Data Pipelines Implementation Guide (Pilot)

Salesforce, Winter '17

@salesforcedocs

Last updated: December 20, 2016

CONTENTS

DATA PIPELINES OVERVIEW

Salesforce Platform Data Pipelines consists of a new set of tools that help Salesforce.com developers and administrators manage the challenges of the increasing scale of customer data, also known as Big Data.

EDITIONS

Available in: Salesforce Classic

Available in: **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

The Data Pipelines pilot ushers in the era of Apache Hadoop on the Salesforce.com Platform, allowing highly scalable batch processing, transformation, and understanding of customer data. Data Pipelines offers a powerful tool set based on Apache Pig—a broadly used high-level language for expressing data flow control with a set of functions to help evaluate data. Pig offers an abstraction on top of the Salesforce Hadoop infrastructure to permit MapReduce processing within the established Salesforce multi-tenant architecture. Data Pipelines are expressed as Pig Latin scripts that are deployed to your org using the standard Metadata and Tooling APIs already familiar to the Salesforce.com developer community. The Salesforce Pig distribution supports Apache Pig 0.13. It delivers a whitelisted set of Apache DataFu and Piggybank user-defined function (UDF) libraries to address as many use cases as possible.

Note: This feature is available to select customers through a pilot program. To be nominated to join this pilot program, contact salesforce.com. Additional terms and conditions may apply to participate in the pilot program. Pilot programs are subject to change, and as such, we cannot guarantee acceptance into this pilot program or a particular time frame that this feature can be enabled. Any unreleased services or features referenced in this or other press releases or public statements are not currently available and may not be delivered on time or at all. Customers who purchase our services should make their purchase decisions based on features that are currently available.

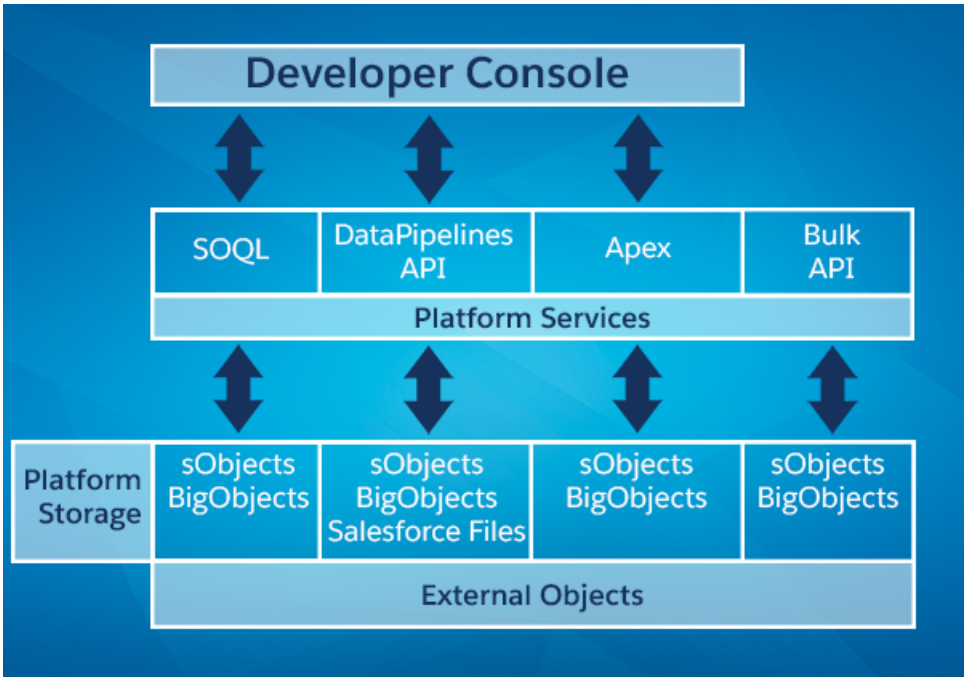
Considerations

- Data Pipelines scripts are limited in size and complexity.
- All data movement is subject to established Platform limits.
- To use Data Pipelines, you must be included in the pilot and have the DatapipelineApi and DataPipelineDevConsole org permissions.

DATA PIPELINES CONCEPTS

To understand Data Pipelines, it's helpful to know the key components and other related concepts.

