



ANALYTICS WORKSHOP

SOFIA2

December 2016

Versión 1



The goal of this workshop is to create a recommendation system based on user ratings, based on one of the exercises proposed at the [Spark Summit](#).

We'll use one of the **Movielens** datasets already reside on the platform. We'll do it in three steps:

- Ingestion and data preparation using Pipelines.
- Creating the model using a Notebook.
- Creating a simple display.

Data Ingestion

Pipeline Creation

We'll perform the data ingestion of films with the Dataflow. First thing to do is to create a Pipeline from scratch. Go to 'Analytics' menu and click on 'My Pipelines'. Click on 'Create'. Write the the name of the Pipeline and a description. The timer does not apply for this practice:

Create Pipeline

Identification

Description

Timer

Generate

CreateCancel

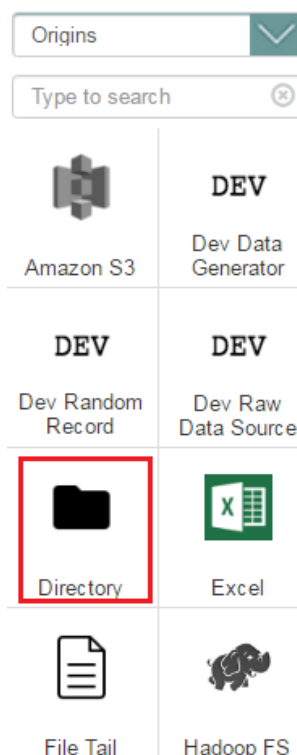
When you create the pipeline, access to workspace where you will create the information flow.

Source Component Definition

Data is already downloaded on Sofia2 machine. Depending on the environment, we'll find it in one route or another. The route is `"/datadrive/movielens"` for `sofia2.com`, while `"/datadrive/ftp/movielens"` is for `sofia2-analytic`. In this directory there should be two files: *movies.dat* and *ratings.dat*. To do this pipeline we are interested in the data of the films.

If they were not in the machine, you have to download them for this workshop.

First of all, you need to create the Data Source. Since the files already reside on Sofia2 machine, the required component is 'Directory'. Click on the component and it will appear in the workspace. You'll see error alerts. Don't worry, when you create the empty component, the required configuration parameters are empty. That is just what to do in the next step.



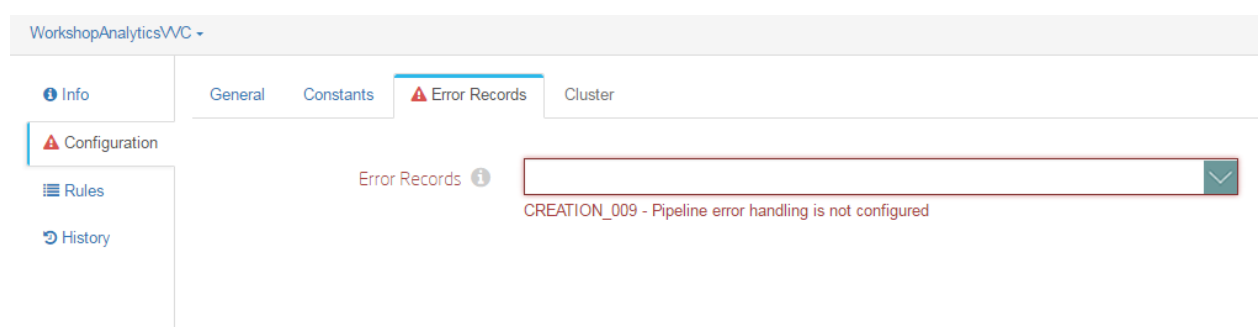
Click on the component and you will access its configuration. For the local directory source, the required configuration parameters are:

- **Files → Data Format:** Represents the input data format. There are different options, but the one needed in this example is 'Text'.
- **Files → Files Directory:** It is the input directory where the files to read reside. In our case, this route is /datadrive/ftp/movielens. (If you work from Sofia2.com/console the route is /datadrive/movielens).
- **Files → Name Pattern:** It is the regular expression with you will search the files to load within the directory configured in the previous parameter. We are interested in reading a single file, so we have to assign this field 'movies.dat'.

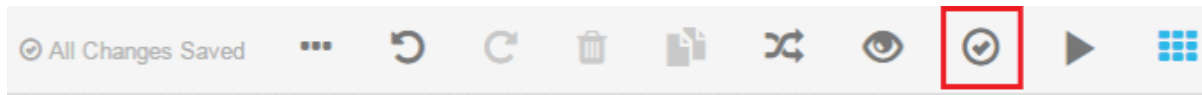
Depending on the input format chosen, the corresponding tab is activated in the configuration window. You will see that, in this case, the active tab is 'Text'. It only has one parameter ('Max Line Length') which has a default value that we won't modify.

