    <default to="kill" />
  </switch>
</decision>
```

Oozie Coordinator

- Oozie Coordinator allows the user to schedule the execution of workflow jobs based on an event (time/date, data availability, or other external factors)
 - According to Oozie documentation: “Expressing the conditions that trigger a workflow job can be modeled as a predicate that has to be satisfied. The workflow job is started after the predicate is satisfied. A predicate can reference to data, time and/or external events.”
- Oozie Coordinator uses parameterization of workflow definitions using the `${}` script notation)
 - At execution time, all the parameterized values are resolved to actual values

Notes:

For parameterization, Oozie Coordinator uses the JSP Expression Language syntax as defined by the JSP 2.0 Specification

Time-based triggers

- In a time aware coordinator, a workflow is executed at fixed intervals or frequency
- A time trigger in the coordinator using three attributes:
 - Start time
 - Frequency
 - End time

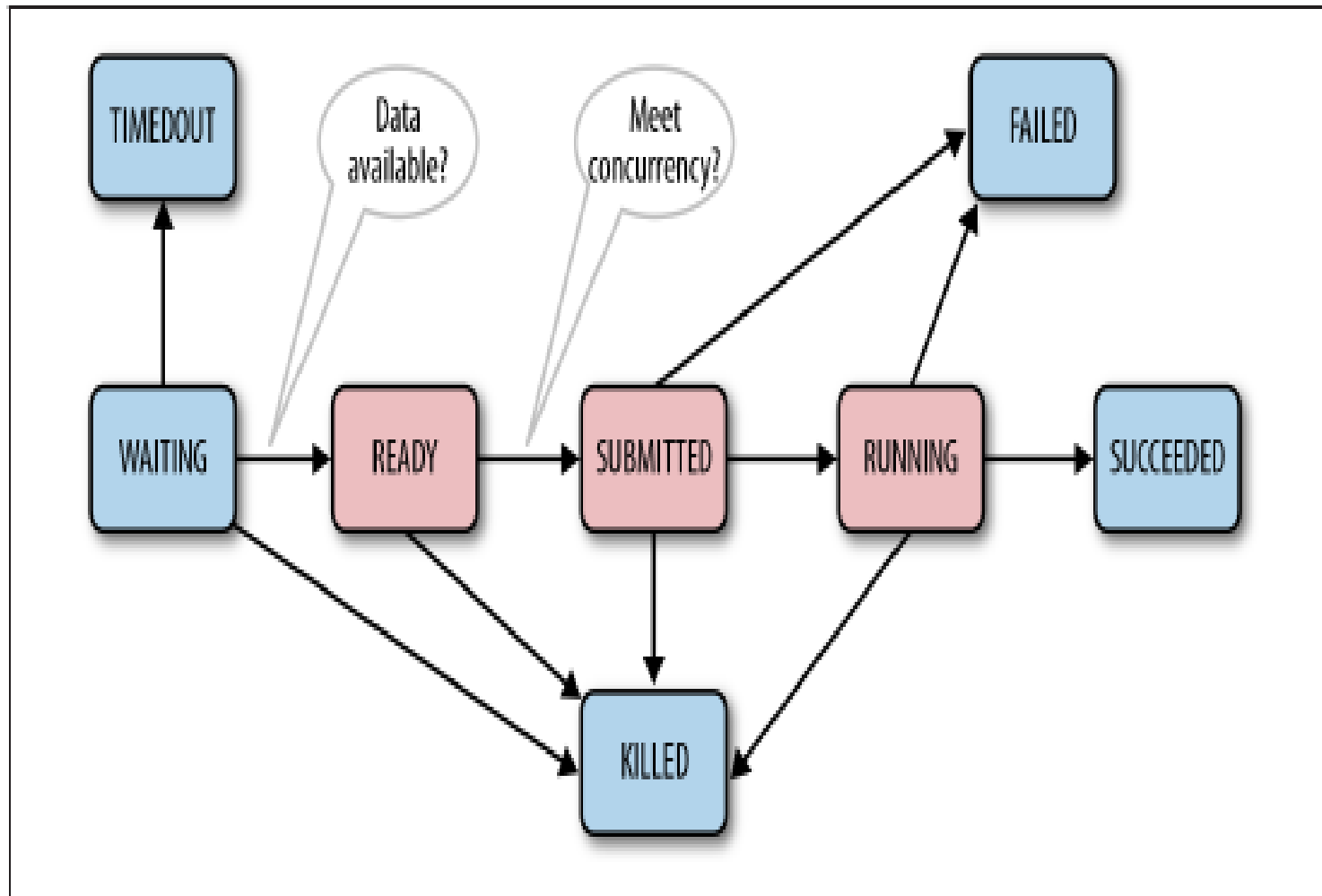
Scheduled Coordinator Example

```
<coordinator-app name="AirawatCoordJobTimeDep"
    frequency="${coord:days(1)}"
    start="${startTime}"
    end="${endTime}"
    timezone="${timeZoneDef}"
    xmlns="uri:oozie:coordinator:0.1">
  <controls>
    <timeout>20</timeout>
    <concurrency>6</concurrency>
    <execution>FIFO</execution>
  </controls>
  <action>
    <workflow>
      <app-path>${workflowAppPath}</app-path>
    </workflow>
  </action>
</coordinator-app>
```

Coordinator Action

- A coordinator job regularly creates/materializes a new coordinator action for each time instance based on its start time and frequency
- The coordinator action actually checks for data availability and ultimately submits the workflow

Coordinator Action Lifecycle



Coordinator Submission

- Job submission using the oozie CLI
- job.property

```
nameNode=hdfs://localhost:8020
```

```
jobTracker=localhost:8032
```

```
appBaseDir=${nameNode}/user/${user.name}/first-coord
```

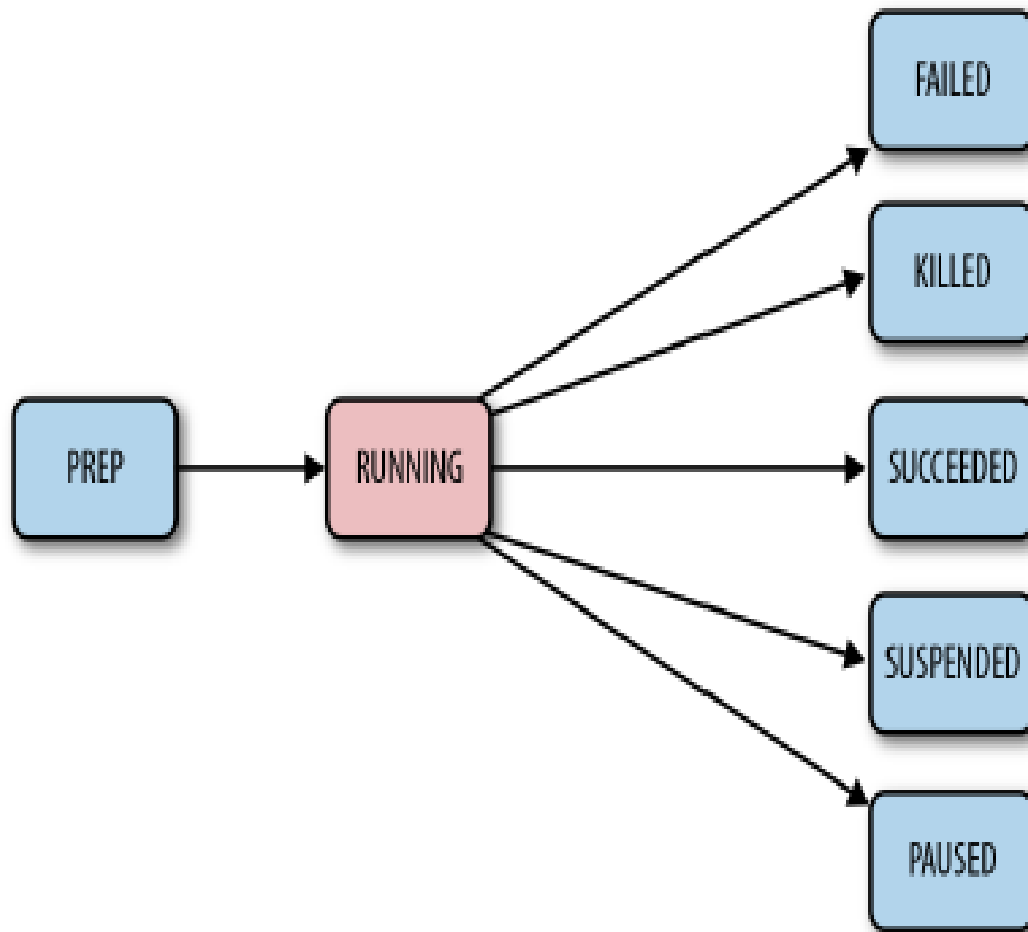
```
oozie.coord.application.path=${appBaseDir}/app
```

- submit the job

```
export OOZIE_URL=http://localhost:11000/oozie
```

```
oozie job -run -config job.properties
```

Coordinator Job Lifecycle



Execution Controls

- A coordinator job continuously creates coordinator actions until it reaches the end time. In an ideal situation
- backlog is lot of coordinator actions being concurrently active in the system due to the following reason:
 - Delayed data
 - Reprocessing
 - Late submission

Execution Controls

- Oozie provides four control parameters for any coordinator to address these catch-up scenarios:
 - throttle - controls how many maximum coordinator actions can be in the WAITING state for a coordinator job at any instant
 - timeout - waiting
 - execution order - multiple actions in the READY state (FIFO, LIFO, and LAST_ONLY)
 - concurrency - the RUNNING state

Data Triggered Coordinator

- Oozie coordinator allows users to express both data and time dependency together and to kick off the workflow accordingly

- Expressing Data Dependency

➤ Dataset

- ✓ user can define a dataset either using its directory location or using metadata
- ✓ the data in a dataset can be produced in a fixed interval (synchronous) or ad hoc/random (asynchronous)
- ✓ oozie currently supports only the synchronous datasets

Data Triggered Coordinator

➤ Dataset

✓ five attributes to define a dataset in Oozie:

- ☐ name
- ☐ initial-instance
- ☐ frequency
- ☐ uri-template
- ☐ done-flag

Data Triggered Coordinator

➤ Dataset

✓ example:

```
<dataset name="ds_input1" frequency="${coord:hours(6)}" initial-  
instance="2014-12-29T02:00Z">  
  <uri-template>${baseDataDir}/revenue_feed/${YEAR}-${MONTH}-  
  ${DAY}-${HOUR}</uri-template>  
  <done-flag>_trigger</done-flag>  
</dataset>
```

Data Triggered Coordinator

➤ Input-event

- ✓ describe the actual instance(s) of dependent dataset for the coordinator, a workflow will not start until all the data instances defined in the input-events are available
- ✓ only one <input-events> section in a coordinator, but it can include one or more data-in sections, data-in can include one or more data instances of that dataset
- ✓ example:

```
<input-events>
<data-in name="event_input1" dataset="ds_input1">
<start-instance>${coord:current(-4)}</start-instance>
<end-instance>${coord:current(-1)}</end-instance>
</data-in>
</input-events>
```

Data Triggered Coordinator

➤ Output-event

- ✓ `<output-events>` specifies the data instance produced by a coordinator action

- ✓ example:

```
<output-events>  
<data-out name="event_output1" dataset="daily-feed">  
<instance>${coord:current(0)}</instance>  
</data-out>  
</output-events>
```

More Complex Workflow Coordinator

```
<coordinator-app name="my_rollup_job" start="2014-01-01T02:00Z "
  end="2014-12-31T02:00Z" frequency="${coord:days(1)}"
  xmlns="uri:oozie:coordinator:0.4">
  <datasets>
    <dataset name="ds_input1" frequency="${coord:hours(6)}"
      initial-instance="2013-12-29T02:00Z">
      <uri-template>
        hdfs://localhost:8020/user/joe/revenue_feed/${YEAR}-${MONTH}-${DAY}-
        ${HOUR}
      </uri-template>
      <done-flag>_trigger</done-flag>
    </dataset>
    <dataset name="daily-feed" frequency="${coord:days(1)}"
```


More Complex Workflow Coordinator

```
    initial-instance="2013-12-29T02:00Z">
    <uri-template>
      hdfs://localhost:8020/user/joe/revenue_daily_feed/${YEAR}-${MONTH}-
      ${DAY}
    </uri-template>
  </dataset>
</datasets>
<input-events>
  <data-in name="event_input1" dataset="ds_input1">
    <start-instance>${coord:current(-4)}</start-instance>
    <end-instance>${coord:current(-1)}</end-instance>
  </data-in>
</input-events>
<output-events>
  <data-out name="event_output1" dataset="daily-feed">
    <instance>${coord:current(0)}</instance>
  </data-out>
</output-events>
<action>
  <workflow>
    <app-path>${myWFHomeInHDFS}/app</app-path>
    <property>
      <name>myInputDirs</name>
      <value>${coord:dataIn('event_input1')}</value>
    </property>
    <property>
      <name>myOutputDirs</name>
      <value>${coord:dataOut('event_output1')}</value>
    </property>
  </workflow>
</action>
```

Oozie Bundle

- Oozie Bundle is a top-level abstraction that enables you to group a set of Coordinator applications into a Bundle application
- Coordinator applications grouped into a Bundle can be controlled (start/stop/suspend/resume/rerun) as a whole
- A Bundle is defined by using the Bundle language, which is an XML-based language

Oozie Bundle - Example

```
<bundle-app name='weather-forecast' xmlns:xsi='http://www.w3
xmlns='uri:oozie:bundle:0.1'>
  <controls>
    <kick-off-time>${kickOffTime}</kick-off-time>
  </controls>
  <coordinator name='monitor-datastream' >
    <app-path>${'coord-path'}</app-path>
    <configuration>
      <property>
        <name>monitor-time</name>
        <value>60</value>
      </property>
    </configuration>
  </coordinator>
  <coordinator name='calc-publish-forecast' >
    <app-path>${'calc-coord-path'}</app-path>
    <configuration>
      <property>
        <name>monitor-time</name>
        <value>600</value>
      </property>
    </configuration>
  </coordinator>
</bundle-app>
```

Oozie Bundle

ELEMENTS AND ATTRIBUTES	DESCRIPTION	ATTRIBUTES AND SUB-ELEMENTS
<code>bundle-app</code>	This is the top-level element of the Bundle application.	<code>name</code> , <code>controls</code> , <code>coordinator</code>
<code>name</code>	This is the attribute that specifies the name for the Bundle. For example, you can use it to refer to the Bundle application in the Hadoop CLI command.	
<code>controls</code>	This is an element that contains only one attribute — <code>kick-off-time</code> . That attribute specifies the start time for a Bundle application.	<code>kick-off-time</code>
<code>coordinator</code>	This element describes a Coordinator application included in the Bundle. A Bundle application can have multiple Coordinator elements.	<code>name</code> , <code>app-path</code> , <code>configuration</code>

Passing Arguments to Oozie Jobs

- Using an Oozie Invocation Command
- Using an Oozie Job Property File
- Using the config-default.xml File
- Using a <configuration> Element in the Action Definition
- Using a <job-xml> Element

Deciding How to Pass Arguments to Oozie Jobs

- Oozie uses parameters explicitly defined inside an action's `<arg>` tag.
- If any of the parameters cannot be resolved there, Oozie uses parameters defined in the file specified inside the `<job-xml>` tag
- If any of the parameters cannot be resolved there, Oozie uses parameters defined inside the `<configuration>` tag
- If any of the parameters cannot be resolved there, Oozie uses parameters from the command-line invocation
- If any of the parameters cannot be resolved there, Oozie uses parameters from a job property file
- Once everything else fails, Oozie tries to use `config-default.xml`

Oozie Parameterization with EL

All types of Oozie applications (Workflow, Coordinator, and Bundle) can be parameterized with Oozie Expression Language (EL)

- Basic constants are the simplest elements of EL:
 - KB: 1024 — 1 kilobyte
 - MB: 1024 * KB — 1 megabyte
 - MINUTE — 1 minute
 - HOUR — 1 hour
- Additional EL constants:
 - RECORDS — This is the Hadoop record counters group name
 - MAP_IN — This is the Hadoop mapper input record counters name

Oozie Basic Functions

- `String timestamp():` This gives the current date and time in ISO8601 format
- `String trim(String s):` This gives the trimmed value of the given string

Oozie Workflow Functions

- `wf:id()` — This returns the Oozie ID of the current Workflow job
- `wf:appPath()` — This returns the application path of the current Workflow
- `wf:conf(String name)` — This can be used to obtain the complete content of the configuration for the current Workflow
- `wf:callback(String stateVar)` — This returns the callback URL for the current Workflow node for a given action
- `wf:transition(String node)` — If the node (action) was executed, this function returns the name of the transition node from the specified node

Oozie Coordinator Functions

Coordinator EL functions provide access to Coordinator parameters, as well as parameters of input events and output events

- `coord:current(int n)` — This is used to access the name of (n-th) data set instance in input and output events
- `coord:nominalTime()` — This retrieves the time of the creation of a Coordinator action
- `coord:dataIn(String name)` — This resolves to all the URIs for the data set instances specified in an input event dataset section

Oozie Other EL Functions

HDFS EL functions are used to query HDFS for files and directories

- `fs:exists(String path)` — This checks whether or not the specified path URI exists on HDFS
- `fs: fileSize(String path)` — This returns the size of the specified file in bytes
- `hadoop:counters(String node)` - Obtain the values of the counters for a job submitted by an Hadoop action node
- `fs:isDir(String path)`
- `fs:dirSize(String path)`
- `fs:blockSize(String path)`

Oozie Libraries

The JARs in Oozie largely come from the following sources:

- System JARs
- Hadoop JARs
- Action JARs
- User JARs

Oozie Security

- Oozie Service to Hadoop Services
 - Oozie has to present the appropriate Kerberos credentials to access HDFS files or submit the jobs to Yarn
- Oozie Client to Oozie Service
 - Oozie client (Oozie CLI, REST client, Java client, or any other custom client) needs to present a valid credential to the Oozie service using Kerberos based authentication

Supporting Uber Jar

- Users package their application-specific custom classes into a JAR. Additionally, users might need a set of third-party JARs for their application
- An uber JAR that is comprised of the custom classes as well as the third-party JARs under the lib/ subdirectory in the JAR
- Hadoop understands this predefined directory structure and includes all the JARs from the lib/ subdirectory of the uber JAR on to the application classpath
- `conf.setJar()`
- `oozie.action.mapreduce.uber.jar.enable = true` in `oozie-site.xml`

Supporting Uber Jar

- MapReduce Action

```
<property>
```

```
<name>oozie.mapreduce.uber.jar</name>
```

```
<value>${MY_HDFS_PATH_TO_UBER_JAR}/my-uber-jar.jar</value>
```

```
</property>
```

Cron Scheduling

- Oozie borrows from the cron concept to address the requirement as scheduling the job every hour only on Mondays
- A Simple Cron-Based Coordinator

```
<coordinator-app name="my_second" start="${startTime}" end="${endTime}"  
    frequency="0 11-14 * * MON,WED" timezone="UTC"  
    xmlns="uri:oozie:coordinator:0.4">
```

- Oozie's cron implementation is based on the Quartz Scheduler

Cron Scheduling

Field name	Allowed values	Allowed special chars
Minute	0-59	Commas (,), dashes (-), asterisks, and slashes (/)
Hour	0-23	Commas (,), dashes (-), asterisks, and slashes (/)
Day of Month	1-31	Commas (,), dashes (-), asterisks, question marks (?), slashes (/), and the letters "L" and "W"
Month	1-12 or JAN-DEC	Commas (,), dashes (-), asterisks, and slashes (/)
Day of Week	1-7 or SUN-SAT	Commas (,), dashes (-), asterisks, question marks (?), slashes (/), hash tags (#), and the letter "L"

Emulating Async Data Processing

- Some use cases that depend on data rather than time, which require the workflow to be run as soon as some data is available irrespective of the time
- Basic idea it that if the data is available, the workflow will be run. Otherwise, the action will end up in the TIMEDOUT state
- Preconditions required for implementing this approach:
 - The shortest or the most frequent interval when the dependent data can potentially be available
 - The dependent dataset should be stored in a directory with the timestamp as part of the HDFS path
 - The value of the timeout parameter for the coordinator

Emulating Async Data Processing