The DataPipelines API joins other Salesforce platform services that give you access to data from various sources. You can work with data that's already on the Salesforce platform or load data into custom sObjects or BigObjects by using the Bulk API. With DataPipelines, you can process all your customer data, including sObjects, BigObjects, Salesforce files, and external objects. You can store the results in sObjects and BigObjects for use by other APIs and applications. With the Developer Console, you can write and deploy Data Pipelines jobs that use data from any of these sources.



With Data Pipelines, you can work directly with CSV data in Salesforce files, or you can insert data into the Salesforce platform using Bulk API.

Read on to learn about the Hadoop and Pig frameworks that DataPipelines uses.

Apache Hadoop

Apache Hadoop is an open-source software framework for storage and large-scale processing of data sets on clusters of commodity hardware. Hadoop is an Apache top-level project that's built and used by a global community of contributors and users. Hadoop is based on a distributed programming framework and a distributed filesystem. Hadoop is also a platform on which applications and higher-level programming languages can run.

Apache Pig

Pig is a data processing framework that runs on top of Hadoop. Pig scripts are expressed in a language that's called Pig Latin, which can be extended by using UDFs. Pig scripts are translated into MapReduce jobs and run on a Hadoop cluster, operating on data that you copy to a temporary data processing store.

UDF

A user-defined function (UDF) is an extension of Pig Latin that's created by a user. That is, you use UDFs to specify custom processing. UDFs are usually written in Java, but can be implemented in Python or JavaScript. Data Pipelines currently supports Pig built-ins, DataFu, and Piggybank.

Load and Store Functions

All load and store operations are done through the `ForceStorage` UDF, which lets you copy files between the data processing store and:

- Salesforce files—files that are attached to Chatter
- Salesforce objects (sObjects)—standard or custom objects that represent data in your organization

Apache DataFu Pig

Apache DataFu Pig is a collection of Pig UDFs for working with large data sets in Hadoop. We've whitelisted the UDFs that provide the most power without giving up security.

Piggy Bank

The Piggy Bank is a place where users share their Pig UDFs. The Data Pipelines distribution includes UDFs that come as part of the Pig distribution, including built-in UDFs and user-contributed UDFs in Piggybank. Some UDFs are disallowed. For more information, see UDF and Operator Whitelist on page 19.

DATA PIPELINES EXAMPLES

Use Data Pipelines to create and submit Apache Pig jobs with your Salesforce data.

You can use the Developer Console to write and run Pig scripts. You can also package a Pig script and deploy it by using the Metadata API. In API version 33.0 and later, you can create a DataPipeline job by using the Tooling API.

IN THIS SECTION:

ForceStorage Examples

Here are a few examples of how to use ForceStorage to get data from your Salesforce organization to the data processing store and back again.

Use ForceStorage to Load and Store Data

`ForceStorage` is the Pig UDF that lets you move your Salesforce data and files in and out of the data processing store. The fully qualified name is `gridforce.hadoop.pig.loadstore.func.ForceStorage`. You can use `ForceStorage` to address the following scheme.

- `force://` for objects and Salesforce files

In bulk load and store operations, the header row is in a separate file from the data rows. The org must have Bulk API enabled (asyncApi) for bulk load and store to work. Only users with permission to view all data can perform query, insert, and upsert operations. File names cannot contain whitespace or the ampersand (&) character.

Note: In ForceStorage loads, all column type values are considered type `bytearrray`. You can cast to another type by using the Pig cast operators. sObjects in bulk queries must be deployed to become available.

You can use ForceStorage to copy from Salesforce by using the Bulk API from within your Pig script.

Load and Store Salesforce Files

You can copy data in Salesforce files to and from the data processing store. Each file contains comma-delimited text with no header row. Enclose multi-line fields in quotation marks. There's a 2-GB limit on file size.

To copy a Salesforce file to the data processing store, use the file's `contentId`:

```
FileLoad = LOAD 'force://chatter/069xx000000065X' USING
gridforce.hadoop.pig.loadstore.func.ForceStorage();
```

To copy files from the data processing store to Salesforce, use the file prefix:

```
STORE A INTO 'force://chatter/myfileprefix' USING
gridforce.hadoop.pig.loadstore.func.ForceStorage();
```

Here, the file prefix is a user-defined prefix that's used for the created file names. The file names have the format `<file_prefix>_part-<m/r>-XXXX`. The `m` or `r` indicates whether the output came from the map or reduce phase. The `XXXX` is a four-digit number. For example, if a Pig job produces two files as output from the reduce phase, they're stored as files named `<file_prefix>_part-r-0000` and `<file_prefix>_part-r-0001`. If files with these names exist, they're overwritten.

