ALIBABA CLOUD

Alibaba Cloud

Data Lake Analytics

Serverless Spark

Document Version: 20201009

Alibaba Cloud

Legal disclaimer

Alibaba Cloud reminds you to carefully read and fully understand the terms and conditions of this legal disclaimer before you read or use this document. If you have read or used this document, it shall be deemed as your total acceptance of this legal disclaimer.

1. You shall download and obtain this document from the Alibaba Cloud website or other Alibaba Cloud-authorized channels, and use this document for your own legal business activities only. The content of this document is considered confidential information of Alibaba Cloud. You shall strictly abide by the confidentiality obligations. No part of this document shall be disclosed or provided to any third party for use without the prior written consent of Alibaba Cloud.

2. No part of this document shall be excerpted, translated, reproduced, transmitted, or disseminated by any organization, company or individual in any form or by any means without the prior written consent of Alibaba Cloud.

3. The content of this document may be changed because of product version upgrade, adjustment, or other reasons. Alibaba Cloud reserves the right to modify the content of this document without notice and an updated version of this document will be released through Alibaba Cloud-authorized channels from time to time. You should pay attention to the version changes of this document as they occur and download and obtain the most up-to-date version of this document from Alibaba Cloud-authorized channels.

4. This document serves only as a reference guide for your use of Alibaba Cloud products and services. Alibaba Cloud provides this document based on the "status quo", "being defective", and "existing functions" of its products and services. Alibaba Cloud makes every effort to provide relevant operational guidance based on existing technologies. However, Alibaba Cloud hereby makes a clear statement that it in no way guarantees the accuracy, integrity, applicability, and reliability of the content of this document, either explicitly or implicitly. Alibaba Cloud shall not take legal responsibility for any errors or lost profits incurred by any organization, company, or individual arising from download, use, or trust in this document. Alibaba Cloud shall not, under any circumstances, take responsibility for any indirect, consequential, punitive, contingent, special, or punitive damages, including lost profits arising from the use or trust in this document (even if Alibaba Cloud has been notified of the possibility of such a loss).

5. By law, all the contents in Alibaba Cloud documents, including but not limited to pictures, architecture design, page layout, and text description, are intellectual property of Alibaba Cloud and/or its affiliates. This intellectual property includes, but is not limited to, trademark rights, patent rights, copyrights, and trade secrets. No part of this document shall be used, modified, reproduced, publicly transmitted, changed, disseminated, distributed, or published without the prior written consent of Alibaba Cloud and/or its affiliates. The names owned by Alibaba Cloud shall not be used, published, or reproduced for marketing, advertising, promotion, or other purposes without the prior written consent of Alibaba Cloud. The names owned by Alibaba Cloud include, but are not limited to, "Alibaba Cloud", "Aliyun", "HiChina", and other brands of Alibaba Cloud and/or its affiliates, which appear separately or in combination, as well as the auxiliary signs and patterns of the preceding brands, or anything similar to the company names, trade names, trademarks, product or service names, domain names, patterns, logos, marks, signs, or special descriptions that third parties identify as Alibaba Cloud and/or its affiliates.

6. Please directly contact Alibaba Cloud for any errors of this document.

Document conventions

Style	Description	Example
Danger	A danger notice indicates a situation that will cause major system changes, faults, physical injuries, and other adverse results.	Danger: Resetting will result in the loss of user configuration data.
Warning	A warning notice indicates a situation that may cause major system changes, faults, physical injuries, and other adverse results.	Warning: Restarting will cause business interruption. About 10 minutes are required to restart an instance.
Notice	A caution notice indicates warning information, supplementary instructions, and other content that the user must understand.	Notice: If the weight is set to 0, the server no longer receives new requests.
Note	A note indicates supplemental instructions, best practices, tips, and other content.	Note: You can use Ctrl + A to select all files.
>	Closing angle brackets are used to indicate a multi-level menu cascade.	Click **Settings**> **Network**> **Set network type**.
Bold	Bold formatting is used for buttons , menus, page names, and other UI elements.	Click **OK**.
`Courier font`	Courier font is used for commands	Run the `cd /d C:/window` command to enter the Windows system folder.
Italic	Italic formatting is used for parameters and variables.	`bae log list --instanceid` *Instance_ID*
[] or [a|b]	This format is used for an optional value, where only one item can be selected.	`ipconfig [-all|-t]`
{} or {a|b}	This format is used for a required value, where only one item can be selected.	`switch {active|stand}`