```
<coordinator-app name="SIMULATED_ASYNC_COORD" frequency="${frequency}" start="${start}"
end="${end}" xmlns="uri:oozie:coordinator:0.4">
  <controls><timeout>10</timeout></controls>
  <datasets>
    <dataset name="simulated_dataset" frequency="${min_frequency}" initial-
instance="${start}" timezone="${timezone}">
      <uri-
template>hdfs://localhost:8020/tmp/data/${YEAR}/${MONTH}/${DAY}/${HOUR}/${MINUTE}</uri-
template>
    </dataset>
  </datasets>
  <input-events>
    <data-in name="ds_event" dataset="simulated_dataset">
      <instance>${coord:current(0)}</instance></data-in>
    </input-events>
  <action>
    <workflow>
      <app-path>hdfs://localhost:8020/${AppBaseDir}/mapreduce/ </app-
path>
    </workflow>
  </action>
</coordinator-app>
```

HCatalog Based Data Dependancy

The challenges of the pull approach to periodically checks the HDFS for data availability until the timeout period is reached:

- Oozie is polling HDFS regularly, it creates a lot of load on the Hadoop NameNode
- The polling-based approach increases the processing load on the Oozie server
- Due to the polling interval, the workflow might not be started as soon as the dependent dataset is available

HCatalog Based Data Dependancy

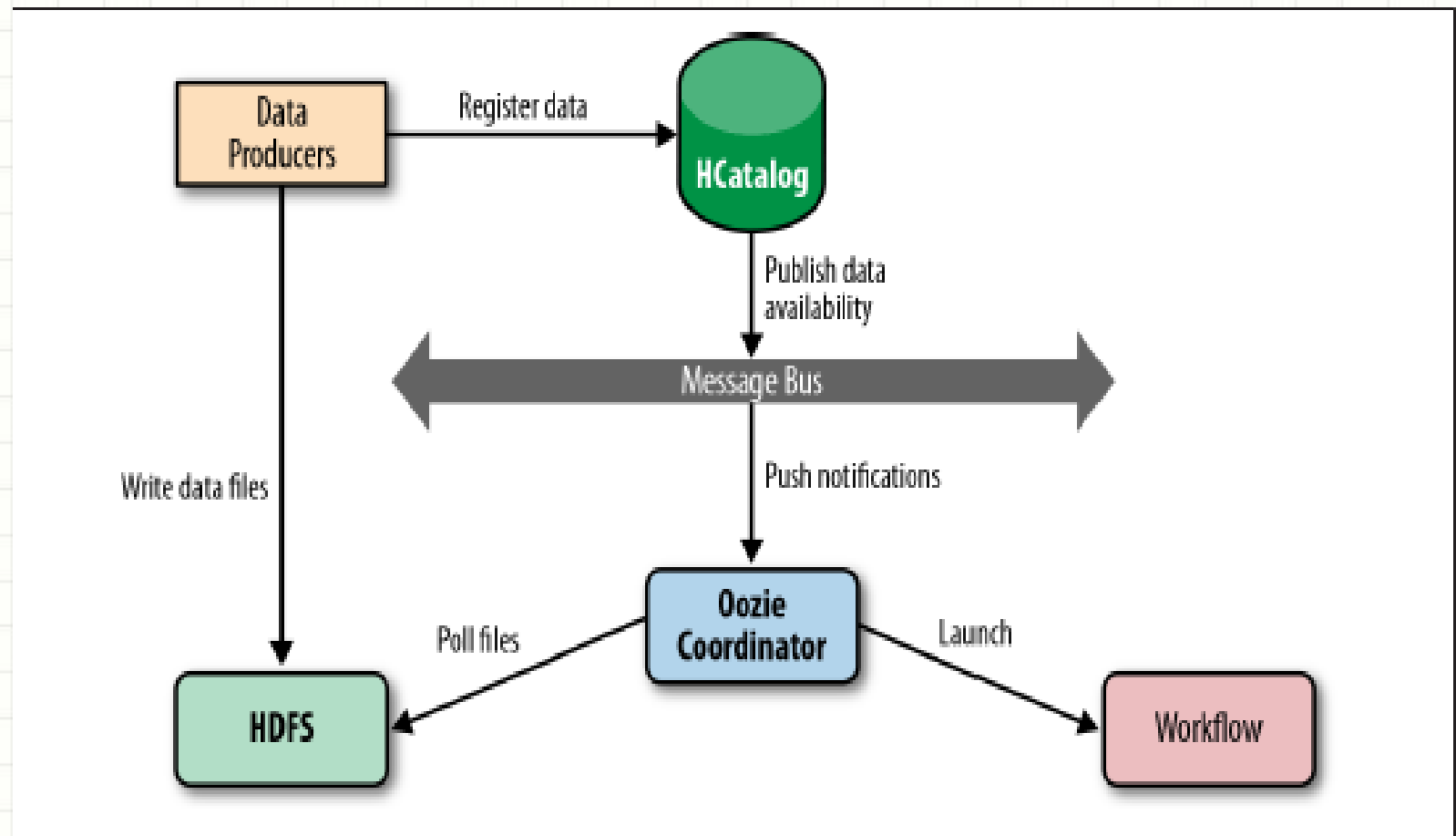
Push model for data availability checks:

- Get notified asynchronously as soon as the data is available
- HCatalog also offers JMS based notification as soon as any new data/partition is registered with HCatalog
- Oozie exploits these notifications to trigger the workflows to avoid HDFS polling

```
<dataset name="hcat_dataset" frequency="{coord:days(1)}" initial-instance="2009-02-15T08:15Z" timezone="America/Los_Angeles">
```

```
  <uri-template>hcat://localhost:9080/database1/table1/datestamp=${YEAR}${MONTH}${DAY}${HOUR};region=USA</uri-template>
</dataset>
```


HCatalog Based Data Dependancy



Oozie Actions with Hive Thrift Services

- **oozie-site.xml**

```
<property>
```

```
  <name>oozie.credentials.credentialclasses</name>
```

```
  <value>hcat=org.apache.oozie.action.hadoop.HCatCredentials</value>
```

```
</property>
```

Oozie Actions with Hive Thrift Services

- **workflow xml file**

```
<credentials>
  <credential name='hive_credentials' type='hcat'>
    <property>
      <name>hcat.metastore.uri</name>
      <value>thrift://localhost:9083</value>
    </property>
    <property>
      <name>hcat.metastore.principal</name>
      <value>hive/sandbox.hortonworks.com@weclouddata.ca</value>
    </property>
  </credential>
</credentials>
```

Oozie Actions with Hive Thrift Services

- **workflow xml file**

```
<action name="hive" cred="hive_credentials">
  <hive xmlns="uri:oozie:hive-action:0.2">
    <job-tracker>${jobTracker}</job-tracker>
    <name-node>${nameNode}</name-node>
    <job-xml>${appPath}/hive-site.xml</job-xml>
    <configuration>
      <property>
        <name>mapred.job.queue.name</name>
        <value>${queueName}</value>
      </property>
    </configuration>
    <script>${appPath}/queries.hql</script>
  </hive>
  <ok to="pass"/>
  <error to="fail"/>
</action>
```

Oozie SLA

- A Service Level Agreement (SLA) is a standard part of a software contract that specifies the quality of the product in measurable terms
- Following are some SLA requirements that could be important for jobs running under Oozie control:
 - Have some of the job instances been delayed (relative to a time specified in the Coordinator), and how long were the delays?
 - Have some of the job instance execution times been outside specified limits, and how much were the deviations?
 - What is the frequency and percentage of violating the start time and execution time?
 - Did some job instances fail?

Oozie SLA

- Oozie SLA provides a way to define, store, and track the desired SLA information for Oozie activities
- Oozie activity can be tracked in different Oozie functional subsystems (for example, Coordinator and Workflow jobs and actions, Hadoop jobs submitted from Oozie, and so on)
- Oozie SLA can be specified in the scope of a Coordinator application, Workflow application, and Workflow action. The specification becomes part of (Coordinator or Workflow) application definition, and is expressed as a fragment of an XML document

Oozie SLA

- Enabling Oozie SLA (oozie-site.xml)

```
<property>
```

```
    <name>oozie.service.SchemaService.wf.ext.schemas</name>
```

```
    <value>oozie-sla-0.1.xsd </value>
```

```
</property>
```


Developing Custom Action

- Adding a new synchronous action type is highly discouraged
- The asynchronous execution model guarantees isolation between the action, the recommended way of writing a heavy-duty action is to use the asynchronous model
- The steps to develop custom action:
 - write a new action executor
 - writing an XML schema (XSD) file for the new action
 - deploy the new action type onto the Oozie server through the oozie-site.xml
 - write and submit the workflow using the new action type

Developing Custom Workflow

- The Oozie Workflow language enables to define Workflows that can be represented as Directed Acyclic Graphs (DAGs) of actions
- There is the approach that enables you to convert many loops and/or dynamic fork/joins to DAG
- Use cases:
 - loops case
 - dynamic fork/join construct
- Oozie Workflow language is an XML dialect that is defined by an XML schema, the programmatic generation of a Workflow is fairly straightforward
- Java Architecture for XML Binding (JAXB) to generate Java binding (based on the Workflow schema), and use these bindings as a Java API to create Oozie Workflows

Developing Custom EL Functions

- Oozie provides a bunch of built-in EL functions for most of the common use cases. However, users often feel the need for EL functions for new or special use cases
- It is highly recommended that the new EL function be simple, fast, and robust
- The new function should not perform any resource-intensive operation or be dependent on external systems and services
- The Oozie web application archive (oozie.war) should contain the new JAR supporting the new EL function

Developing Custom EL Functions

- **Writing a New EL Function**

the function should be a public static method in a Java class