The write operation that copies files back to Salesforce happens via the user who submitted the Pig job. So, make sure that user has the necessary permissions to write Salesforce files.

Load and Store from an Object

You can load data from standard and custom objects with the following syntax:

```
force://soql/<SELECT statement>
```

For example, the following Pig statement loads data from the Account and Opportunity objects.

```
ACCOUNT = LOAD 'force://soql/SELECT Id, Name FROM Account' USING
gridforce.hadoop.pig.loadstore.func.ForceStorage();
OPPORTUNITY = LOAD 'force://soql/SELECT AccountId, CreatedById FROM Opportunity' USING
gridforce.hadoop.pig.loadstore.func.ForceStorage();
```

The result of the query is written to the data processing store as two files: a `pig.schema` file that contains column information, and a second file that contains the data rows returned by the query. When performing an insert or upsert, don't include a header row because it is interpreted as a data row.

You can load data from external objects the same way. For example:

```
A = LOAD 'force://soql/Select AccountID__x,CustomerName__x from  SalesOrder__x ' USING
gridforce.hadoop.pig.loadstore.func.ForceStorage();
```

You can store Pig output by inserting into standard and custom objects with the following syntax:

```
force://entity/<objectName>/fields/<list of comma-separated fields>
```

To store the result set into a standard Salesforce object, list the fields like this:

```
STORE result INTO 'force://entity/Opportunity/fields/Name,Stage,Amount,CloseDate' USING
gridforce.hadoop.pig.loadstore.func.ForceStorage();
```

To store the result set into a custom Salesforce object, the object must already be deployed. Plus, you must define the object and list the fields in the same order as they appear in the result variable. For example, to insert the result into a custom `InsertDemo__c` object, the Pig statement might look like this:

```
STORE result INTO 'force://entity/InsertDemo__c/fields/AccountId__c, Name__c, Description__c,
 Country__c, AccOppId__c, OppName__c, MergeCountry__c' USING
gridforce.hadoop.pig.loadstore.func.ForceStorage();
```

You can upsert instead of inserting by specifying an external field ID with the following syntax:

```
force://entity/<objectName>/fields/<list of comma-separated
fields>/externalIdFieldName/<externalFieldId>
```

The upsert operation uses the specified field to determine whether to create a record or update an existing record. For example, you can use the parent_id as a unique identifier for an upsert operation into a Contact object:

```
force://entity/Contact/fields/FirstName,LastName,Department,Birthdate,Description/externalIdFieldName/parent_id
```

Load Data from an External Object

You can use ForceStorage to load data from external objects. Once your external objects are defined, the syntax is the same as loading from standard or custom objects. For example:

```
A = LOAD 'force://soql/Select AccountID__x,CustomerName__x from  SalesOrder__x '
    USING gridforce.hadoop.pig. loadstore.func.ForceStorage();
```

Store Data into a Wave Analytics Dataset

Often, after processing data using Data Pipelines, you can upload the data into an Analytics Cloud dataset. You use `ForceStorage` UDF to prepare the CSV data file that is then uploaded to the Analytics Cloud. Use the following syntax:

```
force://wave/<datasetContainerName>@<datasetName>/<operationType>/<list of fields>
```

where

<datasetContainerName> is the app where the dataset is stored, which is, by default, `'my personal apps'`.

<datasetName> is the name of the Analytics dataset to create or update.

<operationType> can be either Append or Overwrite. Append appends the data to the dataset or creates a dataset if it doesn't exist. Overwrite creates a dataset with the given data or replaces the dataset if it exists.

<list of fields> specifies the column name and type in the format *columnName-type*, where type can be text, numeric, or date. For example, AccountId-Text, CreateDate-Date, Revenue-Numeric.

You are responsible for converting the date to the format that Data Pipelines needs, which is yyyy-MM-dd'T'HH:mm:ss.SSSZ. You can use a Pig UDF for the data conversion.

A Pig statement that stores the result into a Wave Analytics dataset could look like this:

```
STORE result INTO 'force://wave/AccountInfo@AccountEM/Append/AccountId-Numeric, Name-Text,
 Description-Text, CreateDate-Date USING gridforce.hadoop.pig.loadstore.func.ForceStorage();
```

At this point, you can load this external data into Analytics Cloud as described in the *Analytics Cloud External Data API Developer's Guide*.