Now you have the source configured. To start, it is highly recommended take a look at the data to be read. To do this, we can configure a "Dummy" destination and preview the information. Access to 'Destinations' components and choose 'Trash'. As before, when you click on the icon, the component appears in the workspace. Connect origin and destiny, and this flow is almost ready.

As you'll notice, there are still configuration errors. This is because in the general configuration it is necessary to define the management of erroneous records. Click on any site that is not a component within the workspace. The bottom window will show the general settings, and you will see the alert appears in the 'Error Records' tab.

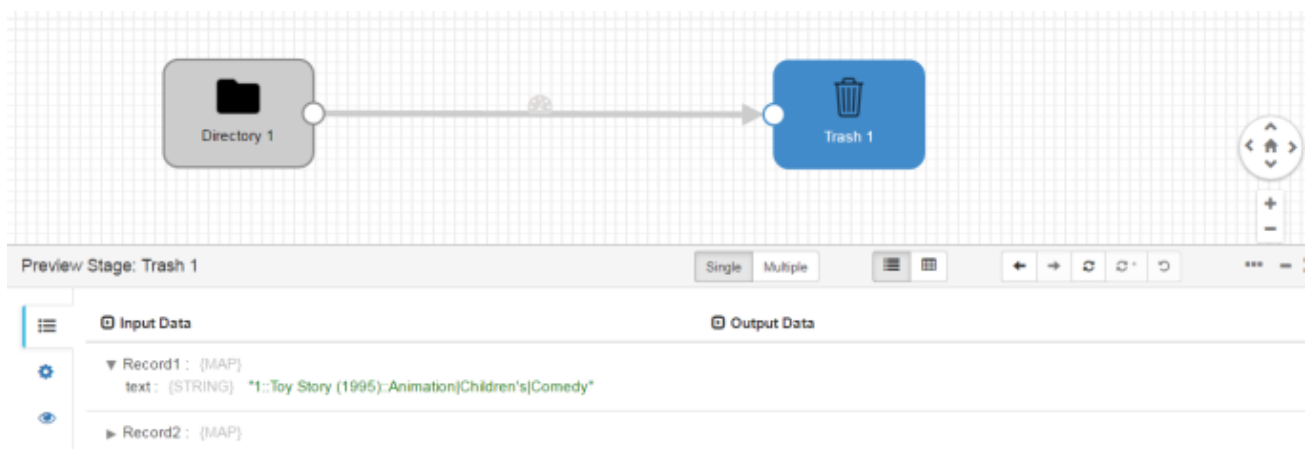


Choose 'Discard' inside this options. With this, there should be no errors, but still, we will validate the flow. Click on 'Validate' button, at menú on top:



If everything is correct, it will display an OK message.

Now we can do the preview. The button just on the left of 'Validate' is 'Preview'. Click on it and a window will appear with a configuration data. Look if 'Write to destinations' is checked. If it is checked, in addition to previewing the data, it will write the data in destination. Uncheck it if it's checked and click on 'Run Preview':

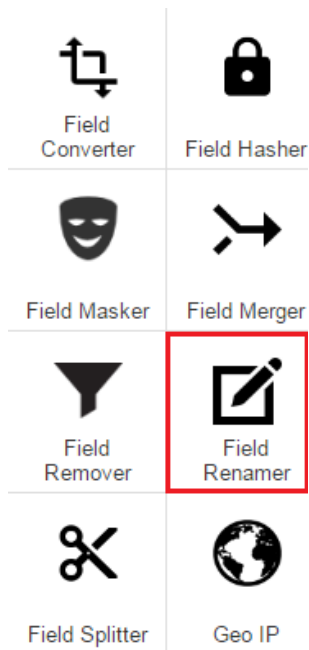


You can see in 'input data' what you read in each record and in each of the components. If you click on the 'directory' component, you will see what it generates and if you click on 'Trash' what it receives. In this case it's the same.