```
public class CustomELFunctions {  
    public static String splitAndGet(String str, String expr, int pos) {  
    }  
}
```

Developing Custom EL Functions

- **Deploy the new EL function**

copy the custom JAR to the libext/ directory before executing the command `bin/oozie-setup.sh prepare-war`

oozie-site.xml

```
<property>
  <name>oozie.service.ELService.ext.functions.workflow</name>
  <value>splitAndGet=CustomELFunctions#splitAndGet</value>
</property>
```

- **Using the new function**

```
<arg>${splitAndGet("Apache Oozie!", " ", 2)}</arg>
```

Oozie REST API

- Oozie supports an HTTP-based REST API. This web service API is JSON-based and all responses are in UTF8.
- API is useful for interacting programmatically with Oozie
- Automating the monitoring of your Oozie jobs and building custom web UIs to render all that information
- Oozie's own internal services like the Oozie CLI, Oozie's web UI, and the Java client API use this REST API
- Example
 - <http://localhost:11000/oozie/v2/job/0000025-140522211231058-oozie-joe-C@80?show=info>

Oozie REST API

- `/v2/admin`
- `/v2/job`
- `/v2/jobs`
- `/v2/sla`
- `curl "http://localhost:11000/oozie/versions"`
- `curl "http://localhost:11000/oozie/v2/admin/status"`

Oozie Monitoring

Oozie provides multiple ways to monitor the jobs:

- Oozie web UI
- Polling REST API
- Oozie Email Action
- Callback URL
- JMS-based Monitoring (Job and SLA)

Oozie Performance Tuning

- **JVM Tuning**

```
oozie-env.sh - export CATALINA_OPTS="$CATALINA_OPTS -Xmx3072m"
```

- **Service Settings**

- **CallableQueueService**

The queue items are various callables, User actions like scheduling a coordinator or submitting a workflow gets translated into many callables internally

Oozie Performance Tuning

➤ CallableQueueService Settings

`oozie.service.CallableQueueService.queue.size (10000)`

`oozie.service.CallableQueueService.threads (10)`

`oozie.service.CallableQueueService.callable.concurrency (3)`

➤ Check the current size of the queue and list all the items that are occupying the queue

`oozie admin -queuedump | wc -l`

`oozie admin -queuedump`

Oozie Performance Tuning

➤ RecoveryService

Oozie has implemented a recovery service, which keeps an eye on the jobs and the queue and recovers actions and jobs that appear to be hung or lost in space. The service itself runs every 60 seconds and looks for actions older than the configured number of seconds

➤ RecoveryService Settings

`oozie.service.RecoveryService.interval`

`oozie.service.RecoveryService.wf.actions.older.than (120)`

`oozie.service.RecoveryService.coord.older.than (600)`

`oozie.service.RecoveryService.bundle.older.than (600)`

Oozie Reprocessing

There are three scenarios when a user needs to rerun the same job:

- The job failed due to a transient error
- The job succeeded but the input data was bad
- The application logic was flawed or there was a bug in the code

Oozie provides convenient ways to reprocess completed jobs through the CLI or the REST API. Reprocessing is driven through the `job -rerun` subcommand and option of the Oozie CLI

Oozie Reprocessing

- Workflow Reprocessing
 - `oozie.wf.rerun.skip.nodes`
 - `oozie.wf.rerun.failnodes`
 - ✓ `oozie job -rerun 0000092-141219003455004-oozie-oozi-W -config job.properties -Doozie.wf.rerun.failnodes=false`
 - ✓ `oozie job -rerun 0000092-141219003455004-oozie-oozi-W -config job.properties -Doozie.wf.rerun.skip.nodes=node_A,node_B`
- Coordinator Reprocessing
- Bundle Reprocessing

Oozie High Availability

- Oozie High AvailabilityThe multiple Oozie servers are usually fronted by a software load-balancer or a virtual IP or a DNS round-robin solution so that the Oozie CLI or the REST API can use a single address to access the server
- The DB runs on other machines
- Oozie HA is built on top of Apache Zookeeper to coordinate the logs fetch

Oozie Application Deployment

Simple App Deployment

```
./my_app_dir
bundle.xml
coordinator.xml
workflow.xml
hive_qry.hql
lib/
    my_jar.jar
```

```
./my_app_dir
bundle_1/
    bundle.xml
    coord_1/
        coordinator.xml/
        workflow_1/
            workflow.xml
            hive_qry.hql
            lib/
                my_jar.jar
        workflow_2/
            ...
    coord_2/
    ...
bundle_2/
...
```

Shared App Deployment