Note: You must have the required Wave licenses and access to the app to create the dataset.

SEE ALSO:

Files Overview

Run Data Pipelines with the Developer Console

Use the Developer Console to create, edit, and submit Data Pipelines jobs. This tutorial shows you how to run the Data Denormalization sample script, which processes data from the Account and Opportunity objects and stores the results in a custom object.

Before you try the tutorial, create a custom object, called MyCustomObject__c, with the following fields.

Table 1: MyCustomObject__c Fields

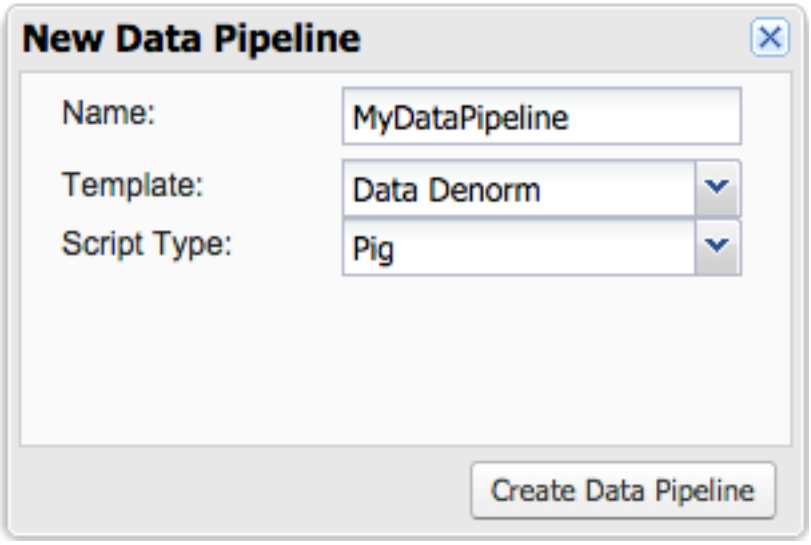
Field	Type
AccountId__c	ID
Name__c	string
Description__c	string
Country__c	string
AccOppId__c	ID
OppName__c	string

EDITIONS

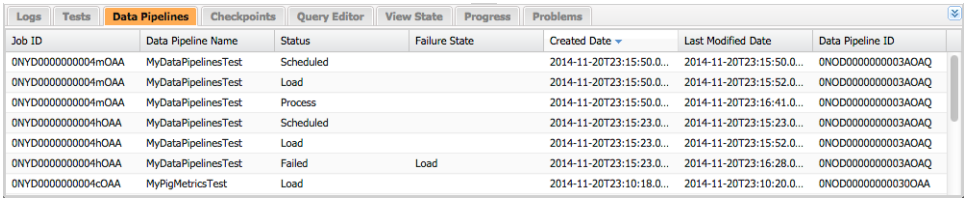
Available in: Salesforce Classic

Available in: **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

The Developer Console includes several templates for testing and to help you get started writing your own scripts.



The Data Pipelines tab shows the status of all submitted jobs. The status values correspond to those in the DataPipelineJob object.



Try the following example, which runs a data denormalization script on the Account and Opportunity objects and stores the results in the MyCustomObject__c object.

1. Create the MyCustomObject__c object.
2. In the Developer Console, click **File** > **New** > **Data Pipeline** to create a Data Pipeline.
3. Choose the **Data Denorm** template, and then type a name for the new Data Pipeline.
4. Click **Submit Data Pipeline** to submit the job.
5. Track the status of the job on the Data Pipelines tab.

Data Pipelines Deployment

Deploy and run a Pig script by using the Metadata API.

The Metadata API takes a `.zip` file that contains a package of files that describe the `DataPipeline` object and its metadata.

- `<filename>.dataPipeline`: a text file that contains the Pig script
- `<filename>.dataPipeline-meta.xml`: an XML file that contains metadata about the Pig script
- `package.xml`: an XML file that describes the contents of the package

Data Pipeline files are stored in a folder that's called `dataPipelines`. An XML file that's called the package manifest (`package.xml`) tells the Metadata API what's contained in the package.