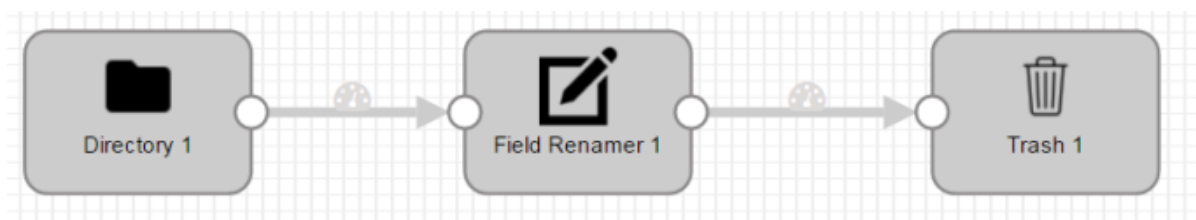
Data Processing

Now we are going to prepare the data. As you can see in the 'preview' of previous step, fields are separated by '::.'. Dataflow interprets separators as a single character, so it cannot be defined as a delimiter '::.'. That will be the next step..

We will include, before the delimiter change, a field renamer. In the 'preview', when displaying each record, appear the defined fields. By reading as 'Text' format, for each line, a field is called 'text', and is generated by default. This is the one we are going to rename. To do this, within 'Processors', click on 'Field Renamer'.



Create a stream like this:



We are going to configure it. This component is very simple. Click on it, and in the configuration go to the 'Rename' tab. In 'Fields to Rename' you have to enter the source field and the name to change it. Write in 'From Field' '/text' and in 'To Field' '/data'.

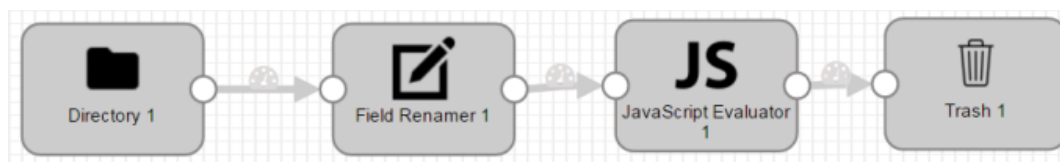
Fields to Rename 	From Field 	To Field 
	<input type="text" value="/text"/>	<input type="text" value="/data"/>  

You can try previewing to verify you are effectively renaming the field.

Now we can create the component that replaces the delimiter. To carry out this task you can use different processors, specifically all those who are 'Evaluators'. We'll do it with JavaScript.

 Groovy Evaluator	 HBase Lookup
 Hive Metadata	 JavaScript Evaluator
 JSON Parser	 Jython Evaluator
 Log Parser	 Record Deduplicator

Click on the component and create a flow like the following:

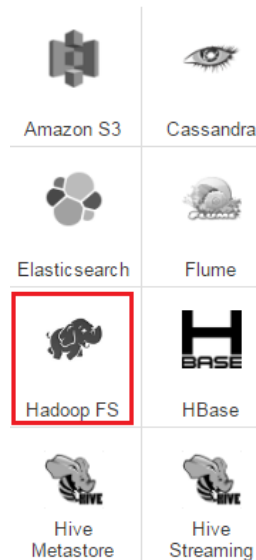


Access to the component configuration, and go to 'Javascript' tab. You will see a text editor called 'script', which already has predefined code inside. It's the template on which we will define our changes. Inside of the 'for' loop, add the following line of code:

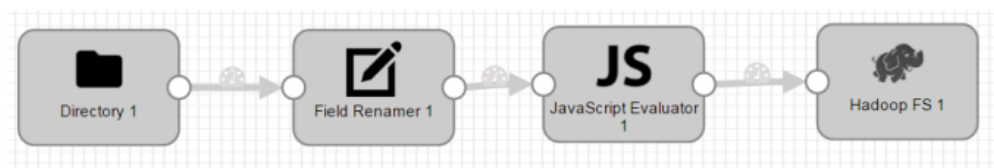
```
records[i].value['data'] = records[i].value['data'].replace(/::/g, "%");
```

This line replace '::' for '%'. We have chosen this delimiter because the typical ones that are usually ';', ',' and '|' appear in the dataset as part of the fields. Launch the 'preview' again and verify the change has been made correctly.

Destination Component Definition



Click on the component and create a flow like the following:



Access to the settings of destination. You have to modify 3 tabs:

Hadoop FS: Corresponds to HDFS connections and routes.