```
./my_app_dir
shared_coord/
    shared_wf_1/
    ...
    shared_wf_2/
    ...
    common_lib/
    ...
    bundle_1/
        [shared_coord]
        coord_1/
            [shared_wf_1]
            ...
        coord_2/
            [shared_wf_1]
            ...
    bundle_2/
        [shared_coord]
        coord_3/
        ...
    ...
    ...
```

Testing Oozie

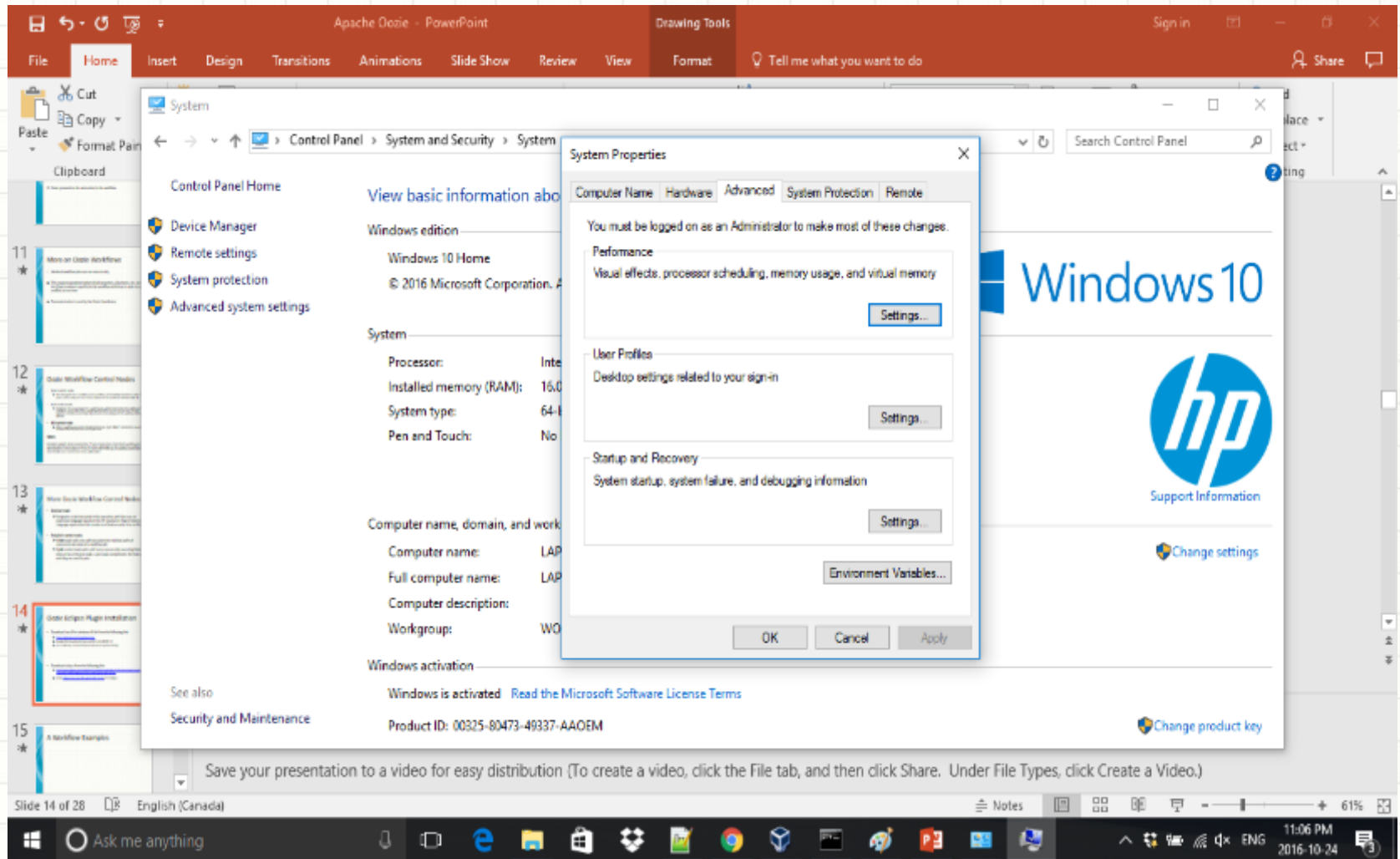
There are ways to test and verify Oozie applications locally in a development environment

- MiniOozie - JUNIT test class for testing workflow and coordinator applications
- LocalOozie - Embedded Oozie to simulates an Oozie deployment locally

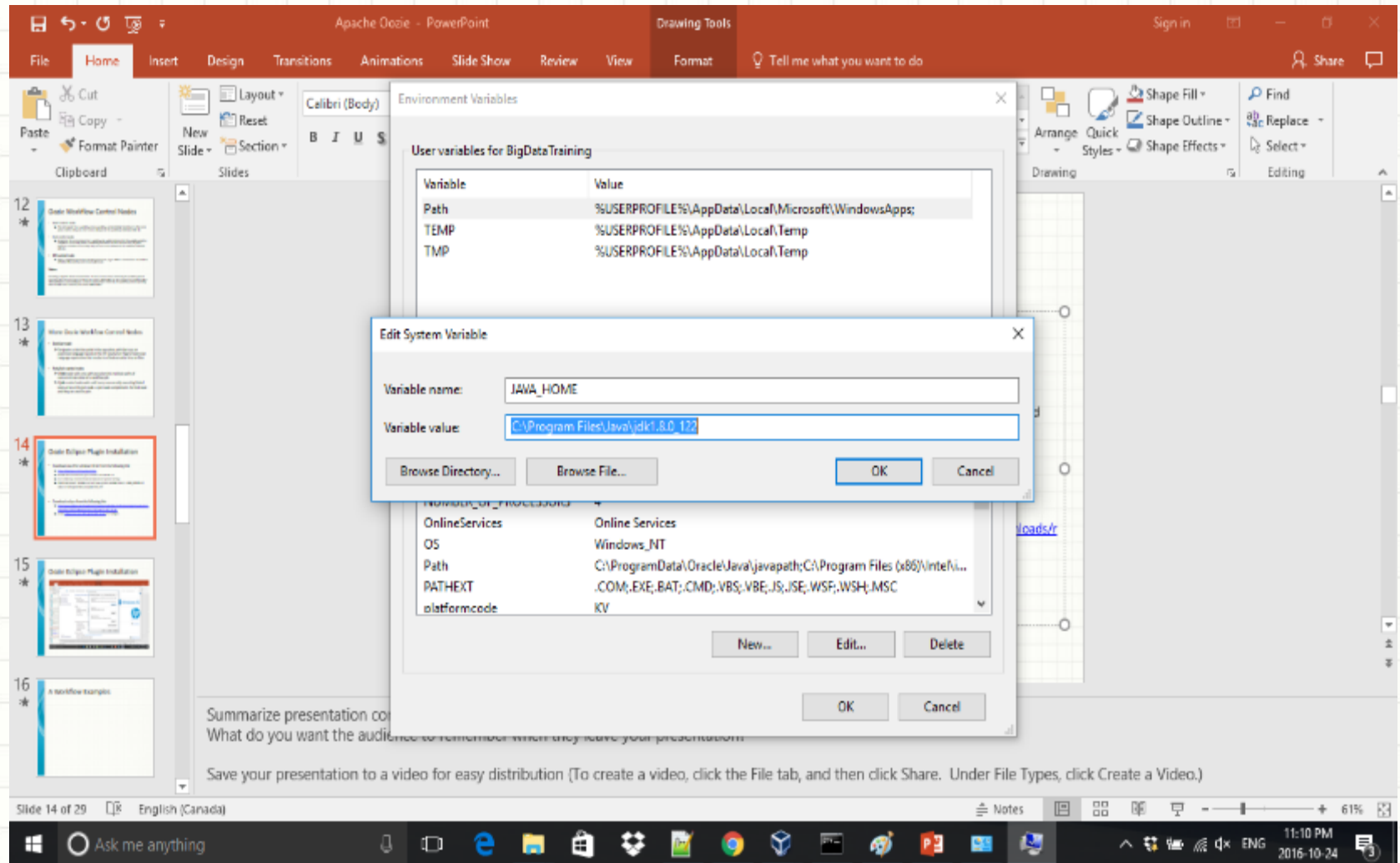
Oozie Eclipse Plugin Installation

- Download Java 8 for windows 64 bit from the following link:
 - <https://jdk8.java.net/download.html>
 - Double click downloaded java exe file to install JDK 1.8
 - Go to Windows Control Panel and Advanced System Settings
 - Click Environment Variables and add new system variable name is JAVA_HOME and value is C:\Program Files\Java\jdk1.8.0_122
- Download eclipse from the following link:
 - http://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/luna/SR2/eclipse-java-luna-SR2-win32-x86_64.zip
 - Unzip [eclipse-java-luna-SR2-win32-x86_64.zip](#) to C:\eclipse
 - Go to C:\eclipse and double click eclipse.exe

Oozie Eclipse Plugin Installation



Oozie Eclipse Plugin Installation



Oozie Eclipse Plugin Installation

- Go to the following link in your chrome browser:
 - <https://marketplace.eclipse.org/content/oozie-eclipse-plugin>
- Hover over the install icon and drag it, move to **Eclipse** window and drop the icon there, Wait till the **Eclipse Marketplace Client** window shows up.
- if you are experiencing the following error:
 - Cannot complete the install because one or more required items could not be found.
Software being installed: Oozie Eclipse Plugin 0.1.0.201508312352
(com.mashin.oep.feature.feature.group 0.1.0.201508312352)
 - Go to **Help** -> **Install New Software...**

Oozie Eclipse Plugin Installation

- Go to **Help -> Install New Software...**
- Add this site "<http://download.eclipse.org/tools/gef/updates/releases/>", select "GEF (Graphical Editing Framework" and continue GEF installation.
- Re-install Oozie Eclipse Plugin
- Select the plugin and press **Confirm >**
- Choose "**I accept the terms of the license agreement**" and press **Finish**
- When you asked "**Do you trust these certificates?**", check-mark the certificate "**Ahmed Mahran; Oozie Eclipse Plugin; Mashin**" and press **OK**