Example: Deploy a DataPipeline Object

This example shows how to deploy a DataPipeline object by using the Metadata API. The package consists of a `.zip` file that contains a `dataPipelines` folder and a project manifest that lists the objects and the API version to use. The `dataPipelines` folder contains the XML that defines the DataPipeline object and its metadata,

1. Create a directory to contain the package files.
2. Create the Pig script and save it in a `.dataPipeline` file in a directory that's called `dataPipelines` inside the package directory.
3. Inside the `dataPipelines` directory, create the `.dataPipeline-meta.xml` file:

```
<?xml version="1.0" encoding="UTF-8"?>
<DataPipeline xmlns="http://soap.sforce.com/2006/04/metadata">
    <apiVersion>32.0</apiVersion>
    <label>Example Script</label>
    <scriptType>Pig</scriptType>
</DataPipeline>
```

   The only valid value for `scriptType` is `Pig`.

4. In the package directory, create the project manifest:

```
<?xml version="1.0" encoding="UTF-8"?>
<Package xmlns="http://soap.sforce.com/2006/04/metadata">
    <types>
        <members>df</members>
        <name>DataPipeline</name>
    </types>
    <version>32.0</version>
</Package>
```

5. Create a `.zip` file of the contents of the package directory (not the directory itself).

6. Use the `deploy()` function to deploy the DataPipeline object to your production environment. For more information, see the Metadata API Guide.

   Note: This pilot doesn't support deployment from sandbox to production environments.

7. From your Java code, execute the Pig script: pass the ID returned by the `deploy()` function into the call that creates a `DataPipelineJob` object. Example:

```
DataPipelineJob job = new DataPipelineJob();
job.setDataPipelineId("<ID>");
SaveResult[] saveResults = sforce.create(new DataPipelineJob[] {job});
```

DATA PIPELINES TOOLING API

DataPipeline

The `DataPipeline` object is the container for your custom Pig script.

To submit a Pig job, write a Pig script and create a `DataPipeline` object to contain it. You can create a DataPipeline this way in API version 33 and later. The Pig script must include code to load the data from the Salesforce database before execution and to store the data again after execution is complete. This object is available in API version 32.0 and later.

Supported SOAP API Calls

The `DataPipeline` object supports the following calls: `describeSObjects()`, `query()`, `retrieve()`, `create()`, `update()`

Fields

Field Name	Details
`ApiVersion`	**Type** double **Properties** Create **Description** The API version for this class. Every pipeline has an API version specified at creation.
`CreatedDate`	**Type** dateTime **Properties** Defaulted on create **Description** The date and time when the DataPipeline was created.
`CreatedById`	**Type** reference **Properties** Create **Description** The UserId of the user who created the DataPipeline.
`DeveloperName`	**Type** string

Field Name	Details
	Properties Defaulted on create **Description** The internal name of the DataPipeline object, as specified by the developer.
`LastModifiedById`	**Type** dateTime **Properties** Defaulted on create **Description** The date and time when the `DataPipeline` object was created.
`LastModifiedDate`	**Type** dateTime **Properties** Defaulted on create **Description** The date and time when the `DataPipeline` object was created.
`MasterLabel`	**Type** string **Properties** Create **Description** The display label for the DataPipeline object, as specified by the developer. Required.
`ScriptContent`	**Type** textarea **Properties** Create **Description** The script to be executed in the DataPipelines API. Limited to 100,000 characters. Required.
`ScriptType`	**Type** enum **Properties** Create **Description** The type of script to be executed in the DataPipelines API. Currently only “PIG” is supported. Required.

DataPipelineJob

Use the `DataPipelineJob` object to submit a Pig script that's contained in a `DataPipeline` object for execution.

This object is available in API version 32.0 and later.

Supported SOAP Calls

The DataPipeline object supports the following calls.

- `create()`
- `describeSObjects()`
- `query()`
- `retrieve()`

Fields

Field Name	Details
`CreatedById`	**Type** reference **Properties** Create **Description** The ID of the user who created the job.
`CreatedDate`	**Type** dateTime **Properties** Create **Description** The date and time when the job was created.
`DataPipelineId`	**Type** reference **Properties** Create **Description** The Id of the Datapipeline record to execute.
`FailureState`	**Type** enum **Properties** Create

Field Name	Details
	Description The state in which the job failed: • Load • Process • Scheduled • Store
`LastModifiedDate`	**Type** dateTime **Properties** Create **Description** The date and time when the job was last modified.
`Status`	**Type** test **Properties** Create **Description** The status of the job: • Failed • Killed • Load • Process • Scheduled • Store • Success

Example:

create()

```
DataPipelineJob job = new DataPipelineJob();
job.setDataPipelineId("dummyId");
SaveResult[] saveResults = sforce.create(new DataPipelineJob[] {job});
```

query()

```
String query = "Select CreatedById,Status,DataPipelineId from DataPipelineJob where
 id = 'something'";
QueryResult result = conn.query(query);
result.getSize();
```

```
DataPipelineJob job = result.records[0];
```

DATA PIPELINES METADATA API

DataPipeline

The `DataPipeline` object is the container for your custom Pig script.