- **Hadoop FS URI**: Hadoop FS URI: `hdfs://localhost:8020`
- Hint: If we're doing the workshop in Sofia2.com/console we have to change '`localhost:8020`' by '`sofia2-hadoop.cloudapp.net:8020`'
- **HDFS User**: `cloudera-scm`

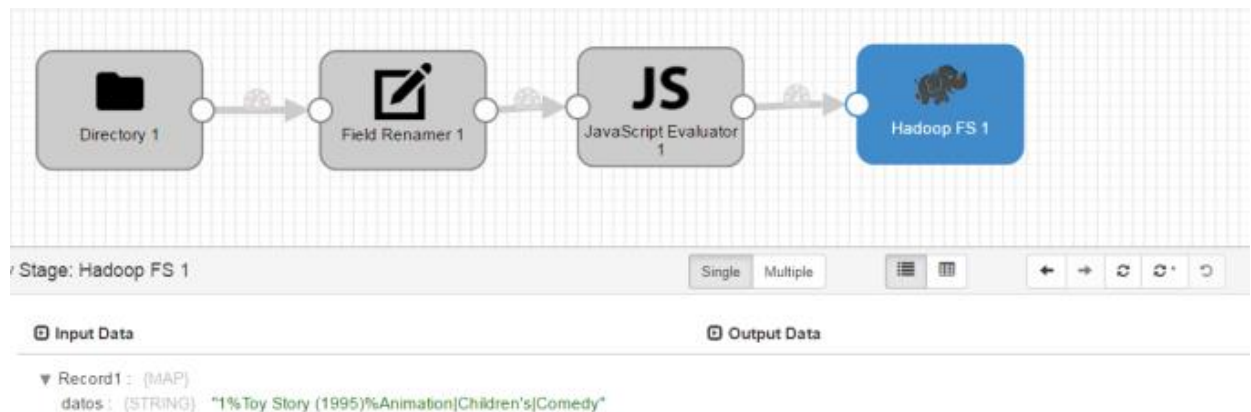
Output Files: This is the definition of the output files, routes, format, etc.

- **File Type**: Text Files
- **Data Format**: Text
- **Files Prefix**: `movie`
- **Directory Template**: `/user/cloudera-scm/movielens/student_alias/`

Text: It is the configuration of the format chosen in the previous tab.

- **Text Field Path:** /data

Launch the 'preview' again and verify that the data arrives correctly to the destination.



If everything seems correct, click on the 'Start' button, to the right of the validation button you used previously. You will see that another window opens with the statistics of the data being read, processing times of each component, etc. When you see you are no longer reading data, it means you have already run through all the input files. As we don't need more data than those, we can stop the pipeline.

Would you know to do the same for the 'Ratings' file?

Would you know generate the file in the HDFS as delimited, defining the names of the fields separated by ';'?

Note: If you generate the semicolon file as a delimiter, keep in mind that in the next steps you will have to use that same delimiter instead of the "%" as it appears in the document.

NOTEBOOK

With the help of Sofia2 notebooks we are going to generate the movie recommendation model using the data we uploaded on the platform in the previous exercise. We propose to carry it out with Spark using Scala, and more concretely, we'll implement the ALS.

Input data paths Definition

The first step is reading movie data and ratings, and, to do this, you have to define the data path. Define `ratings_path` and `movies_path` variables with the corresponding paths where you have loaded to the platform. For example:

Downloaded data paths Definition

```
val ratings_path = "hdfs://sofia2-analytic:8020/user/cloudera-scm/movielens/ratings*"
val movies_path = "hdfs://sofia2-analytic:8020/user/cloudera-scm/movielens/movies*"
```

Tip: If we do this workshop at [Sofia2.com/console](https://sofia2.com/console) we have to change 'sofia2-analytic:8020' by 'localhost:8020'

Structure the data

The next thing to do is to save movie information and ratings. We'll read this information through Spark RDDs

You need to define a specific format for both movies: (movieId, movieName) and rating: (timestamp % 10, Rating (userId, movieId, rating)).

We also took advantage of importing Mlib libraries we will use in the example. In particular, you need **ALS**, **Rating** y **MatrixFactorizationModel**.

Save data in Rdd

```
import org.apache.spark.mllib.recommendation.{ALS, Rating, MatrixFactorizationModel}

val movies = sc.textFile(movies_path).map { line =>
  val fields = line.split("%")
  // format: (movieId, movieName)
  (fields(0).toInt, fields(1))
}.collect.toMap

val ratings = sc.textFile(ratings_path).map { line =>
  val fields = line.split("%")
  // format: (timestamp % 10, Rating(userId, movieId, rating))
  (fields(3).toLong % 10, Rating(fields(0).toInt, fields(1).toInt, fields(2).toDouble))
}
```

Data checks

Now, check that the data has been read. How many ratings did you download? How many films are in the catalog? How many movies have been scored? And how many users have done it?

```
val numRatings = ratings.count
val numUsers = ratings.map(_._2.user).distinct.count
val numMovies = ratings.map(_._2.product).distinct.count

println("Got " + numRatings + " ratings from "
  + numUsers + " users on " + numMovies + " movies.")
```