Table of Contents

1.Overview of Serverless Spark ---- 05
2.Developer Guide ---- 07
2.1. Use the spark-submit scripts ---- 07
2.2. Python-based Spark job ---- 16

1.Overview of Serverless Spark

Serverless Spark is a data analytics and computing service that is developed based on the cloud-native, serverless architecture. It is designed for Data Lake Analytics (DLA) scenarios. To submit a Spark job, you can perform simple configurations after you activate DLA. When a Spark job is running, computing resources are dynamically assigned based on task loads. After the Spark job is completed, you are charged based on the resources consumed by the job. Serverless Spark helps you eliminate the workload for resource planning and cluster configurations. It is more cost-effective than the traditional mode.

Terms

- virtual cluster

  Serverless Spark uses the multi-tenant mode. The Spark process runs in an isolated and secure environment. A virtual cluster is a secure unit for resource isolation.

  A virtual cluster does not have fixed computing resources. Therefore, you only need to allocate the resource quota based on your business requirements and configure the network environment to which the destination data that you want to access belongs. You do not need to configure or maintain computing nodes. You can also configure parameters for Spark jobs of a virtual cluster. This facilitates unified management of Spark jobs.

- compute unit

  Compute unit (CU) is the measurement unit of Serverless Spark. One CU equals one vCPU and 4 GB of memory. After a Spark job is completed, you are charged based on the CUs that are consumed by the Spark job and the duration for running the Spark job. The pay-as-you-go billing method is used. The computing hours are used as the unit for billing. If you use one CU for an hour, the resources consumed by a job is the sum of resources consumed by each computing unit. The fee required for computing resources of a Spark job is calculated by using the following formula:

  ```
  Fee required for computing resources of a Spark job = Total computing hours of the Spark job × Unit price per computing hour (USD 0.05)
  ```

  Use the following job configuration as an example:

  - Driver: medium (2 CUs), runtime of 50s
  - Executor: medium (2 CUs) × 5, runtime of 40s

  The total fee is calculated by using the following formula: (2 CUs (driver) × 50s + 5 × 2 CUs (executor) × 40s)/3600s × 0.05 ≈ USD 0.007.

- resource specification

  To simplify user configurations, the Serverless Spark feature of DLA performs simplified encapsulation on CPUs and memory. This way, you only need to select resource specifications such as small, medium, and large in the DLA console.

small	1 core, 4 GB of memory	1 CU
medium	2 cores, 8 GB of memory	2 CUs

large	4 cores, 16 GB of memory	4 CUs

Limits

The Serverless Spark feature of DLA has the following limits:

- The Serverless Spark feature supports only three types of CU specifications: small, medium, and large.
- A maximum of 10 virtual clusters can be created under an Alibaba Cloud account.

Use Serverless Spark

1. Manage virtual clusters.
2. .

2.Developer Guide

2.1. Use the spark-submit scripts

This topic describes how to use the spark-submit scripts and provides sample scripts.

Prerequisites

The spark-submit package is obtained.

You can click here to download the spark-submit-toolkit.tar.gz package or download it by using the following wget command.

```
wget https://dla003.oss-cn-hangzhou.aliyuncs.com/SparkSubmit/spark-submit-toolkit.tar.gz
```

After you download the package, decompress it.

```
tar zxvf spark-submit-toolkit.tar.gz
```

> **Note** To use the spark-submit scripts, ensure that JDK 8 or later is installed.