To submit a Pig job, write a Pig script and create a `DataPipeline` object to contain it. The Pig script must include code to load the data from the Salesforce database before execution and to store the data again after execution is complete. This object is available in API version 32.0 and later.

Supported Calls

`create()`, `delete()`, `query()`, `retrieve()`, `update()`, `upsert()`

Declarative Metadata File Suffix and Directory Location

The file suffix for the pipeline file is `.dataPipeline`. The accompanying metadata file is named `pipelineName.dataPipeline-meta.xml`. Data Pipelines files are stored in the `dataPipelines` folder in the corresponding package directory.

Fields

Field Name	Details
`ApiVersion`	**Type** double **Properties** Create **Description** The API version for this class. Every pipeline has an API version specified at creation.
`ScriptContent`	**Type** textarea **Properties** Create **Description** The script to be executed in the DataPipelines API. Limited to 100,000 characters.
`ScriptType`	**Type** string **Properties** Create

Field Name	Details
	Description The type of script to be executed in the DataPipelines API. Currently only “PIG” is supported.

SALESFORCE PIG REFERENCE

Salesforce Pig Limitations

Some limitations apply when working with data or Pig in Data Pipelines.

When you submit a Pig script for processing, keep in mind the following limits.

- You don't have direct access to Hadoop configurations or parameters such as the number of reducers.
- Each Pig script is limited to 20 operators, 20 loads, and 10 stores.
- You can run up to 30 DataPipelines jobs per day.
- Bulk API doesn't support the following SOQL:
  - COUNT
  - ROLLUP
  - SUM
  - GROUP BY
  - CUBE
  - OFFSET
  - Nested SOQL queries
  - Relationship fields

Salesforce Pig Tips and Tricks

Use these best practices to make your Pig scripts effective on Salesforce.

- The only Load/Store function that you can use is `gridforce.hadoop.pig.loadstore.func.ForceStorage`.
- When loading Chatter files, specify the entity ID rather than the chatter filename, since there can be multiple files with the same name.
- When storing data in a Chatter file, specify the filename. The ID is created automatically.
- You can use the following operators.
  - COGROUP
  - CROSS
  - DISTINCT
  - FILTER
  - FOREACH
  - GROUP
  - JOIN
  - LIMIT
  - LOAD
  - ORDER

  - SAMPLE
  - SPLIT
  - STORE
  - UNION

- Avoid using CROSS because it's an expensive operator, especially with large data sets.
- Use the PARALLEL clause carefully: don't set more than five reducers.
- The default datatype is `bytearray`. See the Apache Pig documentation for casting rules.
- Only text data is supported.
- Custom UDFs aren't supported.
- Use FILTER early and often to reduce the size of the data set before performing operations on it.
- Creating zero-length Salesforce files is not supported. Any job that creates a zero-length Salesforce file fails.

Clean Your Data

If a field that you're using for numeric data contains text, number operations such as `SUM()` fails and causes unexpected results. Here is a short example script that looks for non-numeric data in the Amount field of the Opportunity object.

```
-- Load fields from the Opportunity object
opportunities = LOAD 'force://soql/SELECT OwnerId, Amount, Type FROM Opportunity' USING
gridforce.hadoop.pig.loadstore.func.ForceStorage() as (OwnerId,Amount,Type);

-- Cast the Amount field as a double. Non-numeric Amount values will result in null values
 for numericAccount.
typeCast = foreach opportunities generate Amount, (double)Amount as numericAccount;

-- Look for the null values to see the rows containing non-numeric values
filtered = filter typeCast by numericAccount is null;
```

Pig Syntax

Salesforce Files Load and Store

Move Salesforce files to and from the data processing store for use with Pig.