Split Dataset

Before building the model, the dataset must be splitted into smaller parts, one for training (60%), one for validation (20%) and another for testing (20%).

```
val training = ratings.filter(x => x._1 < 6)
  .values
  .cache()
val validation = ratings.filter(x => x._1 >= 6 && x._1 < 8)
  .values
  .cache()
val test = ratings.filter(x => x._1 >= 8).values.cache()

val numTraining = training.count()
val numValidation = validation.count()
val numTest = test.count()

println("Training: " + numTraining + ", validation: " + numValidation + ", test: " + numTest)
```

Function to evaluate the model

Once data is splitted, define the function that will evaluate the performance of the model. In particular we will use **Root Mean Squared Error (RMSE)** and this is the version in Scala:

```
import org.apache.spark.rdd.RDD

/** Compute RMSE (Root Mean Squared Error). */
def computeRmse(model: MatrixFactorizationModel, data: RDD[Rating], n: Long): Double = {
  val predictions: RDD[Rating] = model.predict(data.map(x => (x.user, x.product)))
  val predictionsAndRatings = predictions.map(x => ((x.user, x.product), x.rating))
  .join(data.map(x => ((x.user, x.product), x.rating))).values
  math.sqrt(predictionsAndRatings.map(x => (x._1 - x._2) * (x._1 - x._2)).reduce(_ + _) / n)
}
```

Choice of model

Now you can use this function to define the parameters for the training algorithm. The ALS algorithm requires 3 parameters: the range of the factor matrix, the number of iterations and a lambda. We will define different values for these parameters and try different combinations of them to determine which one is the best:

```
val ranks = List(8, 12)
val lambdas = List(0.1, 10.0)
val numIters = List(10, 20)
var bestModel: Option[MatrixFactorizationModel] = None
var bestValidationRmse = Double.MaxValue
var bestRank = 0
var bestLambda = -1.0
var bestNumIter = -1
for (rank <- ranks; lambda <- lambdas; numIter <- numIters) {
  val model = ALS.train(training, rank, numIter, lambda)
  val validationRmse = computeRmse(model, validation, numValidation)
  println("RMSE (validation) = " + validationRmse + " for the model trained with rank = "
    + rank + ", lambda = " + lambda + ", and numIter = " + numIter + ".")
  if (validationRmse < bestValidationRmse) {
    bestModel = Some(model)
    bestValidationRmse = validationRmse
    bestRank = rank
    bestLambda = lambda
    bestNumIter = numIter
  }
}
```

Wich one do you think is the best model?

Now let's launch our function on data test.

```
// evaluate the best model on the test set
val testRmse = computeRmse(bestModel.get, test, numTest)

println("The best model was trained with rank = " + bestRank + " and lambda = " + bestLambda
+ ", and numIter = " + bestNumIter + ", and its RMSE on the test set is "+
+ testRmse + ".")
```

User recommendations Performance

Once the best model is chosen, the next step is to know the recommendations of movies per user. The idea is to ask for the user, which for the Dataset used is a numeric. Let's do it form type, so first will ask for the user, insert it into a text field and finally release the recommendation. To ask for the user:

```
println("¿A qué usuario hay que recomendar películas? ")
val uid : AnyRef = z.input("User Id")
val userid = z.input("User Id").asInstanceOf[String].toInt
```

User Id

For this example, we configured it to show the 10 best recommendations for the user inserted in the text field.

```
val candidates = sc.parallelize(movies.keys.toSeq)

val recommendations = bestModel.get
    .predict(candidates.map((userid, _)))
    .collect()
    .sortBy(-_.rating)
    .take(10)

var i = 1
println("Movies recommended for you:")
recommendations.foreach { r =>
    println("%2d".format(i) + ": " + movies(r.product))
    i += 1
}
```

Persist the recommendations

Now we only have to save the best recommendations for each user in ontology. The idea is to save records of the form: UserId, MovieName, MovieGenre.

```
val recommendations = bestModel.get.predict(candidates.map((userid, _)))

val all_recommendations = bestModel.get.recommendProductsForUsers(10).flatMap{ case(u, r) =>
  r.map{ rat =>
    val m = movies(rat.product).split("%")
    (u, m(0), m(1))
  }
}.toDF("User", "Movie", "Genre")

all_recommendations.registerTempTable("recomendaciones")
```

We create the HIVE table with data stored in the DataFrame. It modifies the name of the table of the image " recomendaciones_arturo " by a unique identifier, for example, *recomendaciones_yourname*.

```
%sql
create table recomendaciones_arturo as select * from recomendaciones
```