- Restart Eclipse to be able to use the plugin

Oozie Eclipse Plugin Installation

The screenshot shows a web browser window with multiple tabs. The active tab is titled "Oozie Eclipse Plugin" and displays the URL <https://marketplace.eclipse.org/content/oozie-eclipse-plugin>. The page content includes a breadcrumb trail: HOME / MARKETPLACE / TOOLS (1635) / OOZIE ECLIPSE PLUGIN. On the left sidebar, there are sections for "MARKETS", "SEARCH" (with a search bar and "ADVANCED SEARCH" button), and "MORE LIKE THIS" (listing related plugins like Splunk Plug-In for Eclipse, WSO2 Developer Studio, Eclipse Java EE Developer Tools, and Progress DataDirect Connect for JDBC Type 5 Drivers). The main content area is titled "Oozie Eclipse Plugin" and features the Oozie logo, a star rating of 0, and an "Install" button. Below the logo are social media icons. A tabbed interface shows the "Details" tab selected, containing the following text: "An Eclipse plugin for editing Apache Oozie workflows. Follow instructions at this blog post (<http://blog.oep.mashin.io/2015/09/install-oozie-eclipse-plugin.html>) to install Oozie Eclipse Plugin. For further information, visit: - The plugin website: <http://oep.mashin.io/> - The plugin blog: <http://blog.oep.mashin.io/> To report an issue or request a feature, please create a new issue at <https://github.com/mashin-io/oep/issues> Categories: Editor Tags: oozie, apache oozie, workflow, Big Data, apache". At the bottom of the main content area is a button labeled "ADDITIONAL DETAILS". The Windows taskbar at the bottom shows the system clock as 11:13 PM on 2016-10-24, along with various application icons and a search bar.

HOME / MARKETPLACE / TOOLS (1635) / OOZIE ECLIPSE PLUGIN

MARKETS »

SEARCH

Search

ADVANCED SEARCH »

SEARCH

MORE LIKE THIS

- Splunk Plug-In for Eclipse
- WSO2 Developer Studio
- Eclipse Java EE Developer Tools
- Progress DataDirect Connect for JDBC Type 5 Drivers

Oozie Eclipse Plugin

Details | Screenshots | Metrics | Errors | External Install Button

0 stars 0 comments

Install

An Eclipse plugin for editing Apache Oozie workflows.

Follow instructions at this blog post (<http://blog.oep.mashin.io/2015/09/install-oozie-eclipse-plugin.html>) to install Oozie Eclipse Plugin.

For further information, visit:

- The plugin website: <http://oep.mashin.io/>
- The plugin blog: <http://blog.oep.mashin.io/>

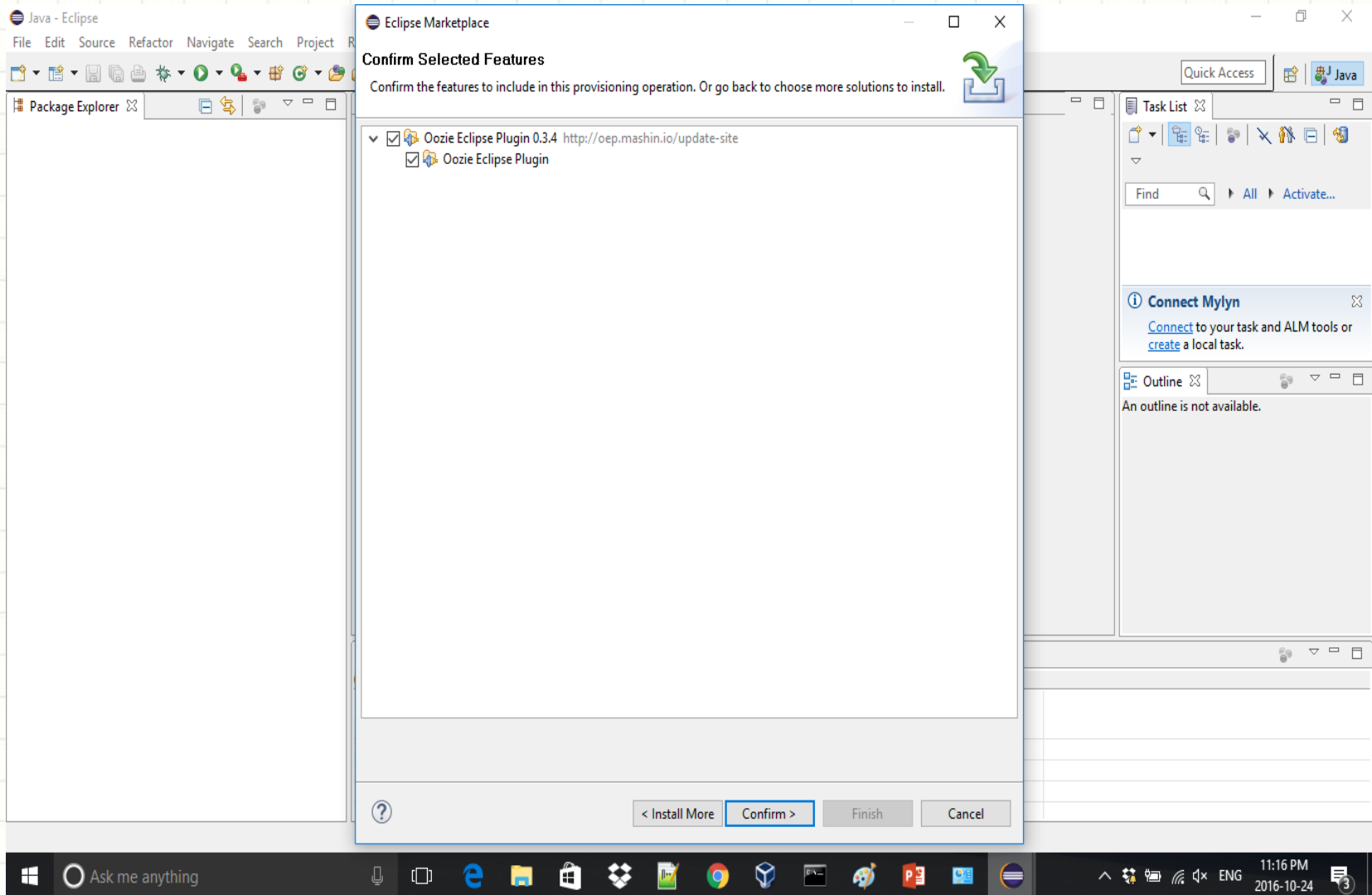
To report an issue or request a feature, please create a new issue at <https://github.com/mashin-io/oep/issues>

Categories: Editor

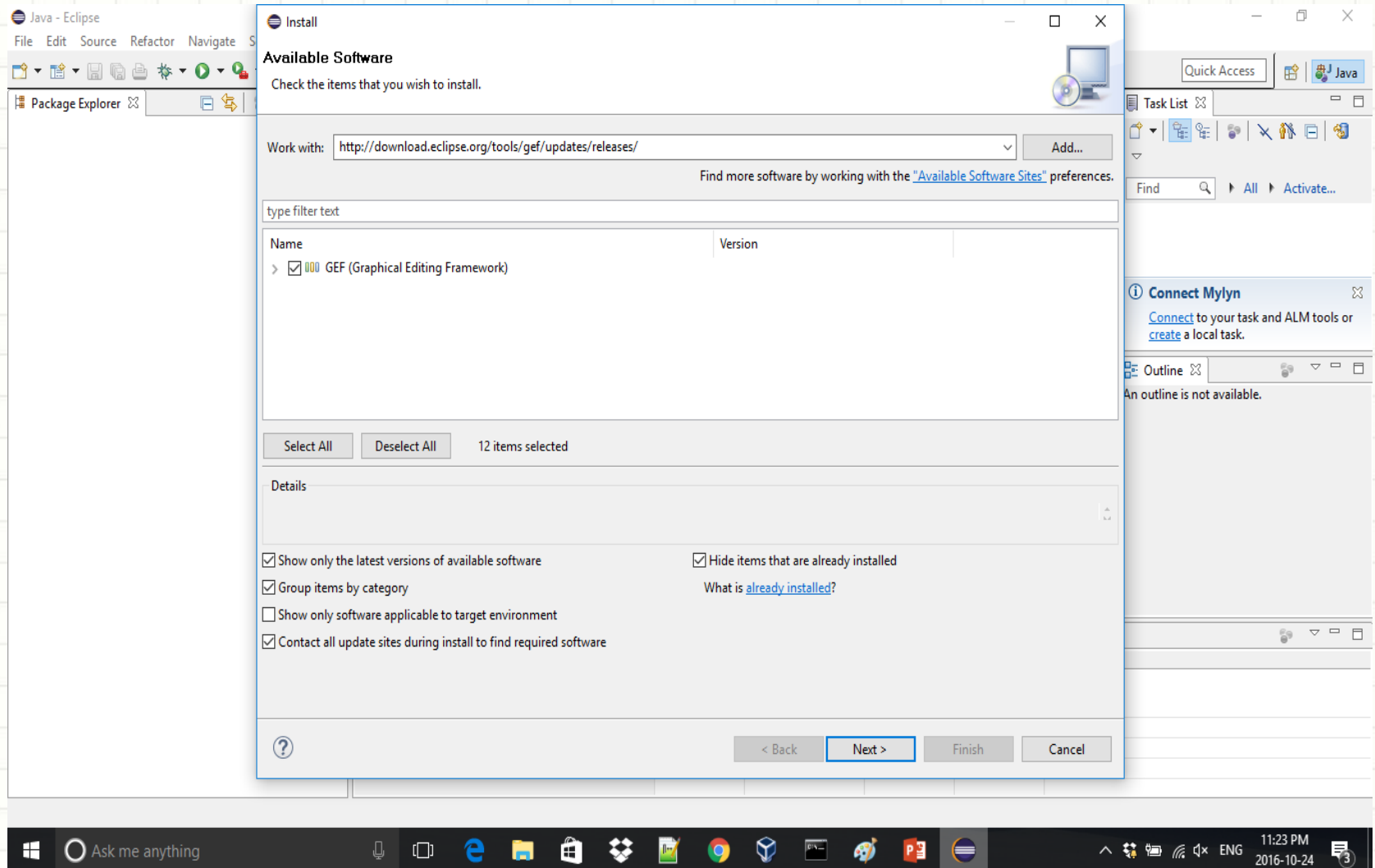
Tags: oozie, apache oozie, workflow, Big Data, apache

ADDITIONAL DETAILS

Oozie Eclipse Plugin Installation



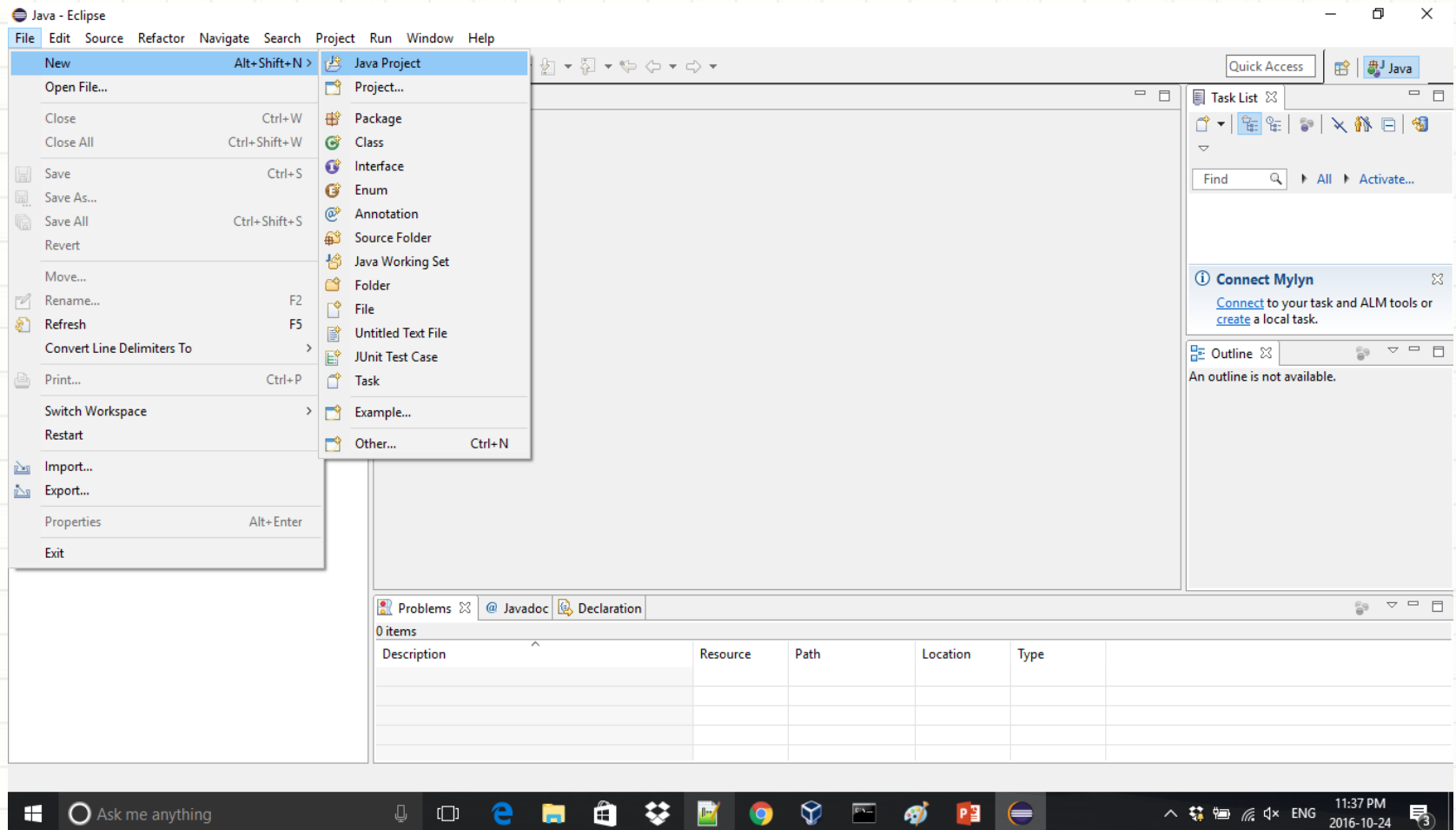
Oozie Eclipse Plugin Installation



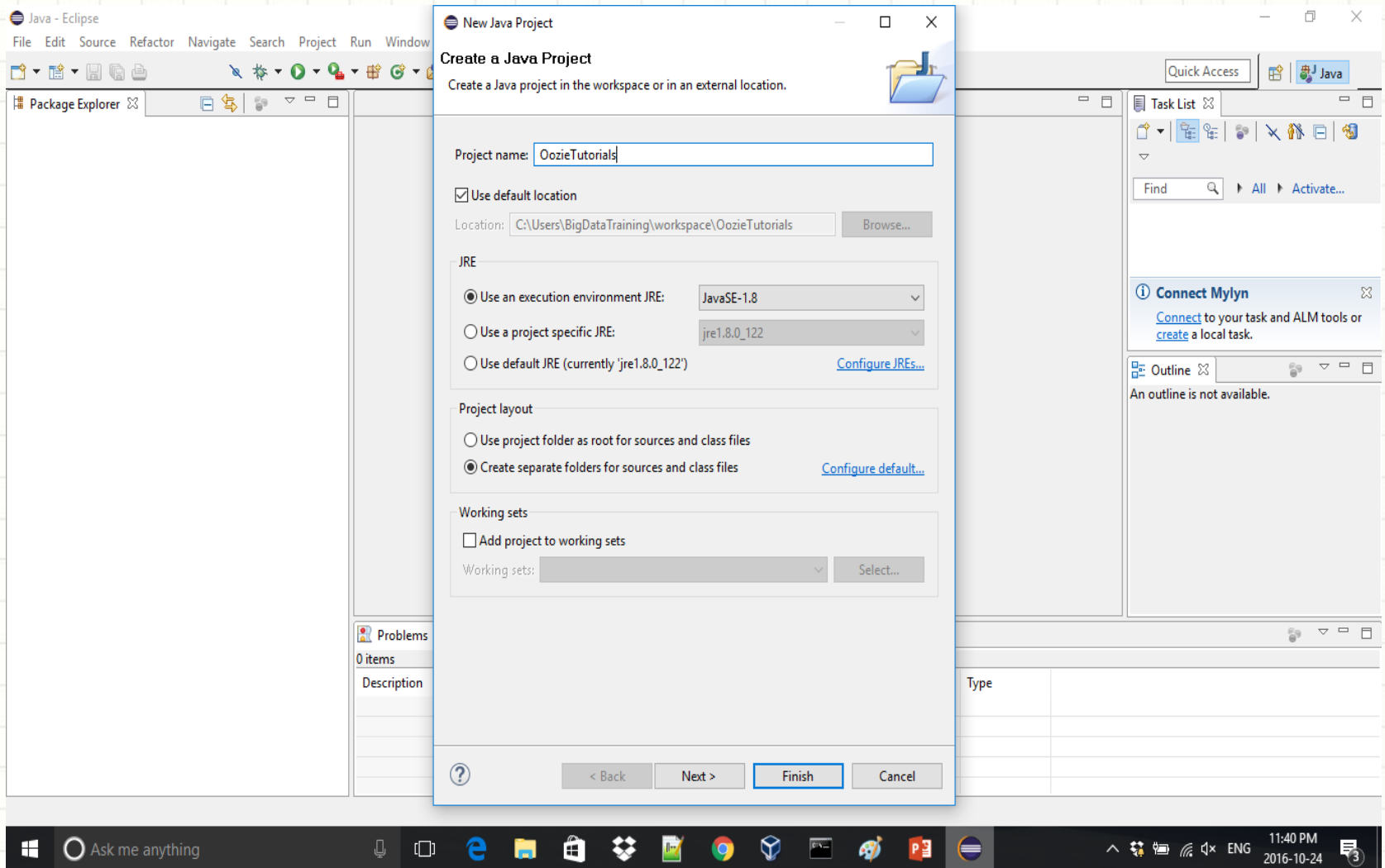
Getting started with Oozie Eclipse Plugin

- Create a new workflow
 - Go to File -> New -> Project -> Java Project
 - Create the project called "OozieTutorials"
 - Create new java package called "oozie"
 - Create new file Under "Apache Oozie" category, select "Apache Oozie Workflow" wizard and press Next
 - A new file will be created and opened in the workflow editor showing a start and an end nodes

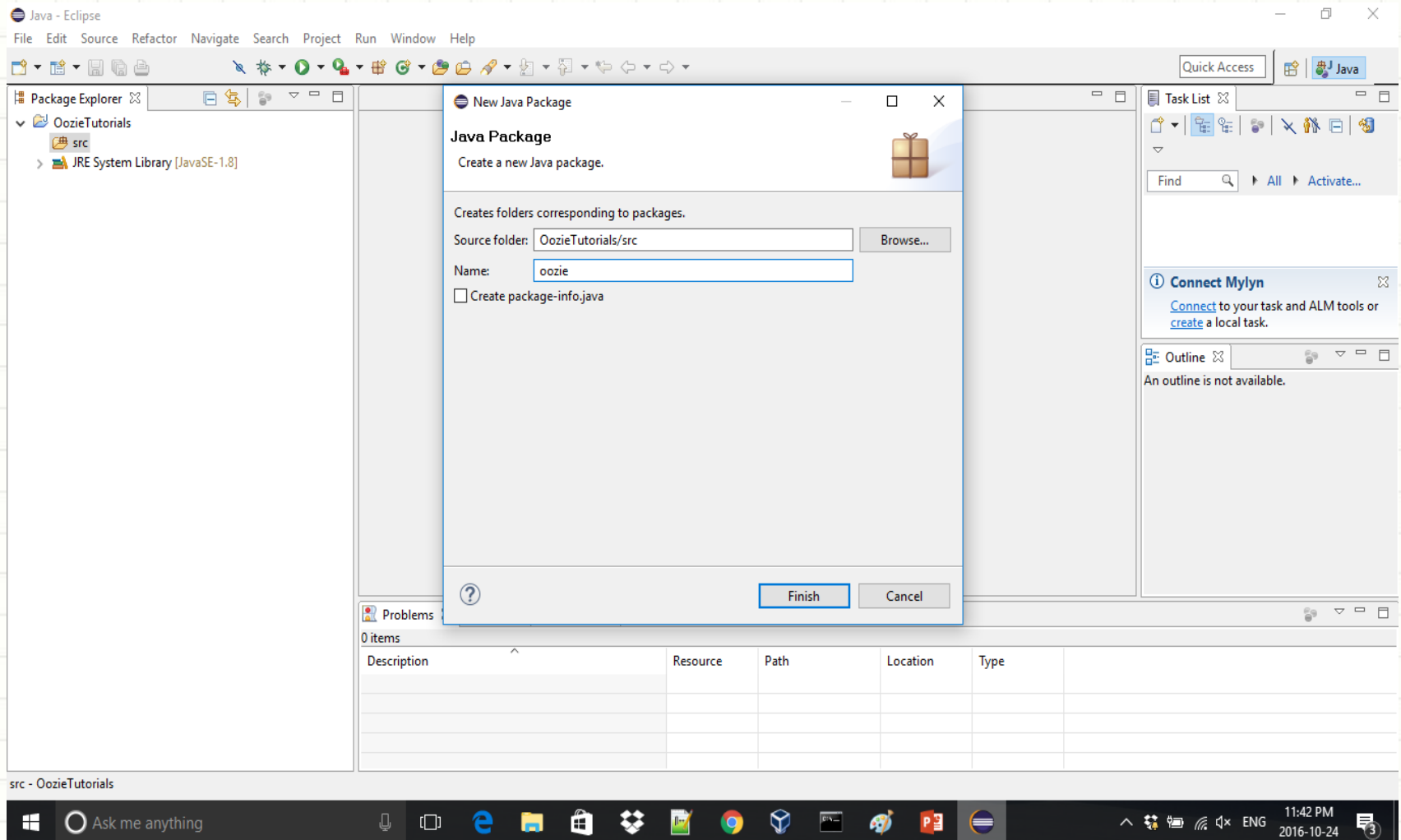
Getting started with Oozie Eclipse Plugin



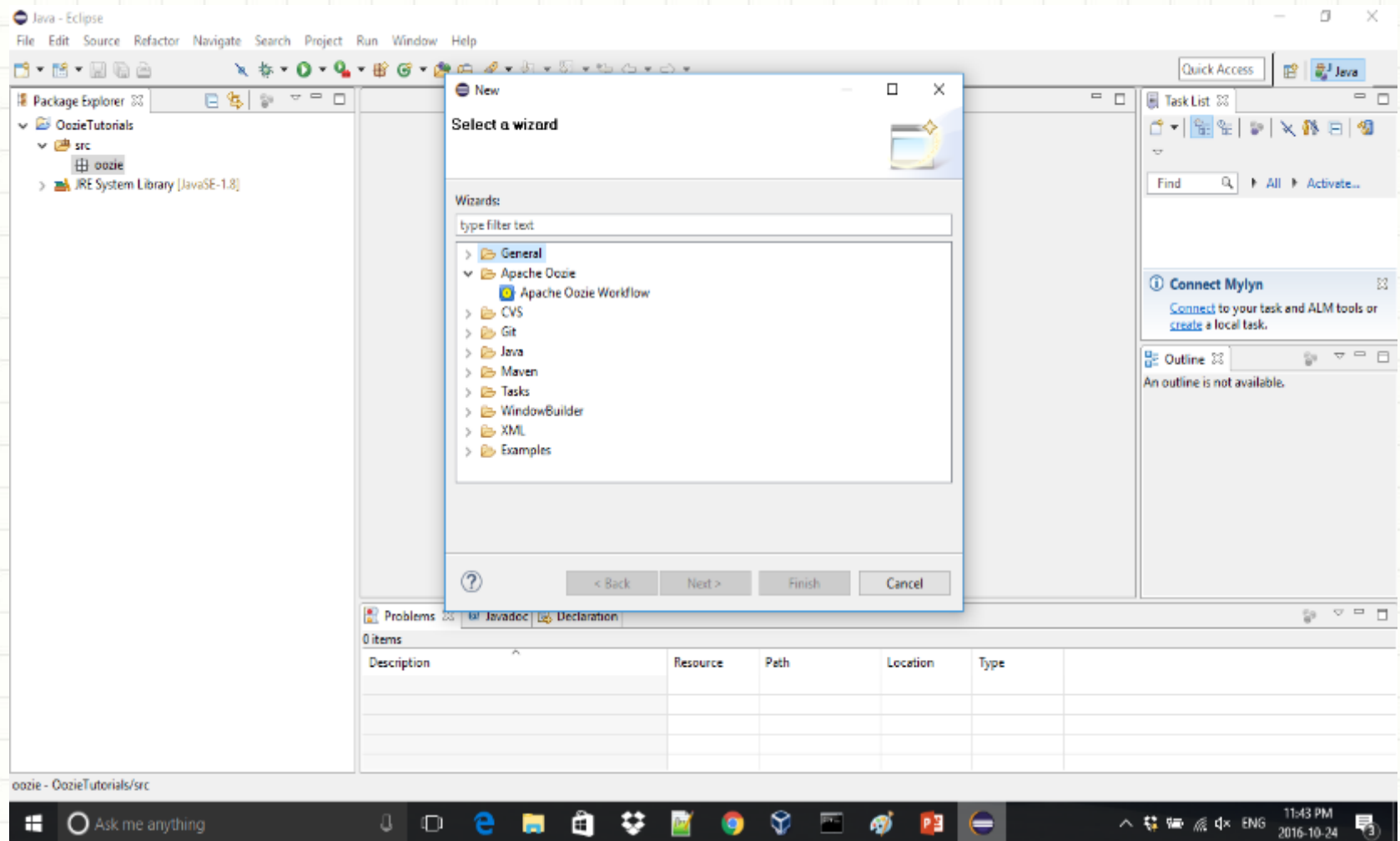
Getting started with Oozie Eclipse Plugin



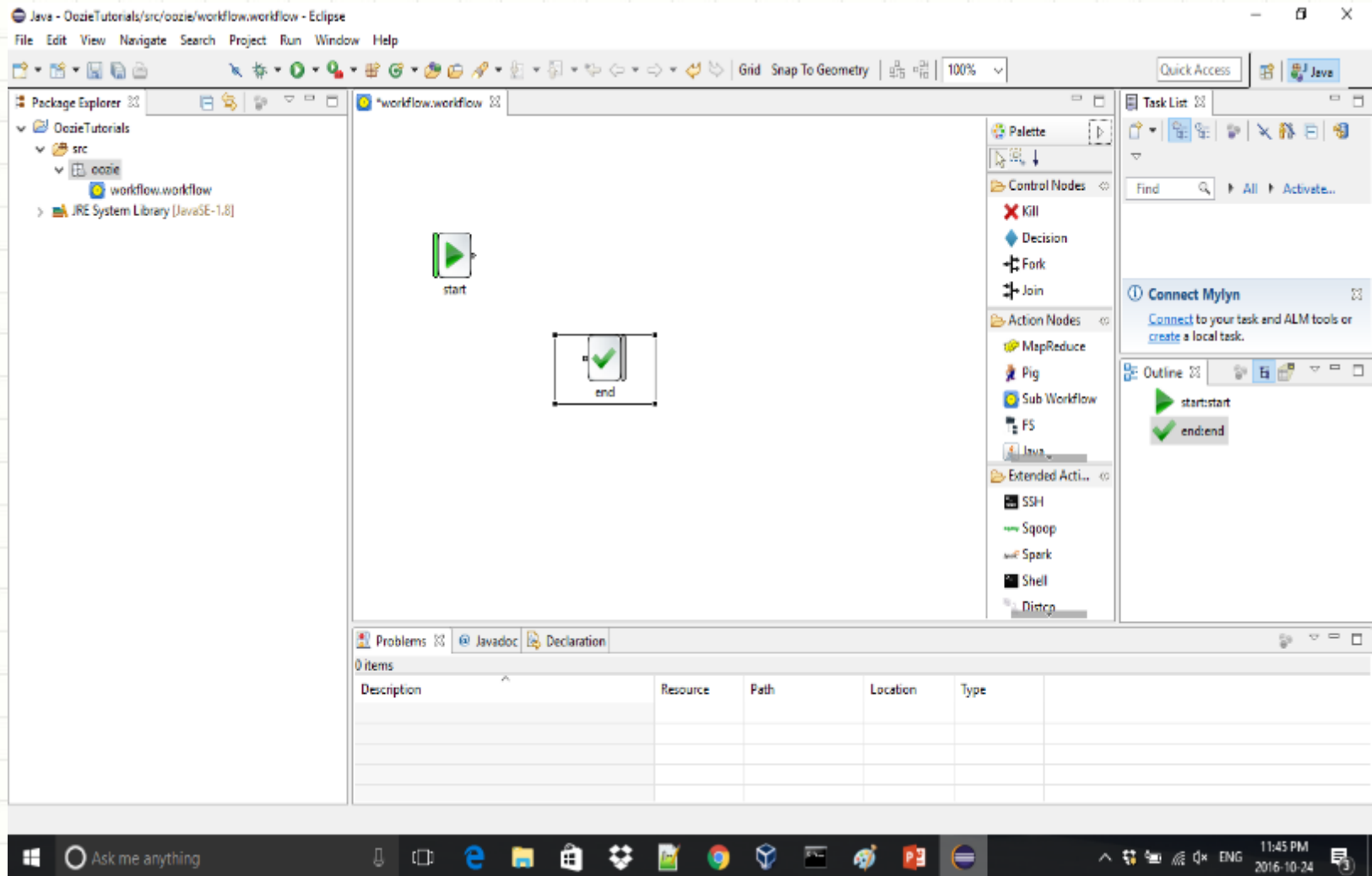
Getting started with Oozie Eclipse Plugin



Getting started with Oozie Eclipse Plugin



Getting started with Oozie Eclipse Plugin



A Workflow Examples

Java - OozieTutorials/src/oozie/oozie_in_action.xml - Eclipse

File Edit View Navigate Search Project Run Window Help

Package Explorer

- OozieTutorials
 - src
 - oozie
 - coordinator.properties
 - job.properties
 - loadintohive.hql
 - movefiles.sh
 - oozie_in_action_coord.xml
 - oozie_in_action.xml
 - oozielab.bak.txt
 - oozielab.txt
 - preparemysql.sh
 - preparemysql.sql
 - ShellActionExample.xml
 - wgetfiles.sh
 - JRE System Library [JavaSE-1.8]

oozie_in_action.xml

Workflow Diagram:

- Start node (green triangle) connects to WGetFiles (shell node).
- WGetFiles connects to PrepareMySQL (shell node).
- PrepareMySQL connects to SqoopImportFromMySQL (sqoop node).
- SqoopImportFromMySQL connects to Fork (fork node).
- Fork splits into two parallel paths:
 - Path 1: CreatePartitionedFolder (shell node) → MoveFiles (shell node) → ForkJoin (join node).
 - Path 2: Kill (kill node).
- The Kill node connects to the main Kill node (kill node) at the bottom.
- The main Kill node connects to the end of the workflow.

Palette

- Control Nodes
 - Kill
 - Decision
 - Fork
 - Join
- Action Nodes
 - MapReduce
 - Pig
 - Sub Workflow
 - FS
 - Java
- Extended Actions
 - SSH
 - Sqoop
 - Spark
 - Shell
 - Distcp

Task List

Find: All Activate...

Connect Mylyn

Connect to your task and ALM tools or create a local task.

Outline

- start:start
- kill:kill
- WGetFiles:shell
- PrepareMySQL:shell
- SqoopImportFromMySQL:sqoop
- CreatePartitionedFolder:fs
- MoveFiles:fs
- Fork:fork

Properties

Property	Value

Windows Taskbar: 1:29 AM 2016-10-29

A More Complex Workflow Example

```
C:\Users\BigDataTraining\workspace\OozieTutorials\src\oozie\oozie_in_action.xml - Notepad++
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?

webistes bt x ApacheHive bt x oozielab bt x oozie_in_action_coord.xml x coordinator.properties x oozie_in_action.xml x

1 <?xml version="1.0" encoding="UTF-8"?>
2
3 <workflow-app xmlns:sla="uri:oozie:sla:0.2" xmlns="uri:oozie:workflow:0.5" name="Oozie In Action">
4   <start to="WGetFiles"/>
5   <kill name="kill">
6     <message>Killed job due to error</message>
7   </kill>
8   <action name="WGetFiles">
9     <shell xmlns="uri:oozie:shell-action:0.3">
10      <job-tracker>${jobTracker}</job-tracker>
11      <name-node>${nameNode}</name-node>
12      <exec>wgetfiles.sh</exec>
13      <file>/user/root/oozie/scripts/wgetfiles.sh#wgetfiles.sh</file>
14      <capture-output/>
15    </shell>
16    <ok to="PrepareMySQL"/>
17    <error to="kill"/>
18  </action>
19  <action name="PrepareMySQL">
20    <shell xmlns="uri:oozie:shell-action:0.3">
21      <job-tracker>${jobTracker}</job-tracker>
22      <name-node>${nameNode}</name-node>
23      <exec>preparemysql.sh</exec>
24      <file>/user/root/oozie/scripts/preparemysql.sh#preparemysql.sh</file>
25      <file>/user/root/oozie/scripts/preparemysql.sql#preparemysql.sql</file>
26      <capture-output/>
27    </shell>
28    <ok to="SqoopImportFromMySQL"/>
29    <error to="kill"/>
30  </action>
31  <action name="SqoopImportFromMySQL">
32    <sqoop xmlns="uri:oozie:sqoop-action:0.4">
33      <job-tracker>${jobTracker}</job-tracker>
34      <name-node>${nameNode}</name-node>
35      <command>job -Dmapred.java.child.opts=1024 -Dmapreduce.map.memory.mb=1024 -Dmapreduce.map.java.opts=-Xmx1024m -Dmapreduce.task.io.sort.mb=1024 -Dhadoop.se
36    </sqoop>
37    <ok to="Fork"/>

```

eXtensible Markup Language file length: 3084 lines: 87 Ln: 1 Col: 1 Sel: 0 | 0 UNIX UTF-8 INS

Windows Taskbar: Ask me anything, File Explorer, Chrome, etc. 1:33 AM 2016-10-29

Summary

- Oozie is a workflow scheduler system that manages a wide range of Hadoop-related jobs:
 - MapReduce (including streaming MR), Spark, Sqoop, Pig, Hive, Shell Scripts, Java etc.
- Arranging workflow actions in Directed Acyclic Graph helps prevent infinite action execution loops
- Oozie flow controls help manage execution paths of workflows using start, fork, join, decision and end nodes
- Oozie Coordinator allows users to schedule the execution of workflow jobs