Salesforce Files Load

Salesforce files load uses the following syntax:

```
force://chatter/<file_id>
```

Each file should be a comma-delimited file with no header row. Multi line fields should be enclosed in quotation marks There's a 2-GB limit on file size.

Salesforce Files Store

Salesforce files store uses the following syntax:

```
force://chatter/<file_prefix>
```

The `file_prefix` is user-defined, and is prepended on multiple files if the Pig result is too large to store in one file. The resulting file names are `<file_prefix>_part-<m/r>-XXXX` where `m` or `r` indicates whether the output is from a mapper or a reducer and `XXXX` is a sequential number from 0000 to 9999.

SOQL Load and Bulk Store

Use SOQL load and bulk store to get data from sObjects into and out of the data processing store for use with your Pig scripts.

SOQL Load

SOQL load uses the following syntax:

```
force://soql/<soql_query>
```

Bulk Store

Bulk store uses the following syntax:

```
force://entity/<object_name>/fields/<comma_separated_field_names>
```

The `object_name` specifies an sObject on which to run a bulk API job. The `comma_separated_field_names` must be in the same order as in the sObject, because there's no header row in the data file.

UDF and Operator Whitelist

Supported operators and UDFs when using Pig with Salesforce.

Built-In Operators

Table 2: Built-In Operators Not Supported

Function Group	Operator
Dynamic invokers	All
Load/store functions	All

Note: For load and store, use `gridforce.hadoop.pig.loadstore.func.ForceStorage`.

Piggy Bank

Table 3: Piggy Bank UDFs Supported

Function Group	UDF
org.apache.pig.piggybank.evaluation	ExtremalTupleByNthField
	MaxTupleBy1stField
org.apache.pig.piggybank.evaluation.datetime	DiffDate
org.apache.pig.piggybank.evaluation.datetime.convert	ISOToUnix
	UnixToISO
org.apache.pig.piggybank.evaluation.datetime.diff	ISODaysBetween
	ISOHoursBetween
	ISOMinutesBetween
	ISOMonthsBetween
	ISOSecondBetween
	ISOYearsBetween
org.apache.pig.piggybank.evaluation.datetime.diff.truncate	ISOToDay
	ISOToHour
	ISOToMinute
	ISOToMonth
	ISOToSecond
	ISOToMonth
	ISOToSecond
	ISOToWeek
	ISOToYear
org.apache.pig.piggybank.evaluation.math	ABS
	ACOS
	ASIN
	ATAN
	ATAN2
	CBRT
	CEIL
	copySign

Function Group	UDF
	COS
	COSH
	DoubleAbs
	DoubleBase
	DoubleCopySign
	DoubleDoubleBase
	DoubleGetExponent
	DoubleMax
	DoubleMin
	DoubleNextAfter
	DoubleNextup
	DoubleRound
	DoubleSignum
	DoubleUlp
	EXP
	EXPM1
	FloatAbs
	FloatCopySign
	FloatGetExponent
	FloatMax
	FloatMin
	FloatNextAfter
	FloatNextUlp
	FloatRound
	FloatSignum
	FloatUlp
	Floor
	getExponent
	Hypot
	IEEERemainder

Function Group	UDF
	IntAbs
	IntMax
	IntMin
	LOG
	LOG10
	LOG1P
	LongAbs
	LongMax
	LongMin
	MAX
	MIN
	nextAfter
	NEXTUP
	POW
	RANDOM
	RINT
	ROUND
	SCALB
	SIGNUM
	SIN
	SINH
	SQRT
	TAN
	TANH
	toDegrees
	toRadians
	ULP
	Util
org.apache.pig.piggybank.evaluation.stats	HashFNV
	INDEXOF

Function Group	UDF
	LASTINDEXOF
	LcFirst
	LENGTH
	LOWER
	RegexExtract
	RegexExtractAll
	RegexMatch
	Replace
	Reverse
	Split
	Substring
	Trim
	UcFirst
	Upper

Table 4: Piggy Bank UDFs Not Supported

Function Group	UDF
org.apache.pig.piggybank.storage	All
org.apache.pig.piggybank.storage.allloader	All
org.apache.pig.piggybank.storage.apachelog	All
org.apache.pig.piggybank.storage.hiverc	All
org.apache.pig.piggybank.storage.partition	All

Note: For load and store, use `gridforce.hadoop.pig.loadstore.func.ForceStorage`.

DataFu

Table 5: DataFu UDFs Supported

Statistics	datafu.pig.stats.StreamingMedian
	datafu.pig.stats.StreamingQuantile
	datafu.pig.stats.VAR

BagOperations	datafu.pig.bags.CountEach
	datafu.pig.bags.BagConcat
	datafu.pig.bags.AppendToBag
	datafu.pig.bags.PrependToBag
JoinBags	datafu.pig.bags.BagLeftOuterJoin
	datafu.pig.util.Coalesce
Set Operations	datafu.pig.sets.SetIntersect
	datafu.pig.sets.SetUnion
	datafu.pig.sets.SetDifference
Sessions	datafu.pig.sessions.Sessionize
	datafu.pig.sessions.SessionCount
Sampling	datafu.pig.sampling.SimpleRandomSample
	datafu.pig.sampling.SimpleRandomSampleWithReplacementVote
	datafu.pig.sampling.SimpleRandomSampleWithReplacementElect
	datafu.pig.sampling.WeightedSample
	datafu.pig.sampling.SampleByKey
	datafu.pig.stats.HyperLogLogPlusPlus
Estimation	datafu.pig.hash.MD5
	datafu.pig.hash.SHA
Link Analysis	datafu.pig.linkanalysis

Data Pipelines Pig Errors

If a Data Pipelines job fails, informative error messages can help you solve the problem.

You can access Data Pipelines error messages via the Tooling API or on the Developer Console with the Job Run Information. Data Pipelines errors fall into the following categories:

- Pig script errors—for example, a typo or syntax error
- Data loading and storing errors—for example, a mistyped object name, a file that doesn't exist, or exceeding data ingest limits
- Other errors—for example, a problem reading the Pig script, a problem submitting the job, or an internal error

EDITIONS

Available in: Salesforce Classic

Available in:

Pig Script Errors

The Data Pipelines framework logs errors when there are problems with the Pig script itself.

Script Syntax Errors
A Pig syntax error has prevented Data Pipelines from saving the job. Make sure that the script follows proper Pig syntax. Syntax errors are reported when you save a Pig script by using the Developer Console or the Tooling API.

Script Validation Errors
A logical error in the Pig script caused the Data Pipelines job to fail. The error message provides information that you can use to correct the problem.

Pig script errors contain text to help you isolate the location and cause of the error. For example:

```
Error Running Pig Script.<line 4, column 38> Invalid field projection.Projected field [Id]
 does not exist
```

You can find information about Pig syntax and errors at pig.apache.org.

Data Loading and Storing Errors

Data loading and storing errors occur when you have a typo in the name of a file or sObject, or when a file or sObject doesn't exist. Different kinds of storage can have different types of errors:

- SOQL—Wrong entity names, incorrect field names, Bulk API errors (error in bulk execution or hitting Bulk API limits)
- Chatter—Non-existing Chatter file, loading a Chatter file by name instead of ID, Chatter API errors

The error message provides information to help you fix the problem. For example, if you try to load a Chatter file by using the name instead of the ID, you see the following error.

```
Source : force://chatter/myChatterFile had an error. Invalid parameter value
    "myChatterFile" for parameter "id".
```

Some errors occur because you try to use a feature not present in the organization or exceed a limit, such as the Bulk API limits for data ingest. For example:

```
Source 'force://soql/select Id, Name, MailingAddress, AccountId from Contact' failed.
    Bulk Job 750xx000000001gAAA, Batch Id: 751xx000000001qAAA Failed.
    Reason : InvalidBatch : Failed to process query: FUNCTIONALITY_NOT_ENABLED:
    Selecting compound data not supported in Bulk Query
```

If you have insufficient privileges to create an object to which you are storing data, an error occurs. For example:

```
Sink:force://entity/Trial__c/fields/Name... has an error. InvalidJob : No create access
for object Trial__c
```

You can see additional information about data loading and storing errors on the reference pages for the Bulk API and the Salesforce Files API.

Other Errors

Internal Errors
An internal error indicates internal problem, such as when the framework is unable to read the Pig script or to save information about the job. Retry the job. If the error recurs, contact Salesforce support. For example:

```
Error Running Pig Script.Internal Error.
```

MapReduce Job Failures

A MapReduce failure indicates that the jobs spawned by the Pig script failed. Contact Salesforce support. For example:

```
Error Running Pig Script.Failed to execute Map/Reduce jobs.
```

INDEX