Ontology Generation

We are going to generate an ontology from the HIVE table we created in the previous step. To do this, go to 'Analytics menu' and select 'UTIL HIVE_To_Ontology'. You will see a window with a list of available tables. The table you just created should not appear. This happens because the table is HIVE and that list shows IMPALA entities. Therefore, you have to give visibility to the table. To do this, click on "Visualize Hive table" button:



You'll see another window, where our table should appear. Select it and click on 'Regenerar Metadatos' (Regenerate Metadata):

ANALYTICS / HIVE TABLES

Hive table list

pruebatwitter_se_jsondump

recomendaciones_vic

recomendaciones_victor

scadatags_alarms

scadatags_alarms_jsondump

scoreendereco

scoreendereco_jsondump

sendor_puc_abn_incendio

Available operations

Regenerar Metadatos

Cancel

Once executed, return to the previous window with the "Cancel" button. Now our table appears in the list:

ANALYTICS / UTIL HIVE_TO_ONTOLOGY

List of available tables

my_first_table

passen_datatres

passenger_data

passenger_data_raw

passenger_datatres

passenger_hola

prueba_jirejas

recomendaciones_arturo

Visualize Hive table

TABLE RESULTS

```
{
  "user": 3747,
  "movie": "Bewegte Mann, Der (1994)",
  "genre": "Comedy"
}
```

Generate Schema

Once the table is chosen, click on "Generate Schema" and finally click on "Create".

A window appears with the data of the created ontology. There is only one more step, which involves activating the ontology. Click on "Modify" button, which is at the bottom of the page. Another window will open, check the "Active" box:

ONTOLOGIES / MODIFY ONTOLOGY

Ontology

Name

Actual Template Version

Original Template Version

recomendaciones_arturo

3

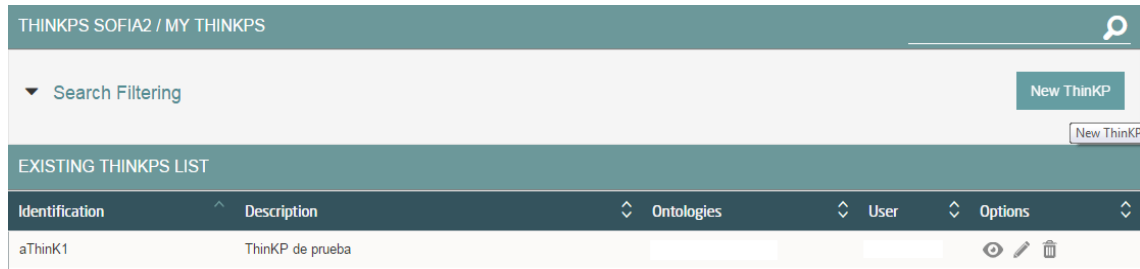
3




☒ Active

☐ Public

Finally, we generate the instance and click on "Save" button. To work with it we have to associate a valid ThinkKP. If you already have one created you can associate it to this ontology in "My ThinkKPs" -> Edit (you have to choose the ThinkKP), adding the ontology to the list associated with the ThinkKP. For this workshop, we will create a new one.

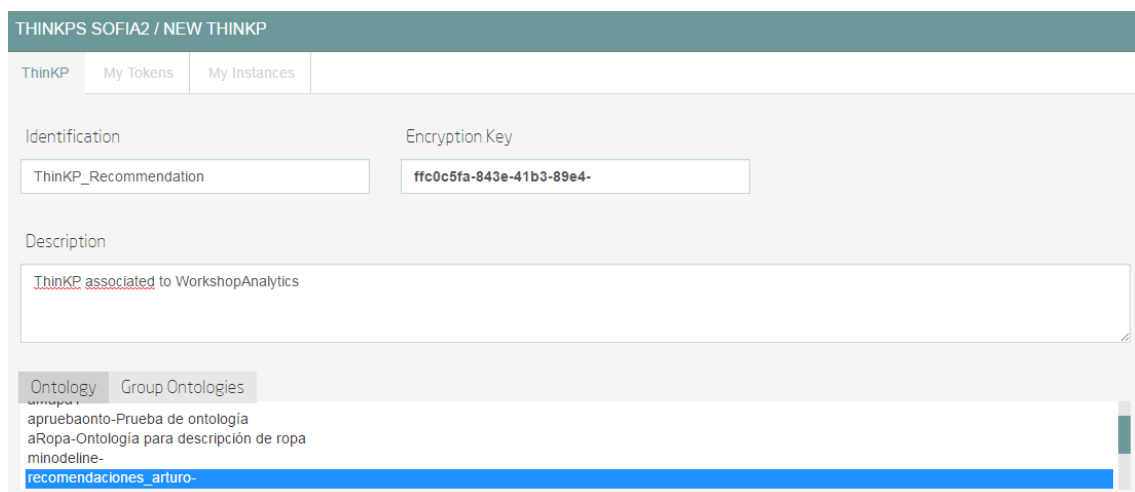
Click on 'THINKPS SOFIA2' -> "My ThinkKPs" and click on 'New ThinkKP'



Identification	Description	Ontologies	User	Options
aThinkK1	ThinkKP de prueba			  

You will see a new window in which you have to fill in "Identification" with the name of the new ThinkKP, and choose the ontologies to which the ThinkKP will have access.