Procedure

1. **View help information.**
   - Run the following command to view the help information:

     ```
     cd /path/to/spark-submit-toolkit
     ./bin/spark-submit --help
     ```

   - After the preceding command is executed, the following result is returned:

     ```
     Usage: spark-submit [options] <app jar> [app arguments]
     Usage: spark-submit --list [PAGE_NUMBER] [PAGE_SIZE]
     Usage: spark-submit --kill [JOB_ID]

     Options:
       --keyId               Your ALIYUN_ACCESS_KEY_ID, required
       --secretId            Your ALIYUN_ACCESS_KEY_SECRET, required
       --regionId            Your Cluster Region Id, required
       --vcName              Your Virtual Cluster Name, required
       --oss-keyId           Your ALIYUN_ACCESS_KEY_ID to upload local resource to oss.
                         The default is the same as --keyId
       --oss-secretId        Your ALIYUN_ACCESS_KEY_SECRET, the default is the same as --secretId
       --oss-endpoint        Oss endpoint where the resource will upload. The default is http://oss-
     $regionId.aliyuncs.com
     ```

```
  --oss-upload-path          The user oss path where the resource will upload
                             If you want to upload a local jar package to the OSS directory,
                             you need to specify this parameter



  --class CLASS_NAME         Your application's main class (for Java / Scala apps).
  --name NAME                A name of your application.
  --jars JARS                Comma-separated list of jars to include on the driver
                             and executor classpaths.

  --conf PROP=VALUE          Arbitrary Spark configuration property
  --help, -h                 Show this help message and exit.
  --driver-resource-spec     Indicates the resource specifications used by the driver:
                             small | medium | large
  --executor-resource-spec   Indicates the resource specifications used by the executor:
                             small | medium | large
  --num-executors            Number of executors to launch
  --properties-file          spark-defaults.conf properties file location, only local files are supporte
d
                             The default is ${SPARK_SUBMIT_TOOL_HOME}/conf/spark-defaults.conf
  --py-files PY_FILES        Comma-separated list of .zip, .egg, or .py files to place
                             on the PYTHONPATH for Python apps.

  --status job_id            If given, requests the status and details of the job specified
  --verbose                  print more messages, enable spark-submit print job status and more job d
etails.

  List Spark Job Only:
  --list                     List Spark Job, should use specify --vcName and --regionId
  --pagenumber, -pn          Set page number which want to list (default: 1)
  --pagesize, -ps            Set page size which want to list (default: 10)

  Get Job Log Only:
  --get-log job_id           Get job log

  Kill Spark Job Only:
  --kill job_id,job_id       Comma-separated list of job to kill spark job with specific ids

  Spark Offline SQL options:
  -e <quoted-query-string>   SQL from command line
```

```
-f <filename>           SQL from files
```

2. **Use the spark-defaults.conf file to configure common parameters.**

   The spark-defaults.conf file allows you to configure the following parameters. Only the common parameters under spark conf are listed.

```
#  cluster information
# AccessKeyId
#keyId =
#  AccessKeySecret
#secretId =
#  RegionId
#regionId =
#  set vcName
#vcName =
#  set OssUploadPath, if you need upload local resource
#ossUploadPath =

##spark conf
#  driver specifications : small 1c4g | medium 2c8g | large 4c16g
#spark.driver.resourceSpec =
#  executor instance number
#spark.executor.instances =
#  executor specifications : small 1c4g | medium 2c8g | large 4c16g
#spark.executor.resourceSpec =
#  when use ram,  role arn
#spark.dla.roleArn =
#  when use option -f or -e, set catalog implementation
#spark.sql.catalogImplementation =
#  config dla oss connectors
#spark.dla.connectors = oss
#  config eni, if you want to use eni
#spark.dla.eni.enable =
#spark.dla.eni.vswitch.id =
#spark.dla.eni.security.group.id =
#  config log location, need an oss path to store logs
#spark.dla.job.log.oss.uri =
#  config spark read dla table
#spark.sql.hive.metastore.version = dla
```

> **Note**
>
> - The spark-submit scripts automatically read the `spark-defaults.conf` file in the conf folder.
> - Command line parameters take precedence over the `spark-defaults.conf` file.
> - For mappings between regions and `regionIds`, see Regions and zones.

3. **Submit a job.**

   For more information, see .

   Before you execute the spark-submit scripts, submit a Spark job in JSON format. Example:

```
{
  "name": "xxx",
  "file": "oss://{bucket-name}/jars/xxx.jar",
  "jars": "oss://{bucket-name}/jars/xxx.jar,oss://{bucket-name}/jars/xxx.jar"
  "className": "xxx.xxx.xxx.xxx.xxx",
  "args": [
    "xxx",
    "xxx"
  ],
  "conf": {
    "spark.executor.instances": "1",
    "spark.driver.resourceSpec": "medium",
    "spark.executor.resourceSpec": "medium",
    "spark.dla.job.log.oss.uri": "oss://{bucket-name}/path/to/log/"
  }
}
```

   After you execute the spark-submit scripts, submit a job in the following format:

```
$ ./bin/spark-submit  \
--class xxx.xxx.xxx.xxx.xxx \
--verbose \
--name xxx \
--jars oss://{bucket-name}/jars/xxx.jar,oss://{bucket-name}/jars/xxx.jar
--conf spark.driver.resourceSpec=medium \
--conf spark.executor.instances=1 \
--conf spark.executor.resourceSpec=medium \
oss://{bucket-name}/jars/xxx.jar \
xxx xxx

## The main program file which can be a JAR package specified by --jars or a file specified by --py-
```

```
files. It supports both the local file directory and the OSS file directory.
## The specified local files must use an absolute path. After the spark-submit scripts are execute
d, the local files are automatically uploaded to the specified OSS directory.
## You can use --oss-upload-path or set ossUploadPath in the spark-defaults.conf file to specify
the OSS directory to which data is uploaded.
## When a local file is being uploaded, the file content is verified by using MD5. If the specified O
SS directory has a file that has the same name and the MD5 value as the local file, the file upload
is canceled.
##
## Format: --jars  /path/to/local/directory/XXX.jar,/path/to/local/directory/XXX.jar
## Separate multiple files with commas (,) and specify an absolute path for each file.

## -- jars and -- py-files also allow you to specify a local directory to upload all files in the directo
ry. Content in sub-directories are not recursively uploaded.
## You must specify an absolute path for the directory. Example: --jars /path/to/local/directory/,
/path/to/local/directory2/
## Separate multiple directories with commas (,) and use absolute paths for directories.


## Program output. You can use the SparkUI listed in the following output to access the SparkUI o
f the job and view Job Detail to check whether the parameters submitted by the job meet your exp
ectations.
job status: starting
job status: starting
job status: starting
job status: starting
job status: starting
job status: starting
job status: running
{
  "jobId": "",
  "jobName": "SparkPi",
  "status": "running",
  "detail": "",
  "sparkUI": "",
  "createTime": "2020-08-20 14:12:07",
  "updateTime": "2020-08-20 14:12:07",
  ...
}
Job Detail: {
```

```
  "name": "SparkPi",
  "className": "org.apache.spark.examples.SparkPi",
  "conf": {
    "spark.driver.resourceSpec": "medium",
    "spark.executor.instances": "1",
    "spark.executor.resourceSpec": "medium"
  },
  "file": ""
}
```

> **Note**
>
> - For more information about how to use the AccessKey ID and AccessKey secret of a RAM user to submit jobs, see here.
> - If the local JAR package needs to be uploaded by using the spark-submit scripts, RAM users must be granted permissions for accessing OSS. You can grant the AliyunOSSFullAccess permission to the RAM user. For more information. see Users.

4. **Submit an offline SQL job.**

   You can use the `-f` option to specify the local SQL file, or use the `-e` option to specify the SQL statement that needs to be uploaded to the serverless Spark engine for execution.

```
## Separate multiple SQL statements with semicolons (;).
$ ./bin/spark-sql \
-e "show databases;show tables;" \
--name "runSQL" \
--verbose \
--conf spark.sql.hive.metastore.version=dla



$ ./bin/spark-sql \
-f /path/to/local/file.sql \
--verbose \
--name "runSQLInFile" \
--conf spark.sql.hive.metastore.version=dla
```

5. **Terminate a job.**

   Run the following commands to terminate a job:

```
$ ./spark-submit \
--kill <jobId>

## Return results
{"data":"deleted"}
```

6. **View the job list.**

   You can view jobs in CLI. The following sample script demonstrates how to view the first page of a job list that contains only one job.

```
$ ./spark-submit \
--list --pagenumber 1 --pagesize 1

## Return results
{
 "requestId": "",
 "dataResult": {
  "pageNumber": "1",
  "pageSize": "1",
  "totalCount": "251",
  "jobList": [
   {
    "createTime": "2020-08-20 11:02:17",
    "createTimeValue": "1597892537000",
    "detail": "",
    "driverResourceSpec": "large",
    "executorInstances": "4",
    "executorResourceSpec": "large",
    "jobId": "",
    "jobName": "",
    "sparkUI": "",
    "status": "running",
    "submitTime": "2020-08-20 11:01:58",
    "submitTimeValue": "1597892518000",
    "updateTime": "2020-08-20 11:22:01",
    "updateTimeValue": "1597893721000",
    "vcName": ""
   }
  ]
 }
}
```

7. **Obtain the parameters and SparkUI to submit jobs.**

```
./spark-submit --status <jobId>

## Return results
Status: success
job status: success
{
 "jobId": "",
 "jobName": "SparkPi",
 "status": "success",
 "detail": "",
 "sparkUI": "",
 "createTime": "2020-08-20 14:12:07",
 "updateTime": "2020-08-20 14:12:33",
 "submitTime": "2020-08-20 14:11:49",
 "createTimeValue": "1597903927000",
 "updateTimeValue": "1597903953000",
 "submitTimeValue": "1597903909000",
 "vcName": "",
 "driverResourceSpec": "medium",
 "executorResourceSpec": "medium",
 "executorInstances": "1"
}
```

8. **Obtain job logs.**

```
./spark-submit --get-log <jobId>

# Return results
20/08/20 06:24:57 WARN NativeCodeLoader: Unable to load native-hadoop library for your platfor
m... using builtin-java classes where applicable
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
20/08/20 06:24:58 INFO SparkContext: Running Spark version 2.4.5
20/08/20 06:24:58 INFO SparkContext: Submitted application: Spark Pi
20/08/20 06:24:58 INFO SecurityManager: Changing view acls to: spark
20/08/20 06:24:58 INFO SecurityManager: Changing modify acls to: spark
20/08/20 06:24:58 INFO SecurityManager: Changing view acls groups to:
20/08/20 06:24:58 INFO SecurityManager: Changing modify acls groups to:
20/08/20 06:24:58 INFO SecurityManager: SecurityManager: authentication disabled; ui acls disabl
ed; users  with view permissions: Set(spark); groups with view permissions: Set(); users  with m
odify permissions: Set(spark); groups with modify permissions: Set()
...
```

2.2. Python-based Spark job

After a Python-based Spark cluster is created, you can submit a job. This topic uses a Python-based Spark job in an example.

Note: When you create a Python-based Spark cluster, you must use the image of version `spark_2_4_5_dla_0_0_2` .

> *Python-based Spark is available in the China (Shenzhen) region and will be available in other regions.*

1. Prepare the test data

Generate a CSV file named `staff.csv` and upload it to Object Storage Service (OSS).

The CSV file lists the information and income of each employee.

```
name,age,gender,salary
Lucky,25,male,100
Lucy,23,female,150
Martin,30,male,180
Rose,31,female,200
```

2. Develop a dependency method

To calculate the after-tax income of each employee, create a file named `func.py` , write a `tax` method to it, and register the method as a Spark user-defined function (UDF) to facilitate subsequent operations.

Sample code:

```
def tax(salary):
    """
    convert string to int
    then cut 15% tax from the salary
    return a float number

    :param salary: The salary of staff worker
    :return:
    """
    return 0.15 * int(salary)
```

To introduce the tax method by using `pyFiles` , store the method in a ZIP-compressed package.

The following figure shows the directory structure of the compressed package.

Based on Python syntax, a module named `tools` is created, and the `func.tax` method is stored under the tools module.

Compress the depend folder as `depend.zip` and upload the package to OSS.

3. Develop the main program

Develop a Python-based Spark program to read the data in the CSV file from OSS, and register the CSV file as a `DataFrame`. Register the `tax` method in the depend.zip dependency package as a `Spark UDF`. Then, use the `Spark UDF` to calculate the `DataFrame` and generate the results.

Replace `{your bucket}` with the name of your OSS bucket in the following sample code:

```
from __future__ import print_function
from pyspark.sql import SparkSession
from pyspark.sql.functions import udf
from pyspark.sql.types import FloatType

# import third part file
from tools import func

if __name__ == "__main__":
    # init pyspark context
    spark = SparkSession\
        .builder\
        .appName("Python Example")\
        .getOrCreate()

    # read csv from oss to a dataframe, show the table
    df = spark.read.csv('oss://{your bucket}/staff.csv', mode="DROPMALFORMED",inferSchema=True, he
ader = True)
    # print schema and data to the console
    df.printSchema()
    df.show()

    # create an udf
    taxCut = udf(lambda salary: func.tax(salary), FloatType())

    # cut tax from salary and show result
    df.select("name", taxCut("salary").alias("final salary")).show()
    spark.stop()
```

Write the preceding code to the `example.py` file and upload the file to OSS.

4. Submit the job

Create a cluster. When you create the cluster, select the latest Spark image version `spark_2_4_5_dla_0_0_2`.

In the left-side navigation pane of the DLA console, choose Serverless Spark > Submit job. On the page that appears, click Create Job. In the dialog box that appears, complete the settings and submit the following job information:

```
{
   "name": "Spark Python",
   "file": "oss://{your bucket name}/example.py",
   "pyFiles": ["oss://{your bucket name}/depend.zip"],
   "conf": {
      "spark.driver.resourceSpec": "small",
      "spark.executor.instances": 2,
      "spark.executor.resourceSpec": "small",
      "spark.dla.connectors": "oss",
      "spark.kubernetes.pyspark.pythonVersion": "3"
   }
}
```

Replace `{your bucket name}` with the name of your OSS bucket. In this example, **Python 3** is used to run this job. You can specify the `spark.kubernetes.pyspark.pythonVersion` parameter to decide which version of Python is used the same as what you do to Spark community versions. **If you do not specify this parameter, Python 2.7 is used.**

5. Parameters

Parameter	Description	Required
name	The name of the Spark job.	No
file	The Python file where the Spark job is located. The value of this parameter must be an OSS endpoint.	Yes
pyFiles	The package of the third-party module on which the Spark job is dependent. You can separate multiple packages with commas (,).	No

Parameter	Description	Required
conf	The configuration parameters that are used by the Spark job. The configuration `"spark.dla.connectors": "oss"` indicates that the Spark job can connect to OSS. The configuration `"spark.kubernetes.pyspark.pythonVersion": "3"` indicates that the Spark job must be run by using Python 3.	No