To mark more than one ontology, use Ctrl + Shift.



THINKPS SOFIA2 / NEW THINKKP

ThinkKP My Tokens My Instances

Identification: ThinkKP_Recommendation

Encryption Key: ffc0c5fa-843e-41b3-89e4-

Description: ThinkKP associated to WorkshopAnalytics

Ontology Group Ontologies

- apruebaonto-Prueba de ontología
- aRopa-Ontología para descripción de ropa
- minodeline-
- recomendaciones_arturo-**

Once you have filled in the data, click on "New" and a summary window of the ThinkKP will appear:

THINKPS SOFIA2 / SHOW THINKP

ThinkKP My Tokens My Instances

Identification

ThinkKP_Recommendation

Encryption Key

ffc0c5fa-843e-41b3-89e4-

User

Description

ThinkKP associated to WorkshopAnalytics

Meta-Inf

Ontologies

recomendaciones_arturo

Cancel New Modify Delete

Now the ontology is ready. Go to 'TOOLS' menu and make some query on the newly created ontology.

It's a good practice to restrict queries results in Sofia2 console with "limit number_registers" (Eg select * from ontology limit 5)

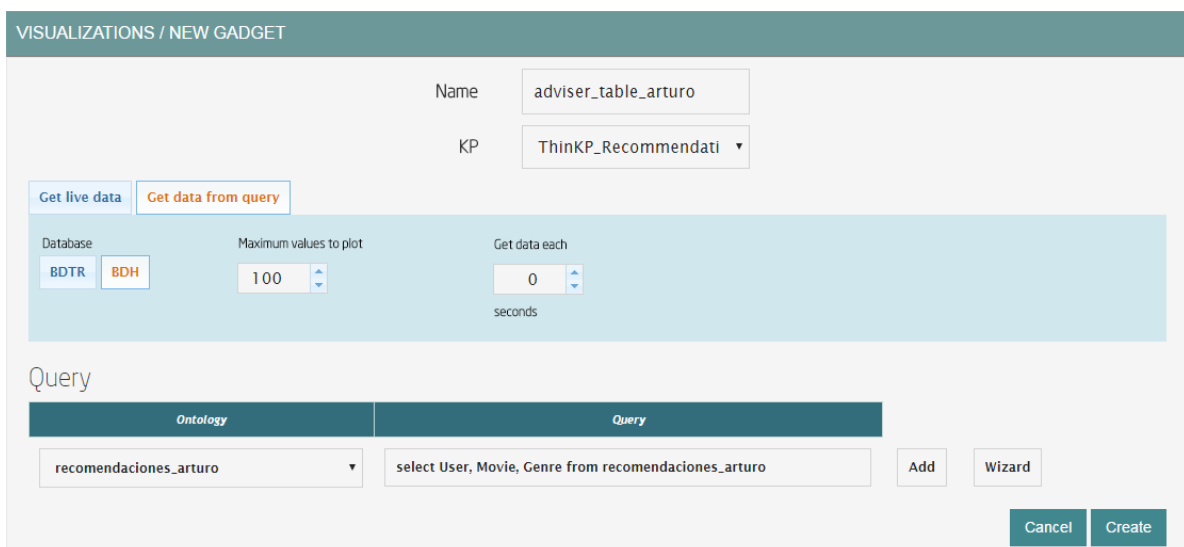
In the final step we're going to create a dashboard on the created ontology.

Gadget Creation

First we'll create the gadgets that will be displayed in the dashboard. Go to 'My Gadgets' menu. Click on 'New Gadget'. The first gadget we'll create is one of 'table' type. Choose that option in the catalog. You will have to fill in the necessary data for its creation:

- **Name:** E.g. adviser_table_yourname
- **KP:** The ThinkKP you created in previous steps
- **Get data from query**
 - o **Database:** BDH
 - o **Maximum values to plot:** 100
 - o **Get data each** (seconds): 0
- **Query**
 - o **Ontology:** Choose the ontology you have created for this workshop.
 - o **Query:** *select User, Movie, Genre from name_ontology*

It should look something like this:



The screenshot shows the 'VISUALIZATIONS / NEW GADGET' interface. At the top, there are two tabs: 'Get live data' and 'Get data from query', with the latter being selected. Below the tabs, there are three input fields: 'Database' with a dropdown menu showing 'BDTR' and 'BDH' (BDH is selected), 'Maximum values to plot' with a numeric input set to '100', and 'Get data each' with a numeric input set to '0' and a unit label 'seconds'. Below these fields, there is a 'Query' section with a table-like structure. The table has two columns: 'Ontology' and 'Query'. The 'Ontology' column has a dropdown menu with 'recomendaciones_arturo' selected. The 'Query' column has a text input field containing 'select User, Movie, Genre from recomendaciones_arturo'. To the right of the table, there are 'Add' and 'Wizard' buttons. At the bottom right, there are 'Cancel' and 'Create' buttons.

Click on the add button next to the query. You'll see more options you have to fill in with the name of the field to display along with its transformation (It is not mandatory), and the alias that will appear in the Gagdet. Add the User, Movie and Genre fields.

Measures

Column	Data Transform	Column Name	
User		user	<button>Remove</button>
Movie		Title	<button>Remove</button>
Genre		genre	<button>Remove</button>

Genre

genre

Add

Token

01425e9528914a1f9d3657 ▼

CancelCreate

If everything is correct, below this block should appear a preview of the table. To finish, click on "Create".

Let's go for the second Gagdet. Go to Gadget creation menu and choose in the catalog one of 'Pie' type. Again we have to fill in a number of attributes:

- **Name:** adviser_pie_yourname
- **KP:** The same ThinKP as for the table
- **Get data from query**
 - **Database:** BDH
 - **Maximum values to plot:** 100
 - **Get data each** (seconds): 0
- **Query**
 - **Ontology:** Choose the ontology you have created for this workshop.
 - **Query:** *select Genre, count(distinct Movie) as num from name_ontology group by Genre*

VISUALIZATIONS / NEW GADGET

Name:

KP:

Database:

Maximum values to plot:

Get data each: seconds

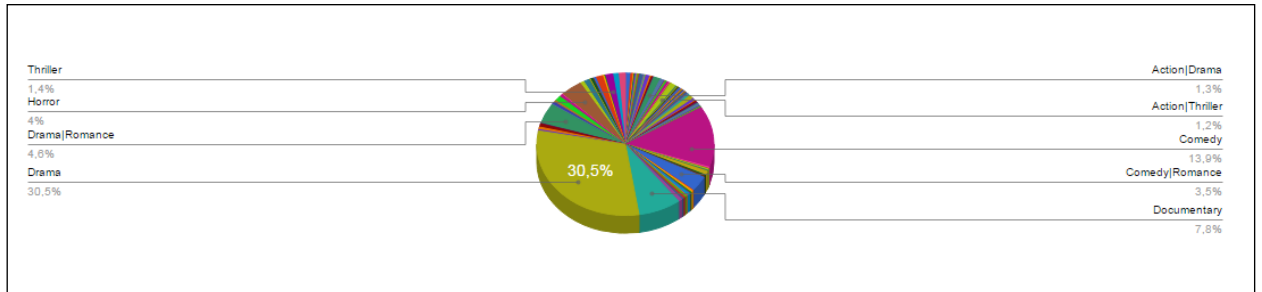
Query

Ontology	Query
recomendaciones_arturo	select Genre, count(distinct Movie) as num from recomendaciones_arturo group by Genre

As in the table, we have to fill in the measurements:

- **Category:** Genre
- **Value:** num

Once you have added the measurements, a preview of the chart will appear:



Just need to create the dashboard. To do this, go to 'My Dashboards' menu, click on 'New Dashboard'. Fill in the name, choose the theme you prefer and click on 'New page'. A window like this will open:

Sofia 2

Page_1479403930814

Click on + button to choose a created Gadget and choose 'Table Gadget' type. Once done, click on the Gadget configuration button, choose the one you created in the previous steps, put a name and click on the "Close" button.



The image shows a configuration window for a 'Table Gadget'. It has a title field with the value 'Tabla', a 'Gadget' dropdown menu with 'recomendador_tabla' selected, and a 'Master Gadget' dropdown menu with 'None...' selected. A 'Close' button is located at the bottom right.

You should show the table you created with the Gadget.

Similarly, add the 'Pie' Gadget you created in the previous steps. Once this is done, you can move the gadgets by using the move button on each of them. Finally, the dashboard might look something like this:

