

Open Data Analytics for z/OS
1.1

Installation and Customization Guide



Note

Before using this information and the product it supports, read the information in [“Notices” on page 191.](#)

This edition applies to Version 1 Release 1 of IBM® Open Data Analytics for z/OS® (5655-OD1) and to all subsequent releases and modifications until otherwise indicated in new editions.

Last updated: 2022-07-29

© **Copyright International Business Machines Corporation 2016, 2021.**

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

© **Rocket Software, Inc. 2016, 2021.**

Contents

Figures.....	vii
Tables.....	ix
About this information.....	xi
How to send your comments to IBM.....	xiii
If you have a technical problem.....	xiii
Summary of changes for IBM Open Data Analytics for z/OS Installation and Customization Guide.....	xv
Part 1. Introduction.....	1
Chapter 1. Introduction to IBM Open Data Analytics for z/OS.....	3
Chapter 2. Planning for installation.....	5
Planning considerations.....	5
Product overview.....	5
Skill requirements.....	5
Time requirements.....	5
Preinstallation considerations.....	5
Installation user ID.....	5
Requisite products.....	6
Required resources.....	6
Part 2. Installation.....	9
Chapter 3. Installing IBM Open Data Analytics for z/OS.....	11
Part 3. Customization.....	13
Chapter 4. Customizing your environment for z/OS Spark.....	15
Using the Spark configuration workflow	16
Upgrading Spark configuration workflows	20
Assigning an owner to new or changed steps	22
Verifying the Java and bash environments.....	26
Verifying configuration requirements for z/OS UNIX System Services.....	28
Setting up a user ID for use with z/OS Spark.....	28
Verifying the env command path.....	32
Customizing the Apache Spark directory structure.....	32
Creating the Apache Spark configuration directory.....	33
Updating the Apache Spark configuration files.....	34
Creating the Apache Spark working directories.....	35
Configuring networking for Apache Spark.....	37
Configuring z/OS Spark client authentication.....	41
Creating and configuring digital certificates and key rings.....	43
Configuring Policy Agent.....	46

Defining security authorization for Policy Agent.....	47
Creating the Policy Agent configuration files.....	48
Configuring PROFILE.TCPIP for AT-TLS.....	49
Defining the AT-TLS policy rules.....	49
Starting and stopping Policy Agent.....	53
Configuring additional authorities and permissions for the Spark cluster.....	53
Restricting the ability to start or stop the Spark cluster.....	56
Starting the Spark cluster.....	57
Configuring IBM Java.....	57
Creating jobs to start and stop Spark processes.....	58
Setting up started tasks to start and stop Spark processes	60
Procedures for each Spark cluster.....	61
Define the routing of the log output.....	61
Set and export common environment variables.....	62
Define the RACF started profile for started tasks.....	63
WLM configuration.....	64
Stopping the started tasks.....	64
Canceling the started tasks.....	64
Automating the starting of tasks.....	64
Configuring memory and CPU options.....	65
Configuring z/OS workload management for Apache Spark.....	73
Overview of Apache Spark Processes.....	73
Assigning job names to Spark processes.....	73
Overview of WLM classification.....	77
Defining WLM service classes for Spark.....	78
Defining WLM report classes for Spark.....	82
Defining WLM classification rules for Spark.....	82
Other Apache Spark attributes.....	83
Chapter 5. Customizing the Data Service server.....	85
Preparing to customize.....	85
Required naming conventions.....	86
Creating server data sets.....	86
Defining security authorizations.....	87
Configuring Workload Manager (WLM).....	87
APF-authorizing LOAD library data sets.....	88
Copying target libraries.....	88
Configuring support for code pages and DBCS	88
Creating the Global Registry log stream.....	89
Customizing the server initialization member.....	89
Configuring the started task JCL.....	90
Configuring the ISPF application.....	91
Configuring generation data set retrieval.....	91
Configuring delimited data support.....	93
Chapter 6. Installing the Data Service Studio.....	95
Verifying the studio installation.....	96
Installing the JDBC driver.....	96
Installing the Python dsdbc module.....	97
Chapter 7. Installing the JDBC Gateway.....	99
Starting the JDBC Gateway server.....	101
Launching the JDBC Gateway administrative console.....	102
Chapter 8. z/OS IzODA Livy Installation and Customization.....	105
Customizing z/OS IzODA Livy	106
Customizing user access for z/OS IzODA Livy	108

Chapter 9. Customizing Anaconda	113
Part 4. Verification.....	115
Chapter 10. Verifying the IBM Open Data Analytics for z/OS customization.....	117
Using the IBM Open Data Analytics for z/OS Spark Configuration Checker	118
Chapter 11. Verifying the Data Service server installation.....	121
Chapter 12. Verifying the IBM Open Data Analytics for z/OS product.....	123
Chapter 13. Verifying the z/OS IzODA Livy installation.....	131
Part 5. Resource monitoring.....	135
Chapter 14. Resource monitoring for Apache Spark.....	137
Spark web interfaces.....	137
Configuring Spark web interfaces.....	140
Securing Spark web interfaces.....	141
Event log directory and file permissions.....	142
Enabling the Spark history service.....	142
Spark log files.....	143
Using RMF to monitor Spark workload.....	144
Interactive performance reports with Monitor III.....	144
Long-term reporting with the Postprocessor.....	148
Using z/OS and z/OS UNIX commands to monitor Spark workload.....	149
Using IBM Health Checker for z/OS to monitor Spark workload.....	151
Part 6. Troubleshooting.....	153
Chapter 15. Troubleshooting issues with Apache Spark.....	155
Appendix A. Migrating to a new version of Apache Spark.....	159
Appendix B. Sample configuration and AT-TLS policy rules for z/OS Spark client authentication.....	163
Appendix C. Sample z/OS IzODA Livy AT-TLS policy rules.....	169
Appendix D. Memory and CPU configuration options.....	171
Appendix E. Spark properties specific to the z/OS environment.....	177
Appendix F. Data sets.....	181
Appendix G. Restrictions.....	185
Appendix H. Apache Spark in a mixed-endian environment.....	187
Accessibility.....	189
Notices.....	191
Trademarks.....	192

Index.....	193
-------------------	------------

Figures

1. Example of components in a typical Spark cluster.....	3
2. Network ports used in a typical Apache Spark environment.....	38
3. Sample job to start the master and worker.....	59
4. Sample job to stop the master and worker.....	59
5. Sample Spark cluster in client deploy mode.....	69
6. Logical view of a sample WLM classification scenario.....	77
7. Livy/Spark key ring setup.....	108
8. The Apache Spark master web UI.....	138
9. The Apache Spark worker web UI.....	139
10. The Apache Spark application web UI.....	140
11. The Spark history server web UI.....	143
12. Example of the RMF Storage Delays report.....	146
13. Example of the RMF Common Storage report.....	146
14. Example of the RMF Storage Frames report.....	146
15. Example of the RMF Storage Memory Objects report.....	147
16. Example of the RMF Processor Delays report.....	147
17. Example of the RMF Processor Usage report.....	147
18. Example of the RMF zFS File System report.....	148
19. Example of the RMF Workload Activity report for a Spark service class.....	148

Tables

1. Planning checklist for a first-time installation.....	6
2. Scope of environment variables.....	29
3. Apache Spark working directories.....	35
4. Network ports used by the Spark cluster.....	38
5. Network ports used by the Spark driver.....	39
6. Suggested initial memory sizing for a Spark cluster.....	66
7. Customization checklist.....	85
8. Livy working directories.....	106
9. Supported environments per deploy mode.....	107
10. Spark UI configurations.....	140
11. Apache Spark log files.....	143
12. Selected fields in the RMF Workload Activity report.....	149
13. z/OS system commands to monitor Spark workload.....	150
14. z/OS UNIX shell commands to monitor Spark workload.....	150
15. Example network port configurations.....	163
16. Environment variables that control memory settings.....	171
17. Spark properties that control memory settings.....	171
18. Environment variables that control CPU settings.....	172
19. Spark properties that control CPU settings.....	172
20. Spark properties that affect application and cluster parallelism.....	173
21. IBM JVM runtime options that control resource usage.....	174
22. IBM z/OS configuration parameters.....	175
23. Spark properties specific to the z/OS environment.....	177

24. Data sets created by INSTPAC.....	181
---------------------------------------	-----

About this information

This information supports IBM Open Data Analytics for z/OS (5655-OD1).

Purpose of this information

This information describes how to prepare for installation, install, customize, and verify IBM Open Data Analytics for z/OS in your environment.

Who should read this information

This information is intended for z/OS system programmers and system administrators who are responsible for installing and customizing IBM Open Data Analytics for z/OS. The customization information is also of interest to application developers who want to understand how various customization and tuning actions might affect the performance of their applications.

While IBM values the use of inclusive language, terms that are outside of IBM's direct influence are sometimes required. In this manual, we reference some instances of third-party software that contain non-inclusive command names, and we have had to include those terms in code examples.

How to send your comments to IBM

We invite you to submit comments about the z/OS product documentation. Your valuable feedback helps to ensure accurate and high-quality information.

Important: If your comment regards a technical question or problem, see instead [“If you have a technical problem”](#) on page xiii.

Submit your feedback by using the appropriate method for your type of comment or question:

Feedback on z/OS function

If your comment or question is about z/OS itself, submit a request through the [IBM RFE Community](http://www.ibm.com/developerworks/rfe/) (www.ibm.com/developerworks/rfe/).

Feedback on IBM Knowledge Center function

If your comment or question is about the IBM Knowledge Center functionality, for example search capabilities or how to arrange the browser view, send a detailed email to IBM Knowledge Center Support at ibmkc@us.ibm.com.

Feedback on the z/OS product documentation and content

If your comment is about the information that is provided in the z/OS product documentation library, send a detailed email to mhvrfs@us.ibm.com. We welcome any feedback that you have, including comments on the clarity, accuracy, or completeness of the information.

To help us better process your submission, include the following information:

- Your name, company/university/institution name, and email address
- The following deliverable title and order number: Open Data Analytics for z/OS Installation and Customization Guide, SC27-9033-00
- The section title of the specific information to which your comment relates
- The text of your comment.

When you send comments to IBM, you grant IBM a nonexclusive authority to use or distribute the comments in any way appropriate without incurring any obligation to you.

IBM or any other organizations use the personal information that you supply to contact you only about the issues that you submit.

If you have a technical problem

If you have a technical problem or question, do not use the feedback methods that are provided for sending documentation comments. Instead, take one or more of the following actions:

- Go to the [IBM Support Portal](http://support.ibm.com) (support.ibm.com).
- Contact your IBM service representative.
- Call IBM technical support.

Summary of changes for IBM Open Data Analytics for z/OS Installation and Customization Guide

The following changes are made to Version 1 Release 1.

New

The following information is new.

August 2022

- Changes throughout the manual for software prerequisite of z/OS Java Version 8, SR7 FP10 or higher

December 2021

- Changes throughout the manual for Spark version 2.4.8.2

September 2021

- Changes throughout the manual for the removal of support for Spark version 2.3.4
- Changes throughout the manual for Spark version 2.4.8.0

June 2021

- Added a section to [Chapter 15, “Troubleshooting issues with Apache Spark,” on page 155](#) warning not to use SHAREPORT when assigning TCPIP PORT definitions to Spark
- Changes throughout the manual for Spark version 2.3.4.10
- Changes throughout the manual for Spark version 2.4.7.4

March 2021

- Changes throughout the manual for Spark version 2.3.4.8
- Changes throughout the manual for Spark version 2.4.7.2

November 2020

- Changes throughout the manual for Spark version 2.4.7

August 2020

- Changes throughout the manual for Spark version 2.4.6
- A new section, [“Using _BPX_ACCT_DATA to assign accounting information to Spark processes” on page 76](#)

June 2020

- A new section, [“Event log directory and file permissions” on page 142](#)

Prior to June 2020

- Changes throughout [“Setting up started tasks to start and stop Spark processes ” on page 60](#) for History Server and Shuffle service support, including a new section, [“Procedures for each Spark cluster” on page 61](#) (March 2020)
- Changes throughout the manual for Spark version 2.3.4, including changes to [“Using the Spark configuration workflow ” on page 16](#) and [“Upgrading Spark configuration workflows ” on page 20.](#) (November 2019)
- The default value for the globalmax parameter is increased from 5000 to 50000. (September 2019)
- The newest supported version of Bourne Again Shell (Bash) is 4.3.48. (September 2019)

- A new section, [“Configuring Spark web interfaces”](#) on page 140. (September 2019)
- A new section, [“Restricting the ability to start or stop the Spark cluster”](#) on page 56. (September 2019)
- Updated descriptions for configuration workflows in [“Using the Spark configuration workflow ”](#) on page 16. (September 2019)
- Appendix B, [“Sample configuration and AT-TLS policy rules for z/OS Spark client authentication,”](#) on page 163 and Appendix C, [“Sample z/OS IzODA Livy AT-TLS policy rules,”](#) on page 169 have extensive changes. Note that this information has replaced Appendix C, [“z/OS IzODA Livy Pagent policies.”](#) (September 2019)
- A new chapter, [Chapter 2, “Planning for installation,”](#) on page 5, provides tips on planning the installation and deployment of IzODA. (June 2019)
- New information for z/OS IzODA Livy support (June 2019):
 - Introductory description of Livy in [Chapter 1, “Introduction to IBM Open Data Analytics for z/OS,”](#) on page 3
 - An additional migration note in [Chapter 3, “Installing IBM Open Data Analytics for z/OS,”](#) on page 11
 - [Chapter 8, “z/OS IzODA Livy Installation and Customization,”](#) on page 105
 - [Chapter 13, “Verifying the z/OS IzODA Livy installation,”](#) on page 131
 - z/OS IzODA Livy Pagent policies
- Added **Spark worker fails with ICH408I message with NEWJOBNAME insert** to [Part 6, “Troubleshooting,”](#) on page 153. (June 2019)
- The system messages were moved from this guide to the new [Open Data Analytics for z/OS System Messages](#). (March 2019)
- A new section, [“Upgrading Spark configuration workflows ”](#) on page 20, which includes [“Assigning an owner to new or changed steps ”](#) on page 22. (March 2019)
- The *JDBC Gateway* is a Data Service distributed application server that allows direct connectivity to JDBC data sources. See [Chapter 7, “Installing the JDBC Gateway,”](#) on page 99. (March 2019)
- New sample policy for when AT-TLS is used as the Spark client authentication method, in [Appendix C, “Sample z/OS IzODA Livy AT-TLS policy rules,”](#) on page 169. (December 2018)
- A new table of Spark configuration options, [Table 20 on page 173](#). (December 2018)
- New entries in [Table 23 on page 177](#). (December 2018)
- The Spark REST server port is disabled. See the migration notes in [Chapter 3, “Installing IBM Open Data Analytics for z/OS,”](#) on page 11 for more information. (December 2018)
- New section, [“Other Apache Spark attributes”](#) on page 83, that describes increasing the parallelism of your Spark applications and allowing multiple Spark applications to run simultaneously. (December 2018)
- This version has received editorial updates. (September 2018)
- A new section, [“Using the Spark configuration workflow ”](#) on page 16. (September 2018)
- New blog link locations in the What to Do Next part of [Chapter 12, “Verifying the IBM Open Data Analytics for z/OS product,”](#) on page 123. (September 2018)
- [“Verifying the env command path”](#) on page 32 is updated to include a fixed APAR number. (June 2018)
- [“Updating the Apache Spark configuration files”](#) on page 34 is updated to provide clarification about Apache Derby configuration. (June 2018)
- A note is updated in the task, [“Creating the Apache Spark working directories”](#) on page 35, to provide guidance on temporary file system usage. (June 2018)
- A new network port for the PySpark daemon is added to [“Configuring networking for Apache Spark”](#) on page 37. (June 2018)

- **Step 3b** in [“Configuring memory and CPU options”](#) on [page 65](#) is updated to clarify the amount of native memory that is required. (June 2018)
- A new property, `spark.python.daemon.port`, is added to [Appendix E, “Spark properties specific to the z/OS environment,”](#) on [page 177](#). (June 2018)
- An appendix is added for Data Service server messages and codes. (June 2018)
- [“Automating the starting of tasks”](#) on [page 64](#) is updated to provide clarification about the sample procedures that are included in IBM Open Data Analytics for z/OS. (April 2018)
- A new topic, [“Define the routing of the log output”](#) on [page 61](#), is added for using started tasks. (April 2018)
- The following enhancements are available when customizing the Data Service server (April 2018):
 - You can now manually create the Global registry log stream. See [“Creating the Global Registry log stream”](#) on [page 89](#).
 - Using a virtual table rule, you can read a subset of a generation data group. See [“Configuring generation data set retrieval”](#) on [page 91](#).
 - Delimited data can now be used with virtual tables. See [“Configuring delimited data support”](#) on [page 93](#).
- The following topics are updated to introduce the environment verification function (March 2018):
 - A new migration note is added to [Chapter 3, “Installing IBM Open Data Analytics for z/OS,”](#) on [page 11](#).
 - The Spark property, `spark.zos.environment.verify` is added to [Appendix E, “Spark properties specific to the z/OS environment,”](#) on [page 177](#).
- A note is added to the table, [Table 4 on page 38](#), for the `SPARK_MASTER_PORT` configuration property. (March 2018)
- Step 4, in [“Creating and configuring digital certificates and key rings”](#) on [page 43](#), is updated to include an additional command. (March 2018)
- A note and a new step are added to the task, [“Configuring additional authorities and permissions for the Spark cluster”](#) on [page 53](#). (March 2018)
- A note is added to [“Creating jobs to start and stop Spark processes”](#) on [page 58](#) to provide clarification. (March 2018)
- The following topics introduce the task, [“Setting up started tasks to start and stop Spark processes”](#) on [page 60](#) (March 2018):
 - [“Set and export common environment variables”](#) on [page 62](#)
 - [“Define the RACF started profile for started tasks”](#) on [page 63](#)
 - [“WLM configuration”](#) on [page 64](#)
 - [“Stopping the started tasks”](#) on [page 64](#)
 - [“Canceling the started tasks”](#) on [page 64](#)
 - [“Automating the starting of tasks”](#) on [page 64](#)
- [“Using the IBM Open Data Analytics for z/OS Spark Configuration Checker”](#) on [page 118](#) is added to introduce the Configuration Checker tool. (March 2018)
- Step 4, in [“Securing Spark web interfaces”](#) on [page 141](#), is updated to include an additional option. (March 2018)
- [Chapter 15, “Troubleshooting issues with Apache Spark,”](#) on [page 155](#) includes new troubleshooting information. (March 2018)
- An appendix is added for IBM Open Data Analytics for z/OS system messages. (March 2018)
- A note and performance considerations are added to [Chapter 3, “Installing IBM Open Data Analytics for z/OS,”](#) on [page 11](#). (December 2017)
- [“Verifying the env command path”](#) on [page 32](#) includes an updated task description. (December 2017)

- An important note is added to [“Creating jobs to start and stop Spark processes”](#) on page 58. (December 2017)
- The following topics introduce enhanced job name specification options (December 2017):
 - [Chapter 3, “Installing IBM Open Data Analytics for z/OS,”](#) on page 11
 - [“Assigning job names to Spark processes”](#) on page 73
 - [“Using the spark.zos.executor.jobname.template”](#) on page 74
 - [“Using the spark.zos.driver.jobname.template”](#) on page 75
 - [Appendix E, “Spark properties specific to the z/OS environment,”](#) on page 177
- The following topics introduce a new client authentication method, Trusted Partner (December 2017):
 - [“Configuring z/OS Spark client authentication”](#) on page 41
 - [“Creating and configuring digital certificates and key rings”](#) on page 43
 - [“Defining the AT-TLS policy rules”](#) on page 49
 - [“Configuring additional authorities and permissions for the Spark cluster”](#) on page 53
 - [“Starting the Spark cluster”](#) on page 57
 - The `spark-defaults.conf` configuration file option, `spark.zos.master.authenticate.method` is introduced. For more information, see [Appendix E, “Spark properties specific to the z/OS environment,”](#) on page 177.
- A checklist for customizing the Data Service server is added. See [“Preparing to customize”](#) on page 85. (December 2017)
- [“APF-authorizing LOAD library data sets”](#) on page 88 contains updated information about required APF authorizations. (December 2017)
- [“Configuring additional authorities and permissions for the Spark cluster”](#) on page 53 and [Chapter 12, “Verifying the IBM Open Data Analytics for z/OS product,”](#) on page 123 include updated content and code samples. (December 2017)
- The task, [“Securing Spark web interfaces”](#) on page 141, is added. (December 2017)
- [Chapter 15, “Troubleshooting issues with Apache Spark,”](#) on page 155 includes an updated error message when Spark scripts fail. (December 2017)
- [Appendix A, “Migrating to a new version of Apache Spark,”](#) on page 159 includes updated migration actions. (December 2017)
- [Appendix C, “Sample z/OS IzODA Livy AT-TLS policy rules,”](#) on page 169 is updated to include a new sample policy. (December 2017)
- All of the links to external Apache Spark websites are updated for Spark version 2.2.0. (December 2017)
- All instances of WLM APAR OA50845 are updated to the latest WLM APAR OA52611. (December 2017)
- [“Creating and configuring digital certificates and key rings”](#) on page 43 and [“Defining security authorization for Policy Agent”](#) on page 47 include updated code samples. (September 2017)
- [“Configuring additional authorities and permissions for the Spark cluster”](#) on page 53 provides further clarification on Spark permissions. (September 2017)
- The following topics include updated job names (September 2017):
 - [“Using _BPX_JOBNAME to assign job names to Spark processes”](#) on page 76
 - [“Overview of WLM classification”](#) on page 77
 - [“Defining WLM service classes for Spark”](#) on page 78
 - [“Defining WLM report classes for Spark”](#) on page 82
 - [“Defining WLM classification rules for Spark”](#) on page 82
 - [“Interactive performance reports with Monitor III”](#) on page 144

- [“Long-term reporting with the Postprocessor” on page 148](#)
- [Chapter 12, “Verifying the IBM Open Data Analytics for z/OS product,” on page 123](#) adds two new installation verification procedures to verify the IBM Open Data Analytics for z/OS product. (September 2017)

Part 1. Introduction

Chapter 1. Introduction to IBM Open Data Analytics for z/OS

This topic provides a brief introduction to the product components and terminology in IBM Open Data Analytics for z/OS (IzODA).

Product components

IBM Open Data Analytics for z/OS consists of the following components:

z/OS IzODA Spark (FMID HSPK120)

z/OS IzODA Spark (z/OS Spark) is built on Apache Spark, a high-performance, general execution engine for large-scale data processing. One of its key features is the capability to perform in-memory computing. Unlike traditional large data processing technologies, Spark allows caching of intermediate results in memory rather than writing them to disk, thereby dramatically improving the performance of iterative processing.

z/OS IzODA Mainframe Data Service (FMID HMDS120)

z/OS IzODA Mainframe Data Service (Data Service or MDS) provides integration facilities for both IBM Z data sources and other off-platform data sources. The Data Service provides your Apache Spark application with optimized, virtualized, and parallelized access to a wide variety of data.

z/OS IzODA Anaconda (FMID HANA110)

z/OS IzODA Anaconda includes Python and Anaconda Python packages for data science, which provide data scientists with a comprehensive solution for integrating computations to the data.

Figure 1 on page 3 illustrates the components in a typical Spark cluster.

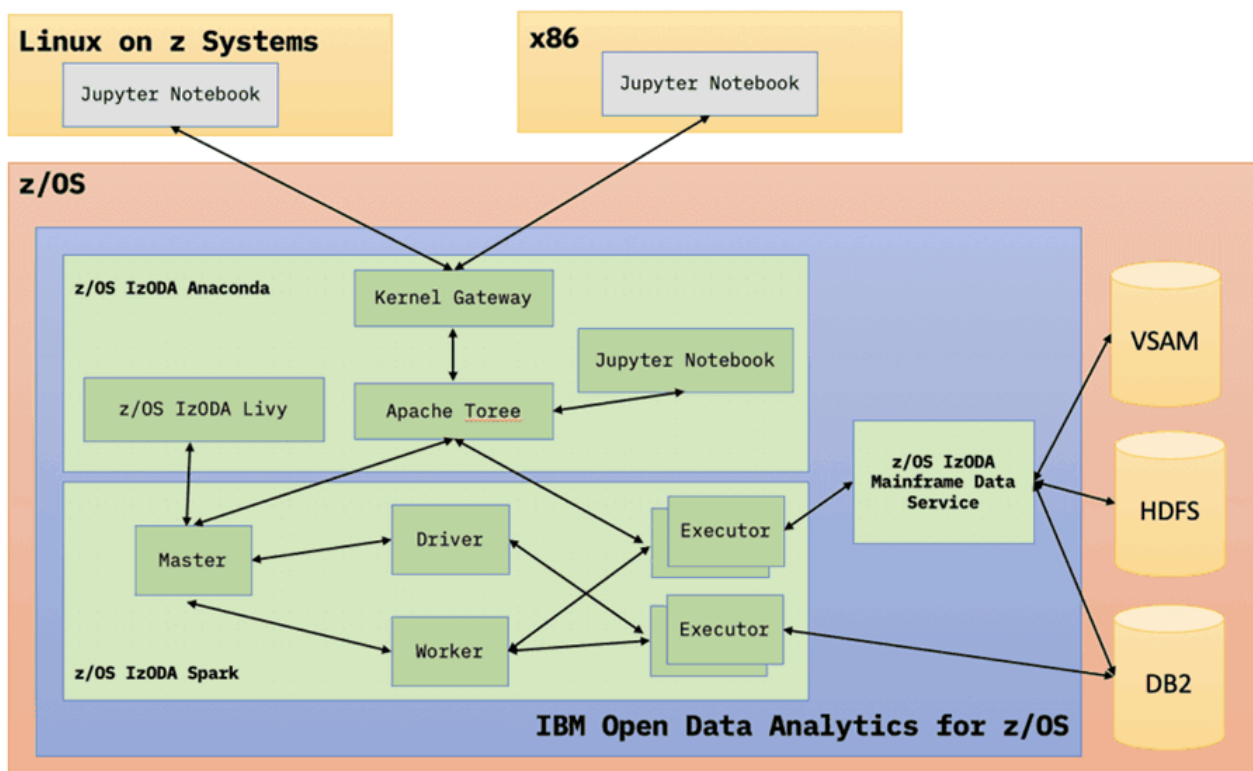


Figure 1. Example of components in a typical Spark cluster

The components are described in “Terminology” on page 4.

z/OS IzODA Livy

z/OS IzODA Livy is built on Apache Livy. It is a REST service used in conjunction with Spark that enables users to submit Spark jobs without having the Spark client installed. This enables developers to harness the data analytics power that Spark is capable of providing from within a web or mobile application. Jobs are submitted to the Livy server through REST API that contains information about the Spark application to be run. An interactive Scala or Python session with Livy can also be started.

Terminology

The following terms and abbreviations appear throughout this documentation:

Master

The Spark daemon that allocates resources across applications.

Worker

The Spark daemon that monitors and reports resource availability and, when directed by the master, spawns executors. The worker also monitors the liveness and resource consumption of the executors.

Executor

A process that the worker creates for an application. The executors perform the actual computation and data processing for an application. Each application has its own executors.

Driver program

The process that runs the main function of the application and creates the SparkContext.

SparkContext

Coordinates all executors in the cluster and sends tasks for the executors to run.

Apache Toree

Open source software that provides the foundation for interactive applications to connect to and use z/OS Spark.

Jupyter Notebook

An open source web application that provides an interactive application development environment for data scientists.

Deploy mode

Distinguishes where the driver process runs. In *cluster* deploy mode, the framework starts the driver inside the cluster. In *client* deploy mode, the submitter starts the driver from outside the cluster. If you use Jupyter Notebook and Apache Toree to interact with Spark, you are likely using client deploy mode. The default is client deploy mode.

Local mode

A non-distributed, single-JVM deployment mode in which all of the Spark execution components—driver, master, worker, and executors—run in the same JVM.

Cluster mode

Not to be confused with cluster deploy mode, Spark in cluster mode means that, unlike local mode, each Spark execution component—driver, master, worker, and executors—runs in a separate JVM. An application can be submitted to a Spark cluster in both cluster deploy mode and client deploy mode.

Cluster manager

The software that manages resources for the Spark cluster. Apache Spark supports Standalone, Mesos, and YARN. Only the Standalone cluster manager is available for Open Data Analytics for z/OS.

Task

A unit of work that is sent to one executor.

Chapter 2. Planning for installation

Use the information in this chapter and the IBM Open Data Analytics for z/OS (IzODA) software requirements to plan the installation and deployment of IzODA.

For a complete listing of the IzODA hardware and software requirements including prerequisites and co-requisites, see *Program Directory for IBM Open Data Analytics for z/OS*.

Planning considerations

Product overview

IBM Open Data Analytics for z/OS (IzODA) consists of three components: Spark, MDS, and Anaconda.

Skill requirements

SMPE skills are required for IBM Open Data Analytics for z/OS installation.

The configuration of IBM Open Data Analytics for z/OS requires more than the typical system programming permissions and expertise. The installation and configuration span several roles that may be performed by one or more individuals. [Table 1 on page 6](#) lists the roles that may be needed for the required and optional customization tasks.

Given the number of tasks and roles involved, close collaboration is key to successfully getting IBM Open Data Analytics for z/OS (IzODA) up and running. Use of this high-level roadmap will help coordinate, organize, and keep track of all installation and configuration tasks.

The amount of time that is required to install and configure IzODA components depends on such factors such as:

- The current z/OS UNIX and TCP/IP configuration
- The availability of prerequisite software and maintenance
- Whether OMVS segments are defined for IzODA users.

Time requirements

Experience has shown that the installation and configuration process of IzODA requires from one to five days to complete. This time requirement is for a clean installation performed by an experienced system programmer. If problems are encountered, or if the required skills are not available, the setup will take longer.

Preinstallation considerations

For detailed instructions on the SMP/E installation of the product, see *Program Directory for IBM Open Data Analytics for z/OS* (GI13-4348-00).

Installation user ID

The user ID that is used to install IzODA, or to install maintenance, must have the following attributes:

- UID(0) or READ access or higher to the BPX.SUPERUSER facility class
- Be connected to a group that has a GID
- Have READ access or higher to the following facility classes:
 - BPX.FILEATTR.PROGCTL
 - BPX.FILEATTR.APF

- BPX.FILEATTR.SHARELIB
- Have WRITE access to the following paths:
 - /usr/lpp/IBM/izoda/spark
 - /usr/lpp/IBM/izoda/anaconda

For a complete listing of the IzODA installation user ID requirements, see *Program Directory for IBM Open Data Analytics for z/OS*.

Requisite products

IzODA has a list of prerequisite software that must be installed and operational before the product will work.

For a complete listing of the IzODA software requirements including prerequisites, see *Program Directory for IBM Open Data Analytics for z/OS*.

Plan ahead to have these requisite products available, as it might take some time, depending on the policies at your site. The key requisites for a basic setup are the following:

- z/OS V2R1 or higher
- z/OS ICSF V2R1 or higher
- IBM 64-bit SDK for z/OS Java Version 8, SR7 FP10 or higher.

Required resources

The configuration of IzODA requires more than the typical system programming permissions and expertise; therefore, assistance from others might be needed. Table 1 on page 6 lists the administrators who are needed for the required and optional customization tasks.

These are the book numbers referenced in Table 1 on page 6:

Table 1. Planning checklist for a first-time installation. Planning checklist			
Complete?	Task	IT Role/Skills	Publications
Installation			
	Planning installation options on z/OS	z/OS system programmer	<i>Program Directory for IBM Open Data Analytics for z/OS and PSP buckets</i>
	Procuring, installing, and configuring prerequisite products, except IzODA	z/OS system programmer	<i>Program Directory for IBM Open Data Analytics for z/OS and PSP buckets</i>
	Installing IzODA	z/OS system programmer with UNIX skills	<i>Program Directory for IBM Open Data Analytics for z/OS, Part 2, “Installation,” on page 9, and PSP buckets</i>
Configuring Spark			
	Verifying environment and UNIX System Services configuration requirements.	z/OS system programmer, security administrator	Chapter 4, “Customizing your environment for z/OS Spark,” on page 15

Table 1. Planning checklist for a first-time installation. Planning checklist (continued)

Complete?	Task	IT Role/Skills	Publications
	Creating a user ID	z/OS system programmer, security administrator	Chapter 4, “Customizing your environment for z/OS Spark,” on page 15
	Customizing Apache Spark directory structure	z/OS system programmer, security administrator	Chapter 4, “Customizing your environment for z/OS Spark,” on page 15
	Configuring network, ports, and firewalls	z/OS system programmer, network administrator	Chapter 4, “Customizing your environment for z/OS Spark,” on page 15 and Chapter 5, “Customizing the Data Service server,” on page 85
	Configuring client authentication for Apache Spark	z/OS system programmer, security administrator, network administrator	Chapter 4, “Customizing your environment for z/OS Spark,” on page 15
	Configuring memory and CPU options	z/OS system programmer, security administrator	Chapter 4, “Customizing your environment for z/OS Spark,” on page 15
	Configuring WLM	z/OS system programmer, security administrator	Chapter 4, “Customizing your environment for z/OS Spark,” on page 15
Configuring MDS			
	Create required data sets and security application to use with the server.	z/OS system programmer, security administrator	Chapter 5, “Customizing the Data Service server,” on page 85
	Configuring WLM	z/OS system programmer, security administrator	Chapter 5, “Customizing the Data Service server,” on page 85
	Setting up security authorizations and APF-authorize LOAD library data sets	z/OS system programmer, security administrator	Chapter 5, “Customizing the Data Service server,” on page 85
	Configure MDS server to read optionally selected z/OS data	IMS administrator, Db2 administrator, CICS administrator, security administrator	<i>Open Data Analytics for z/OS Solutions Guide</i>

Table 1. Planning checklist for a first-time installation. Planning checklist (continued)

Complete?	Task	IT Role/Skills	Publications
	Optionally configure MDS server to read distributed data	Administrator of distributed server	Chapter 7, “Installing the JDBC Gateway,” on page 99 and Open Data Analytics for z/OS Solutions Guide
Configuring Anaconda			
	Environmental setup and Post-SMP/E installation instructions	z/OS system programmer with UNIX skills	Chapter 9, “Customizing Anaconda,” on page 113
Verifying IzODA			
	Verifying IzODA customization	z/OS system programmer with UNIX skills	Chapter 10, “Verifying the IBM Open Data Analytics for z/OS customization,” on page 117
	Verifying Data Service server installation	z/OS system programmer	Chapter 11, “Verifying the Data Service server installation,” on page 121
	Verifying IzODA product	z/OS system programmer with UNIX skills	Chapter 12, “Verifying the IBM Open Data Analytics for z/OS product,” on page 123

Part 2. Installation

Chapter 3. Installing IBM Open Data Analytics for z/OS

You can install IBM Open Data Analytics for z/OS (IzODA) by using CBPDO or, alternatively, SystemPac or ServerPac.

Before you begin

Ensure that the following software requirements for Open Data Analytics for z/OS have been met:

- IBM z/OS V2.1 or later
- The minimum required Java™ level is IBM 64-Bit SDK for z/OS Java Technology Edition V8, Service Refresh 7, FP 10. However, if the RELEASE file in the Spark installation directory indicates that the product was built with a later Java level, IBM urges you to use that Java level.
- Bourne Again Shell (bash) version 4.2.53 or version 4.3.48.

For the latest list of requirements, see the information in the Preventive Service Planning (PSP) bucket.

Migration notes: If you already use IBM z/OS Platform for Apache Spark, note the following differences in Open Data Analytics for z/OS:

- IzODA changes the level of Apache Spark. For more information, see [Appendix A, “Migrating to a new version of Apache Spark,”](#) on page 159.
- IzODA changes the default z/OS Spark installation directory to /usr/lpp/IBM/zspark/spark/sparknnn (for instance, /usr/lpp/IBM/zspark/spark/spark32x).
- IzODA uses UTF-8 encoding. For details, see [“Setting up a user ID for use with z/OS Spark”](#) on page 28 and [“Network port configurations”](#) on page 163.
- As of the December 2018 release, the Spark REST server port is disabled. You can enable connections to the REST port (such as when using cluster deploy mode) in your local spark-defaults.conf file, but the port will not function properly until you complete the setup to secure and enable the REST port.. For details, see [“Configuring networking for Apache Spark”](#) on page 37.
- IzODA introduces client authentication, which is enabled by default and requires additional setup. Apache Spark will not function properly until you complete the setup for client authentication or disable the client authentication function. For details, see [“Configuring z/OS Spark client authentication”](#) on page 41.
- IzODA changes the way you assign job names to executor and driver processes. IzODA no longer honors the specification of **spark.executorEnv._BPX_JOBNAME** on the command line or in an application. For details, see [“Assigning job names to Spark processes”](#) on page 73.
- If the PTF for APAR PI93605 is installed, Spark master and worker daemons will perform environment verification during initialization and will fail to start if the verification fails. The reason for termination can be found in the daemon's log. You can disable this feature by setting the `spark.zos.environment.verify` to false in `spark-defaults.conf`.
- z/OS IzODA Livy (Livy) is delivered through Anaconda (HANA11) and introduced in APAR (PH11339). The SMP/E APPLY process installs the Livy package into the Anaconda directory, but does not make them available for use. See [Chapter 8, “z/OS IzODA Livy Installation and Customization,”](#) on page 105 for instructions on how to get started with using the z/OS IzODA Livy package.

Additional migration note: If you are installing the PTF for APAR PI89136, note the following changes that are introduced by the APAR:

- APAR PI89136 changes the level of Apache Spark to 2.2.0. For more information, see [Appendix A, “Migrating to a new version of Apache Spark,”](#) on page 159.
- If you specify an incorrect job name prefix, Spark worker daemon will fail rather than ignoring the error. For more information, see [“Assigning job names to Spark processes”](#) on page 73.

- If client authentication is enabled, and you submit an application to the Spark master port in cluster-deploy-mode, then the Spark driver will run under the ID of the user who did the submit.

Note: Open Data Analytics for z/OS currently has some restrictions on Apache Spark functionalities. For a list of restrictions, see [Appendix G, “Restrictions,”](#) on page 185.

Note: A new service (PTF) level for Open Data Analytics for z/OS (FMID HSPK120) might provide a new version of Apache Spark. Before installing a new PTF, see [Appendix A, “Migrating to a new version of Apache Spark,”](#) on page 159.

About this task

Open Data Analytics for z/OS is supplied in a Custom-Built Product Delivery Offering (CBPDO, 5751-CS3). For installation instructions, see *Program Directory for IBM Open Data Analytics for z/OS*.

You can also install Open Data Analytics for z/OS with a SystemPac or ServerPac. For information about the various z/OS product installation offerings, see [z/OS Planning for Installation](#).

Service updates for Open Data Analytics for z/OS are provided as PTFs that perform a full replacement of the product. Therefore, you can use a PTF to update your existing installation or to perform a new installation.

IBM recommends that you mount your z/OS Spark file system from the same system on which the Spark cluster will be run.

Procedure

Complete the following steps to install Open Data Analytics for z/OS on your system.

1. Choose the most appropriate method for installing Open Data Analytics for z/OS.
2. Use the information in *Program Directory for IBM Open Data Analytics for z/OS* to install Open Data Analytics for z/OS on your system.

Results

Open Data Analytics for z/OS is installed on your z/OS system.

What to do next

Before you use Open Data Analytics for z/OS for the first time, follow the customization instructions in [Part 3, “Customization,”](#) on page 13.

Part 3. Customization

Customize your z/OS environment for z/OS Spark.

Chapter 4. Customizing your environment for z/OS Spark

Before you can use z/OS Spark, you must customize your environment for it and its dependent products. Complete this task after you install Open Data Analytics for z/OS but before your first use of it.

Before you begin

Follow the instructions in Chapter 3, [“Installing IBM Open Data Analytics for z/OS,”](#) on page 11 to install Open Data Analytics for z/OS on your system.

About this task

The default program location for z/OS Spark is `/usr/lpp/IBM/zspark/spark/`.

Procedure

Complete the following tasks to customize your environment for z/OS Spark. You can use the z/OS Spark configuration workflow as described in [“Using the Spark configuration workflow ”](#) on page 16 or you can follow these individual steps. You will need access to a TSO session, an OMVS session (preferably through a Putty terminal):

- __ a. [“Verifying the Java and bash environments”](#) on page 26
- __ b. [“Verifying configuration requirements for z/OS UNIX System Services”](#) on page 28
- __ c. [“Setting up a user ID for use with z/OS Spark”](#) on page 28
- __ d. [“Verifying the env command path”](#) on page 32
- __ e. [“Customizing the Apache Spark directory structure”](#) on page 32
 - __ i) [“Creating the Apache Spark configuration directory”](#) on page 33
 - __ ii) [“Updating the Apache Spark configuration files”](#) on page 34
 - __ iii) [“Creating the Apache Spark working directories”](#) on page 35
- __ f. [“Configuring networking for Apache Spark”](#) on page 37
- __ g. [“Configuring z/OS Spark client authentication”](#) on page 41
- __ h. [“Configuring IBM Java”](#) on page 57
- __ i. [“Creating jobs to start and stop Spark processes”](#) on page 58
- __ j. [“Setting up started tasks to start and stop Spark processes ”](#) on page 60
- __ k. [“Configuring memory and CPU options”](#) on page 65
- __ l. [“Configuring z/OS workload management for Apache Spark”](#) on page 73

Results

You have customized your environment for z/OS Spark.

What to do next

Continue with [Chapter 5, “Customizing the Data Service server,”](#) on page 85.

Using the Spark configuration workflow

Using the Spark configuration workflow to configure Spark.

About this task

You can use the the Workflows task in IBM z/OS Management Facility (z/OSMF) to configure Spark (FMID HSPK120). The information and commands are similar to those described in this manual.

The z/OSMF interface simplifies the configuration steps, collects input, discovers details about your installation, helps you manage task assignments to members of your team, and runs configuration jobs. The parts of the configuration workflow are as follows:

workflow-spark-config.xml

The workflow definition file. Specify the fully-qualified path to this file on the 'Create Workflow' dialog in the z/OSMF workflow task. The workflow definition and the other supporting files in this directory will then load into z/OSMF

workflow-spark-config-*.rexx

These REXX programs support various workflow tasks. The programs are invoked by the z/OSMF workflow. Do not run them manually unless directed by IBM Support.

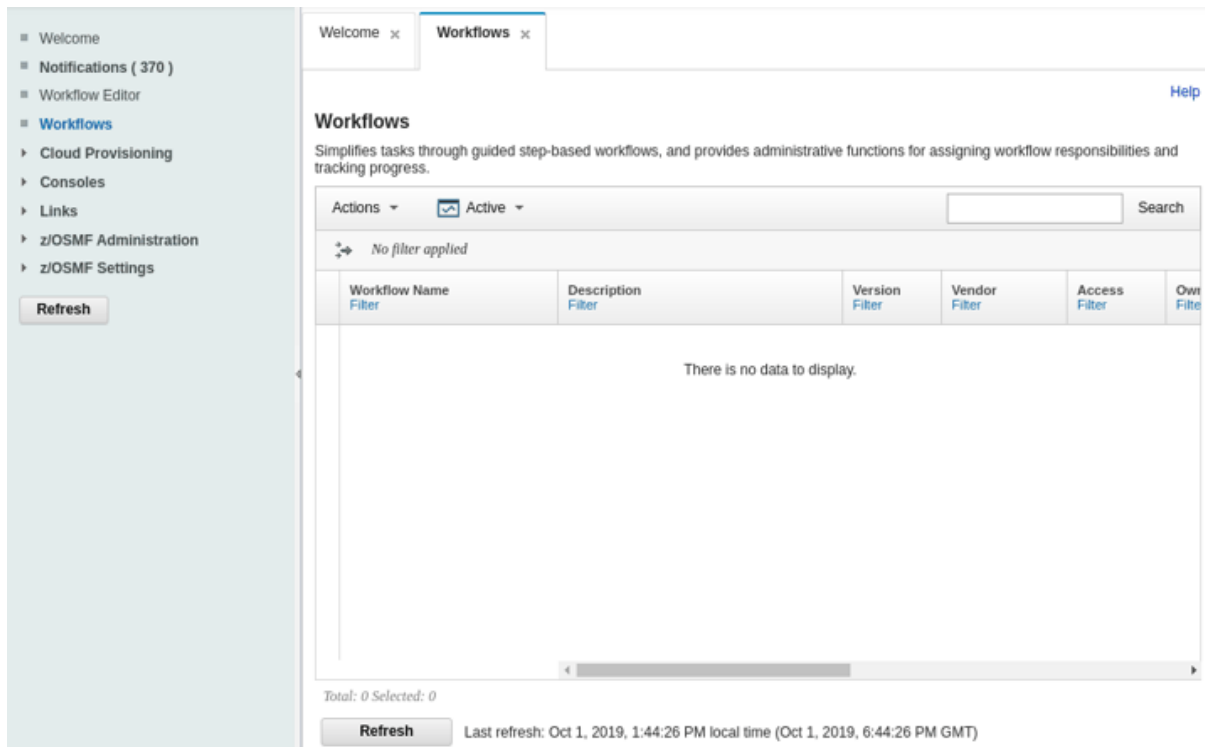
workflow-spark-config-attls*.template

Template files for configuration of Application Transparent Transport Layer Security (AT-TLS) and client authentication. These files are processed as input by the workflow code, but they can serve as a model for manually configuring AT-TLS and client authentication.

These workflow files are located in the Spark installation directory under the zos-workflow directory for the Spark releases supported by the workflow. The default installation directory is `/usr/lpp/IBM/izoda/spark/sparknnn`, where *nnn* is the current Spark version (for example, `/usr/lpp/IBM/izoda/spark/spark24x` for Spark 2.4.8). Note that for all Spark 2.4 releases, *nnn* is 24x. (for example, `/usr/lpp/IBM/izoda/spark/spark24x` for Spark 2.4.8).

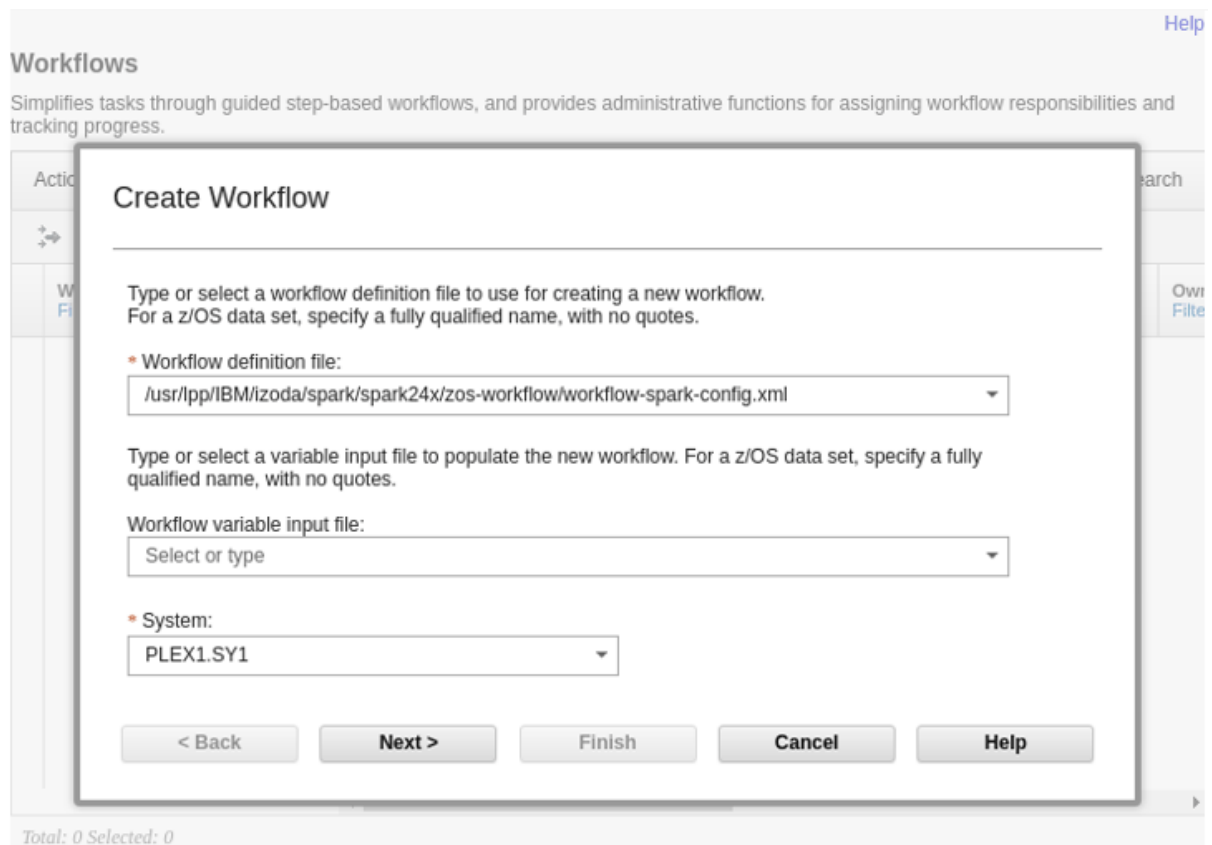
Procedure

1. Create and run the workflow using a UID 0 user.
2. From the Workflows task on z/OSMF, locate the Workflows table.



3. Click the Actions drop down menu and select **Create Workflow...**
4. From the Create Workflow dialog, enter the fully-qualified path to the workflow-spark-config.xml file in the **Workflow definition file** field.

Leave the 'Workflow variable input file' field empty. In the 'System' field, select the local z/OS system name.



5. Click **Next >**.
6. Ensure the 'Owner user ID' field is the same as the current UID 0 user.
Check the **Assign all steps to owner user ID** checkbox.

Create Workflow

Workflow definition file:
/usr/lpp/IBM/izoda/spark/spark24x/zos-workflow/workflow-spark-config.xml

Description:
Configure IBM Open Data Analytics for zOS Spark Software

Vendor:	Version:	Is Callable:
IBM	2.4.x.1	Cannot be called by another workflow

* Workflow name:

* Owner user ID:	System:
<input type="text" value="wellie0"/>	PLEX1.SY1

Comments:

* Access([Learn More](#)):

☒ Open workflow on finish ☒ Assign all steps to owner user ID

7. Click Finish.
Read and run the steps in the order presented in the workflow.

Workflows > Configure IBM Open Data Analytics for zOS Spark Software - Workflow_0 Help

Configure IBM Open Data Analytics for zOS Spark Software - Workflow_0

Description:
Configure IBM Open Data Analytics for zOS Spark Software

Recent completion:

Owner: wellie0

Steps:

System: PLEX1.SY1

Status:

Is Callable:
Cannot be called by another workflow

Associated user:

Notes | History

Contains Parallel Steps: No

Workflow Steps

Actions Search

No filter applied

<input type="checkbox"/> State Filter	No. Filter	Title Filter	CalledWorkflow Filter	Automated Filter
<input type="checkbox"/> Ready	1	How to use this workflow		No
<input type="checkbox"/> Not Ready	2	(Optional) How to use this workflow to copy or upgrade from a previous Spark configuration		No
<input type="checkbox"/> Not Ready	3	Ensure that IBM Open Data Analytics for z/OS Software requirements are met		
<input type="checkbox"/> In Progress	4	Ensure that IBM z/OS UNIX configuration requirements are met		
<input type="checkbox"/> Not Ready	5	Configure IBM Java		
<input type="checkbox"/> Not Ready	6	Ensure that IBM Open Data Analytics is installed		
<input type="checkbox"/> Not Ready	7	Customize IBM Open Data Analytics for z/OS		

Total: 120 Selected: 0

Return to Workflows
Refresh
Last refresh: Oct 1, 2019, 1:49:32 PM local time (Oct 1, 2019, 6:49:32 PM GMT)

Note: The workflow can be used to copy and modify an existing Spark configuration to be a new configuration or to update an existing configuration. To use the workflow for these purposes, read and follow the instructions in step 2 - How to use this workflow to copy or upgrade from a previous Spark configuration.

What to do next

You may assign certain steps to other users who have other skills or authority such as your Security Administrator or Network administrator depending on how your organization divides configuration work. IBM Open Data Analytics for z/OS also provides a workflow that provides step-by-step instructions for tuning and allocating resources. That workflow exists at the following location: <https://github.com/IBM/IBM-Z-zOS/tree/master/zOS-Workflow/IzODA%20Workflows>

Note that customization steps need to be done in sequential order.

For more information about the Workflows task in z/OSMF, see the following location: https://www.ibm.com/support/knowledgecenter/SSLTBW_2.3.0/com.ibm.zosmfworkflows.help.doc/izuWFhpWorkflowsTask.html

Upgrading Spark configuration workflows

Upgrading Spark configuration workflows.

About this task

You may optionally upgrade your existing Spark configuration workflow in order to receive fixes or new function. You can then run any new or changed steps as needed. Note that you may also need to run subsequent steps that have dependencies on any changes. Steps that need to be rerun will no longer be marked as Complete. The Spark configuration workflow supports running all of the steps again; however, any changes you have made to the configuration manually might be overwritten.

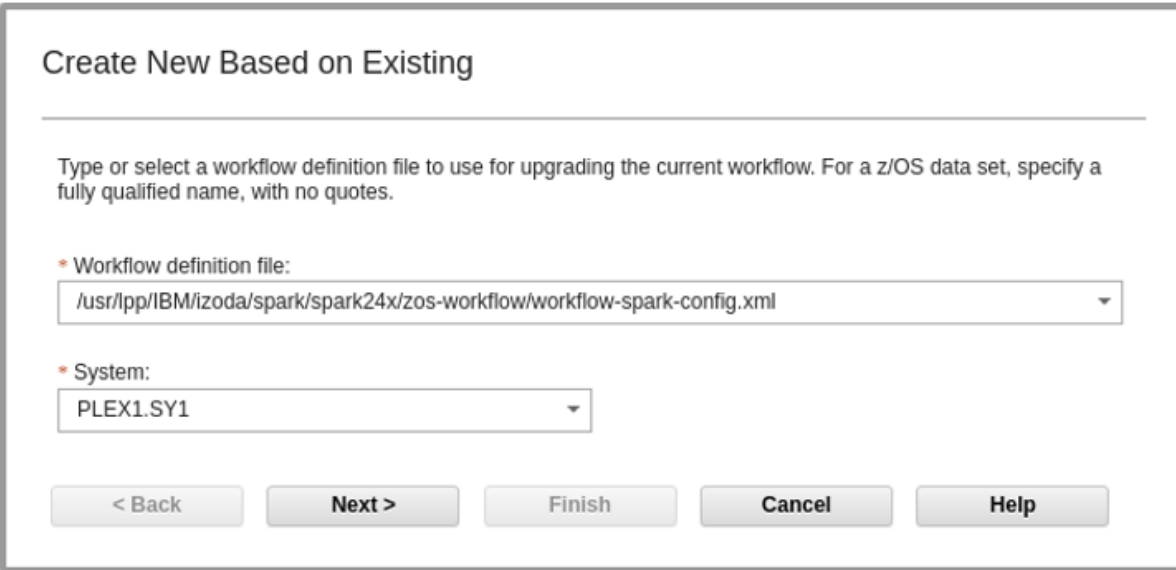
Another option is to delete or archive the old instance of the Spark configuration workflow and create a new one using the new xml file. After creating a new instance, you will need to run all of the steps again.

Only one instance of a given workflow can be active at a time. Follow these steps to upgrade the workflow:

Procedure

1. Select the previous instance of the Spark configuration workflow in the Workflows Task.
2. From the Actions menu, select the 'Create New Based on Existing' action.

The Create New Based on Existing window is displayed.



3. Input the file name of the workflow (for example, /usr/lpp/IBM/izoda/spark/spark24x/zos-workflow/workflow-spark-config.xml) and select the appropriate system. Use the same system name used when you created the original workflow instance.

Click **Next>**.

The Create New Based on Existing - Replace Workflow window is displayed:

Create New Based on Existing - Replace Workflow

Selected workflow:
Configure IBM Open Data Analytics for zOS Spark Software - Workflow_0

Upgrade notes:
Update workflow version (e.g. 2.4.x.1 to 2.4.x.2) to incorporate workflow enhancements and fixes to your existing (possibly in-progress) workflow.

Workflow definition file: /usr/lpp/IBM/izoda/spark/spark24x/zos-workflow/workflow-spark-config.xml	Version: 2.4.x.1	Upgrade version 2.4.x.2
Description: Configure IBM Open Data Analytics for zOS Spark Software	Owner: wellie0	System: PLEX1.SY1

Percent complete:

20%

Steps complete:
20 of 101

Status:
 In Progress

☒ **Copy variable values based on upgrade definition**

* Access([Learn More](#)):

Public

☒ Copy workflow notes
 ☒ Copy workflow history

Step attributes:

☒ Copy step attributes based on upgrade definition
 ☐ Assign all steps to owner user ID
 ☐ No copy and assign

☒ Open workflow on finish

< Back

Next >

Finish

Cancel

Help

4. Ensure 'Copy variable values based on upgrade definition' is checked. Unchecking this means the upgrade will discard your previous data input for the workflow, such as SPARK_HOME and other configuration values.

By default the workflow notes and workflow history are copied to the new workflow instance. You may uncheck 'Copy workflow notes' or 'Copy workflow history' to omit them. Workflow notes are created by users of the workflow. Workflow history is a z/OSMF-managed log of activity in the workflow, which might be useful to IBM support for certain workflow issues.

5. Make a choice under Step Attributes:

Copy step attributes based on upgrade definition

(Default) This copies your previous input and retains the status of the steps and job output from the previous workflow. Steps that have changed, or were affected by changes in the new workflow version may have their step status and job output discarded when the new instance is created. Steps affected by such changes and any new steps will also have their ownership changed to 'Unassigned.' It is easy to locate the steps you need to run after the upgrade by looking at the step status or by filtering on the State column. However, extra work is required as the workflow owner must assign the owner to all of the affected steps.

See “Assigning an owner to new or changed steps ” on page 22 for instructions.

Note: With this option, if you run a step with a state of Complete again, z/OSMF does not update the step status if the step fails. It will also not change the status of steps that depend on the step.

Assign all steps to owner user ID

Assigns all steps to you, the workflow owner. However, it will not copy step status or job output from the previous workflow. It will be more difficult to locate steps that changed.

No copy and assign

Does not copy step attributes and job output, nor will it assign steps. This option is not recommended for the Spark configuration workflow.

6. Click **Finish**.

z/OSMF cancels the original workflow, creates the new instance of the workflow, and copies data as you specified.

What to do next

Go to [“Assigning an owner to new or changed steps”](#) on page 22.

Assigning an owner to new or changed steps

Assigning an owner to new or changed steps while upgrading Spark configuration workflows.

About this task

Any steps IBM added to the workflow, or changed in some important way, must now be assigned an owner. After you perform these actions, the Perform tab of the steps is enabled so that step actions can be performed.

Procedure

1. Look for a state of Unassigned in the State column for all of the workflow steps.

Actions ▾				
↕ 9 of 86 items shown. Clear filter				
<input type="checkbox"/>	State contains "Unassigned"	No. Filter	Title Filter	CalledWorkflow Filter
<input type="checkbox"/>	In Progress	6	<input type="checkbox"/> Customize IBM Open Data Analytics for z/OS	
<input type="checkbox"/>	Unassigned	6.4	<input type="checkbox"/> Consider your planned usage of Dynamic Allocation	
<input type="checkbox"/>	In Progress	7	<input type="checkbox"/> Configure networking for Apache Spark	
<input type="checkbox"/>	Unassigned	7.2	<input type="checkbox"/> Consider your planned usage of the REST server	
<input type="checkbox"/>	Unassigned	7.3	<input type="checkbox"/> Consider your planned usage of the Spark shuffle service	
<input type="checkbox"/>	In Progress	8	<input type="checkbox"/> Configuring AT-TLS server policies and client authentication for z/OS Spark	
<input type="checkbox"/>	Unassigned	8.4	<input type="checkbox"/> Generate AT-TLS policy file	
<input type="checkbox"/>	In Progress	9	<input type="checkbox"/> Policy Agent Configuration	
<input type="checkbox"/>	Unassigned	9.3	<input type="checkbox"/> Create the Policy Agent configuration files	

2. Use the filter functionality on the table to make this easier. Click the Filter link under the State column to specify a filter. Parent steps appear when more than one step matches, even though the parent's status does not match the filter.

The owner of the workflow selects the unassigned step(s) in the workflow and selects Assignment and Ownership → Add Assignees from the Actions menu.

3. Select a user to own the step.

The user that you want to assign could be the workflow owner or you.

[Workflows](#) > [Configure IBM Open Data Analytics for z/OS Spark Software - Workflow_0](#) > Add Assignees [Help](#)

Add Assignees

Select one or more SAF user IDs, SAF groups or z/OSMF roles to be assigned to the selected steps.

Selected Steps

Available assignees

Actions

No filter applied

<input type="checkbox"/>	Name <small>Filter</small>	Type <small>Filter</small>	
<input checked="" type="checkbox"/>	wellie0	SAF user ID	
<input type="checkbox"/>	z/OS Security Administrator	z/OSMF role	
<input type="checkbox"/>	z/OSMF Administrator	z/OSMF role	
<input type="checkbox"/>	z/OSMF User	z/OSMF role	

Add >

Add All >>

< Remove

<< Remove All

* Assignees to be added:

wellie0

Total: 4 Selected: 1










Comments:

☐ Send z/OSMF notifications to assignees (comments are not included on notifications)

OK

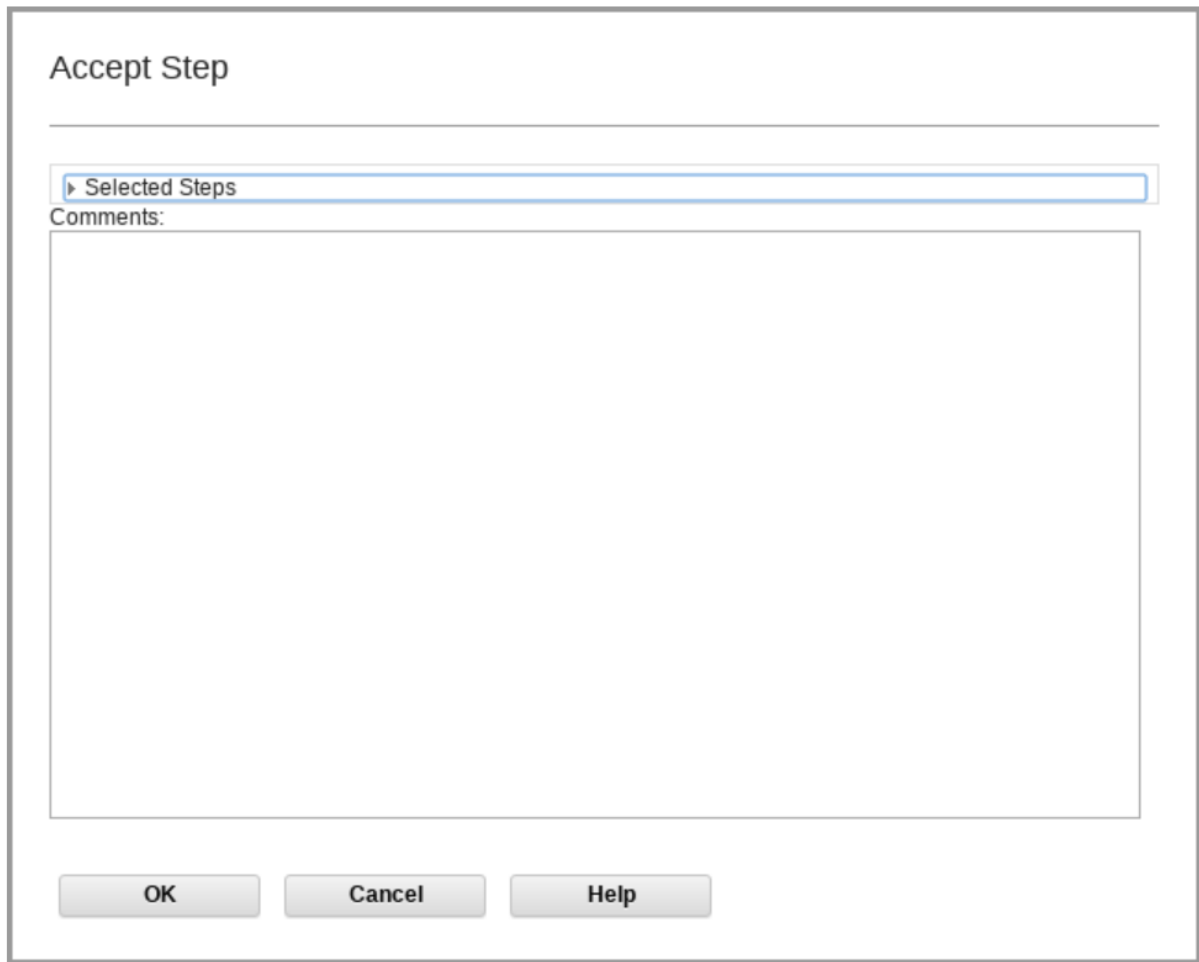
Cancel

4. The step status shows as Assigned. If you used the filter in the table, change it to show the rows with Assigned state.

Actions ▾				
↔ 9 of 86 items shown. Clear filter				
<input type="checkbox"/>	State contains "Assigned"	No. Filter	Title Filter	CalledWorkflow Filter
<input type="checkbox"/>	 In Progress	6	<input type="checkbox"/> Customize IBM Open Data Analytics for z/OS	
<input type="checkbox"/>	 Assigned	6.4	■ Consider your planned usage of Dynamic Allocation	
<input type="checkbox"/>	 In Progress	7	<input type="checkbox"/> Configure networking for Apache Spark	
<input type="checkbox"/>	 Assigned	7.2	■ Consider your planned usage of the REST server	
<input type="checkbox"/>	 Assigned	7.3	■ Consider your planned usage of the Spark shuffle service	
<input type="checkbox"/>	 In Progress	8	<input type="checkbox"/> Configuring AT-TLS server policies and client authentication for z/OS Spark	
<input type="checkbox"/>	 Assigned	8.4	■ Generate AT-TLS policy file	
<input type="checkbox"/>	 In Progress	9	<input type="checkbox"/> Policy Agent Configuration	
<input type="checkbox"/>	 Assigned	9.3	■ Create the Policy Agent configuration files	

The new step owner receives a z/OSMF notification and must open the Spark Configuration workflow.

5. The new step owner selects the steps assigned to them, selects the Accept action from the actions menu, and clicks **OK** on the Accept Step window.












The image shows a dialog box titled "Accept Step". It has a horizontal line below the title. Below the line is a tabbed interface with one visible tab labeled "Selected Steps". Below the tabs is a large text area labeled "Comments:". At the bottom of the dialog are three buttons: "OK", "Cancel", and "Help".

Note that until Accept Step is run, the Perform tab on the step remains disabled, and the step actions cannot be performed

What to do next

The step(s) will show as 'Ready.' Again, you will need to adjust the filter to see the group.

Actions ▾				
🔍 9 of 86 items shown. Clear filter				
<input type="checkbox"/>	State contains "Ready"	No. Filter	Title Filter	CalledWorkflow Filter
<input type="checkbox"/>	 In Progress	6	<input type="checkbox"/> Customize IBM Open Data Analytics for z/OS	
<input type="checkbox"/>	 Ready	6.4	■ Consider your planned usage of Dynamic Allocation	
<input type="checkbox"/>	 In Progress	7	<input type="checkbox"/> Configure networking for Apache Spark	
<input type="checkbox"/>	 Ready	7.2	■ Consider your planned usage of the REST server	
<input type="checkbox"/>	 Not Ready	7.3	■ Consider your planned usage of the Spark shuffle service	
<input type="checkbox"/>	 In Progress	8	<input type="checkbox"/> Configuring AT-TLS server policies and client authentication for z/OS Spark	
<input type="checkbox"/>	 Not Ready	8.4	■ Generate AT-TLS policy file	
<input type="checkbox"/>	 In Progress	9	<input type="checkbox"/> Policy Agent Configuration	
<input type="checkbox"/>	 Not Ready	9.3	■ Create the Policy Agent configuration files	

Verifying the Java and bash environments

Complete this task to verify your Java and bash environments for use with IBM Open Data Analytics for z/OS.

Procedure

1. Ensure that the system on which you intend to run Open Data Analytics for z/OS contains an instance of IBM 64-Bit SDK for z/OS Java Technology Edition V8 Service Refresh 7 (Java 8 SR7). If the RELEASE file in the Spark installation directory indicates that the product was built with a later Java level, IBM urges you to use that Java level.

The default path to IBM 64-Bit SDK for z/OS Java Technology Edition V8 is `/usr/lpp/java/J8.0_64`. If your path is different, make note of it.

Path to IBM 64-Bit SDK for z/OS Java Technology Edition V8:	
---	--

2. Ensure that the installed bash shell level is 4.2.53 or 4.3.48.
 - a) Open an SSH or Telnet shell environment and run the following command:

```
bash -version
```

- If bash is on your path, the command returns information about the installed version. For example, the following sample output indicates that a version of bash that is too old is installed on the system:

```
ID > bash -version
GNU bash, version 2.03.0(1)-release (i370-ibm-mvs)
Copyright 1998 Free Software Foundation, Inc.
```

The following sample output indicates that the minimum required version of bash is already installed on the system:

```
ID > bash -version
GNU bash, version 4.2.53(2)-release (i370-ibm-openedition)
Copyright (C) 2011 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later
<http://gnu.org/licenses/gpl.html>

This is free software; you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
```

- If bash is not on your path, search your system for bash. The default path to bash 4.2.x and 4.3.x are /usr/bin/bash-4.2 and /usr/bin/bash-4.3 respectively, though your system might contain more than one instance of bash. Go to all the directories where you find instances of bash and run ./bash -version to call the instance and check the version.

Tip: You can use the **find** command to search for bash. For instance, the following command searches for all files named bash from the root (/) directory:

```
find / -name "bash"
```

Note the following points about this command:

- This command traverses your whole system and will likely take some time to complete.
 - You will see access errors if you run the command from a user ID that does not have sufficient authority to traverse and access all of the directories and files in your system.
 - You can narrow the search to directories that are likely candidates for product installations, such as /usr/bin and /usr/lpp.
- b) If bash is not installed or a version other than 4.2.53 or 4.3.48 are installed, obtain and install the 4.2.53 or 4.3.48 level of bash. You can obtain the proper levels of bash from z/OS IzODA Anaconda (FMID HANA110), or download them from [Rocket z/OS Open Source Community Downloads \(www.rocketsoftware.com/ported-tools\)](http://www.rocketsoftware.com/ported-tools).

Select to download bash and follow the instructions to register with Rocket. You can then download the archive file and the "Getting Started" document.

Tip: The bash installation process involves the following actions:

- i) Creating a mount point and file system on z/OS to hold the bash files
- ii) Uploading in binary the archive file into your new file system
- iii) Extracting the archive file with `gzip -d filename.tar.gz`
- iv) Extracting the file with `tar -xvfo filename.tar`

When the correct version of bash is installed, a set of directories (/bin and /man) exists that represents the bash shell code. Make note of the path to the bash /bin directory.

Path to the bash /bin directory:	
----------------------------------	--

What to do next

Continue with [“Verifying configuration requirements for z/OS UNIX System Services” on page 28.](#)

Verifying configuration requirements for z/OS UNIX System Services

Spark runs in a z/OS UNIX System Services (z/OS UNIX) environment. Complete this task to ensure that the configuration requirements for z/OS UNIX are met.

Procedure

1. If this is the first time you are running applications in z/OS UNIX, see [z/OS UNIX System Services Planning](#) to ensure that your z/OS UNIX environment is properly configured and customized.

For instance, Spark should not be run as UID 0.

However, if you choose to run Spark as UID 0 in an environment where multiple users are mapped to UID 0, you might encounter problems with the wrong shell profile being read and the required environment variables not being set. For instance, `$HOME/.profile` might be read for user BOB mapped to UID 0, when you really wanted the shell profile for SPARKID (also mapped to UID 0) to be read.

For alternatives to setting multiple user IDs as UID 0, see "Superusers in z/OS UNIX" in [z/OS UNIX System Services Planning](#).

2. Optional: Consider enabling health checks for z/OS UNIX and other z/OS system services. for more information, see [IBM Health Checker for z/OS User's Guide](#).

See ["Using IBM Health Checker for z/OS to monitor Spark workload"](#) on page 151 for a list of health checks that you might find helpful.

What to do next

Continue with ["Setting up a user ID for use with z/OS Spark"](#) on page 28.

Setting up a user ID for use with z/OS Spark

Complete this task to set up a user ID for use with z/OS Spark.

About this task

For this task, you can either create a new user ID to use for z/OS Spark, or you can use an existing user ID.

Note: The user ID of the Spark worker daemon requires READ access to the BPX.JOBNAME profile in the FACILITY class to change the job names of the executors and drivers.

Procedure

1. Choose or create an appropriate user ID for use with z/OS Spark.

Specifically, this is the user ID under which the Spark cluster is started, known as the Spark ID in this documentation. The Spark ID should have a non-zero UID (not a superuser) and should not have access to any data beyond what it needs for running a Spark cluster. Ensure that the default shell program in the OMVS segment for the Spark ID is set to the bash shell. Also, ensure that the user ID has, at a minimum, the authority to create directories and extract archived or compressed files.

Tip: If you need to change or create a user ID, work with your security administrator to do so.

Using an existing user ID

If you intend to use an existing user ID, you might need to first update the OMVS segment to set bash as the default shell program for the user ID. Complete the following steps to determine whether the **PROGRAM** attribute of the OMVS segment is valid for the target user ID.

- a. Use SSH to log on using the user ID.
- b. Run `echo $SHELL` and review the output.

If bash is still not listed as the default shell for the user ID, a potential reason is because `/etc/profile` is explicitly invoking a shell other than bash. If so, work with your system administrator to update `/etc/profile` to define the operative shell in the OMVS segment.

The following code provides an example of how `/etc/profile` might override the bash shell set in the OMVS segment with another shell:

```
if [ -z "$STEPLIB" ] && tty -s;
then
    export STEPLIB=none
    exec -a $0 $SHELL -
fi
```

Creating a new user ID

If you intend to create a new user ID for z/OS Spark, establish the OMVS segment during creation.

The following JCL example shows how to create a new user ID and group for the Spark ID, SPARKID, which will be used to run z/OS Spark:

```
//SPARK    JOB (0), 'SPARK RACF',CLASS=A,REGION=0M,
//          MSGCLASS=H,NOTIFY=&SYSUID
//*-----*
//RACF      EXEC PGM=IKJEFT01,REGION=0M
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD *
ADDGROUP SPKGRP OMVS(AUTOUID) OWNER(SYS1)
ADDUSER SPARKID DFLTGRP(SPKGRP) OMVS(AUTOUID HOME(/u/sparkid) -
PROGRAM(/shared/rocket/bash-4.2/bin/bash)) -
NAME('Spark ID') NOPASSWORD NOIDCARD
ALTUSER SPARKID PASSWORD(SPARKID) NOEXPIRED
/*
```

Notes:

- Use of **AUTOUID** and **AUTOUID** in the example is based on a local preference. Your coding might differ.
- Set the **PROGRAM** attribute to define the path to your own installation of bash 4.2.53 or 4.3.48 that you noted previously.

The chosen user ID is now properly set up to run z/OS Spark. Use this user ID for all remaining customization steps that require a user ID.

2. Configure the z/OS UNIX shell environment for both your Spark ID and all users of z/OS Spark.

z/OS Spark requires certain environment variables to be set. Consider the scope under which you want this environment to take effect. For example:

- Do you want to configure Spark for all users or a subset of users?
- Do you have other Java applications that require a different level of Java or require different (conflicting) Java settings?

At a high level, this environment can be set for all users of both shells, an individual user's shell environment, or, for some settings, for users only when they issue Spark commands. Minimally, you must set up the environment for the Spark ID and for each user of Spark.

Use the information in [Table 2 on page 29](#) to decide where to set each environment variable. This information applies for users with either a login shell of bash or `/bin/sh`.

Table 2. Scope of environment variables	
Environment variables set in this file...	Have this scope...
<code>/etc/profile</code>	All users, all the time
<code>\$HOME/.profile</code> for specific users	Specific users, all the time
<code>spark-env.sh</code>	Specific users, only for Spark commands

Note: The `spark-env.sh` file is discussed in more detail in [“Updating the Apache Spark configuration files”](#) on page 34.

Values that you set for environment variables in the `$HOME/.profile` file override the values for those variables in the `/etc/profile` system file. Values that you set in `spark-env.sh` override any values previously set in either `/etc/profile` or `$HOME/.profile`.

Tip: If the Spark ID does not already have a `$HOME/.profile` file, create one now.

- a) Determine which of the files listed in [Table 2](#) on page 29 you want to update.
(Creation and customization of the `spark-env.sh` file will be discussed later.)
- b) For the files (listed in [Table 2](#) on page 29) that you determined need to be updated, edit each to set the environment variables, as follows:

- Set **JAVA_HOME** to point to the location of IBM 64-Bit SDK for z/OS Java Technology Edition V8.
- Set **PATH** to include the `/bin` directory of IBM 64-Bit SDK for z/OS Java Technology Edition V8.

Tip: You can set this value by using **\$JAVA_HOME**.

- Set **PATH** to prioritize the path to the `/bin` directory of bash 4.2.53 or 4.3.48 higher than any earlier version of bash that exists on your system.
- Set **IBM_JAVA_OPTIONS** to provide file encoding to UTF-8.
- Set **_BPXK_AUTOCVT** to ON to enable the automatic conversion of tagged files. APAR PH01619 adds this to `spark-env.sh` and `spark-env.sh.template` by default.
- Include an `export` statement to make all the variables available to the z/OS UNIX shell environment.

The following example illustrates how to code the `.profile` file for these environment variable settings:

```
# Spark ID .profile
JAVA_HOME=/shared/java/java_1.8_64
PATH=$JAVA_HOME/bin:/shared/rocket/bash-4.2/bin:$PATH:$HOME:
IBM_JAVA_OPTIONS="-Dfile.encoding=UTF8"
_BPXK_AUTOCVT=ON

# This line sets the prompt
PS1='$LOGNAME': '$PWD': ' >'

# This line exports the variable settings
export JAVA_HOME PATH IBM_JAVA_OPTIONS _BPXK_AUTOCVT PS1
```

The same syntax applies for `/etc/profile`, `$HOME/.profile`, and `spark-env.sh`.

- c) If you set the environment variables in the profile (as in either of the first two rows in [Table 2](#) on page 29), skip to step “2.d” on page 30 now. Otherwise, if you set the environment variables only in `spark-env.sh` (as in the third row in [Table 2](#) on page 29), issue the following command in an SSH or Telnet shell environment to source the `spark-env.sh` file:

```
source spark-env.sh
```

- d) In an SSH or Telnet shell environment, run the following command to verify that the bash version is set properly.

```
bash -version
```

The command returns a version number of 4.2.53 or 4.3.48. If it does not, ensure that the **PATH** value in the file you updated in step “2.b” on page 30 lists the latest version of the bash `/bin` directory before any other bash installations.

- e) In an SSH or Telnet shell environment, run the following command to verify that the correct level of bash is set as the default.

```
ps -p $$
```

The command returns the value of the process ID and indicates the shell program that is used, for example:

```
SPARKID:/u/sparkid: >ps -p $$
    PID TTY          TIME CMD
 16777299 ttty0000    0:00 /shared/rocket/bash-4.2/bin/bash
```

This example output shows that the installation path is correctly set to the 4.2.53 installation of bash as provided on the **PROGRAM** attribute of the user ID OMVS segment.

If the latest copy of bash is not listed, something in /etc/profile is overriding the shell. Ensure that /etc/profile is correct.

- f) In an SSH or Telnet shell environment, issue the following command to verify that **JAVA_HOME** is set to IBM 64-Bit SDK for z/OS Java Technology Edition V8.

```
java -version
```

You should see output similar to the following example:

```
java version "1.8.0_231"
Java(TM) SE Runtime Environment (build 8.0.7.0 - pmz6480sr7 - 20191107_01(SR7))
IBM J9 VM (build 2.9, JRE 1.8.0 z/OS s390x-64-Bit
Compressed References 20191106_4321 (JIT enabled, AOT enabled)
OpenJ9 - f0b6be7
OMR - 18d8f94
IBM - 233dfb5)
JCL - 20191016_01 based on Oracle jdk8u231-b10
```

If the output is incorrect or Java is not found, issue the following command:

```
echo $JAVA_HOME
```

The command returns the path to IBM 64-Bit SDK for z/OS Java Technology Edition V8. If it does not, ensure that the **JAVA_HOME** value is set correctly in the file you updated in step “2.b” on page 30.

- g) In an SSH or Telnet shell environment, run the following command to verify the correct file encoding.

```
echo $IBM_JAVA_OPTIONS
```

The command returns -Dfile.encoding=UTF8. If it does not, ensure that the **IBM_JAVA_OPTIONS** value is set correctly in the file you updated in step “2.b” on page 30.

- h) In an SSH or Telnet shell environment, run the following command to verify the automatic conversion of tagged files.

```
echo $_BPXK_AUTOCVT
```

The command returns ON. If it does not, ensure that the **_BPXK_AUTOCVT** value is set correctly in the file you updated in step “2.b” on page 30.

3. Permit the **SPARKID** to spawn USS address spaces with specific job names.

The z/OS Spark worker spawns new address space using the job name specifications in the **spark-defaults.conf** file.

Action required:

Permit the **SPARKID** to the BPX.JOBNAME profile in the security product. For RACF, this would be PERMIT BPX.JOBNAME CLASS(FACILITY) ID(SPARKID) ACCESS(READ)

Results

Your chosen user ID is now ready for use with z/OS Spark.

What to do next

Continue with [“Verifying the env command path”](#) on page 32.

Verifying the env command path

Complete this task to verify the path for the **env** command.

About this task

Before Spark 2.2.0 (introduced by APAR PI89136), the shell scripts for IBM Open Data Analytics for z/OS Spark require `/usr/bin/env`. If your system doesn't have `/usr/bin/env`, complete this task to create a symbolic link for `/usr/bin/env`.

Spark 2.2.0 changes the shell scripts to use `/bin/env`, which is a more common set-up. If your system doesn't have `/bin/env`, use the steps that are outlined in this task as a guideline to create a symbolic link for `/bin/env`.

Procedure

1. Ensure that `/usr/bin/env` exists and provides a correct listing of the environment.

In an SSH or Telnet shell environment, run the following command to verify the location and contents of `env`:

```
/usr/bin/env
```

The command returns a list of name and value pairs for the environment in your shell.

If `/usr/bin/env` does not exist, complete the following steps to set it up:

- a) Locate the **env** program on your system.
A potential location is in `/bin/env`.
- b) Create a symbolic link (symlink) so that `/usr/bin/env` resolves to the true location of **env**.
For example:

```
ln -s /bin/env /usr/bin/env
```

- c) In an SSH or Telnet shell environment, run the following command to verify that the symlink works.

```
/usr/bin/env
```

The command returns a list of name and value pairs for the environment in your shell.

2. Verify that the symbolic link for the **env** command persists across system IPLs.

Depending on how `/usr/bin/` is configured on your system, the symbolic link for `/usr/bin/env` might not persist across an IPL without additional setup. Ensure that your IPL setup includes creation of this symbolic link, if necessary. For instance, you can update your `/etc/rc` file to include the command to create the symbolic link. The `/etc/rc` file is typically used to customize commands for z/OS UNIX application services.

What to do next

Continue with [“Customizing the Apache Spark directory structure”](#) on page 32.

Customizing the Apache Spark directory structure

IBM Open Data Analytics for z/OS installs Apache Spark into a z/OS file system (zFS) or hierarchical file system (HFS) directory. This documentation refers to the installation directory as **SPARK_HOME**. The default installation directory is `/usr/lpp/IBM/zspark/spark/sparknnn`, where *nnn* is the current Apache Spark version (for instance, `/usr/lpp/IBM/izoda/spark/spark24x` for Spark 3.2.0).

By default, Apache Spark runs from the installation directory, and most of its configuration files, log files, and working information are stored in the installation directory structure. On z/OS systems, however, the use of the installation directory for all of these purposes is not ideal operating behavior. Therefore, by default, Open Data Analytics for z/OS installs Apache Spark in a read-only file system. The following tasks describe how to set up customized directories for the Apache Spark configuration files, log files, and temporary work files. While you can customize the directory structure used by Apache Spark, the examples here follow the Filesystem Hierarchy Standard.

Plan to work with your system programmer who has authority to update system directories.

Note: **SPARK_HOME** is an environment variable that is used by many Apache Spark scripts. This variable must contain the path to the z/OS Spark installation directory. In step “2” on page 29 of “[Setting up a user ID for use with z/OS Spark](#)” on page 28, it was determined which files needed to be updated to set the z/OS UNIX shell environment. Update the files modified in that step to set and export the **SPARK_HOME** environment variable. For example:

```
export SPARK_HOME=/usr/lpp/IBM/izoda/spark/spark24x
```

Creating the Apache Spark configuration directory

Complete this task to create a customized directory for the Apache Spark configuration files.

About this task

The default Apache Spark configuration directory is `$SPARK_HOME/conf`. In accordance with the Filesystem Hierarchy Standard (FHS), this task creates a new configuration directory under `/etc`. This task also ensures that the user ID that will run Apache Spark programs has read/write access to the new directory and sets the **SPARK_CONF_DIR** environment variable to point to the new directory.

Procedure

1. Open an SSH or Telnet shell environment and create a new directory under `/etc` for the Apache Spark configuration files.

For example, to create the `/etc/spark/conf` directory, enter the following command:

```
mkdir -p /etc/spark/conf
```

2. Provide read/write access to the new directory to the user ID that runs Open Data Analytics for z/OS.
3. Ensure that the **SPARK_CONF_DIR** environment variable points to the new directory.

For example:

```
export SPARK_CONF_DIR=/etc/spark/conf
```

Note: The **SPARK_CONF_DIR** environment variable can be set and exported as in this example in the `$HOME/.profile` or the `/etc/profile` as determined in Step 2 in “[Setting up a user ID for use with z/OS Spark](#)” on page 28.

Results

You now have a customized directory to hold the Apache Spark configuration files.

What to do next

Continue with “[Updating the Apache Spark configuration files](#)” on page 34.

Updating the Apache Spark configuration files

Complete this task to copy the Apache Spark configuration files to your new configuration directory and update them.

About this task

There are three main Apache Spark configuration files:

spark-env.sh

A shell script that is sourced by most of the other scripts in the Apache Spark installation. You can use it to configure environment variables that set or alter the default values for various Apache Spark configuration settings. For sample contents of this file, see [Appendix B, “Sample configuration and AT-TLS policy rules for z/OS Spark client authentication,” on page 163.](#)

spark-defaults.conf

A configuration file that sets default values for the Apache Spark runtime components. You can override these default values on the command line when you interact with Spark using shell scripts. For sample contents of this file, see [Appendix B, “Sample configuration and AT-TLS policy rules for z/OS Spark client authentication,” on page 163.](#)

log4j.properties

Contains the default configuration for log4j, the logging package that Apache Spark uses.

You can find templates of these configuration files and the default `spark-defaults.conf` and `spark-env.sh` files in the `$SPARK_HOME/conf` directory. Note that `spark-defaults.conf` and `log4j.properties` files are ASCII files. If you have set **_BPXK_AUTOCVT=ON** as specified in [“Setting up a user ID for use with z/OS Spark” on page 28](#), you can edit them without any explicit conversion.

The **spark-shell** and **spark-sql** interactive command line interfaces (`$SPARK_HOME/bin/spark-shell` and `$SPARK_HOME/bin/spark-sql`) have built-in support for the Apache Hive metastore service, which contains an embedded instance of the Apache Derby database. By default, these interfaces automatically create `metastore_db` and `spark-warehouse` directories and the `derby.log` file in the directory from which they are invoked. Therefore, you must either invoke **spark-shell** or **spark-sql** from a writable directory or set up your configuration files to point to writable directories. If you have multiple users running these interfaces, ensure that they use different writable directories so that one user does not attempt to use another user's database.

To set the location of the `metastore_db` directory, configure the **javax.jdo.option.ConnectionURL** property in the `hive-site.xml` file. You can find a sample `hive-site.xml` file in `$SPARK_HOME/conf`. For more information about Hive metastore configuration, see [Hive Metastore Administration \(https://cwiki.apache.org/confluence/display/Hive/AdminManual+MetastoreAdmin\)](https://cwiki.apache.org/confluence/display/Hive/AdminManual+MetastoreAdmin).

To set the location of the `spark-warehouse` directory, configure the **spark.sql.warehouse.dir** property in the `spark-defaults.conf` file, or use the `--conf spark.sql.warehouse.dir` command-line option to specify the default location of the database in warehouse.

To set the location of the `derby.log` file, configure the following property in the `spark-defaults.conf` file or as a command-line option to point to the desired Derby log file location:

```
spark.driver.extraJavaOptions -Dderby.stream.error.file=derby_log_file_location
```

If you do not need separate directories for the `metastore_db` directory and the `derby.log` file, you can configure the Derby system directory by specifying the following property in the `spark-defaults.conf` file:

```
spark.driver.extraJavaOptions -Dderby.system.home=derby_sys_dir
```

By default, both the `metastore_db` directory and the `derby.log` file will be created in this Derby system directory.

For more information about Apache Derby configuration, see <https://db.apache.org/derby/docs/10.14/tuning/>.

For more information about Apache Spark support for Apache Hive, see <http://spark.apache.org/docs/2.4.8/sql-programming-guide.html>.

Procedure

1. Copy the template or default configuration files into your new configuration directory.

For example:

```
cp $SPARK_HOME/conf/spark-env.sh.template $SPARK_CONF_DIR/spark-env.sh
cp $SPARK_HOME/conf/spark-defaults.conf.template $SPARK_CONF_DIR/spark-
defaults.conf
cp $SPARK_HOME/conf/log4j.properties.template $SPARK_CONF_DIR/log4j.properties
```

2. Update the configuration files as necessary as you complete the rest of the customization procedures for Open Data Analytics for z/OS.

If necessary, remember to complete step “2.b” on page 30 now. Also, set and export the SPARK_CONF_DIR environment variable as described in step “3” on page 33 of [“Creating the Apache Spark configuration directory” on page 33](#).

What to do next

Continue with [“Creating the Apache Spark working directories” on page 35](#).

Creating the Apache Spark working directories

Complete this task to create the Apache Spark working directories.

About this task

Table 3 on page 35 lists some of the working directories that Apache Spark uses. The sizes of these directories might need to be large depending on the type of work that is running; this is true particularly for the SPARK_LOCAL_DIRS directory.

Table 3. Apache Spark working directories

Directory contents	Default location	Environment variable	Suggested new directory
Log files	\$SPARK_HOME/logs	SPARK_LOG_DIR	Under /var, such as /var/spark/logs
Working data for the worker process	\$SPARK_HOME/work	SPARK_WORKER_DIR	Under /var, such as /var/spark/work
Shuffle and RDD data	/tmp	SPARK_LOCAL_DIRS	Under /tmp, such as /tmp/spark/scratch
PID files	/tmp	SPARK_PID_DIR	Under /tmp, such as /tmp/spark/pid

Procedure

1. As you did in [“Creating the Apache Spark configuration directory” on page 33](#), follow your file system conventions and create new working directories for Apache Spark.

Note: Consider mounting the \$SPARK_WORKER_DIR and \$SPARK_LOCAL_DIRS directories on separate zFS file systems to avoid uncontrolled growth on the primary zFS where Spark is located. The sizes of these zFS file systems depend on the activity level of your applications and whether auto-cleanup or rolling logs is enabled (see step “4” on page 36). If you are unsure about the sizes, 500 MB is a good starting point. Then, monitor the growth of these file systems and adjust their sizes

accordingly. Avoid using temporary file systems (TFS) for these directories if you expect significant growth, as TFS can use a large amount of real memory.

2. Give the following users read/write access to the newly created working directories:

- The user ID who runs z/OS Spark (SPARKID in these examples)
- The end user IDs who will be using z/OS Spark.

Assuming those users belong to the same UNIX user group, you may issue:

```
chmod ug+rxw /var/spark/logs
chmod ug+rxw /var/spark/work
chmod ug+rxw /tmp/spark/scratch
chmod ug+rxw /tmp/spark/pid
```

3. Update the \$SPARK_CONF_DIR/spark-env.sh script with the new environment variables pointing to the newly created working directories.

For example:

```
export SPARK_WORKER_DIR=/var/spark/work
```

4. Configure these directories to be cleaned regularly.

a) Configure Spark to perform cleanup.

By default, Spark does not regularly clean up worker directories, but you can configure it to do so. Change the following Spark properties in \$SPARK_CONF_DIR/spark-defaults.conf to values that support your planned activity, and monitor these settings over time:

spark.worker.cleanup.enabled

Enables periodic cleanup of worker and application directories. This is disabled by default. Set to true to enable it.

spark.worker.cleanup.interval

The frequency, in seconds, that the worker cleans up old application work directories. The default is 30 minutes. Modify the value as you deem appropriate.

spark.worker.cleanup.appDataTtl

Controls how long, in seconds, to retain application work directories. The default is 7 days, which is generally inadequate if Spark jobs are run frequently. Modify the value as you deem appropriate.

For more information about these properties, see <http://spark.apache.org/docs/2.4.8/spark-standalone.html>.

b) Configure Spark to enable rolling log files.

By default, Spark retains all of the executor log files. You can change the following Spark properties in \$SPARK_CONF_DIR/spark-defaults.conf to enable rolling of executor logs:

spark.executor.logs.rolling.maxRetainedFiles

Sets the number of latest rolling log files that are going to be retained by the system. Older log files will be deleted. The default is to retain all log files.

spark.executor.logs.rolling.strategy

Sets the strategy for rolling of executor logs. By default, it is disabled. The valid values are:

time

Time-based rolling. Use spark.executor.logs.rolling.time.interval to set the rolling time interval.

size

Size-based rolling. Use spark.executor.logs.rolling.maxSize to set the maximum file size for rolling.

spark.executor.logs.rolling.time.interval

Sets the time interval by which the executor logs will be rolled over. Valid values are:

- daily

- hourly
- minutely
- Any number of seconds

spark.executor.logs.rolling.maxSize

Sets the maximum file size, in bytes, by which the executor logs will be rolled over.

For more information about these properties, see <http://spark.apache.org/docs/2.4.8/configuration.html>.

c) Create jobs that clean up or archive the following directories listed in [Table 3 on page 35](#):

\$SPARK_LOG_DIR

\$SPARK_WORKER_DIR, if not configured to be cleaned by Spark properties

\$SPARK_LOCAL_DIRS

z/OS UNIX ships a sample script, **skulker**, that you can use as written or modify to suit your needs. The -R option can be useful, as Spark files are often nested in subdirectories. You can schedule **skulker** to run regularly from **cron** or other in-house automation tooling. You can find a sample **skulker** script in the /samples directory. For more information about **skulker**, see "skulker - Remove old files from a directory" in [z/OS UNIX System Services Command Reference](#).

- Optional: Periodically check all file systems involved in Spark (such as **\$SPARK_HOME** and any others mounted under it or elsewhere).
 - You can specify the FSFULL parameter for a file system so that it generates operator messages as the file system reaches a user-specified threshold.
 - Look for the number of extents, which can impact I/O performance for the disks involved. Perform these steps to reduce the number of extents:
 - Create and mount a new zFS.
 - Use **copytree**, **tar**, or similar utilities to copy the key directories from the old file system to the new one.
 - Unmount the old file system and re-mount the new file system in its place.

For more information, see "Managing File System Size" in [z/OS DFSMSdfp Advanced Services](#).

Note: Update the BPXPRMxx member of parmlib with the new file systems.

Results

You have completed the customization of your Apache Spark directory structure.

What to do next

Continue with ["Configuring networking for Apache Spark" on page 37](#).

Configuring networking for Apache Spark

Complete this task to configure the port access and other networking customization that Apache Spark requires.

About this task

Apache Spark makes heavy use of the network for communication between various processes, as shown in [Figure 2 on page 38](#).

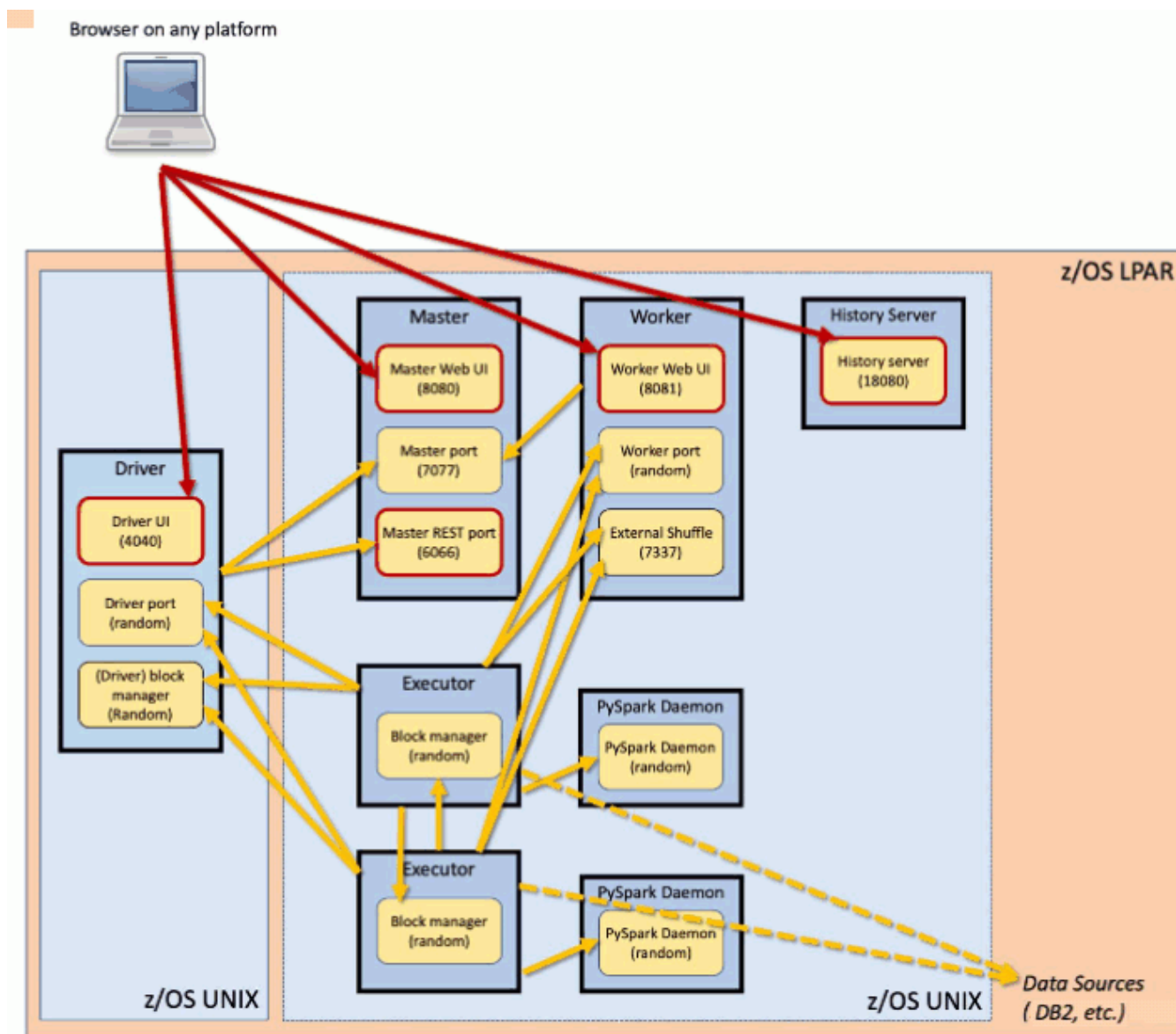


Figure 2. Network ports used in a typical Apache Spark environment

These ports are further described in [Table 4 on page 38](#) and [Table 5 on page 39](#), which list the ports that Spark uses, both on the cluster side and on the driver side.

Table 4. Network ports used by the Spark cluster

Port name	Default port number	Configuration property*	Notes
Master web UI	8080	spark.master.ui.port or SPARK_MASTER_WEBUI_PORT	The value set by the spark.master.ui.port property takes precedence.
Worker web UI	8081	spark.worker.ui.port or SPARK_WORKER_WEBUI_PORT	The value set by the spark.worker.ui.port takes precedence.
History server web UI	18080	spark.history.ui.port	Optional; only applies if you use the history server.

Table 4. Network ports used by the Spark cluster (continued)

Port name	Default port number	Configuration property*	Notes
Master port	7077	SPARK_MASTER_PORT	<p>SPARK_MASTER_PORT (or the default, 7077) is the starting point for connection attempts and not the actual port that might be connected. In addition, the value can be 0, which means it uses a random port number. Therefore, SPARK_MASTER_PORT (or the default, 7077) might not be the port that is used for the master. This statement is true for all methods of starting the master, including BPXBATCH, the start*.sh scripts, and the started task procedure.</p> <p>Note: You should not choose 0 for SPARK_MASTER_PORT if you intend to use client authentication.</p>
Master REST port	6066	spark.master.rest.port	Not needed if the REST service is disabled. If you wish to use the REST service and are planning to use authentication, you should configure AT-TLS port authentication for this port. Note that as of APAR PH03469, it is disabled by default.
Worker port	(random)	SPARK_WORKER_PORT	
Block manager port	(random)	spark.blockManager.port	
External shuffle server	7337	spark.shuffle.service.port	Optional; only applies if you use the external shuffle service.
PySpark daemon	(random)	spark.python.daemon.port	Optional; only applies if you use PySpark. APAR PI98042 (for Spark 2.2.0) is required to use this property.

Table 5. Network ports used by the Spark driver

Port name	Default port number	Configuration property*	Notes
Application web UI	4040	spark.ui.port	
Driver port	(random)	spark.driver.port	
Block manager port	(random)	spark.blockManager.port	The value set by the spark.driver.blockManager.port property takes precedence
Driver block manager port	(Value of spark.blockManager.port)	spark.driver.blockManager.port	If spark.driver.blockManager.port is not set, the spark.blockManager.port configuration is used.

*The Spark properties in the Configuration property column can either be set in the `spark-defaults.conf` file (if listed in lower case) or in the `spark-env.sh` file (if listed in upper case).

Spark must be able to bind to all the required ports. If Spark cannot bind to a specific port, it tries again with the next port number. (+1). The maximum number of retries is controlled by the `spark.port.maxRetries` property (default: 16) in the `spark-defaults.conf` file.

Note: The external shuffle server port does not support the port binding retries functionality.

The port binding retries functionality also implies a limit on the number of simultaneous instances of those ports across all Spark processes that use the same configuration. Assume the `spark.port.maxRetries` property is at default (16), here are a few examples:

- If the Spark application web UI is enabled, which it is by default, there can be no more than 17 Spark applications running at the same time, due to the 18th Spark driver process will fail to bind to an Application UI port.
- When both `spark.blockManager.port` and `spark.driver.blockManager.port` are set, there can be no more than 17 executor processes running at the same time, because the 18th executor process will fail to bind to a Block manager port.
- When `spark.blockManager.port` is set but `spark.driver.blockManager.port` is not set, the combined total of executor and driver processes cannot exceed 17, as the 18th process will fail to bind to a Block manager port.

Careful consideration is needed and you may need to increase the `spark.port.maxRetries` value if you are going to run multiple Spark applications at the same time, and/or planning to utilize a high number of executors within the cluster simultaneously.

Procedure

1. For your planned deployment and ecosystem, consider any port access and firewall implications for the ports listed in [Table 4 on page 38](#) and [Table 5 on page 39](#), and configure specific port settings, as needed.

For instance, if your application developers need to access the Spark application web UI from outside the firewall, the application web UI port must be open on the firewall.

Each time a Spark process is started, a number of listening ports are created that are specific to the intended function of that process. Depending on your site networking policies, limit access to all ports and permit access for specific users or applications.

On z/OS, you can use settings in z/OS Communications Server and RACF to enforce controls. For instance, you can specify `PORT UNRSV DENY` in your `TCPIP.PROFILE` to deny all applications access to unreserved ports for TCP or UDP. You can also specify `PORT UNRSV SAF` to grant specific access to specific users, such as the user ID that starts the Spark cluster and the Spark users. For more information about the **PORT** statement, see [z/OS Communications Server: IP Configuration Reference](#).

2. Consider your planned usage of the REST server.

The REST server interface, which listens on port 6066 by default, is currently disabled by default. As of APAR PH03469, the REST server supports TLS client authentication and Spark applications can be submitted through this interface.

3. Configure Spark environment variables for common enterprise networking configurations.

You can set each of the following environment variables in the `spark-env.sh` file:

SPARK_PUBLIC_DNS

For environments that use network address translation (NAT), set **SPARK_PUBLIC_DNS** to the external host name to be used for the Spark web UIs. **SPARK_PUBLIC_DNS** sets the public DNS name of the Spark master and workers. This allows the Spark Master to present in the logs a URL with the host name that is visible to the outside world.

SPARK_LOCAL_IP

Set the **SPARK_LOCAL_IP** environment variable to configure Spark processes to bind to a specific and consistent IP address when creating listening ports.

SPARK_MASTER_HOST

On systems with multiple network adaptors, Spark might attempt the default setting and give up if it does not work. Set the **SPARK_MASTER_HOST** (known as **SPARK_MASTER_IP** prior to Spark 2.0) to avoid this.

What to do next

Continue with [“Configuring z/OS Spark client authentication”](#) on page 41.

Configuring z/OS Spark client authentication

Complete the following tasks to enable authentication for connections to the Spark master port.

z/OS Spark client authentication is enabled by default. Spark does not function properly until you complete the setup for client authentication or disable the client authentication function.

The z/OS UNIX System Services APAR OA57666 is required for z/OS Spark client authentication to work properly.

If you want to defer the use of client authentication (for instance, for early testing in a secure test environment), you can disable this function by setting the following property in the `spark-defaults.conf` file.

```
spark.zos.master.authenticate      false
```

Note: If client authentication is disabled and you start your Spark cluster and driver under different user IDs, then some functions, such as the `spark-sql` command line interface, might not work properly. This is because the directories that are created by the Spark cluster might not be permissible to the Spark driver.

If client authentication is enabled, you can specify one of the following client authentication methods:

Application Transparent Transport Layer Security (AT-TLS)

This is the default Spark client authentication method that uses digital certificates along with AT-TLS. You need to set up digital certificates for the Spark cluster and its users, as well as an AT-TLS policy.

Trusted Partner

If all connections to the master port are internal, then you can consider using the Trusted Partner client authentication method, which doesn't require client certificates. However, this method continues to use AT-TLS for server authentication. A connection is internal if both endpoints belong in the same sysplex, the data flowing through the connection is never exposed outside of the sysplex, and the link or interface that is used is one of the following types:

- CTC
- HiperSockets interface (iQDIO)
- MPCPTP (including XCF and IUTSAMEH)
- OSA-Express QDIO with CHPID type OSX or OSM
- Loopback
- Both connection partners are owned by the same multihomed stack

Trusted Partner requires additional security configuration for the cluster and its users.

For more information about internal connections, see "Sysplex-specific connection routing information" in [z/OS Communications Server: IP Programmer's Guide and Reference](#).

You can specify the wanted authentication method (ATTLS or TrustedPartner) in the `spark-defaults.conf` file. For example:

```
spark.zos.master.authenticate.method  ATTLS
```

The APAR PI89136 is required to use Trusted Partner client authentication method.

Note: The workers must have the same `spark.zos.master.authenticate` and `spark.zos.master.authenticate.method` options as the master in order for the worker to register.

Note: APAR PH01619 adds a restriction of registering workers with the master. In order to register workers with masters, they must contain the same authentication configurations, `spark.zos.master.authenticate` and `spark.zos.master.authenticate.method`; otherwise, they will be rejected with one of the following messages (and it will appear in the worker log):

- Master's client authentication does not match Worker's
- Master's client authentication method does not match Worker's

About Application Transparent Transport Layer Security (AT-TLS)

AT-TLS is a z/OS Communications Server feature that transparently implements the TLS protocol in the TCP layer of the stack. As defined by the TLS protocol, AT-TLS uses digital certificates to authenticate the server and optionally the client, and encrypts the data that is flowing between the server and the client.

During client authentication, the Spark master acts as a server and accepts connections from the Spark worker and Spark users, which act as clients. Once the Spark master validates the client's digital certificate, a secure connection will be established and all subsequent data-flow between the server and the client will be encrypted.

For more information about AT-TLS, see "Application Transparent Transport Layer Security data protection" in *z/OS Communications Server: IP Configuration Guide*.

Using AT-TLS as the client authentication method

You can use AT-TLS with level 2 client authentication to secure communications between the Spark master and its clients. Specifically, you can create digital certificates for end users and use the certificates to authenticate those users when they connect to the Spark master port. Each of the certificates must map to a valid z/OS user ID, as required by level 2 client authentication. When a client attempts to connect to the Spark master port, the Spark master daemon queries AT-TLS to ensure that the following conditions exist:

- Communication between the client and the server is encrypted.
- A trusted relationship is established.
- A client certificate is matched to a local z/OS user ID.

Using Trusted Partner as the client authentication method

Using AT-TLS as the client authentication method requires a digital certificate for each user that is connecting to the Spark master port. If you know that all connections to the master port are internal, you can consider using the Trusted Partner client authentication method instead, which doesn't require client certificates. However, this method continues to use AT-TLS for server authentication.

Complete the following tasks to configure client authentication for Spark on z/OS.

Note: These tasks show examples using RACF commands and configurations. If you use a different security product, use the equivalent SAF facilities for that product.

- ___ 1. [“Creating and configuring digital certificates and key rings” on page 43](#)
- ___ 2. [“Configuring Policy Agent” on page 46](#)
- ___ 3. [“Defining security authorization for Policy Agent” on page 47](#)
- ___ 4. [“Creating the Policy Agent configuration files” on page 48](#)
- ___ 5. [“Configuring PROFILE.TCPIP for AT-TLS” on page 49](#)
- ___ 6. [“Defining the AT-TLS policy rules” on page 49](#)
- ___ 7. [“Starting and stopping Policy Agent” on page 53](#)
- ___ 8. [“Configuring additional authorities and permissions for the Spark cluster” on page 53](#)

Creating and configuring digital certificates and key rings

Complete this task to create and configure digital certificates and key rings that are needed for z/OS Spark client authentication.

About this task

The TLS protocol relies on digital certificates that are signed by a trusted certificate authority (CA) to authenticate the end points. The following configuration uses an internal CA. You might consider using an internal CA when only users within your company or organization need access to Spark. If you have already configured an internal CA for use with other products, you can reuse any existing end-user certificates for Spark.

This configuration uses a one-to-one certificate-to-user ID association. That is, one certificate maps to one user.

Digital certificates can be managed through RACF, PKI Services, or other security products.

Tip: If you are using AT-TLS as the client authentication method and plan to have a large number of Spark users (50 or more), consider using PKI Services, which provides additional management functionality for larger environments.

- For more information about digital certificates in RACF, see "Planning your certificate environment" and "Setting up your certificate environment" in [z/OS Security Server RACF Security Administrator's Guide](#).
- For more information about PKI Services, see "Introducing PKI Services" in [z/OS Cryptographic Services PKI Services Guide and Reference](#).

The following steps show examples using RACF commands, and subsequent configuration steps assume the CA definitions (such as labels) that appear in these examples.

Procedure

1. If you do not already have an internal CA defined, use RACF as the CA to create a CA certificate.

- AT-TLS as the client authentication method:

```
RACDCERT GENCERT CERTAUTH SUBJECTSDN(OU('SPARK Local CA') O('IBM') C('US'))  
WITHLABEL('SPARK Local CA') NOTAFTER(DATE(2030/01/01)) SIZE(1024)
```

- Trusted partner as the client authentication method:

```
RACDCERT GENCERT CERTAUTH SUBJECTSDN(OU('SPARK Local CA TP') O('IBM') C('US'))  
WITHLABEL('SPARK Local CA TP') NOTAFTER(DATE(2030/01/01)) SIZE(1024)
```

For more information about the **RACDCERT** command, see [z/OS Security Server RACF Command Language Reference](#).

2. Create a server certificate and key ring for the user ID that will be used to start the Spark cluster (SPARKID in these examples).

- **AT-TLS as the client authentication method:**

- a) Create a certificate for the Spark cluster that is signed by the CA.

```
RACDCERT GENCERT ID(SPARKID) SIGNWITH(CERTAUTH LABEL('SPARK Local CA'))  
KEYUSAGE(HANDSHAKE) WITHLABEL('Spark Server Cert') SUBJECTSDN(CN('SPARK TEST  
SERVER') O('IBM') L('Poughkeepsie') SP('New York') C('US'))  
NOTAFTER(DATE(2030/01/01))
```

- b) Create an SSL keyring (SparkRing in these examples).

```
RACDCERT ADDRING(SparkRing) ID(SPARKID)
```

- c) Connect the Spark server certificate to the SSL key ring.

```
RACDCERT ID(SPARKID) CONNECT(ID(SPARKID) LABEL('Spark Server Cert')
RING(SparkRing) USAGE(PERSONAL) DEFAULT)
```

- d) Connect the CA certificate to the SSL key ring.

```
RACDCERT ID(SPARKID) CONNECT(CERTAUTH LABEL('SPARK Local CA') RING(SparkRing))
```

- e) Allow the Spark cluster ID (SPARKID) to access its key ring.

```
PERMIT IRR.DIGTCERT.LISTRING CLASS(FACILITY) ID(SPARKID)
ACCESS(READ)
PERMIT IRR.DIGTCERT.LIST CLASS(FACILITY) ID(SPARKID)
ACCESS(READ)
```

- f) Refresh the RACF FACILITY class profiles.

```
SETROPTS RACLIST(FACILITY) REFRESH
```

You can issue the following command to verify your setup:

```
RACDCERT LISTRING(SparkRing) ID(SPARKID)
```

• **Trusted partner as the client authentication method:**

- a) Create a server certificate that is signed by the CA.

```
RACDCERT GENCERT ID(SPARKID) SIGNWITH(CERTAUTH LABEL('SPARK Local CA TP'))
KEYUSAGE(HANDSHAKE) WITHLABEL('Spark Server Cert TP') SUBJECTSDN(CN('SPARK TEST
SERVER') O('IBM') L('Poughkeepsie') SP('New York') C('US'))
NOTAFTER(DATE(2030/01/01))
```

- b) Create an SSL keyring (SparkRingTP in these examples).

```
RACDCERT ADDRING(SparkRingTP) ID(SPARKID)
```

- c) Connect the Spark server certificate to the SSL key ring.

```
RACDCERT ID(SPARKID) CONNECT(ID(SPARKID) LABEL('Spark Server Cert TP')
RING(SparkRingTP) USAGE(PERSONAL) DEFAULT)
```

- d) Connect the CA certificate to the SSL key ring.

```
RACDCERT ID(SPARKID) CONNECT(CERTAUTH LABEL('SPARK Local CA TP') RING(SparkRingTP))
```

- e) Allow the Spark cluster ID (SPARKID) to access its key ring.

```
PERMIT IRR.DIGTCERT.LISTRING CLASS(FACILITY) ID(SPARKID)
ACCESS(READ)
PERMIT IRR.DIGTCERT.LIST CLASS(FACILITY) ID(SPARKID)
ACCESS(READ)
```

- f) Refresh the RACF FACILITY class profiles.

```
SETROPTS RACLIST(FACILITY) REFRESH
```

You can issue the following command to verify your setup:

```
RACDCERT LISTRING(SparkRingTP) ID(SPARKID)
```

3. Configure the Spark end users (SPARKUSR in these examples).

• **AT-TLS as the client authentication method:**

Create and connect a client certificate and a key ring for each Spark end user.

- a) Create a client certificate that is signed by the CA.


```
RACDCERT GENCERT ID(SPARKUSR) SIGNWITH(CERTAUTH LABEL('SPARK Local
CA')) KEYUSAGE(HANDSHAKE) WITHLABEL('Spark Client Cert')
SUBJECTSDN(CN('SPARK TEST SERVER') O('IBM') L('Poughkeepsie') SP('New
York') C('US')) NOTAFTER(DATE(2030/01/01))
```

Note: For end users who will be using off-platform Jupyter Notebook environments to connect to Spark on this z/OS system, export their certificates and send them to the system administrator of the system from which those clients will connect (for instance, a distributed system administrator for JupyterHub or a z/OS system administrator using a different security database). The remote system administrator will need to set up these client certificates to be used when connecting to this z/OS system. Specifically, to export the client certificate (public and private key) and CA certificate (public key) into a PKCS#12 (.p12) certificate package, issue the following RACF command:

```
RACDCERT EXPORT(LABEL('Spark Client Cert')) ID(SPARKUSR)
DSN('SPARKADM.SPARKUSR.P12') FORMAT(PKCS12DER) PASSWORD('password')
```

In this example:

SPARKADM

The system programmer who is configuring certificates.

SPARKUSR

The end user.

password

The password used to access the contents of the package.

This creates a p12 package in the SPARKADM.SPARKUSR.P12 data set.

- b) Create an SSL keyring. The same key ring name (SparkRing) is used here for setup simplicity.

```
RACDCERT ADDRING(SparkRing) ID(SPARKUSR)
```

- c) Connect the client certificate to the end user's key ring.

```
RACDCERT ID(SPARKUSR) CONNECT(ID(SPARKUSR) LABEL('Spark Client Cert')
RING(SparkRing) USAGE(PERSONAL) DEFAULT)
```

- d) Connect the CA certificate to the end user's key ring.

```
RACDCERT ID(SPARKUSR) CONNECT(CERTAUTH LABEL('SPARK Local CA')
RING(SparkRing))
```

- e) Allow the end user's user ID to access its key ring.

```
PERMIT IRR.DIGTCERT.LISTRING CLASS(FACILITY) ID(SPARKUSR) ACCESS(READ)
PERMIT IRR.DIGTCERT.LIST CLASS(FACILITY) ID(SPARKUSR) ACCESS(READ)
```

- f) Refresh the RACF FACILITY class profiles

```
SETROPTS RACLIST(RDATALIB) REFRESH
```

- Trusted partner as the client authentication method:

With Trusted Partner as the client authentication method, you do not need to setup client certificate for each end user. Instead, you give the clients access to the CA's virtual key ring. The clients need to have access to the CA's certificate on the key ring to validate the server's certificate. You can accomplish this by administering a profile in either the FACILITY or the RDATALIB class. Using the FACILITY class provides global control of all rings, whereas using the RDATALIB class provides granular control of a specific ring. The RDATALIB class profile takes precedence over the FACILITY class profile if both are defined.

- When using the RDATALIB class:

```
RDEFINE RDATALIB CERTIFAUTH.IRR_VIRTUAL_KEYRING.LST UACC(NONE)
PERMIT CERTIFAUTH.IRR_VIRTUAL_KEYRING.LST CLASS(RDATALIB) ID(SPARKUSR)
```

```
ACCESS(READ)
SETROPTS RACLIST(RDATALIB) REFRESH
```

Note: If the RDATALIB class is not already active, activate and RACLIST it first:

```
SETROPTS CLASSACT(RDATALIB) RACLIST(RDATALIB)
SETROPTS RACLIST(RDATALIB) REFRESH
```

- When using the FACILITY class:

```
PERMIT IRR.DIGTCERT.LISTRING CLASS(FACILITY) ID(SPARKUSR) ACCESS(READ)
SETROPTS RACLIST(FACILITY) REFRESH
```

For more information about virtual key rings, see "RACF and key rings" in [z/OS Security Server RACF Security Administrator's Guide](#).

What to do next

Continue with ["Configuring Policy Agent" on page 46](#).

Configuring Policy Agent

Complete this task to configure Policy Agent to run as a z/OS started task. You can skip this task if a Policy Agent started task is already configured on your system.

About this task

Policy Agent reads, parses, and installs AT-TLS policies in the TCP/IP stack. The policies contain information that is necessary to negotiate secure connections.

Policy Agent runs as a z/OS UNIX process, so it can be started either from the z/OS UNIX shell or as a z/OS started task. This task uses a z/OS started task procedure to start Policy Agent.

For more information about Policy Agent and running Policy Agent as a started task, see "Policy Agent and policy applications" in [z/OS Communications Server: IP Configuration Reference](#).

Procedure

Create the following sample procedure to start Policy Agent as a z/OS started task.

```
//PAGENT PROC
//PAGENT EXEC PGM=PAGENT,REGION=0K,TIME=NOLIMIT,
// PARM=('ENVAR("_CEE_ENVFILE=DD:STDENV")/-1 SYSLOGD')
//*
/* For information on the above environment variables, refer to the
/* IP CONFIGURATION GUIDE. Other environment variables can also be
/* specified via STDENV.
/*
/* UNIX file containing environment variables:
//STDENV DD PATH='/etc/pagent.env',PATHOPTS=(ORDONLY)
/*
/* Output written to stdout and stderr goes to the data set or
/* file specified with SYSPRINT or SYSOUT, respectively. But
/* normally, PAGENT doesn't write output to stdout or stderr.
/* Instead, output is written to the log file, which is specified
/* by the PAGENT_LOG_FILE environment variable, and defaults to
/* /tmp/pagent.log. When the -d parameter is specified, however,
/* output is also written to stdout.
/*
//SYSPRINT DD SYSOUT=*
//SYSOUT DD SYSOUT=*
//CEEDUMP DD SYSOUT=*,DCB=(RECFM=FB,LRECL=132,BLKSIZE=132)
```

In this example, /etc/pagent.env points to the Policy Agent environment file.

- For more information about starting Policy Agent as a z/OS started task, see "Starting Policy Agent as a started task" in [z/OS Communications Server: IP Configuration Reference](#).

- For more information about starting Policy Agent from the z/OS UNIX shell, see "Starting Policy Agent from the z/OS Shell" in *z/OS Communications Server: IP Configuration Reference*.

Tip: For information about the overall configuration of Policy Agent, see "Steps for configuring the Policy Agent" in *z/OS Communications Server: IP Configuration Guide*.

What to do next

Before you can start Policy Agent, you must define the appropriate security authorizations.

Continue with [“Defining security authorization for Policy Agent” on page 47](#).

Defining security authorization for Policy Agent

Complete this task to define the appropriate security authorizations for Policy Agent.

About this task

The policies managed by Policy Agent can significantly affect system operation. Therefore, you must restrict the list of z/OS user IDs under which Policy Agent is allowed to run. To do this, you must define certain resources and controls in your system’s security management product, such as RACF.

Procedure

Complete the following steps to set up security definitions for Policy Agent in RACF.

1. Define the PAGENT user ID.

In this example, Policy Agent runs under the z/OS user ID named PAGENT and has a default group (DFLTGRP) of OMVSGRP and an OMVS segment with a UID of 0.

```
ADDUSER PAGENT DFLTGRP(OMVSGRP) OMVS(UID(0) HOME('/'))
```

2. Define the PAGENT started task to RACF.

In this example, Policy Agent runs as a z/OS started task named PAGENT. To define the Policy Agent started task to RACF, use the RDEFINE command to create the PAGENT.* profile in the STARTED class. (The **SETROPTS** commands are included for completeness. These commands have no effect when the STARTED class is already activated.)

```
SETROPTS CLASSACT(STARTED)
SETROPTS RACLIST(STARTED)
SETROPTS GENERIC(STARTED)
RDEFINE STARTED PAGENT.* STDATA(USER(PAGENT))
SETROPTS RACLIST(STARTED) REFRESH
SETROPTS GENERIC(STARTED) REFRESH
```

3. Grant Policy Agent the ability to make socket requests during TCP/IP stack initialization.

A TCP/IP stack initializes before Policy Agent installs policies into the stack. During the initialization window, only user IDs that are permitted to the EZB.INITSTACK.sysname.tcpname profile in the SERVAUTH class can make socket requests.

The following example shows the RACF commands to define a generic EZB.INITSTACK.** resource profile and grants READ access to the PAGENT user ID.

```
SETROPTS GENERIC(SERVAUTH)
SETROPTS CLASSACT(SERVAUTH) RACLIST(SERVAUTH)
RDEFINE SERVAUTH EZB.INITSTACK.** UACC(NONE)
PERMIT EZB.INITSTACK.** CLASS(SERVAUTH) ACCESS(READ) ID(PAGENT)
SETROPTS RACLIST(SERVAUTH) REFRESH
```

In addition to PAGENT, also grant READ access to the following applications:

- OMPROUTE
- SNMP agent and subagents

- NAMED
 - Other applications that do not require AT-TLS but that you want to start prior to general applications
4. Grant access to authorized users to manage the PAGENT started task.

To restrict management access to the PAGENT started task, define a MVS.SERV MGR.PAGENT profile in the OPERCMDS resource class and permit authorized users access to this profile, as in the following example:

```
SETROPTS CLASSACT(OPERCMDS)
SETROPTS RACLIST (OPERCMDS)
RDEFINE OPERCMDS (MVS.SERV MGR.PAGENT) UACC(NONE)
PERMIT MVS.SERV MGR.PAGENT CLASS(OPERCMDS) ACCESS(CONTROL) ID(PAGENT)
SETROPTS RACLIST(OPERCMDS) REFRESH
```

5. Consider restricting access to the **pasearch** command.

You can use the z/OS UNIX **pasearch** command to display policy definitions. The output from this command indicates whether policy rules are active and shows the policy definition attributes. However, you might not want every user to be able to see the policy definitions. To restrict access to the **pasearch** command, define an appropriate resource profile in the SERVAUTH resource class, as described in step 1 of "Steps for configuring the Policy Agent" in [z/OS Communications Server: IP Configuration Guide](#).

What to do next

Continue with ["Creating the Policy Agent configuration files"](#) on page 48.

Creating the Policy Agent configuration files

Complete this task to create Policy Agent configuration files.

About this task

You can use the IBM Configuration Assistant for z/OS Communications Server to update the Policy Agent configuration files, or you can update them manually. For more information, see "Options for configuring AT-TLS security" in [z/OS Communications Server: IP Configuration Guide](#).

Procedure

1. Create a file to contain the environment variable that points to the PAGENT configuration file.

The default configuration file is `/etc/pagent.env`, but you can specify a different location in the Policy Agent started task JCL that you created in ["Configuring Policy Agent"](#) on page 46.

In this example, the `pagent.env` file defines the following PAGENT environment variables:

PAGENT_CONFIG_FILE

Points to the PAGENT configuration file (`pagent.conf`)

PAGENT_LOG_FILE

Points to the PAGENT log file. In this example, Policy Agent logs messages to the syslog daemon and is the recommended practice. For information about setting up the syslog daemon, see "Configuring the syslog daemon" in [z/OS Communications Server: IP Configuration Guide](#).

LIBPATH

Policy Agent needs access to one or more DLLs at run time. Set the **LIBPATH** environment variable to include the `/usr/lib` directory, which normally includes all of the required DLLs.

TZ

Defines the local time zone.

```
PAGENT_CONFIG_FILE=/etc/pagent.conf
PAGENT_LOG_FILE=SYSLOGD
```

```
LIBPATH=/usr/lib
TZ=EST5EDT
```

2. Create the PAGENT configuration file (pagent.conf).

The following example shows the contents of the pagent.conf file:

```
# LOGLEVEL 511 turns on all trace levels for Policy Agent
LOGLEVEL 511
TcpImage TCPIP FLUSH PURGE
TTLSConfig /etc/pagent/TCPIP_TTLS.policy
```

The /etc/pagent/TCPIP_TTLS.policy file is the AT-TLS policy to be defined in [“Defining the AT-TLS policy rules”](#) on page 49.

What to do next

Continue with [“Configuring PROFILE.TCPIP for AT-TLS”](#) on page 49.

Configuring PROFILE.TCPIP for AT-TLS

Complete this task to configure TCP/IP to enable AT-TLS support.

About this task

You must enable AT-TLS support in the TCP/IP profile data set (PROFILE.TCPIP).

Procedure

Add the following statement to the PROFILE.TCPIP data set to enable AT-TLS support:

```
TCPCONFIG TTLS
```

For more information about the **TCPCONFIG TTLS** statement, see [z/OS Communications Server: IP Configuration Reference](#).

Wait until you finish [“Defining the AT-TLS policy rules”](#) on page 49 before recycling TCP/IP or refreshing its profile.

What to do next

Continue with [“Defining the AT-TLS policy rules”](#) on page 49.

Defining the AT-TLS policy rules

Complete this task to define the AT-TLS policy rules in the policy configuration file.

About this task

An AT-TLS policy configuration file contains the AT-TLS rules that identify specific types of TCP traffic, along with the type of TLS/SSL to be applied to those connections. If a rule match is found, AT-TLS transparently provides TLS protocol control for the connection based on the security attributes that are specified in the actions that are associated with the rule.

Procedure

1. Configure your AT-TLS policy rules.

The content of your AT-TLS policy depends on the Spark client authentication method that you choose.

Sample AT-TLS policies are provided in [Appendix B, “Sample configuration and AT-TLS policy rules for z/OS Spark client authentication,”](#) on page 163. For more information about AT-TLS policy file

syntax, structure, and the like, see "AT-TLS policy configuration" in *z/OS Communications Server: IP Configuration Guide*.

- AT-TLS as the client authentication method:
 - a. Create the PortGroup and the associated PortRange statements. By default Spark uses port 7077 as the master port, and supports port binding retries via the `spark.port.maxRetries` configuration setting (default: 16) mentioned in the previous section. Hence, we include the initial value and the entire retry range in the PortRange statement.

```
PortGroup                               SparkClusterGrp_ATTLS
{
  PortRangeRef                          SparkMaster_ATTLS
}
PortRange                               SparkMaster_ATTLS
{
  Port                                  7077-7093
}
```

You may optionally include the Master REST port (default: 6066, supports port binding retries) and the External shuffle server port (default: 7337) in the PortGroup by creating the corresponding PortRange statements and reference to them:

```
PortGroup                               SparkClusterGrp_ATTLS
{
  PortRangeRef                          SparkMaster_ATTLS
  PortRangeRef                          SparkMasterRest_ATTLS
  PortRangeRef                          SparkExtShuffleServer_ATTLS
}
PortRange                               SparkMaster_ATTLS
{
  Port                                  7077-7093
}
PortRange                               SparkMasterRest_ATTLS
{
  Port                                  6066-6082
}
PortRange                               SparkExtShuffleServer_ATTLS
{
  Port                                  7337
}
```

Note: If the Master REST port is protected by the AT-TLS policy rules then it will not be accessible from outside of the sysplex (e.g., from a web browser) until you export the client certificates into a, for example, PKCS#12 (.p12) certificate package. For instructions, see [“Creating and configuring digital certificates and key rings”](#) on page 43.

- b. Define the Inbound and Outbound rules for the port group, and the associated TTLSTLSGroupAction(GroupAct_TTLS_On) and TTLSEnvironmentAction(EnvAct_SparkServer_ATTLS and EnvAct_SparkClient_ATTLS) when the rules are triggered.

```
TTLSTLSRule                             SparkServer_ATTLS
{
  Direction                             Inbound
  LocalPortGroupRef                     SparkClusterGrp_ATTLS
  TTLSTLSGroupActionRef                 GroupAct_TTLS_On
  TTLSEnvironmentActionRef              EnvAct_SparkServer_ATTLS
}
TTLSTLSRule                             SparkClient_ATTLS
{
  Direction                             Outbound
  RemotePortGroupRef                   SparkClusterGrp_ATTLS
  TTLSTLSGroupActionRef                 GroupAct_TTLS_On
  TTLSEnvironmentActionRef              EnvAct_SparkClient_ATTLS
}
```

- c. Create the `TTLSTLSGroupAction` statement. This statement enables TLS security for the selected connections.

```
TTLSTLSGroupAction          GroupAct_TTLS_On
{
  TTLS-enabled               On
}
```

- d. Create the corresponding `TTLSEnvironmentAction` statements for the Inbound and Outbound `TTLSTLSRules`. The server side (Inbound) assumes the `ServerWithClientAuth` handshake role and the client side (Outbound) assumes the `Client` role. For simplicity, both sides use the same `TTLSTLSKeyRingParmsRef(KeyRing_ATTLS)` and `TTLSEnvironmentAdvancedParmsRef(EnvAdv_TLS)`.

```
TTLSEnvironmentAction      EnvAct_SparkServer_ATTLS
{
  HandshakeRole             ServerWithClientAuth
  EnvironmentUserInstance    0
  TTLSKeyRingParmsRef       KeyRing_ATTLS
  TTLSEnvironmentAdvancedParmsRef EnvAdv_TLS
}
TTLSEnvironmentAction      EnvAct_SparkClient_ATTLS
{
  HandshakeRole             Client
  EnvironmentUserInstance    0
  TTLSKeyRingParmsRef       KeyRing_ATTLS
  TTLSEnvironmentAdvancedParmsRef EnvAdv_TLS
}
```

- e. Create the `TTLSTLSKeyRingParm` and `TTLSEnvironmentAdvancedParms` statements. Use the same SSL keyring name (`SparkRing`) as during the server/client certificate creation steps in the previous section. `ClientAuthType` of `SAFCheck` is needed to enforce level 2 client authentication.

```
TTLSTLSKeyRingParms        KeyRing_ATTLS
{
  Keyring                   SparkRing
}
TTLSEnvironmentAdvancedParms EnvAdv_TLS
{
  ClientAuthType            SAFCheck
  TLSv1                     Off
  TLSv1.1                   Off
  TLSv1.2                   On
}
```

- Trusted Partner as the client authentication method:

- a. Create the `PortGroup` and the associated `PortRange` statements. By default Spark uses port 7077 as the master port, and supports port binding retries via the `spark.port.maxRetries` configuration setting (default: 16). Therefore, include the initial value and the entire retry range in the `PortRange` statement.

```
PortGroup                   SparkClusterGrp_TP
{
  PortRangeRef              SparkMaster_TP
}
PortRange                   SparkMaster_TP
{
  Port                       7077-7093
}
```

- b. Define the Inbound and Outbound rules for the port group, and the associated `TTLSTLSGroupAction(GroupAct_TTLS_On)` and `TTLSEnvironmentAction(EnvAct_SparkServer_TP and EnvAct_SparkClient_TP)` when the rules are triggered.

```
TTLSTLSRule                 SparkServer_TP
{
  Direction                 Inbound
  LocalPortGroupRef         SparkClusterGrp_TP
}
```

```

    TTLSGroupActionRef      GroupAct_TTLS_On
    TTLSEnvironmentActionRef EnvAct_SparkServer_TP
  }
  TTLSRule                  SparkClient_TP
  {
    Direction                Outbound
    RemotePortGroupRef        SparkClusterGrp_TP
    TTLSGroupActionRef        GroupAct_TTLS_On
    TTLSEnvironmentActionRef  EnvAct_SparkClient_TP
  }
}

```

- c. Create the TTLSGroupAction statement. This statement enables TLS security for the selected connections.

```

TTLGroupAction      GroupAct_TTLS_On
{
  TTLS-enabled       On
}

```

- d. Create the corresponding TTLSEnvironmentAction statements for the Inbound and Outbound TTLSRules. The server side (Inbound) assumes the Server handshake role, and uses the SSL keyring specified in the TTLSKeyRingParms statement (KeyRing_TP) that follows. The client side (Outbound) assumes the Client role, and uses the CA's virtual key ring to access the CA's certificate via the "Keyring *AUTH*/*" syntax. For simplicity, both sides use the same TTLSEnvironmentAdvancedParmsRef(EnvAdv_TLS_TP).

```

TTLSEnvironmentAction      EnvAct_SparkServer_TP
{
  HandshakeRole             Server
  EnvironmentUserInstance    0
  TTLSKeyRingParmsRef        KeyRing_TP
  TTLSEnvironmentAdvancedParmsRef EnvAdv_TLS_TP
}
TTLSEnvironmentAction      EnvAct_SparkClient_TP
{
  HandshakeRole             Client
  EnvironmentUserInstance    0
  TTLSKeyRingParms
  {
    Keyring                  *AUTH*/*
  }
  TTLSEnvironmentAdvancedParmsRef EnvAdv_TLS_TP
}

```

- e. Create the TTLSKeyRingParms and TTLSEnvironmentAdvancedParms statements. Use the same SSL keyring name (SparkRingTP) as during the server/client certificate creation steps in the previous section.

```

TTLSKeyRingParms      KeyRing_TP
{
  Keyring              SparkRingTP
}
TTLSEnvironmentAdvancedParms EnvAdv_TLS_TP
{
  TLSv1                Off
  TLSv1.1              Off
  TLSv1.2              On
}

```

2. Recycle TCP/IP to pick up the profile changes that you made earlier, or issue the following MVS system command to refresh its profile:

```
VARY TCPIP,,OBEYFILE,DSN=new_tcpip_profile
```

What to do next

Continue with [“Starting and stopping Policy Agent” on page 53](#).

Starting and stopping Policy Agent

This task assumes that Policy Agent is running as a z/OS started task.

Procedure

- Issue the MVS START command to start Policy Agent as a started task.
For example:

```
S PAGENT
```

- To perform a normal shutdown of Policy Agent, issue the MVS STOP command.
For example:

```
P PAGENT
```

What to do next

Continue with [“Configuring additional authorities and permissions for the Spark cluster”](#) on page 53.

Configuring additional authorities and permissions for the Spark cluster

About this task

Complete this task to configure additional authorities and permissions that are needed within the Spark cluster.

Note: If you change the authority of an active Spark process, you might need to restart the process for the change to take effect.

Procedure

1. Ensure that Spark client authentication is enabled.

Spark client authentication is enabled by default. However, you might have disabled it in a controlled test environment to simplify your initial setup and testing. Enable client authentication now (or verify that it is still enabled) by setting the following property in the `spark-defaults.conf` file:

```
spark.zos.master.authenticate      true
```

2. Specify the Spark client authentication method to use. There are 2 Spark client authentication methods - AT-TLS (ATTLS), which is the default, and Trusted Partner (TrustedPartner). Specify **spark.zos.master.authenticate.method** to be either ATTLS or TrustedPartner in the `spark-defaults.conf` file.

- Use AT-TLS as the client authentication method:

```
spark.zos.master.authenticate.method  ATTLS
```

- Use Trusted Partner as the client authentication method:

```
spark.zos.master.authenticate.method  TrustedPartner
```

3. Configure the Spark cluster user ID (SPARKID in earlier examples) to have authority to change the user IDs of the Spark executors to those of the authenticated end user IDs. To do this, configure the SURROGAT class profile for the surrogate user ID.

For example:

- a) Issue the following RACF commands to define a generic profile that allows SPARKID (a non-UID 0 user ID) to switch to any other z/OS user ID that has a defined z/OS UNIX segment:

```
SETROPTS CLASSACT(SURROGAT) RACLIST(SURROGAT) GENERIC(SURROGAT)
RDEFINE SURROGAT BPX.SRV.** UACC(NONE)
```

```
PERMIT BPX.SRV.** CLASS(SURROGAT) ACCESS(READ) ID(SPARKID)
SETOPTS GENERIC(SURROGAT) RACLIST(SURROGAT) REFRESH
```

b) Issue the following command to verify that the profile setup is successful:

```
RLIST SURROGAT BPX.SRV.** AUTHUSER
```

4. Configure the z/OS system that hosts the Spark cluster to honor ACLs that will be set by the Spark ID that is running the cluster.

For example, issue the following RACF command:

```
SETOPTS CLASSACT(FSSEC)
```

5. Change the permissions on the Spark working directory and Spark local directory so that both SPARKID and the end users can write to them. This customization assumes that the end user and the SPARKID belong to the same UNIX group.

Add the sticky bit so that users can only delete their own files. For example, issue the following commands from the z/OS UNIX shell:

```
chmod ug+rxw $SPARK_WORKER_DIR
chmod +t $SPARK_WORKER_DIR
chmod ug+rxw $SPARK_LOCAL_DIRS
chmod +t $SPARK_LOCAL_DIRS
```

Note: If you enabled event logging (as described in [“Enabling the Spark history service”](#) on page 142), perform this step for the event log directory as well.

You can issue the following command to verify the proper settings:

```
ls -ld $SPARK_WORKER_DIR $SPARK_LOCAL_DIRS
```

The output should be similar to the following example:

```
drwxrwxr-t  2 SPARKID  SYS1      8192 Aug  8 08:51 /var/spark/work
drwxrwxr-t  2 SPARKID  SYS1      8192 Aug  8 08:51 /tmp/spark/scratch
```

Note: **-t** in the output can also appear as **-T** if your userid has a umask that does not set the execute permission for other users.

6. Grant the Spark cluster user ID READ access to the BPX.SERVER FACILITY class profile, so the Spark master daemon can verify whether the client is authorized to connect to the master port.

```
RDEFINE FACILITY BPX.SERVER UACC(NONE)
PERMIT BPX.SERVER CLASS(FACILITY) ID(SPARKID) ACCESS(READ)
SETOPTS RACLIST(FACILITY) REFRESH
```

7. If you are using Trusted Partner as the Spark client authentication method, complete the following:
 - a) Configure the Spark cluster user ID (SPARKID) to have authority to obtain its connection partner's routing information and security credentials. To do this, grant the user ID READ access to the following SERVAUTH class profile:

```
EZB.IOCTL.sysname.tcpprocname.PARTNERINFO
```

where:

sysname

Specifies the system name that is defined in the sysplex.

tcpprocname

Specifies the TCP/IP procedure name.

Tip: You can specify a wildcard on segments of the profile name.

```
RDEFINE SERVAUTH EZB.IOCTL.*.*.PARTNERINFO UACC(NONE)
PERMIT EZB.IOCTL.*.*.PARTNERINFO CLASS(SERVAUTH) ID(SPARKID) ACCESS(READ)
SETOPTS RACLIST(SERVAUTH) REFRESH
```

- b) If the client (Spark driver and worker) is on a different TCP/IP stack than the master daemon, create a common security daemon name within your sysplex.

```
RDEFINE SERVAUTH EZBDOMAIN APPLDATA('security_domain_name')
SETROPTS RACLIST(SERVAUTH) REFRESH
```

where:

security_domain_name

Specifies the name of the security domain. The name is not case sensitive and is limited to 255 characters.

You can display the defined EZBDOMAIN by issuing:

```
RLIST SERVAUTH EZBDOMAIN
```

- c) Mark the file \$SPARK_HOME/lib/zos-native/libspark_zos_ioctl.so as controlled (trusted), if it is not already.

To verify whether the file has the controlled attribute on, issue the following command:

```
ls -E $SPARK_HOME/lib/zos-native
```

The output should be similar to the following:

```
-rwxr-xr-x  -ps- 1 SPARKID  SPKGRP  81920 Oct 13 13:43 libspark_zos_ioctl.so
```

If the file does not have the p extended attribute, then it is not marked as controlled. To mark the file as controlled, issue the following command from a user ID that has READ access to the BPX.FILEATTR.PROGCTL resource in the FACILITY profile:

```
extattr +p $SPARK_HOME/lib/zos-native/libspark_zos_ioctl.so
```

- d) Grant the Spark cluster user ID (SPARKID) and every end user ID (SPARKUSR) READ access to the AZK.SPARK.MASTER.CONNECT XFACILIT class profile.

```
RDEFINE XFACILIT AZK.SPARK.MASTER.CONNECT UACC(NONE)
PERMIT AZK.SPARK.MASTER.CONNECT CLASS(XFACILIT) ID(SPARKID) ACCESS(READ)
PERMIT AZK.SPARK.MASTER.CONNECT CLASS(XFACILIT) ID(SPARKUSR) ACCESS(READ)
SETROPTS RACLIST(XFACILIT) REFRESH
```

- e) If you are using started tasks to start the master and worker, after you installed JZOS, you must define a PROGRAM profile for the JZOS load module (JVMLDM86) and permit SPARKID read access to the profile.

For example, issue the following RACF commands:

```
RDEFINE PROGRAM JVMLDM86 ADDMEM('datasetname'/volser/NOPADCHK)
SETROPTS WHEN(PROGRAM) REFRESH
PERMIT JVMLDM86 CLASS(PROGRAM) ID(SPARKID) ACCESS(READ)
SETROPTS WHEN(PROGRAM) REFRESH
```

For more information about started tasks, see [“Setting up started tasks to start and stop Spark processes”](#) on page 60.

What to do next

- If you previously completed the overall configuration of z/OS Spark, continue with [“Starting the Spark cluster”](#) on page 57.
- If you have not yet completed your initial configuration of z/OS Spark, continue with [“Configuring IBM Java”](#) on page 57.

Restricting the ability to start or stop the Spark cluster

Complete this task to restrict the ability to start or stop the Spark cluster components (such as, Master and Worker) via the shell scripts located in the \$SPARK_HOME/sbin directory.

Note: For restricting the ability to start or stop the Spark cluster components via started tasks, see [“Define the RACF started profile for started tasks” on page 63](#).

About this task

Apache Spark provides Bash shell scripts for starting the individual components, such as the Master, the Worker, or the History server. However, when carefully allocating resources for the Spark cluster via WLM, having extra cluster components can disrupt the expected port, storage and zIIP allocations.

Setting the Spark “sbin” directory contents to be unreadable and unexecutable

By default, the contents of the \$SPARK_HOME/sbin directory have permissions of “755”, or “rwxr-xr-x”. To restrict users from starting their own Spark cluster, the permissions of the directory and its contents can be changed to “700” or “rwx-----”. This would prevent users from using the scripts within the directory to start Spark cluster components, seeing the contents of the Spark “\$SPARK_HOME/sbin” directory, and copying the scripts to their own directories (where they could set their own permissions and bypass the execution restrictions). Alternately, you could use permission value “750” (or “rwxr-x---”) in the commands that follow to allow userids within the Spark group to start or stop the Spark cluster components (and potentially, start their own cluster, as they might do if they were testing a new spark-defaults.conf setting, for example).

Following are commands to alter permission of sbin:

Assume the zFS file system is SYS1.SPARK.ZFS for Spark and the mountpoint is /usr/lpp/IBM/izoda/spark/, from TSO OMVS or a Putty session, logged in as the owner of the SPARK_HOME directory:

```
tsocmd "mount filesystem('SYS1.SPARK.ZFS') type(zfs) mode(rdwr)
mountpoint('/usr/lpp/IBM/izoda/spark') "

cd $SPARK_HOME
chmod -R 700 ./sbin

tsocmd "mount filesystem('SYS1.SPARK.ZFS') type(zfs) mode(read)
mountpoint('/usr/lpp/IBM/izoda/spark') "
```

Similarly, to back out this change, return to the same OMVS or Putty shell and issue these commands:

```
tsocmd "mount filesystem('SYS1.SPARK.ZFS') type(zfs) mode(rdwr)
mountpoint('/usr/lpp/IBM/izoda/spark') "

cd $SPARK_HOME
chmod -R 755 ./sbin

tsocmd "mount filesystem('SYS1.SPARK.ZFS') type(zfs) mode(read)
mountpoint('/usr/lpp/IBM/izoda/spark') "
```

Finally, if you want to allow just one of the scripts (such as the Spark configuration checker) to be open to execution by anyone, use these commands:

```
tsocmd "mount filesystem('SYS1.SPARK.ZFS') type(zfs) mode(rdwr)
mountpoint('/usr/lpp/IBM/izoda/spark') "

cd $SPARK_HOME
chmod -R 700 ./sbin chmod 711 ./sbin/
chmod 755 ./sbin/spark-configuration-checker.sh

tsocmd "mount filesystem('SYS1.SPARK.ZFS') type(zfs) mode(read)
mountpoint('/usr/lpp/IBM/izoda/spark') "
```

Some of these commands use the -R command (recursive) flag; others do not. You can reuse the "755" form for any other shell scripts in that directory for which you want to allow execution. Note that some

scripts invoke other scripts in the `./sbin` directory, and will need to have their permissions changed as well.

Note: The SMP/E install process will reinstall the scripts when PTFs are applied. Because of this, you will need to re-do the `chmod` commands each time after applying service. You may want to consider creating a post-install BPXBATCH job to execute the commands shown previously in this section.

Note: It is still possible for “motivated” users to acquire their own copy of the scripts via download from the open source world and use them to start the cluster component. The changes shown in this section prevent only inadvertent or unplanned usage by users.

Using standard z/OS MVS command authorization

Most installations will have tight controls over which operators are allowed to use `START`, `STOP` and `CANCEL` commands. If you have used the supplied `SAZKSAMP` examples to create Started Task JCL for the Spark Master and Worker, you may want to update the RACF controls over the `START` and `STOP` of those started tasks to further secure the Spark cluster environment. Refer to “MVS Commands, RACF Access Authorities, and Resource Names” in *z/OS MVS System Commands*. For example, you might need to define new resources like `MVS.START.STC.AZKMSTR` and `MVS.STOP.STC.AZKMSTR` to control the Master. (Similar names would exist for the Worker; for example, `MVS.START.STC.AZKWRKR`.)

Starting the Spark cluster

About this task

If you have completed all of the tasks to enable client authentication for Apache Spark **and** you have previously completed all of the tasks in your overall configuration of z/OS Spark, you can start your Spark cluster as usual.

However, if you have just configured client authentication as part of your initial overall configuration of z/OS Spark, skip this procedure and do not start the Spark cluster until you have completed all of the remaining configuration tasks for z/OS Spark, as directed in “What to do next.”

Procedure

After completing all of the tasks to enable client authentication and configure z/OS Spark, start the Spark cluster and run your Spark applications as usual. If a worker or driver is unable to be authenticated, it fails to connect to the master port.

With z/OS Spark client authentication enabled, an application that is submitted to the master port has its executors started under the user ID of that application. An application that is submitted to the REST port, which is the port for cluster deploy mode, is considered part of the Spark cluster and therefore has both the driver and executors run under the user ID of the Spark cluster.

What to do next

If you have not yet completed your initial configuration of z/OS Spark, continue your configuration activities with “[Configuring IBM Java](#)” on page 57.

Configuring IBM Java

Spark runs as several Java virtual machine (JVM) processes. Complete this task to ensure that IBM Java is properly configured.

Procedure

1. See [Hints and Tips for Java on z/OS \(www.ibm.com/systems/z/os/zos/tools/java/faq/javafaq.html\)](http://www.ibm.com/systems/z/os/zos/tools/java/faq/javafaq.html) to ensure that your Java configuration has the appropriate settings.
Also see “[IBM Java configuration options](#)” on page 173 for additional settings.

2. Set `_CEE_DMPTARG` to store Java dumps on a separate mount point outside of `$SPARK_HOME`.

For more information, see [JVM](#)

environment settings (https://www.ibm.com/support/knowledgecenter/en/SSYKE2_8.0.0/com.ibm.java.zos.80.doc/diag/appendixes/env_var/env_jvm.html).

For information about the order in which Java applies dump settings, see [Dump](#)

agent environment variables (https://www.ibm.com/support/knowledgecenter/en/SSYKE2_8.0.0/com.ibm.java.zos.80.doc/diag/tools/dumpagents_env.html).

3. Configure large page memory allocation for Java.

For information about configuration and best practices for setting maximum java heap sizes,

see [Configuring large page memory allocation](#) (https://www.ibm.com/support/knowledgecenter/en/SSYKE2_8.0.0/com.ibm.java.zos.80.doc/user/alloc_large_page.html).

4. If you have Java applications that use compression, ensure that zEDC is properly configured.

For more information, see [zEnterprise Data Compression](#) (https://www.ibm.com/support/knowledgecenter/en/SSYKE2_8.0.0/com.ibm.java.zos.80.doc/user/zedc_compression.html).

What to do next

To use jobs to start and stop Spark processes, continue with [“Creating jobs to start and stop Spark processes”](#) on page 58.

To use started tasks to start and stop Spark processes, continue with [“Setting up started tasks to start and stop Spark processes”](#) on page 60.

Creating jobs to start and stop Spark processes

You can use BPXBATCH to start and stop Spark processes, such as the master and worker.

Note: We recommend that you use started tasks rather than BPXBATCH (see [“Setting up started tasks to start and stop Spark processes”](#) on page 60). However, if you prefer, you can use BPXBATCH, as described in this section.

The examples of jobs are based on the following assumptions:

- The user ID that starts and stops the Spark cluster is SPARKID.
- The default shell program for SPARKID is bash.
- Spark is installed in `/usr/lpp/IBM/zspark/spark/sparknnn`, where `nnn` is the Spark version (for instance, `/usr/lpp/IBM/zspark/spark/spark32x` for Spark 3.2.0).

Important: Be sure that all of the required environment variables, such as `JAVA_HOME`, are set in the environment that is started by BPXBATCH. You can accomplish this in one of the following ways:

- Export the environment variables in one of the bash startup files.

Invoke the bash command with the `-l` option in BPXBATCH. The `-l` option on the bash command instructs bash to run a login shell, in which bash first reads, and runs commands from the file `/etc/profile`, if the file exists. After reading that file, bash looks for `~/ .bash_profile`, `~/ .bash_login`, and `~/ .profile` in that order, reads and runs commands from the first one that exists and is readable. You can export your environment variables in `~/ .bash_profile`, for example, so they are added to the environment.

- Use the `STDENV DD` statement to pass environment variables to BPXBATCH. You can specify a file that defines the environment variables or specify them directly in the JCL. For more information about using the `STDENV` statement, see [“Passing environment variables to BPXBATCH”](#) in [z/OS UNIX System Services User's Guide](#).

Sample job to start the master and worker

[Figure 3](#) on page 59 shows an example of a BPXBATCH job to start the master and worker.

The **-l** and **-c** options on the **bash** command instruct bash to run a login shell with the entire shell command sequence given between the single quotation marks ('). The semicolons (;) in the command sequence separate different shell commands.

```
//SPARKMST JOB 'SPARK START',CLASS=K,MSGCLASS=A,  
// NOTIFY=&SYSUID,SYSTEM=HOST,USER=SPARKID  
//PRTDS EXEC PGM=BPXBATCH,REGION=0M  
//STDPARM DD *  
SH /bin/bash -l -c 'cd /usr/lpp/IBM/izoda/spark/spark23x/sbin;start-master.sh;  
sleep 5;start-slave.sh spark://hostname.yourcompany.com:7077'  
//SYSOUT DD SYSOUT=*  
//STDIN DD DUMMY  
//STDOUT DD SYSOUT=*  
//STDERR DD SYSOUT=*  
//
```

Figure 3. Sample job to start the master and worker

The **bash** command in the sample start job issues the following sequence of commands:

1. Change directories to the Spark installation directory where the administration commands are located.

```
cd /usr/lpp/IBM/izoda/spark/spark23x/sbin
```

2. Start the master.

```
start-master.sh
```

3. Sleep for 5 seconds to allow the master time to start.

```
sleep 5
```

4. Start the worker.

```
start-slave.sh spark://hostname.yourcompany.com:7077
```

where *hostname.yourcompany.com* is the name of the host where the master is listening on port 7077. You can also issue these commands directly from the z/OS UNIX shell as a quick test of your Spark configuration.

Sample job to stop the master and worker

Figure 4 on page 59 shows an example of a BPXBATCH job to stop the master and worker. Its logic is similar to the start job, except that it first stops the worker and then stops the master.

```
//SPARKSTP JOB 'SPARK STOP',CLASS=K,MSGCLASS=A,  
// NOTIFY=&SYSUID,SYSTEM=HOST,USER=SPARKID  
//PRTDS EXEC PGM=BPXBATCH,REGION=0M  
//STDPARM DD * SH /bin/bash -l -c 'cd /usr/lpp/IBM/izoda/spark/spark23x/sbin;  
stop-slave.sh;sleep 5;stop-master.sh'  
//SYSOUT DD SYSOUT=*  
//STDIN DD DUMMY  
//STDOUT DD SYSOUT=*  
//STDERR DD SYSOUT=*  
//
```

Figure 4. Sample job to stop the master and worker

What to do next

Test your start and stop jobs to ensure that your setup is correct. Then, continue with [“Configuring memory and CPU options”](#) on page 65.

Setting up started tasks to start and stop Spark processes

Complete this task to set up started tasks to start and stop Spark processes.

About this task

The following list provides the benefits of using this feature.

- Allows the Spark master, worker, history server, and shuffle service to run on z/OS consistent with running other MVS batch jobs, job steps, or started tasks.
 - Handles START, STOP, and CANCEL command options and writes messages to the MVS console. For more information about the messages that are issued, see [Open Data Analytics for z/OS System Messages](#).
 - Can be extended to take advantage of all of the capabilities that JZOS provides. These capabilities include the use of SYSOUT for output directories, MVS data sets, and DD statements.
- Maintains flexible configuration of the Java execution environment and the z/OS UNIX System Services environment that the Spark master and worker require.
- Automation
 - Allows the Spark master, worker, history server, and shuffle service to be managed through customer automation products and policies.
 - Allows automation products to start and stop the master and worker with no parameters, with the assurance that the worker is started using the master port for which the master is actually started.
 - Allows the worker to retry starting for a period of time if the master is not yet started.
 - Allows enterprise automation management strategies to be applied to the Spark master and worker. These strategies include the following:
 - Started task dependencies such as staging the starting and stopping of Spark started tasks based on the availability of other started tasks. These tasks can include but are not limited to OMVS, MDS, TCP/IP, and Database Servers (Db2, IMS, and more).
 - Failure recovery by restarting the Spark master and worker on any system under automation management control.

The examples of started tasks are based on the following assumptions:

- JZOS Batch Launcher and Toolkit in IBM 64-Bit SDK for z/OS Java Technology Edition V8 is installed and operational. For installation and configuration instructions and information about messages and return codes from JZOS, see *JZOS Batch Launcher and Toolkit: Installation and User's Guide*.
- If you are using Trusted Partner authentication, ensure that a PROGRAM profile is defined for the load module, JVMLDM86. For instructions, see Step 11 in [“Configuring additional authorities and permissions for the Spark cluster”](#) on page 53.
- For each Spark cluster, the `spark-env.sh` contains a unique `SPARK_IDENT_STRING`. Do not specify `$USER` or allow it to default to `$USER`.
- The user ID that starts and stops the Spark cluster is the SPARKID that is previously created the SPKGRP is the group that is previously created.
- The default shell program for SPARKID is `bash`.
- Spark is installed in `/usr/lpp/IBM/zspark/spark/sparknnn`, where *nnn* is the Spark version. For example, `/usr/lpp/IBM/zspark/spark/spark32x` for Spark 3.2.0.
- Spark is configured as described in this document and the required environment variables are set in the following procedures. For more information, see [“Set and export common environment variables”](#) on page 62.
- OMVS must be initialized before the master can start.
- The directories that are specified by the following environment variables, or the defaults taken when not specified, must exist and have the appropriate authorities. For more information, see [“Creating the Apache Spark working directories”](#) on page 35.

- **SPARK_HOME**
- **SPARK_CONF_DIR**
- **SPARK_LOG_DIR**
- **SPARK_PID_DIR**
- **SPARK_LOCAL_DIRS**
- **SPARK_WORKER_DIR**

Procedures for each Spark cluster

To create a procedure (master, worker, history server, shuffle service) for a Spark cluster, copy and edit the applicable sample procedure that is included in IBM Open Data Analytics for z/OS. The procedure needs to be put in a data set in your PROCLIB concatenation, such as SYS1.PROCLIB.

There are four sample procedures. The high-level qualifier (*hlq*) depends on your installation. In this document, the default high-level qualifier is AZK.

- *hlq*.SAZKSAMP(AZKMSTR) - Master
- *hlq*.SAZKSAMP(AZKWRKR) - Worker
- *hlq*.SAZKSAMP(AZKHIST) - History Server
- *hlq*.SAZKSAMP(AZKSHUF) - Shuffle service

Follow the instructions in the sample procedure. For example, **SPARK_CONF_DIR** must be set and exported in the procedure. It will not default to \$SPARK_HOME/conf.

Note that there are instances where the procedure for the Shuffle service cannot be used. The Shuffle service can be started in two ways:

- By invoking **sbin/start-shuffle-service.sh** or the new started task
- Internally, by the worker when **spark.shuffle.service.enabled=true** configuration is set. By default, the Spark Shuffle service starts and runs under the Spark Worker when **spark.shuffle.service.enabled=true**.

Do not use the started task (or shell script) during the following:

1. When starting the Shuffle service inside the worker process with the **spark.shuffle.service.enabled=true** configuration (this is a prerequisite for dynamic allocation).
2. When enabling dynamic allocation with the **spark.dynamicAllocation.enabled=true** config.

If using one or both of these features, the worker or Shuffle service (started via the first service) will fail with a port binding error. The second service to be started will fail as both services bind the Shuffle service port (**spark.shuffle.service.port**, default 7337).

Note: **spark.shuffle.service.enabled** has no effect on the Shuffle service when it is started via the shell script or started task. It is a property used by the worker only.

When starting the Spark started tasks manually, you should ensure that the Master (and optional History and Shuffle servers) have initialized before starting the Worker. When starting via automation, they can be started in parallel, as the Worker will wait a limited amount of time for the Master to initialize. Spark users may encounter errors if Spark jobs are submitted before all started tasks have initialized.

Define the routing of the log output

When using started tasks to start and stop Spark processes, the routing of the log output is dependent upon a combination of settings in the \$SPARK_CONF_DIR/log4j.properties file and the values for the STDERR or STDOUT DD cards in the master and worker procedures.

For example, using the default values in the \$SPARK_CONF_DIR/log4j.properties file, as shown in the following sample, will route the log output to the value in the STDERR DD card:

```
log4j.rootCategory=INFO, console
log4j.appender.console=org.apache.log4j.ConsoleAppender
log4j.appender.console.target=System.err
```

Using the default values for both the \$SPARK_CONF_DIR/log4j.properties file and the STDERR DD card of //STDERR DD SYSOUT=&SOUT directs both the JZOS output and the Spark output to SYSOUT.

Changing the log4j.appender.console.target value to System.out will route the log output to the value in the STDOUT DD card.

Alternatively, to route the Spark log output to a USS file in the \$SPARK_LOG_DIR directory, if you are using the default values for the \$SPARK_CONF_DIR/log4j.properties file, you must specify the full path for the log file by using the value that is set for \$SPARK_LOG_DIR on the STDERR DD card. The following examples show the specifications for the master and the worker log files for a cluster that is denoted as Cluster1 where the value of \$SPARK_LOG_DIR is /var/spark/logs. You must specify a unique file name for the master and a unique file name for the worker. If you are starting more than one cluster, you must specify unique file names for each master and worker across all clusters.

The following example is for the master procedure.

```
//STDERR DD PATH='/var/spark/logs/azkmstrCluster1.err',
//      PATHOPTS=(OWRONLY,OCREAT,OTRUNC),PATHMODE=SIRWXU
```

The following example is for the worker procedure.

```
//STDERR DD PATH='/var/spark/logs/azkwrkrCluster1.err',
//      PATHOPTS=(OWRONLY,OCREAT,OTRUNC),PATHMODE=SIRWXU
```

By default, the master and worker log files are created with PATHMODE=SIRWXU, which is equivalent to z/OS UNIX file access attributes of “700” (“rwx-----”). As such, the user ID that starts/stops the Spark cluster (SPARKID in these examples) has full access to the log files, and no access for anyone else. You may change this behavior by modifying the PATHMODE parameter value. For more information about the PATHMODE parameter, see “PATHMODE parameter” in [z/OS MVS JCL Reference](#).

When using started tasks to start and stop Spark processes, the log files are not rotated as when using other methods for starting the master and worker. It is recommended that you not specify OAPPEND for the PATHOPTS to avoid file growth.

Set and export common environment variables

Complete this task to set and export common environment variables.

About this task

Environment variables are needed by both the AZKMSTR and AZKWRKR procedures. In addition, the environment variables that are needed by the JZOS launcher must be exported. However, the /etc/profile and .profile scripts are not run when the JZOS launcher is run.

Therefore, it is easiest to create a common script to set and export the environment variables. IBM Open Data Analytics for z/OS or IzODA provides a script template.

The template is installed in \$SPARK_HOME/conf and is called spark-zos-started-tasks.sh.template. It needs to be copied and modified, and then put into the \$SPARK_CONF_DIR with your other scripts.

The template is modeled after the sample procedure (PROC) in *JZOS Batch Launcher and Toolkit: Installation and User's Guide*. The environment variables are loaded from this script, rather than set directly in the PROC as shown in the sample. In addition, this script loads the spark-env.sh script and sets up other environment variables as needed for the started tasks.

Procedure

1. Copy the template into your configuration directory. For example:

```
cp $SPARK_HOME/conf/spark-zos-started-tasks.sh.template
$SPARK_CONF_DIR/spark-zos-started-tasks.sh
```

2. Update \$SPARK_CONF_DIR/spark-zos-started-tasks.sh as necessary.

Define the RACF started profile for started tasks

Complete this task to define the RACF started profile for started tasks.

About this task

From TSO, or from JCL that runs IKJEFT01, you must define the RACF profiles to the started tasks. Use the names of the procedures for the started tasks for each cluster that you defined for your installation in place of AZKMSTR, AZKWRKR, AZKHIST, and AZKSHUF in the following example procedure.

Procedure

1. Enter the following command to define the RACF started profile to the master started task:

```
RDEFINE STARTED AZKMSTR.* STDATA(USER(SPARKID) GROUP(SPKGGRP))
```

2. Enter the following command to define the RACF started profile to the worker started task:

```
RDEFINE STARTED AZKWRKR.* STDATA(USER(SPARKID) GROUP(SPKGGRP))
```

3. Enter the following command to define the RACF started profile to the history service:

```
RDEFINE STARTED AZKHIST.* STDATA(USER(SPARKID) GROUP(SPKGGRP))
```

4. Enter the following command to define the RACF started profile to the shuffle service:

```
RDEFINE STARTED AZKSHUF.* STDATA(USER(SPARKID) GROUP(SPKGGRP))
```

5. Enter the following command to set the RACF options previously specified:

```
SETOPTS RACLIST(STARTED) REFRESH
```

Results

After successfully completing the previous tasks, you should have a defined RACF started profile to the master and worker started tasks, the history service, and the shuffle service.

Note: If you attempt to start AZKMSTR before you complete the steps in this task, you might be prompted with the following messages:

```
SY1  s azkmstr
SY1  IRR813I NO PROFILE WAS FOUND IN THE STARTED CLASS FOR
      AZKMSTR WITH JOBNAME AZKMSTR. RACF WILL USE ICHRIN03.
SY1  $HASP100 AZKMSTR  ON STCINRDR
SY1  $HASP373 AZKMSTR  STARTED
SY1  ICH408I JOB(AZKMSTR ) STEP(STARTING) CL(PROCESS )
      OMVS SEGMENT NOT DEFINED
SY1  $HASP395 AZKMSTR  ENDED
```

Note:

The master task and the worker task must be started with the same user unless the user used to start the worker task has UID(0).

Note:

If the master was started as a started task, the worker must be started as a started task.

WLM configuration

Configure the AZKMSTR and AZKWRKR started tasks to use the same service class that you previously specified in the Spark master and worker daemons. For more information about WLM service classes for Spark, see [“Defining WLM service classes for Spark” on page 78](#).

You must define both the master and worker started task procedures to a WLM service class for every cluster for which started task procedures were created.

Stopping the started tasks

The started tasks are stopped by using the following stop commands with no parameters.

```
stop azkwrkr
stop azkmstr
stop azkhist
stop azkshuf
```

If a worker is started for a master task, the worker task is stopped when the master task is stopped. The system looks at the file that is created when the worker is started for this instance, which contains the process ID for the worker process. It does so by using the same naming convention that the shell scripts use (identified by SPARK_IDENT_STRING), and ensures that the worker is either a java process or JVM86 (JZOS).

When the worker is stopped due to the master being stopped, the worker process is ended immediately. This action results in a return code of 0143 in the HASP395 message on the console when the worker stops. This return code is normal and appears as seen in the following example.

```
stop azkmstr
$HASP395 AZKWRKR  ENDED - RC=0143
$HASP395 AZKMSTR  ENDED - RC=0000
```

Canceling the started tasks

The started tasks can be canceled by using the following cancel command:

```
cancel azkwrkr
cancel azkmstr
cancel azkhist
cancel azkshuf
```

If a worker is started for a master task and the master task is canceled, the worker task is not ended and is left in a state with no master associated to it. The worker task is reattached upon the restart of the master or can be manually ended by stopping or canceling it.

Automating the starting of tasks

Complete this task if you want to automate the starting of tasks.

About this task

The following one-time steps are optional, but allow the following benefits:

- Allows automation products to start and stop the master and worker with no parameters, with the assurance that the worker is started using the master port for which the master is actually started.
- Allows the worker to retry starting for a period of time if the master is not yet started.

Procedure

1. If you have modified the \$SPARK_CONF_DIR/log4j.properties file, complete the following tasks. If you do not have a \$SPARK_CONF_DIR/log4j.properties file, no action is required.

- a) Ensure that **log4j.rootCategory** is set to INFO, DEBUG, or ALL, and that the appender is console such as, `log4j.rootCategory=INFO, console`.
- b) The **log4j.appender.console.target** parameter can be specified as either `System.err` or `System.out` if the procedure previously created for starting the master uses the correct corresponding card (STDERR or STDOUT) as described in the next step.

For more information, see [“Define the routing of the log output” on page 61](#).

Note: `log4j.appender.console.target=System.err` is the default and the following examples, which refer to the sample procedure *hlq.SAZKSAMP(AZKMSTR)*, use STDERR. The high-level qualifier (*hlq*) depends on your installation. In this document, the default high-level qualifier is AZK.

2. Change the DD statement in the master procedures for each cluster to specify a unique z/OS UNIX System Services file in an existing directory. If you specified `log4j.appender.console.target=System.err` in the previous step, change the DD statement to STDERR. If you specified `log4j.appender.console.target=System.out` in the previous step, change the DD statement to STDOUT. An example is provided in the sample procedure, AZKMSTR, which shows the STDERR that can be used for a cluster that is denoted as `Cluster1`.

Note: It is recommended, but not required, that you use the directory that you specified for the `$SPARK_LOG_DIR` environment variable in the `spark-env.sh` file. If you don't use the `$SPARK_LOG_DIR` directory path, the path must exist and SPARKID must have authority to read and write to it.

```
//STDERR DD PATH='/var/spark/logs/azkmstrCluster1.err',  
// PATHOPTS=(OWRONLY,OCREAT,OTRUNC),PATHMODE=SIRWXU
```

3. Set and export the `SPARK_ZOS_MASTER_LOG` environment variable in the worker procedure for each cluster, to the same z/OS UNIX System Services file that is specified in the corresponding master procedure. An example is provided in the sample procedure, AZKWRKR. The following example shows the setting for the master for a cluster that is denoted as `Cluster1`.

```
export SPARK_ZOS_MASTER_LOG=/var/spark/logs/azkmstrCluster1.err
```

Results

When the master is started with no parameters, values are either taken from the environment variables, if set, or the defaults are used. `SPARK_MASTER_HOST` does not have a default, and therefore must be set in the `spark-env.sh` file.

When the master is started by using the automation technique for started tasks, the INFO messages are written to the file that is specified in Step “2” on page 65. These messages include the port number to which the master is bound, the master host, and the process ID for the master.

When the worker is started with no parameters, it reads the file that is specified in `SPARK_ZOS_MASTER_LOG` and finds the port number and any other values that are needed to ensure that the master is started. This is the same file that is written by the master when the cluster is started. If the master is not started, it retries for a period of time to allow the master time to start.

Configuring memory and CPU options

Complete this task to configure the memory and CPU options for Apache Spark.

About this task

Apache Spark is designed to consume a large amount of CPU and memory resources in order to achieve high performance. Therefore, it is essential to carefully configure the Spark resource settings, especially those for CPU and memory consumption, so that Spark applications can achieve maximum performance without adversely impacting other workloads.

Setting the memory and CPU options for your Spark workload can be an ongoing task. It is good practice to regularly monitor the actual resource consumption and identify potential bottlenecks by using the variety of monitoring tools described in [Chapter 14, “Resource monitoring for Apache Spark,”](#) on page 137. After obtaining performance measurements, you can fine tune your Spark workload.

For a quick reference to various Spark and z/OS system configuration options that you might wish to consider when tuning your environment for Open Data Analytics for z/OS, see [Appendix D, “Memory and CPU configuration options,”](#) on page 171.

Procedure

1. Determine the system resources that Spark needs.

Before you modify the memory and CPU settings for your Spark cluster, determine an amount of resources to give to your Spark workload. For a detailed discussion about sizing Spark workloads, see [Resource Tuning Recommendations for IBM z/OS Platform for Apache Spark \(https://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP102684\)](#).

a) Determine the amount of memory and processor that your Spark cluster is allowed to use.

When assessing the amount of memory to allot for Apache Spark, consider the following parameters:

- The amount of physical memory available on the system. Workloads perform best with minimal paging.
- The amount of memory required by other (possibly more important) subsystems, such as Db2.
- The amount of memory required by supporting software that interacts with Apache Spark, such as z/OS IzODA Mainframe Data Service (MDS).
- Other work that might run at the same time as Apache Spark.

Guidelines:

- IBM suggests that you start with at least 6 GB of memory for the Spark cluster, not including MDS. If you have installed WLM APAR OA52611 and you use WLM to manage your Spark workload, you can also cap the amount of physical memory that the Spark cluster can use to avoid impacting other workloads. [Table 6 on page 66](#) lists the suggested initial memory sizes for the components in a Spark cluster.

Table 6. Suggested initial memory sizing for a Spark cluster

Component	Default size	Suggested initial size
Spark master	1 GB	1 GB
Spark worker	1 GB	1 GB
Spark driver	1 GB	2 GB
Spark executor	1 GB	2 GB
Total	4 GB	6 GB

- Spark workloads are zIIP-eligible. If you have installed WLM APAR OA52611 and you use WLM to manage your Spark workload, you can use the honor priority attribute to minimize the amount of Spark work that is processed by the general processors (GPs). IBM suggests that you start with at least two zIIPs and one GP.

For more information about using the metering and capping functions provided by WLM APAR OA52611 for your Spark workload, see [“Configuring z/OS workload management for Apache Spark”](#) on page 73.

b) Determine how many concurrent Spark applications your Spark cluster must support.

This depends on your business requirements, the amount of resources available on your system, and the application requirements.

Alternatively, you can run multiple Spark clusters instead of configuring a single Spark cluster for multiple applications. This can be beneficial in the following cases:

- You have applications with different workload priorities (for instance, production vs. development). Although you can configure a single cluster that supports multiple applications, separate clusters might simplify the configuration and administration of each cluster as priority requirements change.
- You need to isolate workloads for security reasons or because of the nature of the data being analyzed.

If you do run multiple Spark clusters on the same z/OS system, be sure that the amount of CPU and memory resources assigned to each cluster is a percentage of the total system resources. Over-committing system resources can adversely impact performance on the Spark workloads and other workloads on the system.

- c) For each Spark application, determine the amount of executor heap that it requires.

An executor heap is roughly divided into two areas: data caching area (also called storage memory) and shuffle work area. In Spark 1.5.2 with default settings, 54 percent of the heap is reserved for data caching and 16 percent for shuffle (the rest is for other use). In Spark 2.0.2 and higher, instead of partitioning a fixed percentage, it uses the heap for each area, as needed. To properly estimate the size of the executor heap, consider both the data caching and shuffle areas.

To achieve optimal results, it is best to cache all of the data that Spark needs in memory to avoid costly disk I/O. When raw data is encapsulated in a Java object, the object is often larger than the size of the data that it holds. This expansion factor is a key element to consider when projecting how large the data cache should be. Ideally, the expansion factor is measured by persisting sample data to an RDD or DataFrame and using the **Storage** tab on the application web UI to find the in-memory size. When it is not possible to measure the actual expansion factor for a set of data, choosing a value from 2x to 5x of the raw data size for RDDs, and 1x to 2x of the raw data size for DataFrames should provide a reasonable estimate. Without considering the shuffle work area, you can determine the preliminary heap size using the equations:

in-memory data size = raw data size × estimated or measured expansion factor

executor heap = in-memory data size ÷ fraction of heap reserved for cache

The amount of memory needed for the shuffle work area depends entirely on the application. Shuffling happens when the application issues operations like `groupByKey` that requires data to be transferred. When the amount of shuffle memory is not adequate for the required shuffle operations, Spark spills the excess data to disk, which has a negative impact on performance. You can find occurrences and amounts of shuffle spills executor stderr log file, such as in the following example:

```
INFO ExternalSorter: Thread 106 spilling in-memory map of 824.3 MB to disk (1 time so far)
INFO ExternalSorter: Thread 102 spilling in-memory map of 824.3 MB to disk (1 time so far)
INFO ExternalSorter: Thread 116 spilling in-memory map of 824.3 MB to disk (1 time so far)
```

For more information about Spark memory management settings, see "Memory Management" in <http://spark.apache.org/docs/2.4.8/configuration.html>.

- d) Determine the number of executors for each application.

To do this, consider the following points:

- There is some startup cost associated with initializing a new Java virtual machine. Each executor runs within its own JVM, so the total overhead associated with JVM startup increases with each additional executor.
- Garbage collection (GC) costs are generally higher for large heaps. IBM Java can typically run with up to 100 GB heap with little GC overhead. If your application requires larger heap space, consider using multiple executors with smaller heaps. You can use the Spark application web UI to monitor the time spent in garbage collection.

- Efficient Spark applications are written to allow a high degree of parallelism. The use of multiple executors can help avoid the contention that can occur when too many threads execute in parallel within the same JVM.
 - There is increased overhead when multiple executors need to transfer data to each other.
- e) For each Spark application, determine the size of the driver JVM heap. This only applies if your driver runs on a z/OS system.

The driver heap should be large enough to hold the application results returned from the executors. Applications that use actions like `collect` or `take` to get back a sizable amount of data may require large driver heaps. If the returned data exceeds the size of the driver heap, the driver JVM fails with an out-of-memory error. You can use the `spark.driver.memory` and `spark.driver.maxResultSize` Spark properties to tune the driver heap size and the maximum size of the result set returned to the driver.

The driver process can either run on behalf of the user invoking the Spark application, inheriting the user's z/OS resource limits (client deploy mode), or it can use the backend cluster resources (cluster deploy mode). Generally, application developers tend to use client deploy mode while building their applications (for instance, using Jupyter notebooks or interactive **spark-shell**). Then, production-ready applications are submitted in cluster deploy mode.

If the Spark driver runs in cluster deploy mode, the driver is considered part of the cluster; therefore, the resources it uses (such as `spark.driver.memory`) are taken from the total amount allotted for the Spark cluster.

2. Update the Spark resource settings.

After you determine the resource requirements for your Spark workload, you can change the Spark settings and system settings to accommodate them.

Note: Applications can override almost all of the settings in the `spark-defaults.conf` file. However, it is good practice to set them in the configuration file or in the command line options, and *not* within the applications. Setting them in applications could cause applications to fail because of conflicts with system-level limits or because of system resource changes.

Figure 5 on page 69 shows a sample Spark cluster with various settings that are described later in this topic. The figure shows the Spark resource settings, their defaults (in parentheses) and their scopes (within the dashed lines). z/OS constraints, such as address space size (ASSIZEMAX) and amount of storage above the 2-gigabyte bar (MEMLIMIT) apply to these processes, as usual.

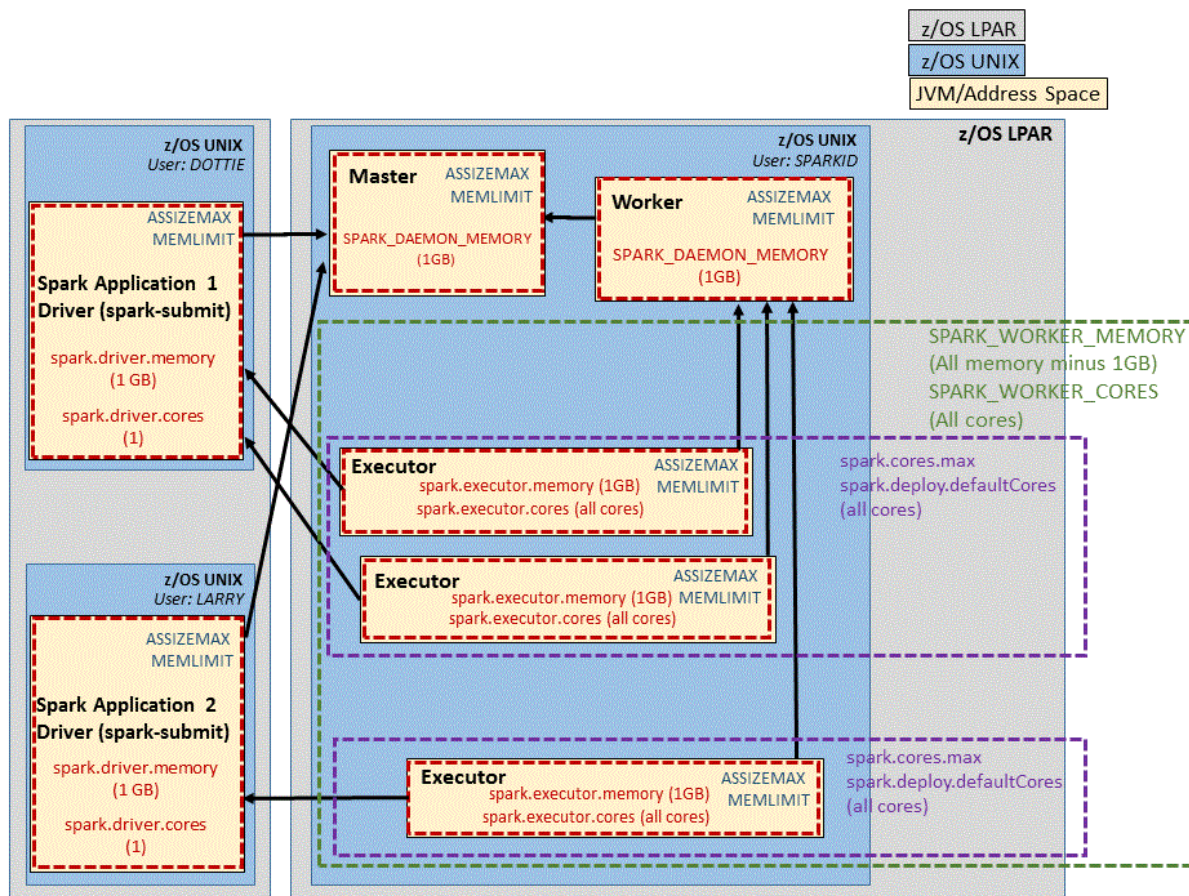


Figure 5. Sample Spark cluster in client deployment mode

- a) Set the number of processors and amount of memory that a Spark cluster can use by setting the following environment variables in the `spark-env.sh` file:

SPARK_WORKER_CORES

Sets the number of CPU cores that the Spark applications can use. The default is all cores on the host z/OS system. Note that Spark sees each zIIP that is enabled for simultaneous multithreading (SMT) as having two cores.

SPARK_WORKER_MEMORY

Sets the amount of virtual memory that Spark applications can use. The default is the total system memory, minus 1 GB.

Both settings apply to the amount of resources that the executors and cluster-deploy-mode drivers can use. They do not include the resources used by the master and worker daemons because the daemons do not process data for the applications.

- b) Set the number of cores that a Spark application (including its executors and cluster-deploy-mode drivers) can use by setting the following properties in the `spark-defaults.conf` file:

spark.deploy.defaultCores

Sets the default number of cores to give to an application if `spark.cores.max` is not set. The default is all the cores on the system.

spark.cores.max

Sets the maximum number of cores to give to an application. The default is to use the `spark.deploy.defaultCores` value.

- c) Set the number of concurrent applications that a Spark cluster can support.

This number is controlled by the amount of resources allotted to the Spark cluster and how much can be used by each application.

Assuming there is enough memory, the number of concurrent applications that a Spark cluster can support is expressed by the following equation:

SPARK_WORKER_CORES ÷ spark.cores.max (or spark.deploy.defaultCores)

Example: If you set SPARK_WORKER_CORES=15 and spark.cores.max=5, the Spark cluster can allow up to $15 \div 5 = 3$ applications to run concurrently, assuming there is enough memory for all of them.

There is no Spark setting that controls the amount of memory that each application can use.

To ensure that the Spark cluster can support the desired number of concurrent applications, set

SPARK_WORKER_MEMORY to an appropriate value. See step “2.g” on page 70 for more information.

- d) Set the amount of resources that each executor can use by setting the following properties in the spark-defaults.conf file:

spark.executor.cores

Sets the number of cores that each executor can use. The default is all CPU cores on the system.

spark.executor.memory

Sets the amount of memory that each executor can use. The default is 1 GB.

- e) Set the amount of resources that each driver can use by setting the following properties in the spark-defaults.conf file:

spark.driver.cores

Sets the number of cores that each driver can use. The default is 1.

spark.driver.memory

Sets the amount of memory that each driver can use. The default is 1 GB.

spark.driver.maxResultSize

Sets a limit on the total size of serialized results of all partitions for each Spark action (such as collect). Jobs will fail if the size of the results exceeds this limit; however, a high limit can cause out-of-memory errors in the driver. the default is 1 GB.

- f) Set the number of executors for each Spark application.

This number depends on the amount of resources allotted to the application, and the resource requirements of the executor and driver (if running in cluster deploy mode).

As discussed earlier, you can use spark.cores.max (or spark.deploy.defaultCores) to set the number of cores that an application can use. Assuming there is enough memory, the number of executors that Spark will spawn for each application is expressed by the following equation:

$(\text{spark.cores.max (or spark.deploy.defaultCores)} - \text{spark.driver.cores (if in cluster deploy mode)}) \div \text{spark.executor.cores}$

Example: If you set spark.cores.max=5, spark.driver.cores=1, and spark.executor.cores=2 and run in cluster deploy mode, the Spark worker spawns $(5 - 1) \div 2 = 2$ executors.

- g) Use the following set of equations to determine a proper setting for **SPARK_WORKER_MEMORY** to ensure that there is enough memory for all of the executors and drivers:

$\text{executor_per_app} = (\text{spark.cores.max (or spark.deploy.defaultCores)} - \text{spark.driver.cores (if in cluster deploy mode)}) \div \text{spark.executor.cores}$

$\text{app_per_cluster} = \text{SPARK_WORKER_CORES} \div \text{spark.cores.max (or spark.deploy.defaultCores)}$

$\text{SPARK_WORKER_MEMORY} \geq (\text{spark.executor.memory} \times \text{executor_per_app} \times \text{app_per_cluster}) + \text{spark.driver.memory (if in cluster deploy mode)}$

- h) Set the amount of memory to allocate to each daemon-like process—specifically, the master, worker, and the optional history server— by setting the **SPARK_DAEMON_MEMORY** environment variable in the spark-env.sh file.

The default is 1 GB and is satisfactory for most workloads.

There is no corresponding processor setting for the Spark daemon processes.

3. Update system resource settings.

In addition to the Spark settings, there are system-level settings that are required or recommended for your Spark workload to function properly and efficiently.

Continue by determining and setting values for the system-level settings shown in [Figure 5 on page 69](#).

a) Set `_BPX_SHAREAS=NO`.

The worker process spawns the executor processes. If the `_BPX_SHAREAS` environment variable is set to YES, the new executor process runs in the same address space as its parent. If `_BPX_SHAREAS` is set to NO, the executor runs in its own address space. Because executor processes typically consume a large amount of system resources, IBM suggests that you set `_BPX_SHAREAS=NO` for easier resource management and isolation, and to increase available resources for the executor.

b) Set the MEMLIMIT value.

z/OS uses the MEMLIMIT setting to control the amount of 64-bit virtual memory that an address space can use. For the user ID that starts the Spark cluster, the MEMLIMIT value must be set to the largest JVM heap size (typically, `spark.executor.memory`) plus the amount of native memory needed. For the user ID that submits the application to the cluster, the MEMLIMIT must be set to `spark.driver.memory` plus the amount of native memory needed. The required amount of native memory varies by application. 1 GB is typically needed if the JVM heap size is less than 16 GB and 2 GB is typically needed if the JVM heap size is greater than 16 GB.

Spark has the ability to use off-heap memory, which is configured through the `spark.memory.offHeap.enabled` Spark property and is disabled by default. If you enable off-heap memory, the MEMLIMIT value must also account for the amount of off-heap memory that you set through the `spark.memory.offHeap.size` property in the `spark-defaults.conf` file.

If you run Spark in local mode, the MEMLIMIT needs to be higher as all the components run in the same JVM; 6 GB should be a sufficient minimum value for local mode.

You can set the MEMLIMIT value in any of the following ways:

- Recommended: Set the MEMLIMIT value in the OMVS segment of the security profile for the user ID that you use to start Spark. For instance, you can use the RACF ALTUSER command to set the MEMLIMIT for a user ID. For more information about the ALTUSER command, see "ALTUSER (Alter user profile)" in [z/OS Security Server RACF Command Language Reference](#).
- Specify the MEMLIMIT parameter on the JCL JOB or EXEC statement, which overrides the MEMLIMIT value in the user's security profile, if you start your Spark cluster from z/OS batch. For more information, see "MEMLIMIT parameter" in [z/OS MVS JCL Reference](#).
- Use the IEFUSI exit, which overrides all of the other MEMLIMIT settings. Therefore, if you use the IEFUSI exit, be sure to make exceptions for Spark jobs. For more information, see "IEFUSI - Step Initiation Exit" in [z/OS MVS Installation Exits](#).
- Set the system default in the SMFPRMxx member of parmlib. The system default is used if no MEMLIMIT value has been set elsewhere. The default value is 2 GB. You can use the DISPLAY SMF,O operator command to display the current settings. For more information, see "SMFPRMxx (system management facilities (SMF) parameters)" in [z/OS MVS Initialization and Tuning Reference](#).

To check the MEMLIMIT setting for a specific user ID, you can issue the `ulimit` command in the z/OS UNIX shell while logged in with that user ID. The following example shows the command response:

```
/bin/ulimit -a
core file             8192b
cpu time              unlimited
data size             unlimited
file size             unlimited
stack size            unlimited
```

file descriptors	1500
address space	unlimited
memory above bar	16384m

- c) Ensure that the settings in your BPXPRMxx parmlib, especially the MAXASSIZE parameter, meet the suggested minimums for Java.

For the suggested minimum values, see [Working with BPXPRM settings \(https://www.ibm.com/support/knowledgecenter/SSYKE2_8.0.0/com.ibm.java.zos.80.doc/user/zos_bpxprm.html\)](https://www.ibm.com/support/knowledgecenter/SSYKE2_8.0.0/com.ibm.java.zos.80.doc/user/zos_bpxprm.html) in IBM SDK, Java Technology Edition z/OS User Guide (www.ibm.com/support/knowledgecenter/SSYKE2_8.0.0/pdf/en/sdkandruntimetimeguide.zos.80_8.0.pdf). For more information about the BPXPRMxx member, see "BPXPRMxx (z/OS UNIX System Services parameters)" in *z/OS MVS Initialization and Tuning Reference*.

You can set some of these parameters at the user ID level, such as the maximum address space size (the ASSIZEMAX value in the OMVS segment of the security profile for the user ID), by using the RACF ALTUSER command. For more information about the ALTUSER command, see "ALTUSER (Alter user profile)" in *z/OS Security Server RACF Command Language Reference*.

- d) Ensure that you have enough extended common service area (ECSA) and extended system queue area (ESQA) memory configured on your system to account for Spark usage.

For instance, Spark memory-maps large files to improve performance. However, the use of memory map services can consume a large amount of ESQA memory. For more information, see "Evaluating virtual memory needs" in *z/OS UNIX System Services Planning*.

You can use the RMF reports to monitor the ECSA and ESQA usage by your Spark cluster. For more information, see [Chapter 14, "Resource monitoring for Apache Spark," on page 137](#).

- e) Consider using z/OS workload management (WLM) to manage Spark workload. With WLM, you define performance goals and assign a business importance to each goal. The system then decides how much resource, such as CPU or memory, to give to a workload to meet the goal. WLM constantly monitors the system and adapts processing to meet the goals. You can also use WLM to cap the maximum amount of CPU time and physical memory used by Spark. For more information, see ["Configuring z/OS workload management for Apache Spark" on page 73](#).
- f) Consider using simultaneous multithreading (SMT) to improve throughput. Beginning with IBM z13[®], you can enable SMT on zIIPs. Open Data Analytics for z/OS is zIIP-eligible and might benefit from SMT. When executing on a system configured with sufficient zIIP capacity, the benchmarks demonstrated in [Resource Tuning Recommendations for IBM z/OS Platform for Apache Spark \(https://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP102684\)](#) showed throughput improvements of 15 to 20 percent when SMT was enabled.
- g) Ensure that the system has enough large pages to accommodate all requests.

When available, 1M pageable large pages are the default size for the object heap and code cache for Java on z/OS. Large pages provide performance improvements to applications such as Spark that allocate a large amount of memory. The use of 1M pageable large pages requires IBM zEnterprise EC12, or later, with the Flash Express feature (#0402).

When the system's supply of pageable 1M pages is depleted, available 1M pages from the LFAREA are used to back pageable 1M pages. Adjust your LFAREA to account for the Spark JVMs. You can set the size of the LFAREA in the IEASYSxx member of parmlib.

For more information about configuring large pages for Java, see [Configuring large page memory allocation \(https://www.ibm.com/support/knowledgecenter/SSYKE2_8.0.0/com.ibm.java.zos.80.doc/user/alloc_large_page.html\)](https://www.ibm.com/support/knowledgecenter/SSYKE2_8.0.0/com.ibm.java.zos.80.doc/user/alloc_large_page.html). For more information about the IEASYSxx parmlib member, see "IEASYSxx (system parameter list)" in *z/OS MVS Initialization and Tuning Reference*.

What to do next

When you have completed configuring memory and CPU options, continue with ["Configuring z/OS workload management for Apache Spark" on page 73](#).

Configuring z/OS workload management for Apache Spark

One of the strengths of the IBM Z platform and the z/OS operating system is the ability to run multiple workloads at the same time within one z/OS image or across a Parallel Sysplex®. Workloads usually have different, often competing, performance and resource requirements that must be balanced to make the best use of an installation's resources, maintain the highest possible throughput, and achieve the best possible system responsiveness. The function that makes this possible is dynamic workload management, which is implemented in the workload management component of z/OS.

With z/OS workload management (WLM), you define performance goals and assign a business importance to each goal. You define the goals for work in business terms, and the system decides how much resource, such as CPU or memory, to assign so as to meet the goal. WLM constantly monitors the system and adapts processing to meet the goals.

You can configure WLM to manage Open Data Analytics for z/OS workloads to optimize system performance.

Overview of Apache Spark Processes

Apache Spark uses a master/worker architecture. A Spark cluster typically has multiple processes, each running in its own Java virtual machine (JVM).

The following list describes the most significant processes:

- The *master* daemon allocates resources across applications.
- The *worker* daemon monitors and reports resource availability and, when directed by the master, spawns executors. The worker also monitors the liveness and resource consumption of the executors.
- The *executor* performs the actual computation and data processing for the application.
- The *driver* program runs the main function of the application and creates a SparkContext.

The master and worker processes are generally lightweight. By contrast, the executors attempt to use all available system resources by default. You can use the `spark-defaults.conf` file, `spark-env.sh` file, and command line parameters to constrain the number of CPU cores and amount of memory that each executor can consume. These parameters, however, are static and require that the Spark processes be restarted for changes to take effect. (For more information about the Apache Spark configuration options, see <http://spark.apache.org/docs/2.4.8/configuration.html>. For more information about tuning the Apache Spark cluster, see <http://spark.apache.org/docs/2.4.8/tuning.html>.)

The worker process spawns the executor processes. If the `_BPX_SHAREAS` environment variable is set to YES, the new executor process will run in the same address space as its parent. If `_BPX_SHAREAS` is set to NO, the executor will run in its own address space. Because executor processes usually consume a large amount of system resources, IBM suggests that you set `_BPX_SHAREAS=NO` for easier resource management and isolation, and to increase available resources for the executor.

The following list describes other processes you might use in your z/OS IzODA Spark configuration:

- The Shuffle service allows executors to transfer (shuffle) data across executors, as needed.
- The History server uses previously saved event logs to display detailed information about running and completed Spark applications. History server reconstructs the Web UI that is normally only available while the SparkContext for an application is running.

WLM provides a more dynamic way to manage the performance of your Spark workloads.

Assigning job names to Spark processes

Assigning unique job names to Spark processes helps to identify the purpose of each process, correlate a process to an application, and facilitate the grouping of processes into a WLM service class.

You can use Spark properties and the `_BPX_JOBNAME` environment variable to assign job names to executors, drivers, and other Spark processes. If you are using started tasks to start various Spark processes, you may use the job names you setup for the started task instead.

Note: The user ID of the Spark worker daemon requires READ access to the BPX.JOBNAME profile in the FACILITY class to change the job names of the executors and drivers.

Note: A specification that yields a job name that is more than 8 characters, raises an exception.

Setting the job name of the executors

A system programmer can specify the **spark.zos.executor.jobname.prefix** or the **spark.zos.executor.jobname.template** property in the `spark-defaults.conf` configuration file to define a job name.

The APAR PI89136 is required to use **spark.zos.executor.jobname.template**.

If **spark.executorEnv._BPX_JOBNAME** is specified in the `spark-defaults.conf` file, the **_BPX_JOBNAME** environment variable takes precedence. Only the executor job name or job name prefix specified in the `spark-defaults.conf` file is honored; any such settings that are specified in an application or on the command line are ignored.

Using the **spark.zos.executor.jobname.prefix**

A system programmer can specify the **spark.zos.executor.jobname.prefix** property in the `spark-defaults.conf` configuration file to define a job name prefix which, when combined with some or all of an application instance number, forms the job name of the executors. An application instance number is the last four digits of an application ID and is unique to each application within a Spark cluster. Application IDs appear on both the master and application web UIs. The actual job names of the executors will consist of the defined prefix plus as many digits of the corresponding application instance number, starting from the last (right-most) digit, as can fit to form an 8-character job name.

For example, if you specify `spark.zos.executor.jobname.prefix=SPARKX`, the job name of the executors for application 0001 will be SPARKX01. Since z/OS job names are limited to eight characters, no special character is used to differentiate separate executor instances for the same application; that is, all executors for application 0001 will have the same job name.

The `spark-defaults.conf` file contains a default setting of `spark.zos.executor.jobname.prefix=0DASX`. If no executor prefix is specified in the `spark-defaults.conf` file, the job names of the executors follow the z/OS UNIX default of a user ID with a numeric suffix. For details, see [“Using _BPX_JOBNAME to assign job names to Spark processes” on page 76](#).

Using the **spark.zos.executor.jobname.template**

A system programmer can specify a template for generating job names by using the **spark.zos.executor.jobname.template** property in the `spark-defaults.conf` configuration file. The template property provides further customization when you are generating job names for executors. The template uses variables that can be substituted for several pieces of information about the work that the executor is running. These variables include *cluster*, *application*, and *executor*.

cluster

The number that is specified by **spark.zos.cluster.instanceNumber**.

Default length: 1

application

The application instance number.

Default length: 4

executor

The executor instance number.

Default length: 1

A number of digits can be specified for each, for example:

<executor:2>

The least 2 significant digits of the executor instance number are used.

For example, ODA<cluster:1><application:2><executor:2> where cluster number is 2, application number is 0312, and executor is 0000, would yield ODA21200.

Note: The **spark.zos.executor.jobname.template** property supersedes **spark.zos.executor.jobname.prefix** options.

Setting the job name of the driver in cluster deploy mode

A system programmer can specify the **spark.zos.driver.jobname.prefix** or **spark.zos.driver.jobname.template** property in the `spark-defaults.conf` configuration file to define a job name.

The APAR PI89136 is required to use **spark.zos.driver.jobname.template**.

Only the driver job name prefix or job name template setting that is specified in the `spark-defaults.conf` file is honored; any such settings that are specified in an application or on the command line are ignored.

Using the **spark.zos.driver.jobname.prefix**

A system programmer can specify the **spark.zos.driver.jobname.prefix** property in the `spark-defaults.conf` configuration file to define a job name prefix which, when combined with some or all of a driver instance number, forms the job name of the driver in cluster deploy mode. A driver instance number is the last four digits of a driver ID and is unique to each driver within a Spark cluster. The driver IDs of cluster deploy-mode applications appears on the master web UI. The actual job name of the driver will consist of the defined prefix plus as many digits of the corresponding driver instance number, starting from the last (right-most) digit, as can fit to form an 8-character job name.

For example, if you specify `spark.zos.driver.jobname.prefix=SPARKD`, the job name of the driver with driver instance number 0001 will be SPARKD01.

Note: The driver prefix does not apply to drivers in client deploy mode since client-deploy-mode driver processes are already up and running before the configuration file is read; thus, their job names cannot be changed.

The `spark-defaults.conf` file contains a default setting of `spark.zos.driver.jobname.prefix=ODASD`. If no driver prefix is specified in the `spark-defaults.conf` file, the job name of the driver in cluster deploy mode follows the z/OS UNIX default of a user ID with a numeric suffix.

Using the **spark.zos.driver.jobname.template**

A system programmer can specify a template for generating job names by using the **spark.zos.driver.jobname.template** property in the `spark-defaults.conf` configuration file. The template property provides further customization when you are generating job names for the driver. The template uses variables that can be substituted for several pieces of information about the work that the driver is running. These variables include *cluster* and *driver*.

cluster

The number that is specified by **spark.zos.cluster.instanceNumber**.

Default length: 1

driver

The driver instance number.

Default length: 4

For example, ODA<cluster:1><driver:2> where cluster number is 6 and driver number is 0312, would yield ODA612.

Note: The `spark.zos.driver.jobname.template` property supersedes `spark.zos.driver.jobname.prefix` options.

Using `_BPX_JOBNAME` to assign job names to Spark processes

When a new z/OS UNIX process is started, it runs in a z/OS UNIX initiator (a BPXAS address space). By default, this address space has an assigned job name of `userIDx`, where `userID` is the user ID that started the process, and `x` is a decimal number. You can use the `_BPX_JOBNAME` environment variable to set the job name of the new process. Assigning a unique job name to each Spark process helps to identify the purpose of the process and makes it easier to group processes into a WLM service class.

The following example shows a portion of a script that assigns different job names for the master and worker processes before starting them:

```
_BPX_JOBNAME='ODASM1A' /usr/lpp/IBM/izoda/spark/spark23x/sbin/start-master.sh
sleep 5
export _BPX_SHAREAS=NO
_BPX_JOBNAME='ODASW1A' /usr/lpp/IBM/izoda/spark/spark23x/sbin/start-slave.sh spark://
127.0.0.1:7077
```

A system programmer can set the job name of the executor address space by setting `spark.executorEnv._BPX_JOBNAME` in the `spark-defaults.conf` configuration file as in the following example:

```
spark.executorEnv._BPX_JOBNAME    ODASX1A
```

If the job name of the executor is set in this manner, all Spark executors will have the same job name.

If `spark.executorEnv._BPX_JOBNAME` is specified in the `spark-defaults.conf` file, the `_BPX_JOBNAME` environment variable takes precedence. Only the executor job name or job name prefix specified in the `spark-defaults.conf` file is honored; any such settings that are specified in an application or on the command line are ignored.

Tip: If you use the Jupyter Kernel Gateway with Apache Toree to access Spark, it is good practice to use the same naming convention for the Toree instances. For instance, if you set `_BPX_JOBNAME=ODASD` for the Toree instances and set `spark.zos.executor.jobname.prefix=ODASX` for the executors and you have client authentication enabled, you can then classify the driver and the executors for user Elpida's application by using transaction name and user ID qualifiers, such as `TN=ODAS*` and `UI=Elpida`.

In addition to the master, worker, driver, and executors, your Spark environment may have additional processes, such as the Spark history server. You can also use `_BPX_JOBNAME` to assign unique job names to those processes.

Setting `_BPX_JOBNAME` requires appropriate privileges. For more information about `_BPX` environment variables, see [z/OS UNIX System Services Planning](#).

Using `_BPX_ACCT_DATA` to assign accounting information to Spark processes

In addition to using `_BPX_JOBNAME`, you may assign accounting information to your Spark processes by using the `_BPX_ACCT_DATA` environment variable. You may use `_BPX_ACCT_DATA` instead of `_BPX_JOBNAME` to classify work. If you are using started tasks for Spark processes, you will use the job name or job accounting data to classify the work with WLM and you will not need these environment variables.

The following example shows how to start the master specifying the `_BPX_ACCT_DATA` variable:

```
_BPX_ACCT_DATA='SPARKDAT' $SPARK_HOME/sbin/start-master.sh
```

Use this same method (prefixing `_BPX_ACCT_DATA='xxx'` on the command) to start other IzODA processes such as the worker, Shuffle service, History server, and Livy with accounting information. Depending on your WLM and Spark configuration you might also need to specify `_BPX_JOBNAME` as shown in [“Using `_BPX_JOBNAME` to assign job names to Spark processes” on page 76](#) to set the job name.

In client deploy and cluster deploy spark-submits, the accounting information for spawned executors is set by starting the worker with **_BPX_ACCT_DATA**. For local deploy mode spark-submits, only a single address space is produced and can be passed accounting data as shown in the previous example.

Overview of WLM classification

You specify goals for the WLM services for Open Data Analytics for z/OS work in the same manner as for other z/OS workloads, by associating the work with a service class. In the service class, you assign goals to the work, along with the relative importance of each goal. You can also assign limits for CPU and memory capacity to be available to a service class. To associate incoming work with a particular service class, you must also define classification rules.

Example of a WLM classification scenario

Figure 6 on page 77 shows a logical view of a the WLM configuration for a sample classification scenario.

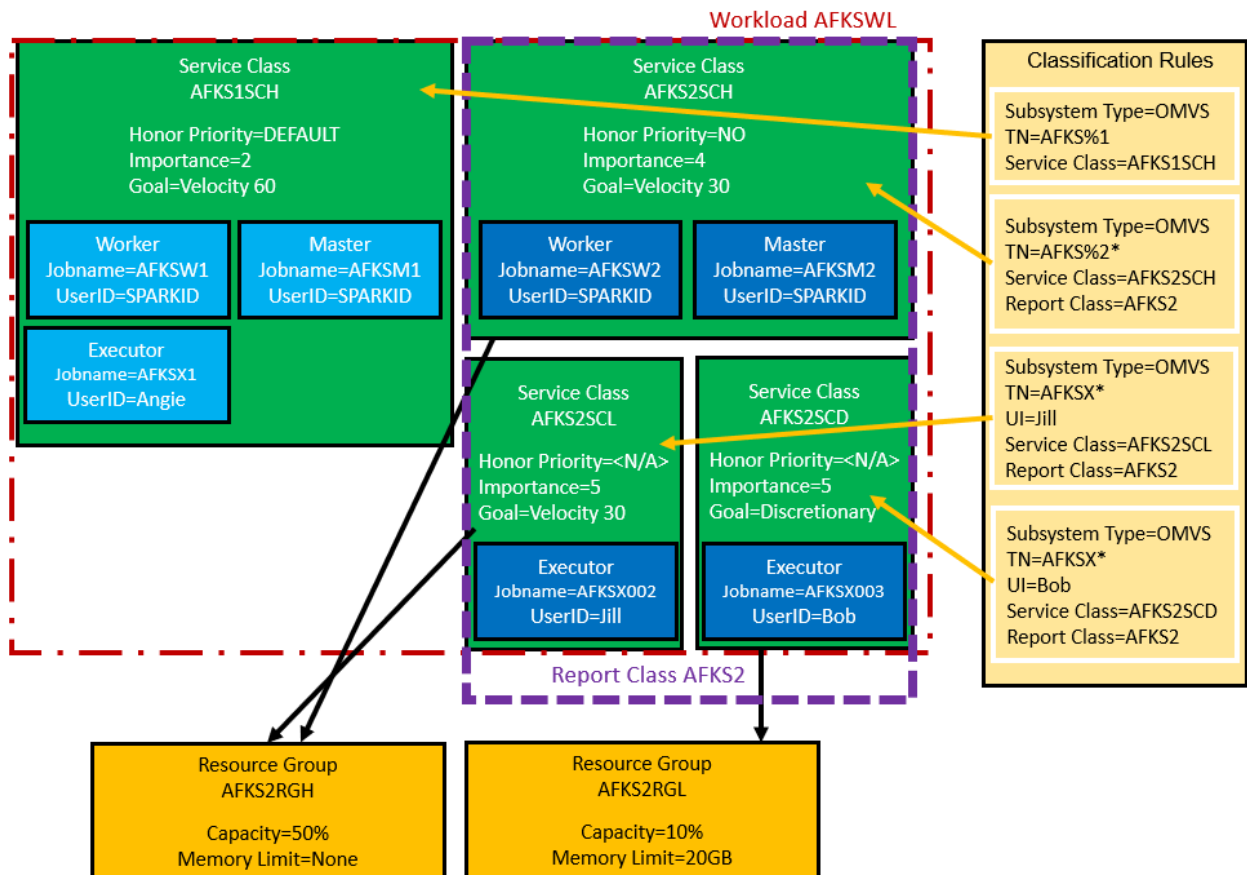


Figure 6. Logical view of a sample WLM classification scenario

In Figure 6 on page 77, four service classes are defined: ODAS1SCH, ODAS2SCH, ODAS2SCL, and ODAS2SCD.

- The ODAS2SCH and ODAS2SCL service classes are associated with the ODAS2RGH resource group, which allows 50 percent GP capacity and has no memory limit.
- The ODAS2SCD service class is associated with the ODAS2RGL resource group, which allows only 10 percent GP capacity and has a memory limit of 20 GB.

Four classification rules are defined that classify work as follows:

- All processes in Spark cluster 1, whose names match the ODAS%1* pattern, are classified into the ODAS1SCH service class. User Angie's executor receives its job name by using the **_BPX_JOBNAME** environment variable.
- Master and worker processes in Spark cluster 2, whose names match the ODAS%2* pattern, are classified into the ODAS2SCH service class.
- The executor for user Jill in Spark cluster 2 has a job name of ODASX002 and is classified into the ODAS2SCL service class. The job name is generated by specifying the `spark.zos.executor.jobname.prefix` property.
- The executor for user Bob in Spark cluster 2 has a job name of ODASX003 and is classified into the ODAS2SCD service class. The job name is generated by specifying the `spark.zos.executor.jobname.prefix` property.

All service classes in Spark cluster 2 (ODAS2SCH, ODAS2SCL, and ODAS2SCD) are grouped together into the ODAS2 report class. Further, the ODASWL workload groups both Spark clusters together.

The topics that follow discuss each of these WLM components in more detail.

Defining WLM service classes for Spark

A *service class* is a named group of work with similar performance goals, resource requirements, or business importance. Based on your business needs, you can define one or more service classes for your Apache Spark cluster. For example, you might choose to define any of the following types of service classes:

- One service class for all Spark processes, including z/OS IzODA Mainframe Data Service (FMID HMDS120)
- One service class with a higher importance for the master, worker, and driver, and another service class with a lower importance for the executors
- One service class for each type of Spark application
- One service class for each Spark cluster. For instance, you might have one Spark cluster for application development and another for production applications.

Note: You can define up to 100 service classes in a Parallel Sysplex. However, to simplify policy management, you might want to reuse existing service classes for your Spark workload rather than define new ones. For instance, you can place the Spark master and worker daemons in a service class with other OMVS daemons.

To define a service class, you first need to determine the following information:

- Service class name and description
- Name of the workload associated with the service class
- Name of the resource group associated with the service class
- Whether the service class is CPU-critical or requires high I/O priority
- Performance periods, goals, and importance
- Whether honor priority needs to be set, if WLM APAR OA52611 is installed
 - This option ensures that eligible work will run on zIIPs and not GPs. See Honor Priority and zIIP-eligibility considerations later in this section.
- Memory limits (if any) to be set on the resource group, if WLM APAR OA52611 is installed

Workloads

A WLM *workload* is a named collection of work to be reported as a unit. You might consider defining a Spark workload for easy resource monitoring. To create a workload, select the **2. Workloads** option on the Definition Menu in the WLM ISPF application.

Example: The following figure shows an example of creating a workload called ODASWL:

```

                                Create a Workload
Command ==> -----
Enter or change the following information:
Workload Name . . . . . ODASWL_ (Required)
Description . . . . . Sample Spark workload_____

```

Resource groups

A WLM *resource group* is a way of limiting or guaranteeing the availability of general purpose processor capacity to one or more service classes. WLM APAR OA52611 also provides the ability to limit the physical memory, at a system level, consumed by all address spaces that belong to a resource group.

If the physical memory consumption by address spaces associated with a resource group is at or near its memory limit, the system may take action against the address spaces. For instance, the system might initiate paging (within the address spaces that are associated with the resource group), suspend storage requesters, or, in certain situations, abnormally end (abend) entire address spaces. IBM suggests using the memory limit attribute as an upper limit so that, under normal conditions, resource consumption will operate well below the limit.

Typically, Spark workloads are resource intensive, so you might wish to consider defining resource groups to ensure that your Spark workload does not impact other workloads that, in business terms, are more important. In addition, Spark executors are generally good candidates to have their physical memory usage capped, since they typically consume a lot of memory. However, you might see performance degradation or even termination of your Spark applications if they reach their memory limit. In the case of termination, Spark applications usually observe an ABEND SEC6. Another situation that can occur is if an executor is not executing or responding for an extended period, such as in a heavy paging situation that is initiated by approaching a resource group's memory limit, the executor might be declared lost by the driver. This might prompt the spawning of a new replacement executor and it becomes possible to have two or more executors running in the same resource group, performing the same computation. In a highly competitive resource situation, the driver might declare that it cannot finish successfully and terminates. IBM suggests that you *not* place the Spark master and worker daemons in a memory-limited resource group, so that the Spark cluster stays alive even if the system terminates executors.

IBM suggests that you consider your Spark configuration when using WLM to limit resources available to Spark processes. The **SPARK_WORKER_CORES** parameter should be set to a value less than or equal to the number of processors available to the Spark executors after WLM-imposed limits. The **SPARK_WORKER_MEMORY** parameter should also be set to a value less than or equal to the memory limit defined for the executors' resource group. Setting appropriate values for these parameters helps Spark to dispatch work within its limits and help reduce unintended consequences. (For more information about the Apache Spark configuration options for standalone mode, see <http://spark.apache.org/docs/2.4.8/spark-standalone.html>.)

For more information about memory limits for resource groups, see "Storage Management Overview" in *z/OS MVS Initialization and Tuning Guide* and "Defining service classes and performance goals" in *z/OS MVS Planning: Workload Management*. For more information about monitoring resource consumption by Spark applications, see Chapter 14, "Resource monitoring for Apache Spark," on page 137.

To create a resource group, select the **3. Resource Groups** option on the Definition Menu in the WLM ISPF application. You can define up to 32 resource groups in a Parallel Sysplex; however, to simplify policy management, you might wish to reuse existing resource groups for your Spark workload rather than define new ones.

Example: The following figure shows an example of creating a resource group called ODASRG with a maximum capacity of 10 percent of the LPAR share in the general processor pool and a memory limit of 20 GB:

```

                                Create a Resource Group
Command ==> -----
Enter or change the following information:

Resource Group Name . . . . : ODASRG_ (required)
Description . . . . . Sample Spark resource group

Define Capacity:
2  1. In Service Units (Sysplex Scope)
   2. As Percentage of the LPAR share (System Scope)
   3. As a Number of CPs times 100 (System Scope)
Minimum Capacity . . . . . 10-----
Maximum Capacity . . . . . 10-----
Include Specialty Processor Consumption NO____(YES or NO)

Memory Limit (System Scope)    20----- GB

```

Other service class attributes

You can define the rest of the service class attributes on the Create a Service Class panel. To create a service class, select the **4. Service Class** option on the Definition Menu in the WLM ISPF application.

CPU Critical and I/O Priority Group

You can use the CPU Critical attribute and the I/O Priority Group attribute to give higher CPU and I/O priorities to the work in the service class. Spark workloads do not typically require higher CPU or I/O priorities.

Honor Priority and zIIP-eligibility considerations

Running z/OS workloads on System z Integrated Information Processors (zIIPs) reduces software licensing costs as compared to running on general processors (GPs). Most Spark workloads are zIIP-eligible. IBM suggests that you start with at least two zIIPs and one GP. You may use WLM to prevent Spark work from spilling over to GPs.

WLM APAR OA52611 also provides the ability to exclude a service class from the system-wide Honor Priority defaults. By setting the Honor Priority attribute to NO on a service class, work in that service class that is eligible to run on specialty processors (such as zIIPs) does not overflow to general-purpose CPs (GPs) when there is insufficient capacity on the specialty processors. The default is to use the IFAHONORPRIORITY and IIPHONORPRIORITY parameters that are specified in the IEAOPTxx member of parmlib.

You can set the Honor Priority attribute to NO if you want to minimize the amount of Spark work processed by the GPs. However, give careful consideration to setting Honor Priority attributes to NO on service classes, especially in the case of highly utilized specialty processors. For instance, a Spark service class restricted to zIIPs may consume much of the zIIP capacity and cause other zIIP-eligible workloads, such as IBM Db2, to overflow to GPs.

In addition, Spark applications, by default, fall into the default OMVS service class. If the default OMVS service class is restricted to specialty engines, other processes in the same service class, such as terminals or ssh sessions, might become unresponsive. Spark applications might also experience timeouts during driver and executor communication if they are waiting for CPU time for too long. Monitor these timeouts and adjust the timeout values in the Spark configuration accordingly. For more information about monitoring Spark resources, see [Chapter 14, “Resource monitoring for Apache Spark,” on page 137](#).

zIIP-eligible work can incur some serialization cost and will resolve on GPs. For Spark, this might happen as the worker splits work amongst multiple executors. Consider the following trends when planning your environment:

1. A greater number of executors allows work to resolve faster, but incurs GP runtime to resolve serialization issues.
2. A smaller number of executors allows work to stay on zIIPs, but finishes at a slower speed.

See [“Configuring memory and CPU options” on page 65](#) for the configuration values that influence the number of executors created by Spark.

Performance periods

Performance periods are available for work that has variable resource requirements and for which your goals change as the work uses more resources. For each performance period, you specify a goal, and importance level, and an optional duration.

Goals

There are three types of goals:

- *Response time goals* indicate how quickly you want your work to be processed. The response time represents the time that WLM is aware of the unit of work and it is measured differently for different subsystems. It is not an end-to-end response time. Response time goals are not appropriate for all types of work, such as long-running jobs or workloads that do not have frequent transaction completions. There are two types of response time goals:
 - *Average response time* goals are typically used for transactions that have extremely homogenous response times.
 - *Percentile response time* goals are used to set a response time target for a given percentage of transactions (for instance, 80 percent of transactions within 3 seconds).
- *Execution velocity goals* define how fast work should run when ready, without being delayed for processor, storage, I/O access, and queue delay. Execution velocity goals are intended for work for which response time goals are not appropriate, such as started tasks or long-running work.
- *Discretionary goals* are for low-priority work for which you do not have any particular performance goal. WLM processes the work using resources not required to meet the goals of other service classes. Note that the honor priority option is insignificant for discretionary service classes because work assigned to these service classes never gets help from GPs.

The type of goal that is best suited for your Spark workload depends on the workload itself.

- For long-running Spark applications, such as model training or streaming workloads, IBM suggests using velocity goals.
- For short-running Spark applications, such as machine learning model evaluations, a response time goal might be more appropriate.
- For exploratory data science and experimental or new workloads, a discretionary goal might be a better fit.

When determining workload goals, also consider how honor priority has been set for the service class. For instance, if a Spark service class has high velocity goals and is set to use only zIIPs, these two settings might interfere with each other under certain conditions and cause an undesired state for the Spark applications and their workload priorities.

Duration

Duration is the length of the performance period, in service units. For a given period, if the work in that period does not complete by the time the specified number of service units have been used, the work moves into the next performance period and its associated goal. You do not specify a duration if there is only one period or on the last period in a service class.

Importance

Importance is the relative importance of the service class goal and is only used when work is not meeting its goal. WLM uses the importance value to decide the order in which work should receive resources when that work is not achieving its goal. Importance is expressed in five levels: 1 to 5, with 1 being the highest importance.

Example: The following figure shows an example of creating a service class called ODASSC1 for the ODASWL workload:

```

                                Create a Service Class                                Row 1 to 2
Command ==> -----
Service Class Name . . . . . ODASSC1 (Required)
Description . . . . . Sample Spark service class
Workload Name . . . . . ODASWL (name or ?)
Base Resource Group . . . . . ODASRG (name or ?)
Cpu Critical . . . . . NO (YES or NO)
I/O Priority Group . . . . . NORMAL (NORMAL or HIGH)
Honor Priority . . . . . NO (DEFAULT or NO)

Specify BASE GOAL information. Action Codes: I=Insert new period,
E=Edit period, D=Delete period.

-- Period -- ----- Goal -----
Action # Duration Imp. Description
-- 1 ----- 3 Execution velocity of 50

```

The following figure shows the definition of the velocity goal for period 1:

```

                                Execution velocity goal
Enter an execution velocity for period 1

Velocity . . . 50 (1-99)

Importance . . 3 (1=highest, 5=lowest)
Duration . . . ----- (1-999,999,999, or
                        none for last period)

F1=Help      F2=Split      F5=KeysHelp  F9=Swap
F12=Cancel

```

Defining WLM report classes for Spark

WLM allows you to specify report classes to report on a subset of work running in a single service class and to combine work running in different service classes within one report class. Although report classes are optional, they can be useful if you have multiple service classes defined for your Spark workload or if you commingle your Spark and non-Spark workloads in the same service classes. For instance, you might have separate service classes set up for Spark executor processes and non-executor processes. You can put both service classes in the same report class and monitor the resource usage as a whole. The RMF workload activity report shows the performance metrics and resource utilization for work in each service class and report class. For more information about the RMF workload activity report, see [z/OS RMF Report Analysis](#).

To create a report class, select the **7. Report Classes** option on the Definition Menu in the WLM ISPF application.

Example: The following figure shows an example of creating a report class called ODASRC:

```

                                Create a Report Class
Command ==> -----

Enter or change the following information:

Report Class name . . . . . ODASRC_ (Required)
Description . . . . . Sample Spark Report Class -----

```

Defining WLM classification rules for Spark

After you have the service classes and, optionally, the report classes defined for Spark, you can define classification rules for your Spark work. WLM uses *classification rules* to map work coming into the system to a specific service class and report class. The classification is based on work qualifiers. The first qualifier is the subsystem type that receives the work request. The subsystem type for work that is processed in z/OS UNIX (which Spark workloads are) is OMVS.

The OMVS subsystem type supports the following secondary qualifiers:

- Sysplex name (PX)
- System name (SY)
- Transaction name or job name (TN)
- User ID (UI)
- Accounting Information (AI)

For instance, if you use a dedicated user ID to start the Spark cluster, this user ID could serve as a classification mechanism. If you have enabled client authentication, you can also define a separate service class for each user based on the user ID and job name. For more information about enabling client authentication, see [“Configuring z/OS Spark client authentication” on page 41](#).

You can also use masking (%) or wildcard (*) notation as a way to group work to the same service class or report class. In earlier examples, all Spark job names are prefixed with ODAS. You can easily group these jobs into the same WLM service class by specifying a classification rule for the OMVS subsystem by using a qualifier of transaction name ODAS*. Alternatively, you can place the master, worker, and driver in one service class (based on transaction name qualifiers of ODASM*, ODASW*, and ODASD*, respectively) and place the executors in another service class (based on a transaction name qualifier of ODASX*).

Tip: IBM suggests classifying any significant Spark workload into its own service class or classes.

Note: z/OS IzODA Mainframe Data Service (FMID HMDS120) uses different classification rules. For more information about configuring WLM for the Data Service server, see [Chapter 5, “Customizing the Data Service server,” on page 85](#).

To create classification rules, select the **6. Classification Rules** option on the Definition Menu in the WLM ISPF application.

Example: The following figure shows an example of creating a classification rule that uses the transaction name qualifier ODAS* to group work into the ODASSC1 service class and ODASRC report class:

```

      Modify Rules for the Subsystem Type          Row 1 to 1 of 1
Command ==> _____ Scroll ==> PAGE
Subsystem Type . : OMVS          Fold qualifier names?   Y   (Y or N)
Description . . . Sample OMVS classification rule

Action codes:   A=After      C=Copy      M=Move      I=Insert rule
                B=Before     D=Delete row R=Repeat    IS=Insert Sub-rule
                                           More ==>

Action   -----Qualifier-----          -----Class-----
Type     Name      Start                Service      Report
-----  ----
----  1 TN        ODAS*   ---          DEFAULTS: OMVSDFSC  OMVSDFRC
                                   ODASSC1      ODASRC

```

For more information about planning for and using WLM, see [z/OS MVS Planning: Workload Management](#) and [z/OS MVS Programming: Workload Management Services](#).

Other Apache Spark attributes

Once you have secured your Apache Spark environment and defined it to WLM, you can further customize your environment. This section describes two ways in which to do that:

- Increasing the parallelism of your Spark applications
- Allowing multiple Spark applications to run simultaneously.

Increasing parallelism attributes

Spark is well-known for its ability to parallelize the processing of data. Often, this involves careful tuning of the attributes within `spark-defaults.conf`. Even then, the Spark Master may decide to execute the program such that it is not consuming all the resources available to it. In such a case, you may want to consider using dynamic allocation.

Dynamic Allocation (see [Appendix D, “Memory and CPU configuration options,” on page 171](#)) allows the Spark Master to start and stop executors as needed and as system resources allow. For example, if an

application has partitioned its data appropriately, it may be possible for the Spark Master to request more executors be added to the application, allowing the application to execute more efficiently and complete earlier.

When using Dynamic Allocation, it is more likely that data will be shuffled among the executors. Toward that end, you must also enable the Spark Shuffle server (also described in [Appendix D, “Memory and CPU configuration options,”](#) on page 171).

This causes the executors to make their shuffle data accessible to other executors by storing it on the Worker. (This may increase the amount of storage needed by the worker.)

You can also read about the shuffle server and dynamic allocation on the Apache Spark website.

Scheduling multiple Apache Spark applications

Any Apache Spark cluster has the ability to run multiple applications. Generally, however, most Spark installations are configured by default to run just one application, using all the resources available memory and CPU. z/OS IzODA Spark has addressed this issue.

Starting with APAR PI03469, Spark 2.2.0.8 and higher includes a new setting (`spark.zos.master.app.alwaysScheduleApps=true`) that enables z/OS IzODA Spark to schedule multiple applications to execute at the same time, even if it seems to use more memory or CPU than the Spark Master sees in the system. In this way, more applications can execute and Workload Manager can better manage resources.

By setting `spark.zos.master.app.alwaysScheduleApps` to true in `spark-defaults.conf`, each application that registers with the master will be allowed to execute in a “virtual” cluster (subject to limitations; more details later). New applications can get all resources on the cluster, independent of the resources already allocated to other applications. This gives z/OS Workload Manager a chance to manage cores and memory on Spark's behalf instead of requiring Spark and the system programmer to tune the available resources for applications to run concurrently.

The number of concurrently running applications can be limited by setting `spark.zos.maxApplications` in `spark-defaults.conf`. The default is 5. `spark.zos.maxApplications` is valid only if `spark.zos.master.app.alwaysScheduleApps` is true.

Note that there are still limits on the memory and cores an application can use. A single application cannot receive more cores than `SPARK_WORKER_CORES` or more memory than `SPARK_WORKER_MEMORY` specifies across all its executors. However, be aware that multiple applications could each receive `SPARK_WORKER_CORES` or `SPARK_WORKER_MEMORY`. Also be aware that the memory and cores allocated to the driver do not introduce limits. Furthermore, the scheduler will still obey the “Memory and CPU configuration options” in [Appendix D, “Memory and CPU configuration options,”](#) on page 171. Resource contention could result if too many applications are concurrently running with too many resources, so it is important to adjust your configuration options accordingly and tailor your WLM policy to ensure proper operation within your system.

Therefore, if all settings other than `spark.zos.maxApplications` and `spark.zos.master.app.alwaysScheduleApps` are default, the total amount of memory the master could allocate is $(\text{spark.zos.maxApplications} * (\text{SPARK_WORKER_MEMORY} + \text{spark.driver.memory}))$ and the total for cores is $(\text{spark.zos.maxApplications} * (\text{SPARK_WORKER_CORES} + \text{spark.driver.cores}))$. It is important to adjust your `SPARK_WORKER_MEMORY` and `SPARK_WORKER_CORES` accordingly.

Note: Spark Drivers only appear in the master Web UI if applications are submitted in cluster-deploy mode. If an application is submitted via client-deploy mode, its driver does not contribute to the used memory and cores on the Web UI.

Note that on the master Web UI, you may observe more memory or cores in use than are available. This is normal behavior in this “virtual cluster” environment.

Chapter 5. Customizing the Data Service server

After you install Mainframe Data Service, customize the server for use.

Before you begin

You must install Data Service and apply all available maintenance before customizing the server. To apply server maintenance, you should acquire available PTFs and apply them to the server so you will have the most current available code for your installation.

Preparing to customize

Before you start to customize IBM Open Data Analytics for z/OS, familiarize yourself with the customization tasks.

The following table describes each significant customization task. Use this checklist to guide you through the customization process.

Table 7. Customization checklist		
Step	Task description	For more information
1	Review the required naming conventions that must be followed when configuring the server subsystem ID and the server initialization member.	See “Required naming conventions” on page 86.
2	Create the server data sets using the <i>hlq.SAZKCNTL</i> members AZKDFDIV, AZKGNMP1 and AZKEXSWI.	See “Creating server data sets” on page 86.
3	Set up the security application to use with the server using one of the following <i>hlq.SAZKCNTL</i> members: AZKRAVDB, AZKA2VDB, AZKTSVDB.	See “Defining security authorizations” on page 87.
4	Configure Workload Manager (WLM) for optimum performance of the server.	See “Configuring Workload Manager (WLM)” on page 87.
5	APF-authorize the product LOAD library data sets.	See “APF-authorizing LOAD library data sets” on page 88.
6	Create a copy of the product libraries (optional).	See “Copying target libraries” on page 88.
7	Configure the server to support DBCS (optional).	See “Configuring support for code pages and DBCS” on page 88.
8	Create the Global Registry log stream (optional).	See “Creating the Global Registry log stream” on page 89.
9	Customize the server to access your data sources in <i>hlq.SAZKEXEC</i> (AZKSIN00).	See “Customizing the server initialization member” on page 89. Then, see "Configuring access to data sources" in the <i>Solutions Guide</i> for the specific types of data sources the server should access.
10	Configure the started task JCL located in <i>hlq.SAZKCNTL</i> (AZK1PROC) before you can start the server.	See “Configuring the started task JCL” on page 90.

Table 7. Customization checklist (continued)		
Step	Task description	For more information
11	Configure the CLIST that invokes the ISPF panels by using <i>hlq.SAZKEXEC(AZK)</i> .	See “Configuring the ISPF application” on page 91 .
12	Verify the installation by creating a virtual table and accessing its underlying VSAM file (optional).	See Chapter 11, “Verifying the Data Service server installation,” on page 121 .

Required naming conventions

You must follow the Data Service server naming conventions when configuring the server subsystem ID and the server initialization member.

The server subsystem name must follow the pattern *xZKy*, where *x* is any alphabetic character A - Z and *y* is any alphanumeric character A-Z or 0-9.

Depending on what you name the server subsystem, the server initialization member must follow the same naming convention as the server subsystem name, for example, *xZKyIN00*.

Note: The default server naming conventions used throughout this guide are *AZKS* for the server subsystem name and *AZKSIN00* for the server initialization member.

Creating server data sets

The *AZKDFDIV* and *AZKGNMP1* members of *hlq.SAZKCNTL* create data sets for the Trace Browse, the global variable checkpoint, and the data-mapping facility (DMF) that are used by the Data Service server. The *AZKGNMP1* member also copies distributed data sets into user-modifiable data sets. The *AZKEXSWI* member builds the Web interface objects.

Procedure

1. Customize the *AZKDFDIV* member in *hlq.SAZKCNTL* to meet your requirements. The *AZKDFDIV* member contains comments that describe how to customize the variables.
2. Submit the *AZKDFDIV* member.
3. Customize the *AZKGNMP1* member in *hlq.SAZKCNTL* to meet your requirements. The *AZKGNMP1* member contains comments that describe how to customize the variables.
4. Submit the *AZKGNMP1* member.

Note: The map data set created in this step should be the first concatenated data set in the DD statement *AZKMAPP* located in the server started task. See *hlq.SAZKCNTL(AZK1PROC)*. The user and server should have read and write permissions to this data set. The system-provided data set (*hlq.SAZKSMAP*) should be the last data set in the *AZKMAPP* concatenation. The user and server should only have read access to the data set. The administrator will need read and write permissions.

5. Customize the *AZKEXSWI* member in *hlq.SAZKCNTL* to meet your requirements. The *AZKEXSWI* member contains comments that describe how to customize the variables.

Note: The data set named on the *RECEIVE* command in the *AZKEXSWI* member is later used in the server initialization member *AZKSIN00* for the **SWICNTLDSN** parameter definition, as follows:

```
swiobj = SHLQ2||".SAZK0BJ"
"MODIFY PARM NAME(SWICNTLDSN) VALUE("||swiobj||")"
```

6. Submit the *AZKEXSWI* member.

Defining security authorizations

To use an external security product, such as RACF, ACF2, or Top Secret, define the started task name to the security product and authorize the data set.

Procedure

To define the server and other required permissions for your security product, customize the appropriate security option located in the *hlq.SAZKCNTL* library, and submit the job:

- AZKRAVDB is for IBM Resource Access Control Facility (RACF) security.
- AZKA2VDB is for CA ACF2 (Access Control Facility) security.
- AZKTSVDB is for CA Top Secret Security (TSS).

Results

Make sure that your z/OS Security Administrator reviews the security definitions. You might need to change definitions to meet requirements at your site.

Configuring Workload Manager (WLM)

To get optimum performance from the server, define the server to WLM. The Data Service server should be prioritized slightly below the data provider in your WLM environment. It is not sufficient to simply add the STC to a WLM service class as the server will create independent enclaves for each connection.

About this task

The server should be configured to use a medium to high performing WLM velocity goal as its default service class.

Procedure

1. Create a WLM Classification rule.
 - a) Go to the WLM ISPF application, and select option **6** (Classification Rules).
 - b) Select option **1** to Create.
 - c) Set the Subsystem Type to AZK, and provide an optional description.
 - d) Under the Class/Service Column next to DEFAULTS, set the desired default service class name. If a desired service class does not exist, then create one using option 4 (Service Classes) under the **Primary WLM** menu. Press enter and PF3 to save.
2. Define the Data Service started task AZK1PROC to a WLM service class.
 - a) Go to the WLM ISPF application, and select option 6 (Classification Rules).
 - b) For the STC WLM-subsystem type, select **Modify**.
 - c) Add an entry for AZK1PROC.
 - d) Add an appropriate service class for the started task and define it relative to existing workload resource management objectives.
 - e) Add a unique Report class for the started task.
3. Activate the new WLM policy definition.

APF-authorizing LOAD library data sets

You must authorize for APF (Authorized Program Facility) all LOAD library data sets allocated to the Data Service server.

About this task

All LOAD library data sets allocated to the Data Service server in the server started task JCL must be APF-authorized.

These LOAD library data sets are allocated to the following ddnames:

- STEPLIB

You must authorize the LOAD library AZK.SAZKLOAD.

- AZKRPCLB

You must authorize the LOAD library AZK.SAZKRPC.

If any data sets allocated to these ddnames are not APF-authorized, the Data Service server will issue the error message AZK0051S during startup identifying the ddname and data set name of each unauthorized library. Startup processing will discontinue and the server will shut down.

Procedure

The APF authorize should be done dynamically using the SETPROG APF command, and then made permanent for the next IPL (initial program load) by updating the appropriate system PARMLIB member.

Copying target libraries

It is recommended that copies be made of the target libraries to preserve any prior customization, as applying new maintenance often replaces existing PDS members.

Configuring support for code pages and DBCS

You can configure the server to support Japanese code pages and double-byte character sets (DBCS).

About this task

To support different code pages and double-byte character sets, you must manually customize the server initialization member.

Procedure

1. Locate the server configuration member. The server initialization member is shipped in data set member *hlq.SAZKEXEC(AZKSIN00)* and may have been copied to a new data set for customization in the step [“Copying target libraries”](#) on page 88.
2. In the member, locate the DEFINE DATABASE statement for your subsystem, and verify that the CCSID value is set correctly for the subsystem.
3. Locate the comment Set CCSID for non-DB2 data, as shown in the following example:

```
/*-----*/
/* Set CCSID for non-DB2 data          */
/*-----*/

if DoThis then
do
    "MODIFY PARM NAME(SQLENGDFLTCCSID)    VALUE(1047) "
```

4. Change DontDoThis to DoThis to enable the parameters.
5. Update the following parameter:

Parameter	Description	Valid values
SQLENGDFLTCCSID	Specifies the CCSID to use for SQL engine tables. All host tables except for Db2 are assumed to be stored in this CCSID. Where possible, this CCSID should match the client CCSID used when connecting.	CCSID value Sample values: <ul style="list-style-type: none"> • 1047 (LATIN OPEN SYS EB) • 931 (JAPAN MIX EBCDIC) • 1390 (JAPAN MIX EBCDIC)

Creating the Global Registry log stream

You can manually create the Global Registry log stream instead of having the server dynamically create it.

Before you begin

Confirm the following prerequisites:

- The GLBLREGISTRYEHLQ parameter is not set to NOLOGSTREAMS in the server initialization file AZKSIN00.
- The server is not running. This customization step should be performed before any instance of the server is started unless the NOLOGSTREAMS parameter is specified.

About this task

To manually create the Global Registry log stream, you must customize and run either the AZKLSGBL or AZKLSGBO customization job before the server is started.

Procedure

Customize and run one of the following jobs before the server is started:

- If a coupling facility is used, use batch job *hlq.SAZKCNTL(AZKLSGBL)*.
- If a coupling facility does not exist within your SYSPLEX environment, then create the DASD-only log stream structure using the batch job *hlq.SAZKCNTL(AZKLSGBO)*.

Customizing the server initialization member

The server initialization member AZKSIN00 is a REXX program that you use to set product parameters and define links and databases. You must customize the server initialization member for your installation environment.

About this task

The server initialization member is shipped in data set member *hlq.SAZKEXEC(AZKSIN00)* and may have been copied to a new data set for customization in the step [“Copying target libraries”](#) on page 88.

As you go through the installation, you accept or set parameter values in the server initialization member.

If you are installing the server for the first time, it is recommended that all the default values be accepted. You can change the values as needed later.

If you are installing a new version of the server over a previous version, the previous server member might contain parameter values that you modified to meet specific requirements in your environment. In this case, you should review the initialization member for the previous version for any customizations that need to be made to the initialization member for the new version.

Procedure

1. Find the line that contains "SHLQ1" and provide your own high-level qualifier to define the ISPF data sets. For example: "SHLQ1=AZK"
2. If you created copies of your target libraries to preserve customizations, find the line that contains "SHLQ2" and provide your own high-level qualifier to define the Event Facility (SEF) data sets. Ensure that the HLQ results in proper data set references for these features.
For example: "SHLQ2=AZK.AZKS". If you did not create copies of the target libraries, then "SHLQ2" should contain the same value as "SHLQ1".
3. Review the following default values for the TCP/IP parameters and change the values as necessary. The following example shows the section of the initialization member in which to make the changes:

```
"MODIFY PARM NAME(OEPORTNUMBER) VALUE(1200)"  
"MODIFY PARM NAME(WSOEPORT) VALUE(1201)"  
"MODIFY PARM NAME(TRACEOERW) VALUE(YES)"  
"MODIFY PARM NAME(OEKEEPALIVETIME) VALUE(30)"  
"MODIFY PARM NAME(PARALLELIO) VALUE(YES)"  
"MODIFY PARM NAME(OEPIOPORTNUMBER) VALUE(1204)"
```

Configuring the started task JCL

To configure the started task JCL, modify the AZK1PROC (subsystem default ID) member that is in the *hlq.SAZKCNTL* library.

About this task

The AZK1PROC member contains the JCL procedure that is required to run the main address space (started task).

Procedure

1. Add the HLQ name of the libraries to the *hlq* parameter.
This parameter sets the server data set allocations to the correct data set names.
2. Confirm that the SYSEXEC DD statement allocates the correct data set name that contains the customized server initialization member AZKSIN00. This data set was created in job AZKGNMP1 previously in the step ["Creating server data sets" on page 86](#). The default name is *hlq.SAZKEXEC(AZKSIN00)*.
3. Ensure that the DD AZKMAPP concatenation points to the *hlq.SAZKMAP* data set created in the previous installation job AZKGNMP1. This data set should be first in the concatenation and is used for storing user-defined virtual table maps. The *hlq.SAZKMAP* data set, which contains the default virtual table maps that are part of the product distribution, should be placed last.
4. The server runs as a z/OS started task. Under normal circumstances, the server starts at system startup and stops before the system shuts down. To start the server on demand, use the following console command:

```
S AZKS
```

where AZKS is the subsystem name of the server instance you defined.

Note: If you use a procedure name other than the SSID provided in the example, then you issue the start command using that procedure name.

5. If you use an automation package to start the system, associate the **START** command with the VTAM initialization complete message (IST020I), the TCP/IP initialization complete message (EZB6473I), or both messages.
6. To verify that the startup is successful, look for the following entries in the server Job Entry Subsystem (JES) log.

```
SD74391I OE stack binding port 1200 to IP address 0.0.0.0
SD74391I OE stack binding port 1201 to IP address 0.0.0.0
SD74391I OE stack binding port 1202 to IP address 0.0.0.0
```

What to do next

If you want to stop the server, issue the following console command:

P AZKS

If you issue a **CANCEL** command, all available connections terminate with an abend, and the server shuts down immediately.

Configuring the ISPF application

Configure and invoke the ISPF application.

Before you begin

The server must be started before you can invoke the ISPF application.

Procedure

1. Edit the *hlq*.SAZKEXEC (AZK) member, and replace the data set name in the following statement with the data set name that you chose for the *hlq*.SAZKLOAD library:

```
llib='hlq.SAZKLOAD'
```

2. Copy the *hlq*.SAZKEXEC (AZK) member to a data set that is allocated to the SYSPROC allocation for all TSO users.

Before starting the ISPF application, you must configure and start your server. See [“Configuring the started task JCL”](#) on page 90

When the server starts, the ISPF data sets are dynamically allocated.

3. To invoke the ISPF application, go to the ISPF command shell and enter the following command:
EX 'hlq.SAZKEXEC(AZK)' 'SUB(AZKS)'

Where:

- *hlq* is the high level qualifier.
- AZKS is the subsystem name of the server instance you defined.

All ISPF clients will communicate with the specified subsystem.

Configuring generation data set retrieval

You can configure the server to read only a subset of generation data sets (GDSs) by activating a VTB rule.

About this task

To read only a subset of generation data sets in a generation data group (GDG), you must enable virtual rule AZKGDGS1 and use the prefix GDG__ in your SQL statement.

A VTB rule is provided that allows a subset of the GDG to be read. VTB rule AZKGDGS1 is invoked by the SEF every time a table with the prefix GDG__ is found in the SQL statement.

The table name in the SQL statement must be of the form:

```
GDG__NumGens_RelGen_MapName
```

Where:

- GDG__ is a constant indicating a generation data set request.

- *NumGens* is a required number 0 through 999 indicating the number of generations to read.
- *RelGen* is an optional number 0 through 999 indicating the relative generation at which to start reading. A value of 0 is equivalent to a suffix of (0) in a JCL allocation; a value of 1 is equivalent to (-1), and so on.
- *MapName* is the table defined in the map data set.

For example, the following request will result in generations HLQ.GDG.STAFF(-3) through HLQ.GDG.STAFF(-6) being retrieved:

```
SELECT * FROM GDG__4_3_STAFF
```

Where the STAFF table specifies a base data set name of HLQ.GDG.STAFF. In other words, with this request, four generations will be read in descending generation order beginning with relative generation 3 (that is, generations 3, 4, 5, and 6).

Use the procedure in this task to enable sample rule AZKGDGS1.

Additional details:

When a request is made to allocate a data set, it will first be determined if the data set name represents a GDG base name. If so, a CSI lookup call will be made to return the associated GDS data set names. If a VTB rule does not specify the number of generations to read and MapReduce is disabled, or if there is a single generation, the GDG will be allocated using its base data set name, and normal system concatenation of generation data sets will occur. If MapReduce is enabled and there are multiple active generation data sets, a number of I/O processing tasks will be created. The number of I/O tasks is determined as follows:

1. If VPD is in use, the number of VPD I/O threads specified.
2. If MRC is in use, the number of active Client threads defined in the MRC request.
3. If neither VPD nor MRC is in use, the number of I/O threads will be equal to the lesser of the following:
 - The number of active generation data sets in the GDG
 - The number of generations requested by a VTB rule
 - The number of MapReduce tasks specified in the ACIMAPREDUCETASKS configuration

When the number of I/O tasks is equal to or less than the number of generation data sets, each task will read one or more complete data sets. When the number of I/O tasks exceeds the number of generation data sets, some tasks will be idle.

Procedure

1. Customize the server configuration member (AZKSIN00) to enable virtual table rule events by configuring the SEFVTBEVENTS parameter in the member, as follows:

```
"MODIFY PARM NAME(SEFVTBEVENTS) VALUE(YES) "
```

2. Access the VTB rules, as follows:
 - a) In the IBM Open Data Analytics for z/OS - Primary Option Menu, specify option E, **Rules Mgmt.**
 - b) Specify option 2, **SEF Rule Management.**
 - c) Enter VTB for **Display Only the Ruleset Named.**
3. Enable the rule by specifying E next to AZKGDGS1 and pressing Enter.
4. Set the rule to Auto-enable by specifying A next to AZKGDGS1 and pressing Enter.
Setting a rule to Auto-enable activates the rule automatically when the server is re-started.

Configuring delimited data support

To be able to process delimited data using virtual tables, you must configure a virtual table rule to activate delimited data processing and optionally define delimiter values.

About this task

Data Service provides the ability to process delimited data from files, MQ data, and log streams using virtual tables mapped to MQ or z/OS files. The most common form of delimited data is comma separate value files (.csv).

When delimited data processing is activated, processing occurs in column order, so the delimited data must include a value for each column in the map in the correct order to prevent errors. Data conversion errors will occur if the delimited data is not compatible with the host types of the columns. If conversion fails, diagnostic information related to the error is automatically logged for troubleshooting problems.

Delimited processing is supported through virtual table rules only. Using virtual table rule options, you can enable delimited data processing, set column and string delimiter values, and control header record processing.

A sample rule, AZKMDDLML, is provided that documents these settings. Use the following procedure to configure the sample rule.

Procedure

1. Customize the server configuration member (AZKSIN00) to enable virtual table rule events by configuring the SEFVTBEVENTS parameter in the member, as follows:

```
"MODIFY PARM NAME(SEFVTBEVENTS) VALUE(YES) "
```

2. Access the VTB rules, as follows:
 - a) In the IBM Open Data Analytics for z/OS - Primary Option Menu, specify option E, **Rules Mgmt.**
 - b) Specify option 2, **SEF Rule Management.**
 - c) Enter VTB for **Display Only the Ruleset Named.**
3. Customize the AZKMDDLML rule, as follows:
 - a) Specify S next to AZKMDDLML to edit the rule.
 - b) Find the **vtb.optbdlcv** variable and set to 1 to activate delimited processing for a map.
 - c) Update additional rule options as needed. The following table describes the VTB rule options that support delimited data processing.

VTB variable	Description
vtb.optbdlcv	Set to 1 to activate delimited processing for a map.
vtb.optbdlco	Set the column delimiter. The default value is the comma character (.). For example, if you use the colon character (:) as the column delimiter, specify <code>vtb.optbdlco = ':'</code> .
vtb.optbdlch	Set the character field or string delimiter. The default value is the quotation mark character ("). For example, if you use the hash character (#) as the string delimiter, specify <code>vtb.optbdlch = '#'</code> .
vtb.optbdlhr	Set to 1 to identify and remove the header record containing column names. If specified without a header prefix, the system compares the first token in each line to the first column name in the table to recognize and discard the header. The default is no header checking.

VTB variable	Description
vtb.optbd1hp	<p>Define prefix data that identifies the beginning of a header line to be discarded. The specified value can contain a maximum of 32 bytes. This value is compared to the beginning of each delimited line of data before any tokenization is performed. For example, <code>vtb.optbd1hp = '"NAME","ADDRESS"'</code>.</p> <p>Note: If an <code>optbd1hp</code> value is defined, it supersedes any <code>optbd1hr</code> setting and the <code>optbd1hr</code> value is ignored.</p>

- d) Save your changes and exit the editor.
4. Enable the rule by specifying E next to AZKMDDLm and pressing Enter.
 5. Set the rule to Auto-enable by specifying A next to AZKMDDLm and pressing Enter.
- Setting a rule to Auto-enable activates the rule automatically when the server is re-started.

Chapter 6. Installing the Data Service Studio

The Data Service Studio is an Eclipse-based user interface that allows you to create and manage metadata on the Data Service server that is required to provide access to your mainframe and non-mainframe data.

Before you begin

Before installing the Data Service Studio, verify that all installation prerequisites are met:

System component	Requirement
Permissions	You have appropriate user logon credentials and user privileges on your client system to install the Data Service Studio. For example, to install the studio on Windows, you need administrator authority; ensure that your user profile has the appropriate privileges to write to the target system location.
Supported operating systems	Windows 7, 8, 10 Linux – Red Hat Enterprise Linux 6.7 or higher; Ubuntu 16 or higher macOS (Sierra)
System memory	A minimum of 4 GB is recommended.
Hard disk space	A minimum of 1 GB is recommended.
Software	Installing the Data Service Studio as a plug-in to your existing Eclipse environment requires Eclipse Kepler 4.3 (or higher) and Java 1.7 or Java 1.8.

About this task

You can choose to install the Data Service Studio software in a new Eclipse environment (a *full install*) or as a plug-in within an existing Eclipse environment (a *plug-in install*):

Full install

A *full install* installs the Data Service Studio software in a new Eclipse environment on Windows. This method includes the installation of Windows Eclipse (64-bit), JRE 1.7, and the Data Service plug-in. This installation method is recommended for Windows 64-bit users who are installing the studio for the first time.

Plug-in install

A *plug-in install* installs only the Data Service plug-in. This installation method is recommended for the following users:

- Users on all supported platforms other than Windows 64-bit
- Existing Eclipse users that want to reuse their own Java runtime and Eclipse installation
- Users wanting to upgrade their existing Data Service Studio installation with a newer version of the Data Service plug-in

Procedure

1. From the z/OS mainframe, transfer the installation member *hlq.SAZKBIN(AZKBIN1)* to a folder on your workstation using the File Transfer Protocol (FTP) in binary mode.
2. Rename the file to *azk-studio.zip*.
3. Create a new installation folder for the Data Service Studio.
4. Double-click the *azk-studio.zip*, and then extract the contents to the installation folder.

5. In the installation folder, navigate to the `studio\install` folder that was created, and then select one of the following installation methods:
 - To perform a *full install*, installing the Data Service Studio software in a new Eclipse environment, complete the following steps:
 - a. In the `studio\install` folder, run the `setup.bat` script.
 - b. After the installation completes, launch the Data Service Studio using the shortcut created on the desktop or in the **Start** menu.
 - To perform a *plug-in install*, installing the Data Service Studio software as a plug-in to an existing Eclipse environment, complete the following steps:
 - a. From your Eclipse application, click **Help > Install New Software**, and then click **Add**.
 - b. On the Add Repository dialog box, click **Archive**.
 - c. Locate the `mds.zip` file in the `studio\install` folder, and then click **Open**.
 - d. Enter the software file name, a name for the repository, and then click **OK**.
 - e. Select the check box next to the file, and then click **Next**.
 - f. Complete the remaining installation wizard steps, and then restart Eclipse when prompted.
6. To begin using Data Service Studio, open the Data Service perspective using the menu option **Window > Open Perspective**.

Verifying the studio installation

Verify that you can connect from the Data Service Studio to the server and browse metadata on the server.

Procedure

1. On the **Server** tab, click **Set Server**.
2. Provide information for the following fields and click **OK**:

Field	Description
Host	The z/OS LPAR name on which the Data Service server is running.
Port	The JDBC port number that the server is using. During customization, the port number is specified in the server configuration file; the parameter name is OEPORNUMBER . To locate this number, use SDSF on the mainframe to browse the server JOB output and search for OEPORNUMBER.
Userid	The user ID that the server will use to authenticate the connection.
User Password	The password that corresponds to the user ID being used to connect to the server.

The **Server** tab displays the new server connection. You can now browse the server metadata and configure the interfaces for the solutions that you want to use.

Installing the JDBC driver

Java-based applications and tools use the JDBC driver to access data that is made available through Data Service.

About this task

The JDBC driver is a Type 4 driver that is written in Java and implements the network protocol for the Data Service.

The JDBC driver requires Java 1.7 or higher and is supplied as a ZIP archive file that contains the following components:

- dv-jdbc-[version #].jar (the driver core implementation file)
- log4j-api-[version #].jar (logging framework API file)
- log4j-core-[version #].jar (logging framework implementation file)
- log4j2.xml (sample logging configuration file)

Procedure

1. From the mainframe, transfer the driver installation member *hlq.SAZKBIN(AZKBIN2)* to your development workstation using the File Transfer Protocol (FTP) in binary mode.
2. Rename the file to JDBCdriver.zip.
3. On your development workstation, create a directory, and then extract the contents of the JDBCdriver.zip archive into that directory.
4. To use the drivers with Data Service instead of the JDBC driver shipped with the studio, create a directory on the mainframe and FTP the driver files in binary mode to that directory. For example: <mount_point>/DSDriver. The <mount_point>/DSDriver path is used with the spark-submit command.

What to do next

For details about the JDBC driver, see the *Solutions Guide*.

Installing the Python dsdbc module

You can access z/OS data from Python applications using Mainframe Data Service and the IzODA Anaconda component.

Querying Data Service virtual tables from Python applications requires the IzODA Anaconda component and use of the Python dsdbc module. The Python dsdbc module is an implementation of the Python DB 2.0 API (<https://www.python.org/dev/peps/pep-0249/>) which enables Python developers on z/OS to use a simple, portable framework to access the Data Service. The dsdbc module is available with the IzODA Anaconda component.

Note: When using the IzODA Anaconda component, Python applications do not use the JDBC driver for data access; the non-Java based Python dsdbc module is used for data access.

To install and configure IzODA Anaconda, see [Chapter 9, “Customizing Anaconda,”](#) on page 113. No additional modifications to the Data Service server are required.

Chapter 7. Installing the JDBC Gateway

The *JDBC Gateway* is a Data Service distributed application server that allows direct connectivity to JDBC data sources. Install the JDBC Gateway to connect directly to JDBC data sources.

Before you begin

Before installing the JDBC Gateway, review the following points:

- For an overview of the JDBC Gateway solution, see "JDBC Gateway" in the *Solutions Guide*.
- The following terminology is used in the installation procedure:
 - *JDBC Gateway server*. The server is the backend component that allows communication with the Data Service server.
 - *JDBC Gateway administrative console*. The administrative console is the front-end web component that you use to configure your data sources. Only a single user (web client) can access the JDBC Gateway administrative console at a time. When installing the JDBC Gateway, you must specify a specific user ID for this purpose. This user ID is an internal application ID that allows access to the web user interface.
 - *Port for the Web UI*. This port will be used to access the Web-based administrative console and is specified during the installation procedure.

Note: The JDBC Gateway also uses another port to listen for incoming DRDA requests. This DRDA listener port is set later when configuring the JDBC Gateway.

- Before installing the JDBC Gateway, verify that all installation requirements are met, as follows:

System component	Requirement
Permissions	You have appropriate user logon credentials and user privileges on your client system to install the JDBC Gateway. For example, to install and deploy the JDBC Gateway on Windows, you may need to run with administrator privileges depending on the target location.
Supported platforms	The JDBC Gateway is a pure Java application and therefore can be deployed on any platform that supports Java 8 or higher.
System memory	Minimum of 1 GB
Hard disk space	Minimum of 500 MB
Software	<ul style="list-style-type: none">– Java 8 is required to install and deploy JDBC Gateway.– One of the following web browsers (with JavaScript support enabled) must be used to access the JDBC Gateway administrative console:<ul style="list-style-type: none">- Google Chrome browser V50.0.2661.102 or later- Mozilla Firefox V47.0.1 or later- Microsoft Edge V25.10586.0.0 or later- Microsoft Internet Explorer V10 or later- Apple Safari browser V9.0.3 or later– Database connectivity requires an appropriate JDBC driver for each type of data source that is accessed.

About this task

Use the following procedure to install the JDBC Gateway. This installation installs the JDBC Gateway server and administrative console.

During the installation, you must specify a user ID to be used for the JDBC Gateway administrative console. When using the JDBC Gateway administrative console, only a single user can access the administrative console at a time.

As part of the installation, the following actions occur:

- The `jgate.properties` file is created, which contains the site-specific settings.
- Start and stop scripts appropriate to the platform are created. The installer creates `cmd` scripts if you are running on Windows and `sh` scripts if you are running on Unix or Linux.

Considerations for USS installation: For installation in USS, it is recommended that you define the following environment variables:

```
export IBM_JAVA_OPTIONS="-Dfile.encoding=ISO8859-1"
export _BPXK_AUTOCVT=ON
```

When the installer generates start and stop scripts, the following actions occur depending on these variables:

- If you have not set the recommended environment variables, the scripts will be generated in EBCDIC. You can run the gateway as normal for Unix using the following command: `sh startServer.sh`
- If you set the `IBM_JAVA_OPTIONS` variable, the scripts will be generated in ASCII, and you will need to use the following command: `ctag -tc ISO8859-1 <file>`. (Tagging in USS basically means `_BPXK_AUTOCVT` must be ON if you want to edit or execute the script in the shell.)

Files generated by the JDBC Gateway, such as log files and the `jgate.properties` file, will be generated in ASCII regardless of the aforementioned environment variable settings (except for `jetty.out`, which is in EBCDIC). In order to browse these files natively in USS, you must use the `ctag` command and set `_BPXK_AUTOCVT=ON`.

Procedure

1. From the z/OS mainframe, transfer the installation member `hlq.SAZKBIN(AZKBINJ)` to your workstation using the File Transfer Protocol (FTP) in binary mode.
2. Rename the file to `jdbc-gateway.zip`.
3. On your host machine, create a directory to host the JDBC Gateway, and then extract the contents of the installation file into that directory.

The extracted contents will include the `JDBCGatewaySetup11.jar` file.

Note: If your host machine does not have an unzip utility, extract the contents of the installation file on a Windows workstation and copy the `JDBCGatewaySetup11.jar` file to the host machine.

4. At a command prompt in the directory, run the following command:

```
java -jar JDBCGatewaySetup11.jar
```

The installer launches.

5. Enter the following information at the prompts:

Prompt	Description
You are about to install JDBC Gateway. Do you want to proceed? (Y/n)	Enter Y to continue with the installation, or enter n to cancel the installation.
Specify the installation directory (<i>local directory</i> \JDBCGateway):	Enter the path of the directory where to install the application, or press Enter to use the default value as indicated.
Set login for JDBC Gateway admin Web page (admin):	Enter the user ID to be used for the JDBC Gateway administrative console, or press Enter to use the default value admin.

Prompt	Description
Set password for JDBC Gateway admin Web page:	Enter the password for the administrative console user ID. The password must be at least five characters in length.
Confirm your password:	Re-enter the password for the administrative console user ID.
Set port for the Web UI (8080):	Enter the number of an available TCP/IP port for the application, or press Enter to use the default value 8080. This port number will be used when launching the JDBC Gateway administrative console.
Installation completed. Do you want to start the JDBC Gateway now? (Y/n)	Enter Y to start the server, or enter n to exit the installation. Note: If you enter Y, the server starts within the same shell.

Results

The JDBC Gateway has been installed and is ready for use. Information about the activity of the JDBC Gateway is available in the Java Console and in the log files.

If you specified to start the server, information about the startup process is displayed.

What to do next

- To start the server, see “Starting the JDBC Gateway server” on page 101.
- To launch the administrative console, see [“Launching the JDBC Gateway administrative console” on page 102](#).

Starting the JDBC Gateway server

Start the JDBC Gateway server so that you can connect directly to JDBC data sources.

Before you begin

The JDBC Gateway must be installed. See [Chapter 7, “Installing the JDBC Gateway,” on page 99](#).

About this task

Use the following procedure to start the JDBC Gateway server.

Information about the startup and additional activity of the JDBC Gateway is available in the Java Console and in the following log file:

```
home_dir_for_user_profile\Application Data\IBM\JDBC Gateway\log\jetty.out
```

Procedure

1. At a command prompt in the JDBC Gateway installation directory, run one of the following commands:

- For Windows: `startServer`
- For Linux or Unix: `sh startServer.sh`

Information about the startup process is displayed using the following format:

```
Using settings file: home_dir_for_user_profile\Application Data\IBM\JDBC Gateway\Settings\jgate.properties
Server is starting. It will be available on: http://localhost:port
```

```
Server process ID: processID  
See home_dir_for_user_profile\Application Data\IBM\JDBC Gateway\log\jetty.out for server status information.
```

2. Wait for the JDBC Gateway server startup process to complete, which is indicated by the following message in the `jetty.out` log file:

```
date time : JGATE Server started and ready to accept connections on port port_number
```

3. Optional: To stop the JDBC Gateway server, run the following command in the JDBC Gateway installation directory:
 - For Windows: `stopServer`
 - For Linux or Unix: `sh stopServer.sh`

Results

The JDBC Gateway server has been started and is ready for use. Information about the activity of the JDBC Gateway is available in the Java Console and in the log files.

What to do next

Start the JDBC Gateway administrative console. See [“Launching the JDBC Gateway administrative console” on page 102.](#)

Launching the JDBC Gateway administrative console

Launch the JDBC Gateway administrative console so that you can configure connections to JDBC data sources.

Before you begin

The JDBC Gateway server must be installed and active. See [Chapter 7, “Installing the JDBC Gateway,” on page 99](#) and [“Starting the JDBC Gateway server” on page 101.](#)

About this task

Use the following procedure to start the JDBC Gateway administrative console.

Only a single user (web client) can access the JDBC Gateway administrative console at a time.

Note: The JDBC Gateway does not require an external web application server. It contains its own Jetty web application server.

Procedure

1. In a web browser, launch the JDBC Gateway administrative console using the following URL:

```
http://server:port
```

where:

- *server* is the machine name or address where the JDBC Gateway server is running
 - *port* is the port specified during the installation
2. Enter the **Username** and **Password** specified during installation.

The JDBC Gateway administrative console launches.

Results

The JDBC Gateway administrative console is running and ready for use. Information about the activity of the JDBC Gateway is available in the Java Console and in the log files.

What to do next

Configure access to data sources in the JDBC Gateway and the Data Service server. See "Configuring access to data sources using the JDBC Gateway" in the *Solutions Guide*.

Chapter 8. z/OS IzODA Livy Installation and Customization

Before you can use z/OS IzODA Livy, you must customize your environment for it and its dependent products.

Before you begin

Follow the instructions in Chapter 3, “Installing IBM Open Data Analytics for z/OS,” on page 11 to install Open Data Analytics for z/OS on your system.

About this task

Once installed, the default program location for z/OS IzODA Livy is `/usr/lpp/IBM/izoda/anaconda/extras/apache-livy`.

Complete the tasks in this topic to customize your environment for z/OS Livy. You will need access to a TSO session and an OMVS session (preferably through a Putty terminal).

About z/OS IzODA Livy

The Livy server is a REST service used in conjunction with Spark that allows users to submit Spark jobs without having the Spark client installed. It provides two modes of job submission for the end users:

- Batch mode: For submitting Spark application files such as Scala JARs and Python `.py` files. Conceptually this is equivalent to users using `spark-submit` for application submission.
- Session mode: For submitting Scala or Python application code interactively. Conceptually this is equivalent to users using interactive shell such as `spark-shell` or `pyspark` shell.

After the Spark job is received, the Livy server submits the application to the Spark cluster on behalf of the user.

Installing z/OS IzODA Livy

Complete this task to install the z/OS IzODA Livy package onto your system using Anaconda. IBM recommends the System Programmer to perform the install of Livy into the Anaconda base environment.

1. Determine the Anaconda environment that you will be using for the install. The System Programmer will be using the base environment (already created).
 - If you are not the System Programmer, IBM recommends first creating your own conda environment for the install. For example:

```
conda create -n myenv python
```

2. Activate your environment. The System Programmer will use the base environment.

```
conda activate myenv
```

3. Enter the following:

```
conda install apache-livy
```

Setting up a user ID for use with z/OS IzODA Livy

Complete this task to set up a user ID for use with z/OS IzODA Livy. For this task, you can either create a new user ID to use for z/OS IzODA Livy, or you can use an existing user ID.

Choose or create an appropriate user ID for use with z/OS IzODA Livy. Specifically, this is the user ID under which the Livy server is started, known as the Livy server user ID in this documentation.

The Livy server will be submitting Spark application to the Spark cluster on behalf of the Livy end users. Therefore, the Livy server user ID must have the appropriate authorities for submitting Spark application to the Spark cluster as any other Spark end user. If z/OS Spark client authentication is enabled, refer to [“Configuring z/OS Spark client authentication” on page 41](#) for details about setting up the Spark end user ID with the appropriate authorities.

Before starting the Livy server, you must have the appropriate authorities to either access or create (if they do not already exist) the directories that are specified by the following environment variables, or the defaults taken when not specified:

```
LIVY_CONF_DIR
LIVY_LOG_DIR
LIVY_PID_DIR
```

IBM also recommends you set the LIVY_HOME environment variable to the Livy installation directory.

Customizing z/OS IzODA Livy

Before you can use z/OS IzODA Livy, you must customize your environment for it and its dependent products.

Configuring the z/OS IzODA Livy working directories

Complete this task to configure z/OS IzODA Livy to run using your desired configurations. The following table lists some of the working directories that z/OS IzODA Livy uses:

Table 8. Livy working directories			
Directory contents	Default location	Environment variable	Suggested new directory
Log files	\$LIVY_HOME/logs	\$LIVY_LOG_DIR	Under /var, such as /var/livy/logs
Configuration files	\$LIVY_HOME/conf	\$LIVY_CONF_DIR	Under /var, such as /var/livy/conf
PID files	/tmp	\$LIVY_PID_DIR	Under /tmp, such as /tmp/livy/pid

1. Follow your file system conventions and create new working directories for z/OS IzODA Livy.
2. Ensure the Livy server user ID (created in [Chapter 8, “z/OS IzODA Livy Installation and Customization,” on page 105](#)) has read/write access to the newly created working directories.
3. Ensure that these directories, specifically \$LIVY_LOG_DIR, get cleaned regularly.

Creating the Livy configuration files from template

This task creates the livy.conf, livy-env.sh and log4j.properties configuration files in the Livy server user ID's \$LIVY_CONF_DIR directory.

Create the files by using the template files located in the \$LIVY_HOME/conf directory. For example:

```
cp $LIVY_HOME/conf/livy-env.sh.template $LIVY_CONF_DIR/livy-env.sh
cp $LIVY_HOME/conf/livy.conf.template $LIVY_CONF_DIR/livy.conf
cp $LIVY_HOME/conf/log4j.properties.template $LIVY_CONF_DIR/log4j.properties
```

Updating the Livy configuration files from template

You will need to configure the Livy server by using the `livy.conf` and `livy-env.sh` configuration files located in `$LIVY_CONF_DIR` directory.

1. The following parameters should be set in the `livy.conf` configuration file:

- a. Set `livy.spark.master` to the Master URL of your Spark cluster. For example:

```
livy.spark.master = spark://master:7077
```

- b. Set `livy.spark.deploy-mode` to the deploy mode that your Spark cluster accepts.

Note: When setting the `livy.spark.deploy-mode` to cluster, Livy session mode submissions are no longer available. In addition, batch job Livy submits with PySpark applications are not supported. Both of these are available only if the `livy.spark.deploy-mode` configuration is set to `client`. See [Table 9 on page 107](#) for more information.

Table 9. Supported environments per deploy mode

	Client	Cluster
Spark session (Scala)	Supported	Not supported
Sparky Session (Python)	Supported	Not supported
Spark batch job	Supported	Supported
Sparky batch job	Supported	Not supported

```
livy.spark.deploy-mode = client
```

- c. Set `livy.file.local-dir-whitelist` to the directories that contain your Spark application files, separated by commas:

```
livy.file.local-dir-whitelist =  
/usr/lpp/IBM/izoda/spark/spark23x/examples/jars/  
path/to/directory2containing/jar-files
```

- d. You may also modify the host IP address and the Livy server port by setting `livy.server.host` and `livy.server.port`:

```
livy.server.host = localhost  
livy.server.port = 8998
```

If you have changed the `SPARK_LOCAL_IP` environment variable, you will likely have to change the `livy.server.host` as well. By default, the Livy server uses port 8998.

- e. You may also modify the session timeout value by setting the `livy.server.session.timeout` configuration. We support the following suffixes: “us” (microseconds), “ms” (milliseconds), “s” (seconds), “m” or “min” (minutes), “h” (hours), and “d” (days):

```
livy.server.session.timeout-check = true  
livy.server.session.timeout = 1h
```

If `livy.server.session.timeout-check` is set to `false`, `livy.server.session.timeout` will be ignored.

2. Update the `$LIVY_CONF_DIR/livy-env.sh` with the environment variables pointing to the newly created working directories during the [“Configuring the z/OS IzODA Livy working directories”](#) on page 106 task.

For example:

```
LIVY_LOG_DIR=/var/livy/logs  
LIVY_PID_DIR=/tmp/livy/pid
```

Customizing user access for z/OS IzODA Livy

Complete this task to configure user access for z/OS IzODA Livy.

About this task

Similar to Spark, z/OS IzODA Livy supports the use of AT-TLS authentication at the Livy server port for enhanced security. It can be achieved by configuring the necessary digital certificates and key ring for the Livy end users and creating the corresponding AT-TLS policy rules.

IBM recommends using AT-TLS authentication at the Livy server port along with AT-TLS as the z/OS Spark client authentication. You may defer setting up authentication on both Livy and Spark during, for instance, early testing in a secure test environment.

Creating and configuring digital certificates and key rings

Complete this task to create and configure digital certificates and key rings that are needed for z/OS Livy authentication.

In these examples, a new set of internal CA, key ring and digital certificates are created for enabling AT-TLS authentication between the Livy end user and the Livy Server, that is separated from the Spark ones discussed in previous chapter. By doing so, the System Programmer can authorize access to the Livy server and the Spark cluster independently. This configuration assumes one-to-one certificate-to-user ID association; that is, one certificate maps to one user.

When both Livy AT-TLS authentication and Spark client authentication are enabled, the certificate setup (as shown in [Figure 7 on page 108](#)) consists of:

- Livy server/client certificates, signed by the Livy internal CA (“LIVY Local CA” in these examples), connected to the Livy SSL key ring (“LivyRing” in these examples).
- Spark server/client certificates, signed by the Spark internal CA (“SPARK Local CA” as discussed previously), connected to the SparkSSL key ring (“SparkRing” as discussed previously).

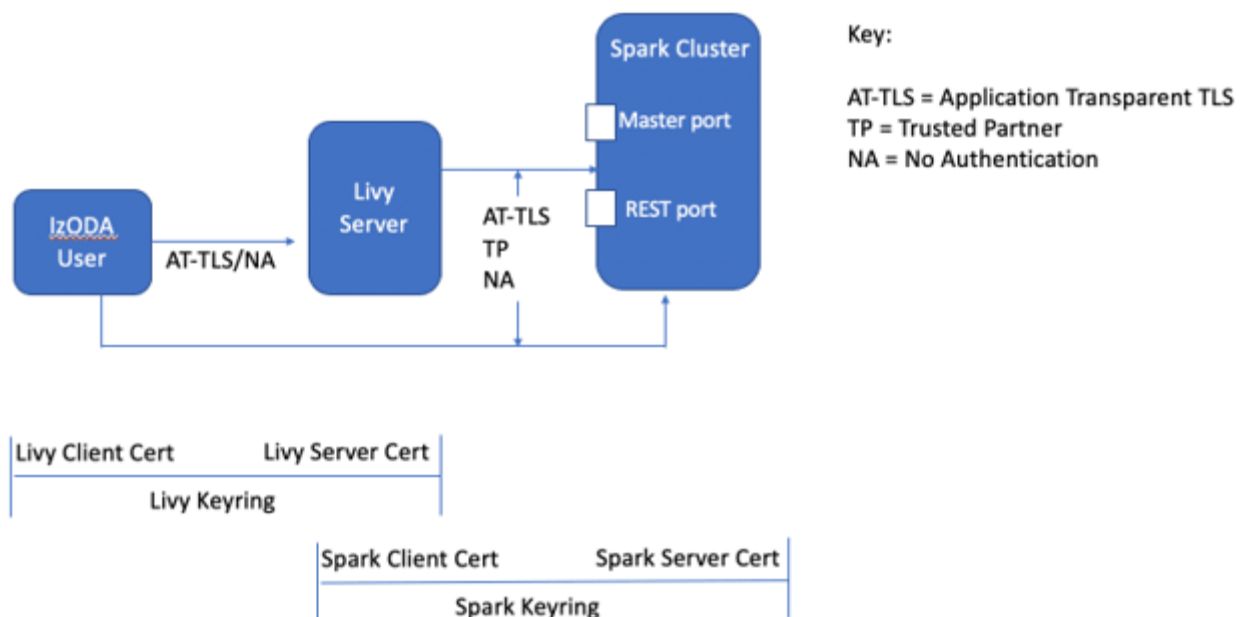


Figure 7. Livy/Spark key ring setup

The Livy end user ID who will be submitting Spark jobs via Livy needs to own a Livy client certificate. End users who own both a Spark and a Livy client certificate can submit Spark jobs either via Livy or to the cluster directly, for example, via spark-submit.

Because the Livy server will be submitting applications to the Spark cluster on behalf of the Livy end users, the Livy server user ID needs to own a Livy server certificate (connected to the Livy key ring) as well as a Spark client certificate (connected to the Spark key ring).

Digital certificates can be managed through RACF, PKI Services, or other security products.

Procedure:

1. If you do not already have an internal Livy CA defined, use RACF as the CA to create a Livy CA certificate. For example:

```
RACDCERT GENCERT CERTAUTH SUBJECTSDN(OU('LIVY Local CA') O('IBM') C('US'))  
WITHLABEL('LIVY Local CA') NOTAFTER(DATE(2030/01/01)) SIZE(1024)
```

For more information about the RACDCERT command, see *z/OS Security Server RACF Command Language Reference*.

2. Create a certificate and key ring for the user ID that will be used to start the Livy server:

The following RACF commands assume that you have already created the Spark Client Certificate for SPARKUSR. If not, see “Creating and configuring digital certificates and key rings” in [“Configuring z/OS Spark client authentication” on page 41](#)

- a. Create a Livy certificate that is signed by the Livy CA.

```
RACDCERT GENCERT ID(SPARKUSR) SIGNWITH(CERTAUTH LABEL('LIVY Local CA'))  
KEYUSAGE(HANDSHAKE)  
WITHLABEL('Livy Server Cert') SUBJECTSDN(CN('LIVY TEST SERVER') O('IBM')  
L('Poughkeepsie') SP('New York')  
C('US')) NOTAFTER(DATE(2030/01/01)) ALTNAME(DOMAIN('livyServer-host-addr')  
IP(xxx.xxx.xxx.xxx))
```

- b. Create a Livy SSL key ring (Livy Ring in these examples).

```
RACDCERT ADDRING(LivyRing) ID(SPARKUSR)
```

- c. Connect the Livy server certificate to the Livy key ring.

```
RACDCERT ID(SPARKUSR) CONNECT(ID(SPARKUSR) LABEL('Livy Server Cert')  
RING(LivyRing) USAGE(PERSONAL) DEFAULT)
```

- d. Connect the Livy CA certificate to the Livy key ring.

```
RACDCERT ID(SPARKUSR) CONNECT(CERTAUTH LABEL('LIVY Local CA') RING(LivyRing))
```

- e. Allow the Livy server user ID (SPARKUSR) to access the its key ring.

```
PERMIT IRR.DIGTCERT.LISTRING CLASS(FACILITY) ID(SPARKUSR) ACCESS(READ)  
PERMIT IRR.DIGTCERT.LIST CLASS(FACILITY) ID(SPARKUSR) ACCESS(READ)
```

- f. Refresh the RACF FACILITY class profiles:

```
SETROPTS RACLIST(FACILITY) REFRESH
```

3. Create and connect a certificate and a key ring for each Livy end user (LIVYUSR in these examples):

- a. Create a client certificate that is signed by the Livy CA.

```
RACDCERT GENCERT ID(LIVYUSR) SIGNWITH(CERTAUTH LABEL('LIVY Local CA'))  
KEYUSAGE(HANDSHAKE)  
WITHLABEL('Livy Client Cert') SUBJECTSDN(CN('LIVY TEST SERVER') O('IBM')  
L('Poughkeepsie') SP('New York')  
C('US')) NOTAFTER(DATE(2030/01/01))
```

Note: In order to connect to the Livy server from off-platform Livy interfaces, this certificate will need to be exported. Refer to [“Exporting Livy user certificates” on page 110](#).

- b. For users who will be connecting to the Livy server, create an SSL key ring for each user with the same name as in step 2b.

```
RACDCERT ADDRING(LivyRing) ID(LIVYUSR)
```

- c. Connect the client certificate to the end user's key ring.

```
RACDCERT ID(LIVYUSR) CONNECT(ID(LIVYUSR) LABEL('Livy Client Cert'))  
RING(LivyRing) USAGE(PERSONAL) DEFAULT)
```

- d. Connect the Livy CA certificate to the end user's key ring.

```
RACDCERT ID(LIVYUSR) CONNECT(CERTAUTH LABEL('LIVY Local CA') RING(LivyRing))
```

- e. Allow the end user's user ID to access its key ring.

```
PERMIT IRR.DIGTCERT.LISTRING CLASS(FACILITY) ID(LIVYUSR)  
ACCESS(READ)  
  
PERMIT IRR.DIGTCERT.LIST CLASS(FACILITY) ID(LIVYUSR) ACCESS(READ)
```

- f. Refresh the RACF FACILITY class profiles:

```
SETROPTS RACLIST(FACILITY) REFRESH
```

Exporting Livy user certificates

Complete this task to export the Livy user certificates for use by off-platform Livy interfaces.

Digital certificates can be exported through RACF PKI services or other security products. In the following example, RACF is used to extract the certificates into a PKCS#12 certificate package (.p12) and openssl to convert the .p12 into the PEM format. PEM is one of many formats available. Work with your security administrator if certificates are needed in other formats.

1. Create the .p12 package.

For example, from TSO, issue the following command to create a .p12 packages in the LIVYUSR's dataset where 'password' is the password used to access the contents of the package:

```
RACDCERT EXPORT(LABEL('Livy Client Cert'))  
ID(LIVYUSR)  
DSN('LIVYUSR.P12')  
FORMAT(PKCS12DER)  
PASSWORD('password')
```

2. Transfer (for example, FTP) the .p12 package as binary to the remote system. Issue the openssl command on the remote system to display the contents of the package. You will be prompted twice for the password used during the export. For example:

```
openssl pkcs12 -info -in LIVYUSR.P12 -passin pass:"password"
```

3. In PEM format, extract the client certificate, the CA certificate, and the private key from the .p12 package to three separate files.

- a. Extract the client certificate.

For example, to extract the client certificate into a pem-encoded file, issue the following:

```
openssl pkcs12 -in LIVYUSR.P12 -passin pass:"password" -out  
livyusr.orig.pem -clcerts -nokeys
```

This creates a file "livyusr.orig.pem". This file cannot be used until some informational lines are removed.

Specifically, lines outside the BEGIN and END lines should be removed. Here is an example of what these look may like:

```
Bag Attributes  
friendlyName: Livy Client Cert  
localKeyID: 00 00 00 01
```

```
subject=/C=US/ST=New York/L=Poughkeepsie/O=IBM/CN=Livy Client
issuer=/C=US/O=IBM/OU=LIVY Local CA
```

To remove these lines, you may edit the file directly or issue the following command to convert it:

```
openssl x509 -in livyusr.orig.pem -out livyusr.pem
```

- b. Extract the CA certificate into a PEM-encoded file. For example:

```
openssl pkcs12 -in LIVYUSR.P12 -passin pass:"password" -out
ca.orig.pem -cacerts -nokeys
```

To remove the informational lines, as above, either edit the file, or issue the following commands:

```
openssl x509 -in ca.orig.pem -out ca.pem
```

Note: If you have a chain of certificate authorities, you may need to edit the file directly to remove each instance instead of issuing the `openssl x509` command.

- c. Extract the private key into a PEM-encoded file. When issuing the following command, you will be prompted for the password twice. For example:

```
openssl pkcs12 -in LIVYUSR.P12 -passin pass:"password" -out
livyusrkey.pem -nocerts -nodes
```

Edit `livyusrkey.pem` to remove the informational lines. Here is an example of what these look may like:

```
Bag Attributes
  friendlyName: Livy Client Cert
  localKeyID: 00 00 00 01
  subject=/C=US/ST=New York/L=Poughkeepsie/O=IBM/CN=Livy Client
  issuer=/C=US/O=IBM/OU=LIVY Local CA
```

4. Verify the end user certificate (once CA certificate is extracted). For example:

```
openssl verify -CAfile ca.pem livyusr.pem
```

You should see output similar to the following:

```
livyusr.pem: OK
```

Defining the AT-TLS policy rules for z/OS IzODA Livy

Complete this task to define the AT-TLS policy rules for z/OS IzODA Livy in the policy configuration file.

About this task:

These example policy rules assume you are using AT-TLS as the z/OS IzODA Spark client authentication method, and previously completed the steps described in [“Configuring z/OS Spark client authentication” on page 41](#) for setting up the AT-TLS policy rules.

You can find sample AT-TLS policy rules for z/OS IzODA Livy in [Appendix C, “Sample z/OS IzODA Livy AT-TLS policy rules,” on page 169](#). Work with your network administrator to configure a policy that incorporates AT-TLS or Trusted Partner client authentication for Spark in conjunction with AT-TLS authentication for Livy.

Procedure:

1. Create the PortRange statement. By default, Livy uses port 8998 as the server port.

```
PortRange                               LivyServer_ATTLS
{
Port                                     8998
}
```

2. Define the Inbound and Outbound TTLSRules. These rules reference the same TTLSGroupAction (GroupAct_TTLS_On) that has already been set up for Spark.

```

TTLSRule                               LivyServer_ATTLS
{
  Direction                             Inbound
  LocalPortRangeRef                     LivyServer_ATTLS
  TTLSGroupActionRef                     GroupAct_TTLS_On
  TTLSEnvironmentActionRef               EnvAct_LivyServer_ATTLS
}
TTLSRule                               LivyClient_ATTLS
{
  Direction                             Outbound
  RemotePortRangeRef                     LivyServer_ATTLS
  TTLSGroupActionRef                     GroupAct_TTLS_On
  TTLSEnvironmentActionRef               EnvAct_LivyClient_ATTLS
}

```

3. Create the corresponding TTLSEnvironmentAction statements for the Inbound and Outbound TTLSRules. These rules reference the same TTLSEnvironmentAdvancedParms (EnvAdv_TLS) that has already been set up for Spark.

```

TTLSEnvironmentAction                   EnvAct_LivyServer_ATTLS
{
  HandshakeRole                         ServerWithClientAuth
  EnvironmentUserInstance                 0
  TTLSKeyRingParmsRef                     KeyRing_Livy
  TTLSEnvironmentAdvancedParmsRef         EnvAdv_TLS
}
TTLSEnvironmentAction                   EnvAct_LivyClient_ATTLS
{
  HandshakeRole                         Client
  EnvironmentUserInstance                 0
  TTLSKeyRingParmsRef                     KeyRing_Livy
  TTLSEnvironmentAdvancedParmsRef         EnvAdv_TLS
}

```

4. Create the TTLSKeyRingParms statement. Use the same SSL keyring name (LivyRing) as during the server/client certificate creation steps in the previous section.

```

TTLSKeyRingParms                       KeyRing_Livy
{
  Keyring                               LivyRing
}

```

Chapter 9. Customizing Anaconda

z/OS IzODA Anaconda (FMID HANA110) contains the Anaconda data science ecosystem, which consists of the Anaconda package manager and a library of packages that data scientists and application developers can assemble into specific stacks of runtime capabilities to support particular data science applications.

Anaconda is primarily a Python-based environment favored by data scientists who are familiar with the analytics capabilities that have been built around the Python language. This environment complements and overlaps some of the capabilities that are available in z/OS IzODA Spark (FMID HSPK120). Together, Anaconda and z/OS Spark provide a complete data science platform that can support a broad range of analytics applications.

A more complete description of the Anaconda environment is available at [Anaconda \(https://www.continuum.io/\)](https://www.continuum.io/).

Note: Users of Spark 2.2.0 and earlier who are migrating to a newer level of Python should continue to use Python 3.6.

Anaconda package management

Anaconda package management is provided through the **conda** command line interface. This is used to manage the root environment of installed packages. A base set of packages is provided with the Anaconda component, and a repository of additional packages is available through the IzODA channel hosted in the Anaconda Cloud. For a complete list of available packages, see [Package Repository for IzODA \(https://anaconda.org/IzODA/repo\)](https://anaconda.org/IzODA/repo).

Anaconda repositories

Anaconda repositories are dynamic and flexible. The conda package manager can be used to install or uninstall any version of a package whenever it is published to a repository. This capability allows Anaconda to be framework for managing all types of functional packages, and makes it ideal for handling runtime environments based on open source.

Detailed information about all of the capabilities of the Anaconda ecosystem on z/OS, as well as the installation and configuration of the z/OS IzODA Anaconda component, is available on github at [Anaconda Overview](#).

Part 4. Verification

Chapter 10. Verifying the IBM Open Data Analytics for z/OS customization

Some simple checks can verify that the basic installation and customization of IBM Open Data Analytics for z/OS was successful.

Before you begin

Follow the instructions in [Part 3, “Customization,” on page 13](#) to customize your environment and Open Data Analytics for z/OS.

About this task

Complete the following steps to verify the successful installation and customization of Open Data Analytics for z/OS on your system.

Procedure

The **spark-shell** command attempts to create `metastore_db` in the current directory. If you do not have the `hive-site.xml` file set up, and if your **SPARK_HOME** directory is read-only, you must invoke **spark-shell** from a writable directory. For more information, see [“Updating the Apache Spark configuration files” on page 34](#).

The following steps assume that you are using a writable directory other than **SPARK_HOME**.

1. Open an SSH or Telnet shell environment and navigate to a user-writable directory, such as your home directory.
2. Run the following command to open the Open Data Analytics for z/OS shell:

```
$SPARK_HOME/bin/spark-shell
```

where:

SPARK_HOME

The environment variable that contains the path to the Open Data Analytics for z/OS installation directory.

After a series of messages, the Open Data Analytics for z/OS prompt appears:

```
scala>
```

3. From the Open Data Analytics for z/OS prompt, run the following command, which should return the value 1000:

```
sc.parallelize(1 to 1000).count()
```

4. From the Open Data Analytics for z/OS prompt, enter the following command to verify that the help information is displaying properly:

```
:help
```

The returned help information resembles the following example:

```
All commands can be abbreviated, e.g. :he instead of :help.
Those marked with a * have more detailed help, e.g. :help imports.

:help [command]          print this summary or command-specific help
:history [num]           show the history (optional num is commands to show)
:h? <string>             search the history
:imports [name name ...] show import history, identifying sources of names
:implicit [-v]           show the implicit in scope
:javap <path|class>      disassemble a file or class name
```

```
:load <path>          load and interpret a Scala file
...
```

5. Enter the following command to exit the spark-shell:

```
:quit
```

6. From the z/OS UNIX prompt, run the following command to verify that Open Data Analytics for z/OS can generate an approximate value for Pi:

```
$SPARK_HOME/bin/run-example SparkPi
```

The returned information resembles the content in the following sample, with "Pi is roughly 3.13742" within the content:

```
SPARKID:~ $ $SPARK_HOME/bin/run-example SparkPi
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
18/08/21 16:33:15 INFO SparkContext: Running Spark version 2.2.0
18/08/21 16:33:16 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform...
using builtin-java classes where applicable
18/08/21 16:33:16 INFO SparkContext: Submitted application: Spark Pi
...
18/08/21 16:33:20 INFO TaskSchedulerImpl: Removed TaskSet 0.0, whose tasks have all completed, from pool
18/08/21 16:33:20 INFO DAGScheduler: ResultStage 0 (reduce at SparkPi.scala:38) finished in 0.713 s
18/08/21 16:33:20 INFO DAGScheduler: Job 0 finished: reduce at SparkPi.scala:38, took 1.057874 s
Pi is roughly 3.140575702878514
18/08/21 16:33:20 INFO SparkUI: Stopped Spark web UI at http://...
...
18/08/21 16:33:20 INFO ShutdownHookManager: Shutdown hook called
18/08/21 16:33:20 INFO ShutdownHookManager: Deleting directory
/tmp/spark/scratch/spark-3d5858dc-bb8b-4538-8f6a-7712e9151b12
```

Results

You have successfully verified the basic installation and customization of Open Data Analytics for z/OS on your z/OS system. For information about verifying other aspects of the product, such as the master and workers, see <http://spark.apache.org/docs/2.4.8/>.

Using the IBM Open Data Analytics for z/OS Spark Configuration Checker

About this task

The Configuration Checker tool, which is included in IBM Open Data Analytics for z/OS Spark starting from IzODA Spark 2.2.0, verifies and displays some of your IzODA Spark settings.

Note: If you are using Spark 2.2.0, APAR PI93605 is required to use the checker.

Procedure

The IzODA Spark Configuration Checker must be invoked from the user ID under which the Spark cluster is started. In this documentation, it is referred to as the Spark ID. To run the IzODA Spark Configuration Checker, invoke the following command:

```
$SPARK_HOME/sbin/spark-configuration-checker.sh
```

Where:

\$SPARK_HOME

Specifies the IzODA Spark installation directory.

The tool also supports the following parameters:

-v, --verbose

Provides lists of checks, configuration files, settings, and solutions for errors or warnings as output.

-c, --color

Adds colors to the output.

-h, --help

Displays the usage of the checker.

For example, to run the IzODA Spark Configuration Checker in colored verbose mode, you can issue the following command:

```
$SPARK_HOME/sbin/spark-configuration-checker.sh -v -c
```

Results

After the tool is executed and completes successfully, the IzODA Spark Configuration Checker will produce a report of possible errors and warnings. IBM suggests that you review the report carefully, fix the reported errors, and take any warnings under advisement.

Example

The following example is sample output from the Configuration Checker. Note that for the purposes of brevity, some lines have been removed and replaced with an ellipsis.

```
./sbin/spark-configuration-checker.sh -v -c
===== CHECKS =====
Pass    Obtaining environment variables
Pass    Checking SPARK_HOME
Pass    Checking if SPARK_CONF_DIR is default
Pass    Checking Java version
Warn    Checking if Worker is using a random port
...
===== INFORMATION =====
SPARK_HOME: /spark/Spark_Srvlib/spark24x
Spark Version: 2.4.8

JAVA_HOME: /usr/lpp/java/java800/J8.0_64
Java Location: /usr/lpp/java/java800/J8.0_64/bin/java
Java Version: 1.8.0 SR7 FP10 - 64-bit

Bash Location: /sparklg01/MiniConda/bin/bash
Bash Version: 4.3.48(2)

Spark Configuration
  /u/Wellie1/conf/spark-defaults.conf
    spark.zos.driver.jobname.prefix      ODASD
    spark.master.rest.enabled             true
    spark.sql.orc.impl                    hive
    spark.files.override                  true
    spark.zos.executor.jobname.prefix     ODASX
    spark.zos.master.authenticate         true
    spark.executorEnv.PYTHONHASHSEED     0
    spark.zos.master.authenticate.method  ATTLS
    spark.sql.warehouse.dir               /u/Wellie1/conf/..

  /u/Wellie1/conf/spark-env.sh
    SPARK_MASTER_HOST                     alps.pok.ibm.com
    SPARK_DAEMON_JAVA_OPTS                -Dfile.encoding=UTF8
    SPARK_LOCAL_DIRS                      /tmp/spark/scratch
    _BPX_SHAREAS                          NO
    SPARK_PID_DIR                          /tmp/spark/pid
    _BPXK_AUTOCVT                          ON
    JAVA_HOME                             /usr/lpp/java/java800/J8.0_64
    SPARK_MASTER_PORT                      7077
    USER                                  ${USER:-$(whoami)}
    _EDC_ADD_ERRNO2                        1
    SPARK_WORKER_DIR                      /u/Wellie1/conf/../../work
    SPARK_LOG_DIR                         /u/Wellie1/conf/../../logs

  Environmental Variables
    SPARK_HOME                           /spark/Spark_Srvlib/spark24x
    _BPX_SHAREAS                          NO
    SPARK_LOGS                            /u/Wellie1/logs
    _BPXK_AUTOCVT                          ON
    JAVA_HOME                             /usr/lpp/java/java800/J8.0_64
    IBM_JAVA_OPTIONS                      -Dfile.encoding=UTF8
-Dconsole.encoding=IBM-1047 -
Xmx4g -Xss1024k
odeCacheSize=512m
...
```

```

Spark Ports
    Master          7077 - Range (7077-7093)
    Master Web UI   8080 - Range (8080-8096)
...
Spark Directories
    Name           Spark Worker Directory
    Path           /u/Wellie1/conf/../../work
    File System     OMVS.WELLIE1
    Size           843.8 MB
    Used Space      17.8 MB - 3%
    Free Space      825.9 MB - 97%
    Mount          /u/Wellie1
    Permissions     rwxrwxr-t
...
OMVS Information
    UID            0000000011
    PROCUSERMAX    NONE
    MMAPAREAMAX    NONE
...
===== ERRORS =====
===== WARNINGS =====
Spark: WARNING:      Worker port is set to random.
      INFO:          Having a Spark port set as random may prevent proper port bindings.
      SOLUTION:      Set the Spark port to a valid port number.
...
Please refer to the IzODA Installation and Customization Guide for more details.

```

Chapter 11. Verifying the Data Service server installation

To verify the server installation, create a sample VSAM file and a virtual table, and then run a query that accesses the VSAM data.

Procedure

1. Create the sample VSAM file on the mainframe that hosts the server. Run the AZKGNSTF member in the *hlq.SAZKCNTL* data set to allocate and load the sample VSAM file.
The job should complete with a condition code of 0.
2. Create the *staffvs* virtual table. Run the AZKIVVS1 member in the *hlq.SAZKCNTL* data set to perform a batch extract of the sample VSAM file listing and create a virtual table that formats the result set that is returned from the VSAM file.

This step runs a query against the sample VSAM file. The job should complete with a condition code of 0.

Chapter 12. Verifying the IBM Open Data Analytics for z/OS product

After you verify the two individual FMIDs that are part of the IBM Open Data Analytics for z/OS product, verify that both components work together.

About this task

In the following procedure, the **SPARK_HOME** environment variable contains the path to the z/OS Spark installation directory. By default, the configuration settings for your Spark cluster reside in the `$SPARK_HOME/conf` directory. These configuration settings may be kept in any directory and pointed to by the **SPARK_CONF_DIR** environment variable. This approach can be useful if you want to isolate your cluster settings from a specific installation, such as for migration or for multi-cluster installation scenarios.

Complete the following steps to verify the operation of the Open Data Analytics for z/OS product on your system.

Procedure

1. Start the Spark master daemon on your z/OS image.

From a z/OS UNIX session, issue the following command:

```
$SPARK_HOME/sbin/start-master.sh -h host_IP_address -p sparkMaster-port
```

where:

\$SPARK_HOME

An environment variable that contains the installation path for z/OS Spark.

host_IP_address

The IP address for this z/OS system. In this documentation, this value is shown as `xxx.xxx.xxx.xxx`.

sparkMaster-port

The Spark master daemon port. The default is 7077.

The Spark master daemon attempts to define the port that you specify on the **-p** parameter. If this port is not available, it increments the **-p** value by one and attempts to bind to the next available port. It is imperative to check the Spark master daemon log file (located in the **\$SPARK_LOG_DIR** directory) to determine the port number on which the Spark master daemon is listening.

It is helpful to look at an example of an actual Spark master daemon log file for some pertinent information that you will need.

Example: The following output shows a portion of a valid Spark master daemon log file, `spark-${USER}-org.apache.spark.deploy.master.Master-1-host.out`, that was generated after issuing the following command:

```
$SPARK_HOME/sbin/start-master.sh -h xxx.xxx.xxx.xxx -p 7077
```

If your Spark master daemon log does not look like this example, compare the contents of the Spark Command: line in your log file to the one shown here and look for any differences. (You will not see the Registering worker message in your log file until you start the Spark worker daemon in the next step.) Different versions of Apache Spark can also generate different log file contents.

```
Spark Command: /usr/lpp/java/java800/J8.0_64/bin/java -cp
/usr/lpp/IBM/Spark/sbin/./conf/:/usr/lpp/IBM/Spark/jars/*
-Dfile.encoding=UTF8 -Xmx1g org.apache.spark.deploy.master.Master --host
xxx.xxx.xxx.xxx --port 7077 --webui-port 8080 -h xxx.xxx.xxx.xxx -p 7077
=====
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
```

```

18/08/21 10:39:39 INFO Master: Started daemon with process name: 65639@hostname
18/08/21 10:39:39 WARN NativeCodeLoader: Unable to load native-hadoop library for your
platform... using builtin-java classes where applicable
18/08/21 10:39:40 INFO SecurityManager: Changing view acls to: SPARKID
18/08/21 10:39:40 INFO SecurityManager: Changing modify acls to: SPARKID
18/08/21 10:39:40 INFO SecurityManager: Changing view acls groups to: ...
18/08/21 10:39:40 INFO SecurityManager: Changing modify acls groups to: ...
A 18/08/21 10:39:40 INFO SecurityManager: z/OS client authentication is
enabled, using method ATTLS. Make sure your environment is configured properly for this
method.
See IzODA Installation and Customization Guide for more information.
18/08/21 10:39:40 INFO SecurityManager: SecurityManager: authentication disabled; ui acls
disabled; users with view permissions: Set(SPARKID); groups with view permissions: Set();
users
with modify permissions: Set(SPARKID); groups with modify permissions: Set()
18/08/21 10:39:40 WARN NetUtil: Failed to find the loopback interface
B 18/08/21 10:39:40 INFO Utils: Successfully started service 'sparkMaster' on port 7077.
B 18/08/21 10:39:40 INFO Master: Starting Spark master at spark://xxx.xxx.xxx.xxx:7077
18/08/21 10:39:40 INFO Master: Running Spark version 2.2.0
18/08/21 10:39:40 INFO Master: Performing master environment verification
C 18/08/21 10:39:40 INFO Utils: Successfully started service 'MasterUI' on port 8080.
C 18/08/21 10:39:40 INFO MasterWebUI: Started MasterWebUI at http://xxx.xxx.xxx.xxx:8080
D 18/08/21 10:39:40 INFO Utils: Successfully started service on port 6066.
D 18/08/21 10:39:40 INFO StandaloneRestServer: Started REST server for submitting
applications on port 6066
18/08/21 10:39:40 INFO Master: I have been elected leader! New state: ALIVE
18/08/21 10:40:03 WARN ZosNativeUtil: TLS connection pending, will retry.
TTLSi_Stat_Policy=4, TTLSi_Stat_Conn=1, TTLSi_Sec_Type=6
18/08/21 10:40:04 INFO Master: Registering worker xxx.xxx.xxx.xxx:1026 with 10 cores,
4.0 GB RAM, from user ID SPARKID

```

Note the following information from the log file:

- The messages that are prefixed by **A** identify the z/OS Client authentication options that were selected. In this example, z/OS Client Authentication is enabled and using ATTLS as its authentication method.
- The messages that are prefixed by **B** identify the port number that is being used for the Spark master daemon. In this example, the Spark master daemon successfully bound to and is using host IP address xxx.xxx.xxx.xxx and port 7077.
- The messages that are prefixed by **C** identify the port number that the Spark master daemon is using for the MasterUI port. In this example, the Spark master daemon successfully bound to and is using port 8080 for the MasterUI port.
- The messages that are prefixed by **D** identify the port number that the Spark master daemon is using for its REST port. Note that finding these message is optional for verifying the IzODA product. The REST server interface, which listens on port 6066, is disabled by default. As of APAR PH03469, IzODA supports TLS client authentication on the REST server port and applications can be securely submitted through this interface. For details, see [“Configuring networking for Apache Spark”](#) on page 37. In this example, the Spark master daemon successfully bound to and is using port 6066 for the REST port. .

Write down each of these port numbers for your configuration; you will need them later in this procedure.

sparkMaster port:	
MasterUI port:	
REST port:	

2. Start the Spark worker daemon on your z/OS image.

From a z/OS UNIX session, issue the following commands:

```
$SPARK_HOME/sbin/start-slave.sh spark://host_IP_address:sparkMaster-port
```

where:

\$SPARK_HOME

An environment variable that contains the installation path for z/OS Spark.

host_IP_address

The IP address (xxx.xxx.xxx.xxx) for this z/OS system.

sparkMaster-port

The Spark master daemon port. The default is 7077.

It is helpful to look at an example of an actual Spark worker daemon log file for some pertinent information that you will need.

Example: The following output shows a portion of a valid Spark worker daemon log file, `spark--org.apache.spark.deploy.worker.Worker-1-host.out`, that was generated after issuing the following command:

```
$SPARK_HOME/sbin/start-slave.sh spark://xxx.xxx.xxx.xxx:7077
```

```
Spark Command: /usr/lpp/java/J8.0_64/bin/java -cp
/usr/lpp/IBM/Spark/sbin/./conf:/usr/lpp/IBM/Spark/jars/* -Dfile.encoding=UTF8
-Xmx1g org.apache.spark.deploy.worker.Worker --webui-port 8081
spark://xxx.xxx.xxx.xxx:7077
=====
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
18/08/21 10:42:48 INFO Worker: Started daemon with process name: 16842871@hostname
18/08/21 10:42:48 WARN NativeCodeLoader: Unable to load native-hadoop library for your
platform... using builtin-java classes where applicable
18/08/21 10:42:48 INFO SecurityManager: Changing view acls to: SPARKID
18/08/21 10:42:48 INFO SecurityManager: Changing modify acls to: SPARKID
18/08/21 10:42:48 INFO SecurityManager: Changing view acls groups to: ...
18/08/21 10:42:48 INFO SecurityManager: Changing modify acls groups to: ...
18/08/21 10:42:48 INFO SecurityManager: SecurityManager: authentication disabled;
ui acls disabled;
users with view permissions: Set(SPARKID); groups with view permissions: Set();
users with modify permissions: Set(SPARKID); groups with modify permissions: Set()
18/08/21 10:42:48 WARN NetUtil: Failed to find the loopback interface
E 18/08/21 10:42:48 INFO Utils: Successfully started service 'sparkWorker' on port 1028.
E 18/08/21 10:42:49 INFO Worker: Starting Spark worker xxx.xxx.xxx.xxx:1028
with 10 cores, 4.0 GB RAM
18/08/21 10:42:49 INFO Worker: Running Spark version 2.2.0
18/08/21 10:42:49 INFO Worker: Spark home: /usr/lpp/IBM/Spark

18/08/21 10:42:49 INFO Worker: Performing worker environment verification
18/08/21 10:42:49 INFO ZosSafUtil: READ access to SAF BPX.SRV.* SURROGAT class profile
permission check
for SPARKID is successful
F 18/08/21 10:42:49 INFO Utils: Successfully started service 'WorkerUI' on port 8081.
F 18/08/21 10:42:49 INFO WorkerWebUI: Bound WorkerWebUI to 0.0.0.0, and started at
http://xxx.xxx.xxx.xxx:8081
18/08/21 10:42:49 INFO Worker: Connecting to master xxx.xxx.xxx.xxx:7077...
18/08/21 10:42:49 INFO TransportClientFactory: Successfully created connection to
xxx.xxx.xxx.xxx/9.57.2.76:7077 after 37 ms (0 ms spent in bootstraps)
18/08/21 10:42:50 INFO Worker: Successfully registered with master
spark://xxx.xxx.xxx.xxx:7077
```

Note the following information from the log file:

- The messages that are prefixed by **E** identify the port number that is being used for the Spark worker daemon. In this example, the Spark worker daemon successfully bound to and is using port 1028.
- The messages that are prefixed by **F** identify the port number that the Spark worker daemon is using for the WorkerUI port. In this example, the Spark worker daemon successfully bound to and is using port 8081 for the WorkerUI port.

Write down each of these port numbers for your configuration; you will need them later in this procedure.

sparkWorker port:	
WorkerUI port:	

Now that you have successfully started the Spark master daemon and Spark worker daemon processes on the z/OS system, you can now use the **spark-submit** command to run the same

SparkPi sample that you ran via the interactive **spark-shell** command in Chapter 10, “Verifying the IBM Open Data Analytics for z/OS customization,” on page 117.

3. Issue the following command on a single line, to run the SparkPi sample in client-deploy mode:

- ```
$SPARK_HOME/bin/spark-submit --class org.apache.spark.examples.SparkPi
--master spark://host-IP-addr:sparkMaster-port
$SPARK_HOME/examples/jars/spark-examples_2.11-sparkVersion.jar
```

where:

**\$SPARK\_HOME**

An environment variable that contains the installation path for z/OS Spark.

**host\_IP\_address**

The IP address (xxx.xxx.xxx.xxx) for this z/OS system.

**sparkMaster-port**

The Spark Master port. The default is 7077.

**sparkVersion**

z/OS Spark version number (2.4.8)

After you issue the **spark-submit** command, you should see the application’s driver log as **stderr** and an approximation of **pi** as **stdout**.

**Example:** The following output shows an example of the messages that result from the Spark submit.

```
18/08/21 11:54:14 INFO DAGScheduler: Job 0 finished: reduce at SparkPi.scala:38, took 3.470622
s
G Pi is roughly 3.140555702778514
18/08/21 11:54:14 INFO SparkUI: Stopped Spark web UI at http://XXX.XXX.XXX.XXX:4040
18/08/21 11:54:14 INFO StandaloneSchedulerBackend: Shutting down all executors
18/08/21 11:54:14 INFO CoarseGrainedSchedulerBackend$DriverEndpoint: Asking each executor
to shut down
18/08/21 11:54:14 INFO MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint
stopped!
18/08/21 11:54:14 INFO MemoryStore: MemoryStore cleared
18/08/21 11:54:14 INFO BlockManager: BlockManager stopped
18/08/21 11:54:14 INFO BlockManagerMaster: BlockManagerMaster stopped
18/08/21 11:54:14 INFO OutputCommitCoordinator$OutputCommitCoordinatorEndpoint:
OutputCommitCoordinator stopped!
18/08/21 11:54:14 INFO SparkContext: Successfully stopped SparkContext
18/08/21 11:54:14 INFO ShutdownHookManager: Shutdown hook called
18/08/21 11:54:14 INFO ShutdownHookManager: Deleting directory
/tmp/spark/scratch/spark-95ed4711-ec2d-41fd-9b64-ab5cd6a03ebc
```

The messages that are prefixed by **G** are **stdout** of the application. All other messages are **stderr**. In this example, the SparkPi application approximated Pi to be 3.140555702778514.

To determine whether the **spark-submit** command completed successfully, look for the approximation of Pi. You can also use the Spark MasterUI port via a web browser to view the state of the completed application (step 5).

4. (Optional) Issue the following command on a single line, to run the SparkPi sample in cluster-deploy mode:

- ```
$SPARK_HOME/bin/spark-submit --class org.apache.spark.examples.SparkPi  
--master spark://host-IP-addr:sparkRest-port  
$SPARK_HOME/examples/jars/spark-examples_2.11-sparkVersion.jar
```

where:

\$SPARK_HOME

An environment variable that contains the installation path for z/OS Spark.

host_IP_address

The IP address (xxx.xxx.xxx.xxx) for this z/OS system.

sparkREST-port

The Spark Master REST port. The default is 6066.

sparkVersion

z/OS Spark version number (2.4.8)

Example: The following output shows an example of the messages that result from the submit. You will not see any response back from your z/OS UNIX session beyond these messages.

Running Spark using the REST application submission protocol.

Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties

18/08/21 13:35:34 INFO RestSubmissionClient: Submitting a request to launch an application in spark://xxx.xxx.xxx.xxx:6066.

18/08/21 13:35:36 INFO RestSubmissionClient: Submission successfully created as

driver-20180821133535-0000. Polling submission state...

18/08/21 13:35:36 INFO RestSubmissionClient: Submitting a request for the status of submission driver-20180821133535-0000 in spark://xxx.xxx.xxx.xxx:6066.

18/08/21 13:35:36 INFO RestSubmissionClient: State of driver driver-20180821133535-0000 is now RUNNING.

18/08/21 13:35:36 INFO RestSubmissionClient: Driver is running on worker worker-20180821104249-9.57.2.76-1028 at 9.57.2.76:1028.

18/08/21 13:35:36 INFO RestSubmissionClient: Server responded with

CreateSubmissionResponse:

```
{
  "action" : "CreateSubmissionResponse",
  "message" : "Driver successfully submitted as driver-20180821133535-0000",
  "serverSparkVersion" : "2.2.0",
  "submissionId" : "driver-20180821133535-0000",
  "success" : true
}
```

To determine whether the **spark-submit** command completed successfully, use the Spark MasterUI port via a web browser (steps 4, 5, 6).

5. Point your web browser to the following URL:

```
http://host_IP_address:MasterUI-port
```

where:

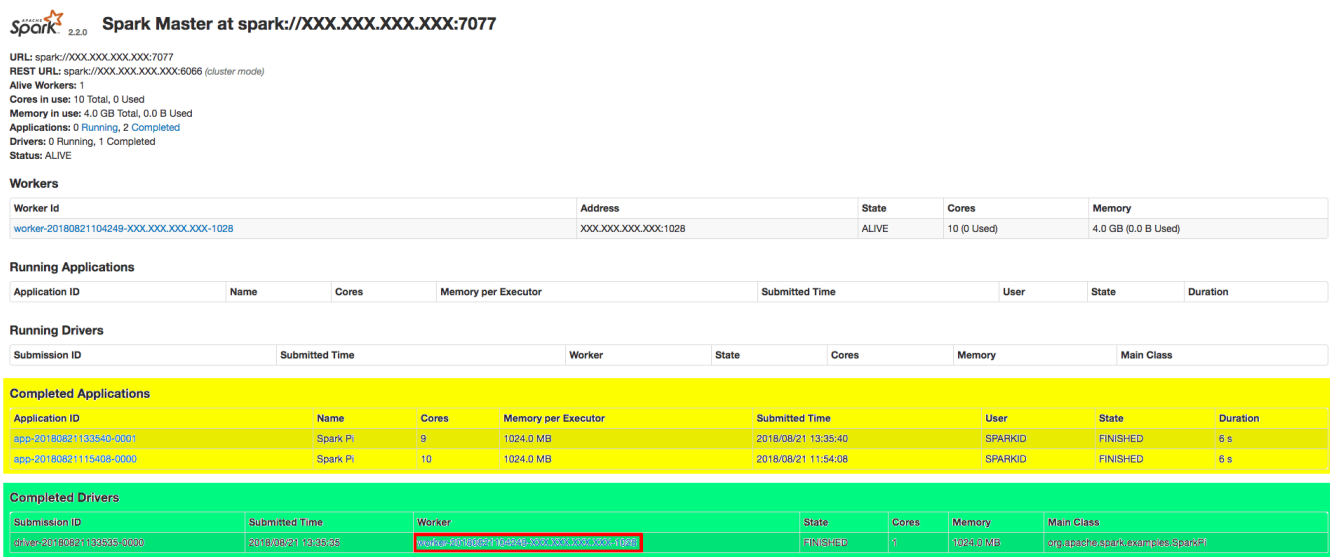
host_IP_address

The IP address (xxx.xxx.xxx.xxx) for this z/OS system.

MasterUI-port

The Spark MasterUI port. The default is 8080.

In your browser window, you see a page that resembles the following example:



Look for the following indicators of successful completion:

- In the **Completed Applications** section (second-to-bottom highlighted section of the figure), look for the Spark Pi in the **Name** column and FINISHED in the **State** column.

- In the **Completed Drivers** section (bottom highlighted section of the figure), look for `org.apache.spark.examples.SparkPi` in the **Main Class** column and **FINISHED** in the **State** column.

If you did not perform the optional cluster-deploy mode submission in step 4, skip to step 8.

6. In the **Completed Drivers** section, click the link in the **Worker** column for the `org.apache.spark.examples.SparkPi` main class.

The link has the format `worker-yyyymmddttttttt-host_IP_address-WorkerPort`. When you click it, you see a page that resembles the following example:

Spark Worker at XXX.XXX.XXX.XXX:1028

ID: worker-20180821104249-XXXX-XXXX-XXXX-XXXX-1028
 Master URL: spark://XXXX-XXXX-XXXX-XXXX-XXXX-1028
 Cores: 10 (0 Used)
 Memory: 4.0 GB (0.0 B Used)
[Back to Master](#)

Running Executors (0)

ExecutorID	Cores	State	Memory	Job Details	Logs
No running executors.					

Finished Executors (2)

ExecutorID	Cores	State	Memory	Job Details	Logs
0	10	KILLED	1024.0 MB	ID: app-20180821115408-0000 Name: Spark Pi User: SPARKID	stdout stderr
0	9	KILLED	1024.0 MB	ID: app-20180821133540-0001 Name: Spark Pi User: SPARKID	stdout stderr

Finished Drivers (1)

DriverID	Main Class	State	Cores	Memory	Logs	Notes
driver-20180821133535-0000	org.apache.spark.examples.SparkPi	FINISHED	1	1024.0 MB	stdout stderr	

7. In the **Finished Drivers** section, click the **stdout** link in the **Logs** column for the `org.apache.spark.examples.SparkPi` main class to display the value of SparkPi.
8. Use the interactive **spark-shell** command to run a small program to access the STAFFVS VSAM data set that was created and loaded in [Chapter 11, “Verifying the Data Service server installation,”](#) on page 121.

- a) Issue the following command to start the interactive spark-shell:

```
$SPARK_HOME/bin/spark-shell --jars path_to_DVDriver/dv-jdbc-version_number.jar
```

Result: The spark-shell provides an automatic, implied Spark context (sc) where you can run a Scala program.

- b) Enter the following Scala statements to access and display the STAFFVS VSAM data set by using the Rocket DV JDBC driver:

```
val dfReader = spark.read
  .format("jdbc")
  .option("driver", "com.rs.jdbc.dv.DvDriver")
  .option("url", "jdbc:rs:dv://host_IP_address:AZKSport;DSN=AZKS;DBTY=DVS")
  .option("dbtable", "AZKSQL.STAFFVS")
  .option("user", "userID")
  .option("password", "password")
val df = dfReader.load()
df.show()
```

where:

host_IP_address

The IP address for this z/OS system.

AZKSport

The Rocket MDSS port. The default is 1200.

userID

The z/OS user ID that has permission to read the STAFFVS data set.

password

The password for *userID*.

Result: You see the first 20 rows of the STAFFVS data set, as in the following example display:

```
scala> val dfReader = spark.read
dfReader: org.apache.spark.sql.DataFrameReader = org.apache.spark.sql.DataFrameReader@53ef76cf

scala> .format("jdbc")
res7: org.apache.spark.sql.DataFrameReader = org.apache.spark.sql.DataFrameReader@53ef76cf

scala> .option("driver", "com.rs.jdbc.dv.DvDriver")
res8: org.apache.spark.sql.DataFrameReader = org.apache.spark.sql.DataFrameReader@53ef76cf

scala> .option("url", "jdbc:rs:dv://xxx.xxx.xxx.xxx:1200;DSN=AZKS;DBTY=DVS")
res9: org.apache.spark.sql.DataFrameReader = org.apache.spark.sql.DataFrameReader@53ef76cf

scala> .option("dbtable", "AZKSQL.STAFFVS")
res10: org.apache.spark.sql.DataFrameReader = org.apache.spark.sql.DataFrameReader@53ef76cf

scala> .option("user", "xxxxxxx")
res11: org.apache.spark.sql.DataFrameReader = org.apache.spark.sql.DataFrameReader@53ef76cf

scala> .option("password", "xxxxxxx").load()
res12: org.apache.spark.sql.DataFrame = [STAFFVS_KEY_ID: int, STAFFVS_DATA_NAME_L: int ... 4 more fields]

scala> val df = dfReader.load()
df: org.apache.spark.sql.DataFrame = [STAFFVS_KEY_ID: int, STAFFVS_DATA_NAME_L: int ... 4 more fields]

scala> df.show()
+-----+-----+-----+-----+-----+-----+
|STAFFVS_KEY_ID|STAFFVS_DATA_NAME_L|STAFFVS_DATA_NAME|STAFFVS_DATA_DEPT|STAFFVS_DATA_JOB|STAFFVS_DATA_YRS|
+-----+-----+-----+-----+-----+-----+
|10|7|SANDERS|20|MGR|7|
|20|6|PERNAL|20|SALES|8|
|30|8|MARENGHI|38|MGR|5|
|40|7|O'BRIEN|38|SALES|6|
|50|5|HANES|15|MGR|10|
|60|7|QUIGLEY|38|SALES|0|
|70|7|ROTHMAN|15|SALES|7|
|80|5|JAMES|20|CLERK|0|
|90|7|KOONITZ|42|SALES|6|
|100|5|PLOTZ|42|MGR|7|
|110|4|NGAN|15|CLERK|5|
|120|8|NAUGHTON|38|CLERK|0|
|130|9|YAMAGUCHI|42|CLERK|6|
|140|5|FRAYE|51|MGR|6|
|150|8|WILLIAMS|51|SALES|6|
|160|8|MOLINARE|10|MGR|7|
|170|8|KERMISCH|15|CLERK|4|
|180|8|ABRAHAMS|38|CLERK|3|
|190|7|SNEIDER|20|CLERK|8|
|200|8|SCOUTTEN|42|CLERK|0|
+-----+-----+-----+-----+-----+-----+
only showing top 20 rows
```

Results

You have now verified that the z/OS Spark and Mainframe Data Service components of Open Data Analytics for z/OS successfully work together on your z/OS system.

What to do next

To perform additional installation verification, run the following installation verification programs that are available on GitHub.

[Anaconda/ODL Installation Verification Program \(IVP\) with Jupyter Notebook \(izoda.github.io/site/anaconda/ivp-jupyter-notebook/\)](https://github.com/izoda/anaconda-ivp-jupyter-notebook/)

IBM provides this installation verification program (IVP) to get started with Anaconda and Optimized Data Layer (ODL) stacks of IzODA. By completing this IVP, you ensure that Anaconda and ODL are installed successfully and that you are able to run data visualizations and analysis on mainframe data sources.

[IzODA Installation Verification Program \(IVP\) with PySpark \(izoda.github.io/site/anaconda/ivp-pyspark/\)](https://github.com/izoda/anaconda-ivp-pyspark/)

IBM provides this installation verification program (IVP) to get started with the Anaconda and PySpark stacks of IzODA. After completing this IVP, you ensure that Anaconda and PySpark are installed

successfully and that you are able to run simple data analysis on mainframe data sources by using Spark dataframes.

Chapter 13. Verifying the z/OS IzODA Livy installation

z/OS IzODA Livy installation verification program.

Before you begin

Follow the instructions in [Part 3, “Customization,” on page 13](#) to customize your environment and Open Data Analytics for z/OS.

About this task

Complete the following steps to verify the successful installation and customization of z/OS IzODA Livy on your system.

Starting the Livy Server

1. If `livy.spark.master` is set to a Spark master URL in the `livy.conf` configuration file, start the corresponding Spark cluster.
2. Start an OMVS session with the user ID chosen as the Livy server user ID, and start the Livy server by issuing the following:

```
$LIVY_HOME/bin/livy-server start
```

A message is issued that indicates the Livy server has started, displaying the location of the Livy server log file.

3. If `livy.spark.master` is set to a Spark master URL in the `livy.conf` configuration file, verify the Livy server user ID has been set up correctly for submitting Spark applications to the Spark cluster via `spark-submit`. From the Livy server user ID OMVS session, issue the following:

```
$SPARK_HOME/bin/spark-submit --class  
org.apache.spark.examples.SparkPi --master spark://host-IP-  
addr:sparkMaster-port $SPARK_HOME/examples/jars/spark-  
examples_2.11-sparkVersion.jar
```

where:

\$SPARK_HOME

An environment variable that contains the installation path for z/OS Spark.

host-IP-addr

The IP address (xxx.xxx.xxx.xxx) for this z/OS system.

sparkMaster-port

The Spark Master port. The default is 7077.

SparkVersion

The z/OS Spark version number (2.4.8).

4. If the Livy server is protected by AT-TLS client authentication, import the PKCS#12 (.p12) certificate package extracted during the "Exporting Livy user certificates" section of [“Customizing z/OS IzODA Livy” on page 106](#) into a web browser. For instructions on importing PKCS#12 certificate package into the web browser, consult the Web browser's documentation.
5. Connect to the Livy server UI by navigating to this web address with a web browser:

`https://livyServer-host-IP-addr:livyServer-port/ui/` (or **`http://`** if the Livy server is not protected by AT-TLS client authentication)

where:

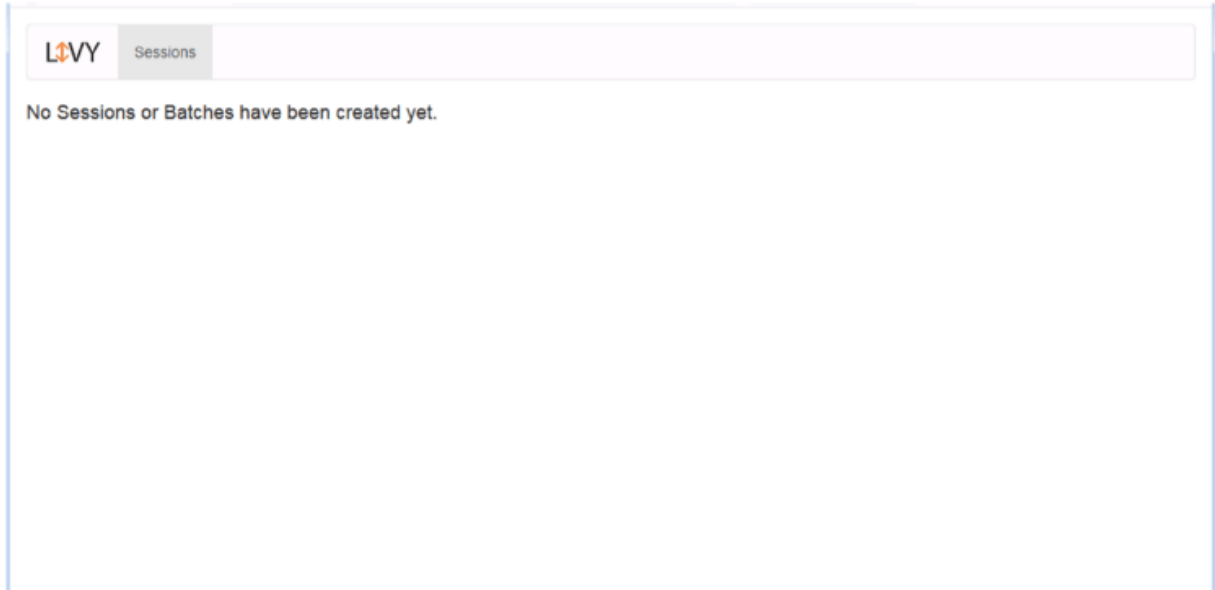
livyServer-host-IP-addr

The IP address (xxx.xxx.xxx.xxx) for this z/OS system.

livyServer-port

The Livy server port. The default is 8998.

6. Confirm the connection is successful and that the Livy server has started:



Performing session mode Livy job submission to the Spark cluster

An application or application code that can perform the HTTP POST, GET and DELETE operations is needed for performing session mode Livy job submission. This section shows how to do so from a remote system using Python with the Requests library.

This section makes the following assumptions:

- Default Livy server port (8998) is used.
 - The Livy server is protected by AT-TLS client authentication.
 - In the `livy.conf` configuration file, `livy.spark.master` is set to a Spark master URL and `livy.spark.deploy-mode` is set to client.
 - The end user's certificate package has been extracted into PEM format (`livyusr.pem`, `ca.pem` and `livyusrkey.pem`) and resides on the remote system.
1. On the remote system, start a Python interactive shell from the directory where the .pem files reside.
 2. Create a Livy session mode session:

```
import json, pprint, requests, textwrap
host = 'https://alpsxxx.pok.ibm.com:8998'
data = {'kind': 'spark'}
headers = {'Content-Type': 'application/json'}
r = requests.post(host + '/sessions',
data=json.dumps(data), headers=headers,
cert=('cert_sprkid5.pem', 'key_sprkid5.pem'), verify=False)
r.json()
```

The output is the following:

```
{'id': 0, 'name': None, 'appId': None, 'owner': None,
'proxyUser': None, 'state': 'starting', 'kind': 'spark',
'appInfo': {'driverLogUrl': None, 'sparkUiUrl': None},
'log': ['stdout: ', '\nstderr: ']}
```

3. Ensure that the session has completed starting up by checking that the state of the session has reached the "idle" phase.


```

session_url = host + r.headers['location']
r = requests.get(session_url, headers=headers)
cert=('livyusr.pem', 'livyusrkey.pem'), verify=False)
r.json()

```

The output is the following:

```

{'id': 0, 'name': None, 'appId': None, 'owner': None,
'proxyUser': None, 'state': 'idle', 'kind': 'spark',
'appInfo': {'driverLogUrl': None, 'sparkUiUrl': None},
'log': ['19/05/16 15:41:41 INFO BlockManager: Using
org.apache.spark.storage.RandomBlockReplicationPolicy for
block replication policy', '19/05/16 15:41:41 INFO
BlockManagerMaster: Registering BlockManager
BlockManagerId(driver, localhost, 57060, None)', '19/05/16
15:41:41 INFO BlockManagerMasterEndpoint: Registering block
manager localhost:57060 with 434.4 MB RAM,
BlockManagerId(driver, localhost, 57060, None)', '19/05/16
15:41:41 INFO BlockManagerMaster: Registered BlockManager
BlockManagerId(driver, localhost, 57060, None)', '19/05/16
15:41:41 INFO BlockManager: Initialized BlockManager:
BlockManagerId(driver, localhost, 57060, None)', '19/05/16
15:41:42 INFO SparkEntries: Spark context finished
initialization in 1842ms', '19/05/16 15:41:42 INFO
SharedState: Setting hive.metastore.warehouse.dir ('null')
to the value of spark.sql.warehouse.dir
('file:$LIVY_HOME/bin/spark-warehouse').", '19/05/16
15:41:42 INFO SharedState: Warehouse path is
'file:$LIVY_HOME/bin/spark-warehouse'.", '19/05/16 15:41:43
INFO StateStoreCoordinatorRef: Registered
StateStoreCoordinator endpoint', '19/05/16 15:41:43 INFO
SparkEntries: Created Spark session.']}

```

4. Pass in the Scala code via a Python dictionary data, with the key being code and the value being the code that is to be run in the session. The output may display the state as either waiting or running.

```

statements_url = session_url + '/statements'
data = {'code': '1 + 1'}
r = requests.post(statements_url, data=json.dumps(data),
headers=headers, cert=('livyusr.pem', 'livyusrkey.pem'),
verify=False)
r.json()

```

The output is the following:

```

{'id': 0, 'code': '1 + 1', 'state': 'waiting', 'output':
None, 'progress': 0.0}

```

5. Obtain the results. Repeat this step if the session is still in 'running' state and has not reached the 'available' state.

```

statement_url = host + r.headers['location']
r = requests.get(statement_url, headers=headers)
cert=('livyusr.pem', 'livyusrkey.pem'), verify=False)
pprint.pprint(r.json())

```

The output is the following:

```

{'code': '1 + 1',
'id': 0,
'output': {'data': {'text/plain': 'res0: Int = 2\n'},
'execution_count': 0,
'status': 'ok'},
'progress': 1.0,
'state': 'available'}

```

6. Close the session.

```

requests.delete(session_url, headers=headers,
cert=('livyusr.pem', 'livyusrkey.pem'), verify=False)

```

Performing batch mode Livy job submission to the Spark cluster

An application or application code that can perform the HTTP POST operation is needed for performing batch mode Livy job submission. This section shows how to do so from a Unix-based remote system using cURL (curl).

This section assumes the following:

- Default Livy server port (8998) is used.
 - The Livy server is protected by AT-TLS client authentication.
 - In the livy.conf configuration file, livy.spark.master is set to a Spark master URL and livy.spark.deploy-mode is set to client.
 - Using z/OS Spark version 2.4.8 at its default installation directory (/usr/lpp/IBM/izoda/spark/spark24x).
 - The end user's certificate package has been extracted into PEM format (livyusr.pem, ca.pem and livyusrkey.pem) and residing on the Unix-based remote system.
 - The end user's certificate PKCS#12 package (.12) has been imported into a web browser.
1. On the Unix-based remote system, go to the directory where the .pem files reside.
 2. Submit the SparkPi sample to the Livy server by issuing the following:

```
curl -X POST --verbose -k --cert ./usera.pem --cacert
./ca.pem --key ./livyusrkey.pem --data '{"file":
"/usr/lpp/IBM/izoda/spark/spark24x/examples/jars/spark-
examples_2.11-2.4.8.jar", "className":
"org.apache.spark.examples.SparkPi"}' -H "Content-Type:
application/json" https://xxx.xxx.xxx.xxx:8998/batches
```

3. Connect to the Livy server web UI with a web browser. Verify a batch session has been created and has reached the 'success' state eventually.



4. Click on 'session' link on the web UI and verify that the 'stdout' section of the log contains the "Pi is roughly" output message:

```
21/06/09 11:43:43 INFO TaskSchedulerImpl: Removed TaskSet 0.0, whose tasks have all completed, from pool
21/06/09 11:43:43 INFO DAGScheduler: ResultStage 0 (reduce at SparkPi.scala:38) finished in 10.087 s
21/06/09 11:43:43 DEBUG DAGScheduler: After removal of stage 0, remaining stages = 0
21/06/09 11:43:43 INFO DAGScheduler: Job 0 finished: reduce at SparkPi.scala:38, took 10.136992 s
Pi is roughly 3.141275706378532
```

Stopping the Livy server

1. From the Livy server user ID OMVS session, issue the following:

```
$LIVY_HOME/bin/livy-server stop
```

Part 5. Resource monitoring

Chapter 14. Resource monitoring for Apache Spark

Whether you use the Apache Spark configuration files or z/OS workload management (WLM) to manage your Spark resources, it is good practice to monitor the actual resource usage over time and fine tune your setup accordingly.

Spark web interfaces

Apache Spark provides a suite of web user interfaces (UIs) that you can use to monitor the status and resource consumption of your Spark cluster.

Apache Spark provides the following UIs:

- Master web UI
- Worker web UI
- Application web UI

The application web UI is particularly useful to identify potential performance issues with your Spark cluster.

Some sections of the web UIs only appear if relevant information is available. For instance, the master web UI omits the driver information section if a driver has not been running in the cluster.

You can access the Spark web UIs by pointing your web browser to:

```
http://host_IP:ui_port
```

where:

host_IP

The IP address for the z/OS system on which Spark runs

ui_port

The Spark web UI port number

For more information about where to find the port numbers, see [“Configuring networking for Apache Spark”](#) on page 37.

Note: The layout of the web UIs that are shown in the following examples are for Apache Spark 2.0.2. The UIs might look different for other releases of Apache Spark.

Master web UI

The master web UI provides an overview of the Spark cluster and displays the following information:

- Master URL and REST URL
- CPUs and memory available to the Spark cluster
- Worker status and its allotted resources
- Information about the active and completed applications, such as their status, allotted resources, and duration
- Information about the active and completed drivers, such as their status and allotted resources

You can use the master web UI to identify the amount of CPU and memory resources that are allotted to the Spark cluster and to each application. Spark sees each SMT-enabled zIIP as having two cores.

The master web UI also provides an overview of the applications. For instance, if an application is in the WAITING state, it likely does not have sufficient resources to run.

[Figure 8 on page 138](#) shows an example of the master web UI.



Spark Master at spark://xxx.xxx.xxx.xxx:7077

URL: spark://xxx.xxx.xxx.xxx:7077

REST URL: spark://xxx.xxx.xxx.xxx:6066 (cluster mode)

Alive Workers: 1

Cores in use: 16 Total, 16 Used

Memory in use: 14.7 GB Total, 2.0 GB Used

Applications: 1 Running, 0 Completed

Drivers: 1 Running, 0 Completed

Status: ALIVE

Workers

Worker Id	Address	State	Cores	Memory
worker-20170223143132-xxx.xxx.xxx.xxx-1082	xxx.xxx.xxx.xxx:1082	ALIVE	16 (16 Used)	14.7 GB (2.0 GB Used)

Running Applications

Application ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration
app-20170223143210-0000	(kill) Spark Pi	15	1024.0 MB	2017/02/23 14:32:10	WELLIE0	RUNNING	5 s

Running Drivers

Submission ID	Submitted Time	Worker	State	Cores	Memory	Main Class
driver-20170223143159-0000	(kill) Thu Feb 23 14:31:59 EST 2017	worker-20170223143132-xxx.xxx.xxx.xxx-1082	RUNNING	1	1024.0 MB	org.apache.spark.examples.SparkPi

Completed Applications

Application ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration
----------------	------	-------	-----------------	----------------	------	-------	----------

Completed Drivers

Submission ID	Submitted Time	Worker	State	Cores	Memory	Main Class
---------------	----------------	--------	-------	-------	--------	------------

Figure 8. The Apache Spark master web UI

Worker web UI

The worker web UI provides an overview of the executors and drivers that are spawned by the worker process. It displays information about the allotted resources, status, and provides links to log files for each of these child processes. You can use this web UI to see how many executors are currently running and the amount of resources allotted to each.

Figure 9 on page 139 shows an example of the worker web UI.

ID: worker-20170223143132-xxx.xxx.xxx.xxx-1082

Master URL: spark://xxx.xxx.xxx.xxx:7077

Cores: 16 (16 Used)

Memory: 14.7 GB (2.0 GB Used)

[Back to Master](#)

Running Executors (1)

ExecutorID	Cores	State	Memory	Job Details	Logs
0	15	RUNNING	1024.0 MB	ID: app-20170223143210-0000 Name: Spark Pi User: WELLIE0	stdout stderr

Running Drivers (1)

DriverID	Main Class	State	Cores	Memory	Logs	Notes
driver-20170223143159-0000	org.apache.spark.examples.SparkPi	RUNNING	1	1024.0 MB	stdout stderr	

Figure 9. The Apache Spark worker web UI

Application web UI

Each Spark application launches its own instance of the web UI. The application web UI provides a wealth of information about the Spark application and can be a useful tool to debug the application. This web UI has the following tabs:

- The **Jobs** tab displays a summary page of all jobs in the Spark application and a detailed page for each job. The summary page shows high-level information, such as the status, duration, and progress of all jobs and the overall event timeline. When you click on a job on the summary page, you see the detailed page for that job. The detailed page further shows the event timeline, DAG visualization, and all stages of the job.
- The **Stage** tab displays a summary page that shows the current state of all stages of all jobs in the Spark application, and, when you click on a stage, a detailed page for that stage. The details page shows the event timeline, DAG visualization, and all tasks for the stage.
- If the application has persisted RDDs, the **Storage** tab displays information about the RDDs. The summary page shows the storage levels, sizes and partitions of all RDDs, and the detailed page shows the sizes and using executors for all partitions in an RDD.
- The **Environment** tab displays the values for the different environment and configuration variables, including Java, Spark, and system properties.
- The **Executors** tab displays summary information about the executors that were created for the application, including memory and disk usage and task and shuffle information. The Storage Memory column shows the amount of memory used and reserved for caching data.
- If the application executes Spark SQL queries, the **SQL** tab displays information, such as the duration, jobs, and physical and logical plans for the queries.
- The web UI includes a **Streaming** tab if the application uses Spark streaming. This tab displays scheduling delay and processing time for each micro-batch in the data stream, which can be useful for troubleshooting the streaming application.

From a resource monitoring perspective, the **Executors** tab provides information about the amount of memory, disk, and cores used by each executor for the application.

From a performance monitoring perspective, the **Executors** tab displays garbage collection (GC) time and shuffle information for each executor. The **Storage** tab displays the percentage of RDD data that can be cached in memory.

Figure 10 on page 140 shows an example of the worker web UI.

Executors

Summary

	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write
Active(2)	2	2.3 KB / 868.8 MB	0.0 B	15	15	0	4601	4616	7.6 m (12.7 s)	0.0 B	0.0 B	0.0 B
Dead(0)	0	0.0 B / 0.0 B	0.0 B	0	0	0	0	0	0 ms (0 ms)	0.0 B	0.0 B	0.0 B
Total(2)	2	2.3 KB / 868.8 MB	0.0 B	15	15	0	4601	4616	7.6 m (12.7 s)	0.0 B	0.0 B	0.0 B

Executors

Executor ID	Address	Status	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Logs	Thread Dump
0	xx.xx.xx.xx:1093	Active	1	1168.0 B / 434.4 MB	0.0 B	15	15	0	4601	4616	7.6 m (12.7 s)	0.0 B	0.0 B	0.0 B	stdout stderr	Thread Dump
driver	xx.xx.xx.xx:1089	Active	1	1168.0 B / 434.4 MB	0.0 B	0	0	0	0	0	0 ms (0 ms)	0.0 B	0.0 B	0.0 B		Thread Dump

Figure 10. The Apache Spark application web UI

Configuring Spark web interfaces

Complete this task to configure and customize the Spark web interfaces.

About this task

The Apache Spark web interfaces can be configured and customized by a number of configuration properties. For more information about Apache Spark web interfaces, see “Spark web interfaces” on page 137.

Table 10. Spark UI configurations

Configuration name	Default	Configuration property	Notes
Spark UI kill	True	spark.ui.killEnabled	Allows jobs and stages to be killed from the web UI

Spark web interface kill button

By default, Spark provides the ability to kill applications, jobs, and stages through the web interface of the Spark master and Spark application. This ability is available to anyone that can access the web interface of the respective process. Depending on your required configuration, you may want to control if this is enabled within your Spark environment.

Depending on which Spark process web interface page you are viewing, the kill button appears in different locations. Within the Spark master web interface, the kill button appears on the Running Applications table and the Running Drivers table. Within the Spark application web interface, the kill button appears on the Active Jobs table, on the Jobs tab, and the Active Stages table, on the Stages tab.

This feature can be disabled by updating your **spark-defaults.conf** with the property, **spark.ui.killEnabled** and setting it to false. If set to false while starting the Spark master, the kill button will not appear on the Spark master web interface and you will no longer be able to kill a running application or running driver. If set to false while starting a Spark application, either through **spark-submit** or **spark-shell**, the kill button will not appear on the Spark application web interface and you will no longer be able to kill an active job or active stage.

Securing Spark web interfaces

Complete this task to secure Spark web interfaces.

About this task

The Apache Spark web interfaces can be secured with https/SSL by way of Spark SSL settings. For more information about Apache Spark web interfaces, see [“Spark web interfaces”](#) on page 137.

Procedure

1. Generate a public-private key pair. Then, wrap the public key in a digital certificate, and store the private key and the certificate in a keystore. The following example uses the Java `keytool` tool to generate a self-signed certificate.

```
keytool -genkeypair -keystore /u/sparkid/.keystore \  
-keyalg RSA -alias selfsigned \  
-dname "CN=myparkcert L=Poughkeepsie S=NY C=US" \  
-storepass examplestorepass -keypass examplekeypass
```

2. Export the generated certificate and import it into a Java truststore. The following example again uses the Java `keytool` tool.

```
keytool -exportcert -keystore /u/sparkid/.keystore \  
-alias selfsigned -storepass examplestorepass -file test1.cer  
  
keytool -importcert -keystore /u/sparkid/.truststore \  
-alias selfsigned \  
-storepass examplestorepass -file test1.cer -noprompt
```

3. Update the `spark-defaults.conf` file to enable SSL for Spark WebUI, by using the keystore and truststore that is setup in the previous steps.

```
spark.ssl.enabled           true  
spark.ssl.trustStore        /u/sparkid/.truststore  
spark.ssl.trustStorePassword examplestorepass  
spark.ssl.keyStore          /u/sparkid/.keystore  
spark.ssl.keyStorePassword examplestorepass  
spark.ssl.keyPassword       examplekeypass  
spark.ssl.protocol          TLSv1.2
```

4. Start your Spark cluster as normal. When you point your web browser to the Spark web interface, it automatically redirects to the SSL port, which is typically the non-SSL port plus 400. For example, `http://127.0.0.1:8080` would be directed to `https://127.0.0.1:8480`.

You can also use the `spark.ssl.ui.port` option to set the SSL port for the Spark web UI. The `spark.ssl.ui.port` option can be specified in `spark-defaults.conf`.

Note: If you are using a self-signed certificate, like the one in the previous example, you might need to install the certificate in your web browser. Self-signed certificates are generally rejected by web browsers, since they are not signed by a known certificate authority and therefore not trusted.

Results

The specified Spark web interfaces are secure.

Event log directory and file permissions

Complete this task to set permissions for your Apache Spark event logs for future reading through the Spark history service.

About this task

The event log directory stores the event logs when enabled for your applications. These files are then used by the Spark history server to reconstruct the application's web UI.

Procedure

Complete the following steps to enable event logging and point the history server to the event logs.

1. Create the event log directory:

```
mkdir -p /var/spark/events
```

2. Ensure the owner and group of the directory correspond to the user who is running the Spark history server:

```
chown SPARKID:SPKGRP /var/spark/events
```

3. Ensure the permissions of the event log directory are set properly:

```
ls -l /var/spark/events
```

4. Change permissions of the event log directory if necessary. Apache Spark advises a default of 777:

```
chmod ### /var/spark/events
```

5. Set the sticky bit on the event log directory. This permits users to delete only the files from this directory which they own:

```
chmod +t /var/spark/events
```

6. Add the following statement to your `spark-defaults.conf` file, where `###` is the permissions you'd like the application to create the file with. Default is 770:

```
spark.zos.app.eventLog.permission ###
```

Enabling the Spark history service

Complete this task to enable the Apache Spark history service to view information about completed applications.

About this task

The information that the application web UI displays is, by default, only available when the application is active. To view the information after an application completes, you must enable event logging before starting the application. The Spark history server can then use the event logs to reconstruct the application's web UI.

Procedure

Complete the following steps to enable event logging and point the history server to the event logs.

1. Add the following statements to your `spark-defaults.conf` file:

```
spark.eventLog.enabled      true
spark.eventLog.dir          event-log-directory
spark.history.fs.logDirectory event-log-directory
```

where *event-log-directory* is the directory you configured in “Event log directory and file permissions” on page 142, where each Spark application stores its event log files, such as `file:///var/spark/events`.

Important: Use the event log directory only for event logging. Unexpected errors can occur if Spark finds non-event log files in the event log directory.

2. To start the history server, issue the following command from bash:

```
$SPARK_HOME/sbin/start-history-server.sh
```

where **SPARK_HOME** is an environment variable that contains the installation path for z/OS Spark. After the history server is started, you can use your browser to access the event logs and reconstructed application web UI for completed applications.

3. Point your web browser to the history server URL:

```
http://host_IP:history_server_port
```

where:

host_IP

The IP address for the z/OS system on which Spark runs.

history_server_port

The Spark history server UI port number. The default port number is 18080.

Figure 11 on page 143 shows an example of the history server web UI.



Figure 11. The Spark history server web UI

4. Click an entry in the **App ID** column of the history server web UI to see the reconstructed application web UI for that application.
5. To stop the history server, issue the following command from bash:

```
$SPARK_HOME/sbin/stop-history-server.sh
```

Spark log files

Apache Spark log files can be useful in identifying issues with your Spark processes.

Table 11 on page 143 lists the base log files that Spark generates.

Table 11. Apache Spark log files

Log file	Location
Master logs	<code>\$SPARK_LOG_DIR/spark-userID-org.apache.spark.deploy.master.Master-instance-host.out</code>
Worker logs	<code>\$SPARK_LOG_DIR/spark-userID-org.apache.spark.deploy.master.Worker-instance-host.out</code>

Table 11. Apache Spark log files (continued)

Log file	Location
Driver logs (client deploy mode)	Printed on the command line by default
Driver logs (cluster deploy mode)	
• stdout	• \$SPARK_WORKER_DIR/ <i>driverID</i> /stdout
• stderr	• \$SPARK_WORKER_DIR/ <i>driverID</i> /stderr
Executor logs	
• stdout	• \$SPARK_WORKER_DIR/ <i>applID</i> / <i>executorID</i> /stdout
• stderr	• \$SPARK_WORKER_DIR/ <i>applID</i> / <i>executorID</i> /stderr

In the Location column:

userID

The user ID that started the master or worker.

instance

The master or worker instance number.

host

The short name of the host on which the master or worker is started

driverID

The ID of the driver. You can find this ID on the application web UI.

executorID

The ID of the executor. You can find this ID on the application web UI.

You can customize the Spark directories shown in the table. For more information about the purpose of these directories and their suggested locations, see [“Creating the Apache Spark working directories” on page 35](#).

Tip: It is a good practice to periodically clean up or archive your Spark directories to avoid errors caused by low file system space. For instance, you can use the z/OS UNIX shell command, **skulker**, in a regularly scheduled tool to remove old files in a directory.

Using RMF to monitor Spark workload

The z/OS Resource Measurement Facility (RMF) is a tool for z/OS performance measurement and management. RMF collects performance data for z/OS and Parallel Sysplex environments and generates reports that allow you to monitor and tune your systems according to your business needs.

RMF uses three monitors to gather data:

- Short-term data collection with Monitor III
- Snapshot monitoring with Monitor II
- Long-term data gathering with Monitor I and Monitor III

The system operator starts all monitors as background sessions with a variety of options that determine the type of data to collect and where to store it. For more information about setting up the RMF monitors, see [z/OS RMF User's Guide](#).

All three RMF monitors create reports, as does the Postprocessor. The following topics provide an overview of the reports that are most relevant to monitoring a Spark workload. For complete information about RMF reports, see [z/OS RMF Report Analysis](#).

Interactive performance reports with Monitor III

The interactive Monitor III reporter runs in a TSO/E session under ISPF and provides system or sysplex performance reports in the following ways:

- Displays your current system status in real-time mode

- Shows previously collected data that is still available in either in-memory buffers or pre-allocated VSAM data sets

You can use Monitor III to quickly identify storage and processor delays for your active Spark workload. To start an interactive Monitor III session, enter the TSO/E command **RMF** and select **Monitor III** from the RMF - Performance Management panel. From the RMF III Primary Menu, you can select the specific performance metrics that you want to see. To further filter the report by job class and service class, you can issue the following command:

```
report_name job_class,service_class
```

where:

report_name

The short name of the report

job_class

One of the following job class names:

- ALL (or A)
- ASCH (or AS)
- BATCH (or B)
- OMVS (or O)
- STC (or S)
- TSO (or T)

service_class

The service class name

Example: To get the Storage Delays report for the ODASSC1 OMVS service class, enter this command:

```
STOR 0,ODASSC1
```

The following Monitor III reports are of particular interest for monitoring Spark workloads:

- Storage Delays report
- Common Storage report
- Storage Frames report
- Storage Memory Objects report
- Processor Delays report
- Processor Usage report
- zFS File System report

Based on the performance measurements that you observe from these reports, you can fine tune the resource assignments for your Spark workload. For instance, you can modify the number of cores and amount of memory for your executors in the `spark-defaults.conf` configuration file. (For more information, see [“Configuring memory and CPU options” on page 65.](#)) Or, if you use WLM to manage your Spark workload, you can adjust the importance and performance goals of your Spark workload. (For more information, see [“Configuring z/OS workload management for Apache Spark” on page 73.](#))

Storage Delays report

The Storage Delays report (STOR) displays storage delay information for all jobs. Here you can find out if your Spark jobs suffer any delays due to memory constraints. A non-zero value in the DLY % column indicates that there is a delay due to memory constraints. [Figure 12 on page 146](#) shows an example of this report.

RMF V2R2		Storage Delays							Line 1 of 7	
Samples: 60		System: SYS1		Date: 03/13/17		Time: 15.55.00		Range: 60		Sec
Jobname	C	Service Class	DLY %	COMM	LOCL	SWAP	OUTR	OTHR	-- Working Set --	Central Expanded
ODASW1A	0	ODASSC1	0	0	0	0	0	0	36823	
ODASM1A	0	ODASSC1	0	0	0	0	0	0	38759	
ODASX1A	0	ODASSC1	0	0	0	0	0	0	183K	

Figure 12. Example of the RMF Storage Delays report

Common Storage report

The Common Storage report (STORC) provides information about the use of common storage (CSA, ECSA, SQA, and ESQA) within a system. You can use this report to identify whether Spark is using an excessive amount of common storage (such as for memory-mapped files). [Figure 13 on page 146](#) shows an example of this report.

RMF V2R2					Common Storage					Line 323 of 340			
Samples: 60		System: SYS1		Date: 03/13/17		Time: 22.42.00		Range: 60		Sec			
System Information IPL Definitions Peak Allocation Values Average CSA to SQA Conversion Average Use Summary Available at End of Range Unalloc Common Area: 5028K					---- Percent ----				----- Amount -----				
					CSA	ECSA	SQA	ESQA	CSA	ECSA	SQA	ESQA	
									1856K	501M	4384K	63M	
					21	25	28	133	393K	124M	1248K	84M	
					0	7			0	34M			
					21	25	14	131	385K	124M	618K	83M	
					79	75	86	23	1471K	377M	3766K	15M	
Service					ELAP	-- Percent Used -				----- Amount Used -----			
Jobname	Act	C	Class	ASID	Time	CSA	ECSA	SQA	ESQA	CSA	ECSA	SQA	ESQA
ODASX1A		0	ODASSC1	0329	12.1M	0	0	0	0	0	0	0	160
ODASW1A		0	ODASSC1	0326	8.3H	0	0	0	0	0	0	0	160
ODASM1A		0	ODASSC1	0326	8.3H	0	0	0	0	0	0	0	160

Figure 13. Example of the RMF Common Storage report

Storage Frames report

The Storage Frames report (STORF) displays detailed frame counts, auxiliary slot count, and page-in rate for each address space. For instance, it tells you the average number of frames used by each Spark process (ACTV column) and the paging rate (PGIN RATE column). Keeping the paging rate as close to zero as possible helps improve performance. For instance, increasing the memory limit for the resource group with which Spark address spaces are associated may help lower the paging rate. [Figure 14 on page 146](#) shows an example of this report.

RMF V2R2				Storage Frames						Line 1 of 7	
Samples: 60		System: SYS1		Date: 03/13/17		Time: 15.55.00		Range: 60		Sec	
Jobname	C	Service Class	Cr	-- Frame Occup.--	- Active Frames -	AUX	PGIN				
				TOTAL	ACTV	IDLE	WSET	FIXED	DIV	SLOTS	RATE
ODASX1A	0	ODASSC1		183K	183K	0	183K	716	0	0	0
ODASM1A	0	ODASSC1		38759	38759	0	38759	669	0	0	0
ODASW1A	0	ODASSC1		36823	36823	0	36823	614	0	0	0

Figure 14. Example of the RMF Storage Frames report

Storage Memory Objects report

The Storage Memory Objects report (STORM) displays information about the use of memory objects for each active address space and within the system. A memory object is a contiguous range of virtual

addresses that is allocated by jobs in units of megabytes on a megabyte boundary. This report can help you assess the total amount of memory that Spark is using. It also shows the fixed and pageable 1M frames used by Spark address spaces. Spark generally does not require the use of fixed large frames, and it might have a negative impact on the overall system health if Spark JVMs are tuned to use them. [Figure 15 on page 147](#) shows an example of this report.

RMF V2R2				Storage Memory Objects				Line 1 of 7				
Samples: 60		System: SYS1		Date: 03/13/17		Time: 15.55.00		Range: 60		Sec		
----MemObj----		---Frames---		--1MB Fixed--		--2GB Fixed--		-1MB Pageable-				
Fixed 1M	10	Shared	424K	Total	111K	Total	12	Initial	22056			
Fixed 2G	1	Common	280K	Common	13	%Used	8.3	Dynamic	4109			
Shared	28	%Used	39.2	%Used	3.9	%Used 100						
Common	631											

Jobname	C	Service Class	ASID	Total	Comm	Shr	Fixed	Pgable	Fixed	Total	Comm	Shr
ODASM1A	0	ODASSC1	0618	351	0	0	0	69	0	13.7G	0	0
ODASX1A	0	ODASSC1	0634	347	0	0	0	541	0	45.8G	0	0
ODASW1A	0	ODASSC1	0345	334	0	0	0	65	0	13.1G	0	0

Figure 15. Example of the RMF Storage Memory Objects report

Processor Delays report

The Processor Delays report (PROC) displays all jobs that were waiting for or using the processor during the reporting interval. Here you can see if your Spark jobs suffer any delays due to processor constraints. [Figure 16 on page 147](#) shows an example of this report.

RMF V2R2		Processor Delays				Line 1 of 10	
Samples: 60	System: SYS1	Date: 03/13/17	Time: 15.55.00	Range: 60	Sec		
Jobname	CX	Service Class	CPU Type	DLY %	USG %	EAppl %	----- Holding Job(s) -----
ODASX1A	0	ODASSC1	CP	3	2	0.360	% Name % Name % Name
			IIP	0	23	24.30	
ODASM1A	0	ODASSC1	CP	2	0	0.000	
			IIP	2	0	0.012	
ODASW1A	0	ODASSC1	CP	2	0	0.000	
			IIP	0	0	0.023	

Figure 16. Example of the RMF Processor Delays report

Processor Usage report

The Processor Usage report (PROCU) displays all jobs that were using a general-purpose or special-purpose processor during the reporting interval. You can use this report to understand the CPU usage of your Spark jobs. Combined with the Processor Delay report, you can assess whether you need to change the performance goals or importance of your Spark workload. [Figure 17 on page 147](#) shows an example of this report.

RMF V2R2			Processor Usage				Line 1 of 5	
Samples: 60		System: SYS1	Date: 03/13/17	Time: 15.55.00	Range: 60	Sec		
Jobname	CX	Service Class	--- Time Total	on CP AAP	% --- IIP	----- EAppl % CP AAP	----- IIP	
ODASX1A	0	ODASSC1	0.360	0.000	0.218	0.360	24.30	
ODASW1A	0	ODASSC1	0.000	0.000	0.000	0.000	0.023	
ODASM1A	0	ODASSC1	0.000	0.000	0.000	0.000	0.012	

Figure 17. Example of the RMF Processor Usage report

zFS File System report

The zFS File System report (ZFSFS) measures zFS activity on the basis of single file systems. With this report, you can monitor the I/O rates and response times associated with the file systems that Spark uses. [Figure 18 on page 148](#) shows an example of this report.

RMF V2R2		zFS File System - SVPLEX3				Line 23 of 46			
Samples: 120	Systems: 4	Date: 03/09/17	Time: 15.07.00	Range: 120	Sec				
-----	File System Name	-----				I/O	Resp	Read	XCF
	System	Owner	Mode	Size	Usq%	Rate	Time	%	Rate
OMVSSPA.SPARK.PLX3.ZFS									
*ALL	D0	R0	1440M	60.5	0.000	0.000	0.0	0.000	

Figure 18. Example of the RMF zFS File System report

Long-term reporting with the Postprocessor

You can use the RMF Postprocessor to generate long-term overview reports for your Spark workload. Unlike the interactive Monitor III sessions, Postprocessor reports allow you to analyze your Spark workload performance over the long term and fine tune its resource assignments. For more information about running the Postprocessor and other report options, see [z/OS RMF User's Guide](#).

Workload Activity report

The Workload Activity report (WLMGL) presents a summarized view of a WLM-managed workload. You can select to view this report for a specific service class, report class, WLM workload, and various other types. For instance, if you place all of your Spark processes in a single service class, you can generate a Workload Activity report that tells you the actual amount of memory and CPU consumed by that workload. [Figure 19 on page 148](#) shows an example of the Workload Activity report for the ODASSC1 service class.

WORKLOAD ACTIVITY														PAGE 48	
z/OS V2R2		SYSPLX ZPETPLX2		DATE 02/13/2017		INTERVAL 30.00.057		MODE = GOAL							
		RPT VERSION V2R2 RMF		TIME 17.29.35											
REPORT BY: POLICY=WLMPOLO1		WORKLOAD=ODASWL		SERVICE CLASS=ODASSC1		RESOURCE GROUP=ODASRG									
				CRITICAL =NONE		HONOR PRIORITY=NO									
				DESCRIPTION =Sample Spark service class											
-TRANSACTIONS- TRANS-TIME HHH.MM.SS.TTT --DASD I/O-- ---SERVICE--- SERVICE TIME ---APPL %--- --PROMOTED-- ---STORAGE---															
AVG	4.19	ACTUAL	716	SSCHRT	5.5	IOC	202338	CPU	276.626	CP	0.57	BLK	0.000	AVG	167913.4
MPL	4.19	EXECUTION	715	RESP	2.3	CPU	16393K	SRB	2.816	AAPCP	0.00	ENQ	0.000	TOTAL	702887.3
ENDED	441	QUEUED	1	CONN	2.2	MSO	1972M	RCT	0.039	IIPCP	0.14	CRM	0.000	SHARED	0.00
END/S	0.24	R/S AFFIN	0	DISC	0.0	SRB	166881	IIT	0.228			LCK	3.098		
#SWAPS	135	INELIGIBLE	0	Q+PEND	0.1	TOT	1989M	HST	0.000	AAP	N/A	SUP	0.000	-PAGE-IN RATES-	
EXCTD	0	CONVERSION	0	IOSQ	0.0	/SEC	1105K	AAP	N/A	IIP	20.85			SINGLE	0.0
AVG ENC	0.00	STD DEV	1.195			ABSRPTN	264K							BLOCK	0.0
REM ENC	0.00					TRX SERV	264K							SHARED	0.0
MS ENC	0.00													HSP	0.0
TRANSACTION APPL% : TOTAL : CP 0.55 AAP/IIP ON CP 0.14 AAP/IIP 20.85															
MOBILE : CP 0.00 AAP/IIP ON CP 0.00 AAP/IIP 0.00															

Figure 19. Example of the RMF Workload Activity report for a Spark service class

Figure 19. Example of the RMF Workload Activity report for a Spark service class

[Table 12 on page 149](#) describes some of the fields in the Workload Activity report that are more relevant to Spark.

Table 12. Selected fields in the RMF Workload Activity report

Field heading	Entries
SERVICE TIME	CPU Time, in seconds, consumed on general-purpose and special-purpose processors. IIT I/O interrupt time, in seconds. IIP zIIP service time, in seconds.
APPL %	CP Percentage of the processor time used by transactions running on general-purpose processors. IIPCP Percentage of the processor time used by zIIP-eligible transactions running on general-purpose processors. This is a subset of the APPL % CP value. IIP Percentage of the processor time used by transactions executed on zIIPs.
STORAGE	AVG Weighted average number of central storage frames allocated to active ASIDs. TOTAL Total number of central storage frames allocated to resident ASIDs.
PAGE-IN RATES	SINGLE Average rate at which pages are read into central storage while transactions are resident in central storage. BLOCK Rate of demand page-ins from DASD for blocked pages.

Using z/OS and z/OS UNIX commands to monitor Spark workload

You can use z/OS system commands and z/OS UNIX shell commands to monitor your Apache Spark workload, especially its usage of system resources.

You can incorporate the system commands or shell commands into automated tools as part of the regular system checkup, or you can issue the commands directly to diagnose a particular problem.

z/OS system commands to monitor Spark workload

Table 13 on page 150 lists some z/OS system commands that you can use to monitor Spark workload. For more information about these commands, see [z/OS MVS System Commands](#). For more information about the response to the DISPLAY OMVS command, refer to the response message IDs in [z/OS MVS System Messages, Vol 3 \(ASB-BPX\)](#).

Table 13. z/OS system commands to monitor Spark workload

z/OS system command	Description
DISPLAY OMVS,LIMITS[,PID= <i>process_id</i>]	<p>Displays information about current z/OS UNIX parmlib limits, their high-water marks, and current system usage. When the PID parameter is specified, LIMITS displays high-water marks and current usage for an individual process.</p> <p>Although you should monitor the usage of all system resources, Spark workload particularly impacts these limits:</p> <ul style="list-style-type: none"> • MAXMMAPAREA, the maximum number of data space pages that can be allocated for memory mapping of z/OS files • MAXPROCSYS, the maximum number of processes that can be active at the same time • MAXPIPES, the maximum number of named or unnamed pipes that can be oped in the system at one time
DISPLAY OMVS,PIPES[, {ALL UID= <i>uid</i> }]	Displays summary information about the z/OS UNIX pipe usage. The default is to list the two UIDs with the highest pipe create count. If you specify ALL, all UIDs with a pipe create count are displayed.
DISPLAY OMVS,U= <i>userid</i>	Displays process information for all processes associated with the specified TSO/E user ID. If you have a dedicated user ID under which all Spark processes run, you can use this command to display information about all of those processes.

z/OS UNIX shell commands to monitor Spark workload

Table 14 on page 150 lists some z/OS UNIX shell commands that you can use to monitor Spark workload. For more information about these commands, see [z/OS UNIX System Services Command Reference](#). For more information about the **zfsadm** command suite, see [z/OS File System Administration](#).

Table 14. z/OS UNIX shell commands to monitor Spark workload

z/OS UNIX shell command	Description
df [-kPStv] [<i>file</i> ...]	Displays the amount of free space left on a file system. You can use this command to monitor free space left on file systems that Spark uses.
skulker [-iw] [-r -R] [-l <i>logfile</i>] <i>directory days_old</i>	Finds and deletes old files in <i>directory</i> , based on comparing the file's access time to the age specified by <i>days_old</i> .
zfsadm fsinfo -exceptions	Lists all file systems that are low on space or that had applications fail due to low space errors.
zlsnf -t [-p <i>pid</i>] [-u <i>user</i>]	Shows a tally of all open files, sockets, and pipes, filtered by process ID (<i>pid</i>) or <i>user</i> , if specified. You can use this command to monitor different file types opened by Spark.

Table 14. z/OS UNIX shell commands to monitor Spark workload (continued)

z/OS UNIX shell command	Description
<code>ulimit -a</code>	Displays the resource limits on processes created by the user. For instance, you can use this command to display the file descriptor limit of your Spark processes.

Using IBM Health Checker for z/OS to monitor Spark workload

IBM Health Checker for z/OS is an MVS component that identifies potential problems before they impact your availability or cause outages.

Health Checker checks the current, active z/OS and sysplex settings and definitions for a system and compares the values to those suggested by IBM or defined by you. It produces output in the form of detailed messages to inform you of potential problems and suggested actions.

Consider enabling the z/OS UNIX System Services health checks and other system-level checks that might be relevant to your Spark workload. The following health checks might be of particular interest when running Spark workload:

- RACF_UNIX_ID
- RSM_MEMLIMIT
- RSM_AFQ IEA_ASIDS
- USS_AUTOMOUNT_DELAY
- USS_FILESYS_CONFIG
- USS_MAXSOCKETS_MAXFILEPROC
- USS_CLIENT_MOUNTS
- USS_KERNEL_PVTSTG_THRESHOLD
- USS_KERNEL_STACKS_THRESHOLD
- VSM_CSA_THRESHOLD
- VSM_SQA_THRESHOLD

For more information about these health checks and about Health Checker in general, see [*IBM Health Checker for z/OS User's Guide*](#).

Part 6. Troubleshooting

Chapter 15. Troubleshooting issues with Apache Spark

Use the following information to troubleshoot issues you might encounter with Apache Spark.

You can also use Apache Spark log files to help identify issues with your Spark processes. See [“Spark log files”](#) on page 143 for more information about where to find these log files.

Multiple Spark applications cannot run simultaneously with the "alwaysScheduleApps" setting enabled

Symptom: Multiple Spark applications cannot run simultaneously with the "alwaysScheduleApps" setting enabled, even when there is sufficient memory and CPU (zIIP) resources:

```
21/02/09 15:34:10 INFO Master: Attempted to re-register application at same address:
dipn.ipc.us.aexp.com:4056
```

The following are examples of repeating Spark messages in this instance:

```
21/02/09 15:34:19 WARN Master: Unknown application app-20210209153410-0352 requested 1 total
executors.
21/02/09 15:34:19 WARN Master: Unknown application app-20210209153409-0351 requested 1 total
executors.
21/02/09 15:34:20 WARN Master: Unknown application app-20210209153407-0350 requested 1 total
executors.
21/02/09 15:34:20 WARN Master: Unknown application app-20210209153410-0352 requested 1 total
executors.
```

Spark issues the following messages when the second and subsequent applications try to use the same port as the first application:

```
21/01/15 10:01:23 DEBUG TransportServer: Shuffle server started on port: 4056
21/01/15 10:01:23 INFO Utils: Successfully started service 'sparkDriver' on port 4056.
```

Cause: Spark expects to find ports in use if they are already being used by another Spark application, and has its own error handling that moves to the next port in the Spark port range. Using SHAREPORT defeats that logic and causes applications to interfere with each other.

Response: Do not use SHAREPORT when assigning TCPIP PORT definitions to Spark.

Spark commands fail with an EDC5111I message, and ICH408I message appears on the z/OS console

Symptom: Spark worker daemon fails to create executors with the following error:

```
18/01/22 13:11:14 ERROR ExecutorRunner: Error running executor
java.io.IOException: Cannot run program "/usr/lpp/java/java800/J8.0_64/bin/java"
(in directory"/u/usr1/work/app-20180122131112-0000/0"): EDC5111I Permission
denied.
    at java.lang.ProcessBuilder.start(ProcessBuilder.java:1059)
    at
org.apache.spark.deploy.worker.ExecutorRunner.org$apache$spark$deploy$worker
$ExecutorRunner$$fetchAndRunExecutor(ExecutorRunner.scala:167)
    at org.apache.spark.deploy.worker.ExecutorRunner$$anon$1.run
(ExecutorRunner.scala:73)
Caused by: java.io.IOException: EDC5111I Permission denied.
    at java.lang.UNIXProcess.<init>(UNIXProcess.java:189)
    at java.lang.ProcessImpl.start(ProcessImpl.java:167)
    at java.lang.ProcessBuilder.start(ProcessBuilder.java:1040)
    ... 2 more
```

The following message appears on the z/OS console:

```

SY1 ICH408I USER(USR1 ) GROUP(SYS1 ) NAME(#####)
/u/usr1/work/app-20180122131112-0000/0
CL(DIRSRCH ) FID(E2D7D2F0F10002000000014BA847EA)
INSUFFICIENT AUTHORITY TO CHDIR
ACCESS INTENT(--X) ACCESS ALLOWED(GROUP ---)
EFFECTIVE UID(0000000012) EFFECTIVE GID(0000000500)

```

Cause: When z/OS Spark client authentication is enabled, the Spark executor processes run under the user ID of the driver. However, the z/OS system that hosts the Spark cluster is not configured to accept ACL's set by Spark, which is needed for the executors to access Spark directories.

Response: Configure the z/OS system that hosts the Spark cluster to accept ACL's. For example, issue the following RACF command:

```

SETROPTS CLASSACT(FSSEC)

```

For more information, see [“Configuring z/OS Spark client authentication” on page 41](#).

Spark scripts fail with FSUM6196 and EDC5129I messages

Symptom: Spark scripts fail with the following error message:

```

env: FSUM6196 bash: not executable: EDC5129I No such file or directory

```

Cause: Apache Spark expects to find the bash shell in the user's **PATH** environment variable, but bash cannot be found when the spark-script attempts to invoke bash.

Response: Ensure that the **PATH** environment variable includes the directory where the bash executable is installed. For example, if bash is located at /usr/bin/bash-4.2/bin/bash, ensure that **PATH** includes /usr/bin/bash-4.2/bin.

Spark scripts fail with FSUM7332 message

Symptom: Spark scripts fail with the following error message:

```

failed to launch org.apache.spark.deploy.master.Master:
/usr/lpp/IBM/izoda/spark/spark23x/bin/spark-class 76:
FSUM7321 Unknown option "posix"

```

Cause: Apache Spark expects to find the **env** command in /usr/bin, but it cannot be found. Either the /usr/bin/env symbolic link is missing or it is not pointing to /bin/env. It is possible that creation of this symbolic link was missed during Spark setup or that the symbolic link was lost after a system IPL.

Response: Ensure that /usr/bin/env exists and is a symbolic link to /bin/env, and that the symbolic link persists across system IPLs. For more information, see [“Verifying the env command path” on page 32](#).

An error occurs when starting the master, but the master starts correctly

Symptom: When starting the master, the following error occurs:

```

bash-4.2$ $SPARK_HOME/sbin/start-master.sh
starting org.apache.spark.deploy.master.Master,
logging to /u/user/Billy/logs/spark--org.apache.spark.deploy.master.Master-1-ALPS.out
failed to launch org.apache.spark.deploy.master.Master:
full log in /u/user/Billy/logs/spark--org.apache.spark.deploy.master.Master-1-
ALPS.out

```

Cause: Apache Spark polls for a number of seconds to repeatedly check to see if the master started successfully. If your system is under heavy load, this message might appear, but it generally means that the check finished polling for success before the master startup completed.

Response: Check the master log, or issue the **ps** command and look for the master process to definitively see whether or not the master started successfully.

Only one Spark executor is started

Symptom: You specified `--num-executors 2` on a **spark-submit**, but only one executor was started.

Cause: The `--num-executors` parameter is only valid in YARN mode, which is not supported on z/OS. Instead, the number of executors is determined by your resource settings.

Response: For more information about resource settings, see [“Configuring memory and CPU options”](#) on page 65.

Shell script displays unreadable characters

Symptom: When running a shell script, it displays unreadable characters on the screen, such as:

```
./start-master.sh: line 1: syntax error near  
unexpected token '$'\101\123\124\105\122^''
```

Cause: Incorrect file encoding or downlevel bash shell.

Response: Ensure that the file encoding is in EBCDIC, not ASCII, and is not tagged as ASCII. You can check the tagging of a file by issuing the **ls -T** shell command. Also, ensure that your bash shell level is 4.2.53 or 4.3.48. You can check the bash level by issuing the **bash -version** command.

Spark-shell fails with java.lang.ExceptionInInitializerError error message

Symptom: Spark-shell fails with the following error message:

```
java.lang.ExceptionInInitializerError ... Scala signature  
package has wrong version expected: 5.0 found: 45.0 in scala.package
```

Cause: Your JVM is likely running with the wrong default encoding.

Response: Ensure that you have the following environment variable set:

```
IBM_JAVA_OPTIONS=-Dfile.encoding=UTF8
```

For more information about setting environment variables, see [“Setting up a user ID for use with z/OS Spark”](#) on page 28.

The Spark master fails with JVMJ9VM015W error

Symptom: The Spark master fails to start and gives the following error:

```
JVMJ9VM015W Initialization error for library j9gc28(2):  
Failed to instantiate compressed references metadata; 200M requested  
Error: Could not create the Java Virtual Machine.  
Error: A fatal exception has occurred. Program will exit.
```

Cause: The master JVM could not obtain enough memory to start. Memory is most likely constrained by your ASSIZEMAX setting.

Response: For more information about setting the ASSIZEMAX parameter, see [“Configuring memory and CPU options”](#) on page 65.

A Spark application is not progressing and shows JVMJ9VM015W error in the log

Symptom: The Spark master and worker started successfully; however, the Spark application is not making any progress, and the following error appears in the executor log:

```
JVMJ9VM015W Initialization error for library j9gc28(2):  
Failed to instantiate heap; 20G requested  
Error: Could not create the Java Virtual Machine.  
Error: A fatal exception has occurred. Program will exit.
```

Cause: The executor JVM could not obtain enough memory to start. Memory is most likely constrained by your MEMLIMIT setting or IEFUSI exit.

Response: For more information about setting the MEMLIMIT parameter or using the IEFUSI exit, see [“Configuring memory and CPU options” on page 65](#). Also see [“Displaying process limits” in z/OS UNIX System Services Planning](#).

Spark commands fail with an EDC5157 message, or BPXP015I and BPXP014I messages appear on the z/OS console

Symptom: Spark commands fail with the following error message:

```
17/08/08 13:51:40 INFO StandaloneAppClient$ClientEndpoint: Executor updated:
app-20170808135140-0000/0 is now FAILED (java.io.IOException: Cannot run program
"/usr/lpp/java/java800/J8.0_64/bin/java" (in directory "/u/user1/work/
app-20170808135140-0000/0")):
EDC5157I An internal error has occurred.)
```

The following messages appear on the z/OS console:

```
BPXP015I HFS PROGRAM /bin/setfacl IS NOT MARKED PROGRAM CONTROLLED.
BPXP014I ENVIRONMENT MUST BE CONTROLLED FOR SURROGATE (BPX.SRV.aaaaaa)
PROCESSING.
```

Cause: Apache Spark 2.1.1 and later requires that the **_BPX_SHAREAS** environment variable be set to NO when starting the cluster, but it is currently set to YES.

Response: Under your Spark user ID (for instance, SPARKID), ensure that the \$SPARK_CONF_DIR/spark-env.sh file contains **_BPX_SHAREAS=NO**, and that the master and worker processes were started using that spark-env.sh file. Verify that the **SPARK_CONF_DIR** and **_BPX_SHAREAS** environment variable are set properly for any BPXBATCH jobs that run start-master.sh and start-slave.sh. Consider restarting the master and worker processes to ensure the proper setting is used.

Spark worker or driver cannot connect to the master and the log files show "java.io.IOException: Failed to connect" error messages

Symptom: The Spark master starts successfully, but the worker or the driver is unable to connect to it. The following error message is repeated several times in the worker or application log:

```
org.apache.spark.SparkException: Exception thrown in awaitResult
...Caused by: java.io.IOException: Failed to connect to ip_address:port
```

Cause: The worker or driver is unable to connect to the master due to network errors.

Response: Check your network connectivity. If you have Spark client authentication enabled, verify that your AT-TLS policy is set up correctly and that the worker or driver has a valid digital certificate. For more information about client authentication, see [“Configuring z/OS Spark client authentication” on page 41](#).

Spark worker fails with ICH408I message with NEWJOBNAME insert

Symptom: Spark worker fails with the following error message:

```
ICH408I USER(SPARKID)
GROUP(SPARKGRP) NAME(#####)
CL(PROCESS )
INSUFFICIENT AUTHORITY TO NEWJOBNAME
```

Cause: The IzODA Apache Spark worker spawns drivers and executors using the jobname prefixes or templates specified in spark-defaults.conf. The SPARKID userid will not be authorized to create jobs with specified jobnames unless authorized.

Response: Permit the SPARKID userid to the BPX.JOBNAME profile with READ access..

Appendix A. Migrating to a new version of Apache Spark

IBM z/OS Platform for Apache Spark (FMID HSPK110) and IBM Open Data Analytics for z/OS (FMID HSPK120) are built on Apache Spark. Different service (PTF) levels of these products might provide different versions of Apache Spark. Perform the following steps if you are migrating from one version of Apache Spark to another.

Before you begin

If you previously installed IBM z/OS Platform for Apache Spark, Version 1.1.0, or IBM Open Data Analytics for z/OS, Version 1.1.0, determine the Apache Spark version that is provided. You can find the Apache Spark version in the RELEASE file in the Spark installation directory. The following sample output indicates that a version of Apache Spark 2.2.0 is provided.

```
IBM Open Data Analytics for z/OS - Spark, Version 2.2.0 built for Hadoop 2.7.7
Built with Java JRE 1.8.0 IBM ZOS build 8.0.5.17 - pmz6480sr5fp17-20180627_01(SR5 FP17)
Built from Git zos_Spark_2.2.0.12 (revision bcb96e0f31e4e4e7f82d85c72ec31478419cbd39)
Built via Jenkins job Spark/zos_Spark_2.2.0.12, Build#20
Build flags: -Phive -Phive-thriftserver -Phadoop-2.7
```

Important: Mixing different Spark components, such as Spark master and worker, from different Apache Spark versions could yield undesirable and unpredictable results. A Spark cluster, for example, might not function properly if the master daemon is started from Apache Spark 2.0.2 whereas the worker daemon is started from Apache Spark 2.2.0.

Important: IBM urges you to install and test the new version of Apache Spark on a test system before you install it on a production system. IBM also recommends that you back up any custom files, such as `spark-defaults.conf` and `spark-env.sh`, before installing the new version.

Before installing the new version of Apache Spark

Complete the following steps to understand the impact of migrating to a newer version of Apache Spark on your applications and to update your level of Java.

1. Determine whether your Spark setup resides in a mixed-endian environment.

A mixed-endian environment exists if part of your Apache Spark setup, typically the worker, runs on an IBM Z platform, and another part, typically the driver, runs on a different platform. The Spark driver is the process that hosts your Spark application, and it can run as an independent process or inside a third-party product, such as Scala Workbench (including Jupyter Notebook) or Spark Job Server. Apache Spark 2.x.x no longer supports mixed-endian environments in client deploy mode. If this is the environment that you use, see [Appendix H, “Apache Spark in a mixed-endian environment,”](#) on page 187 for information about possible alternatives.

2. Review the new functionality in the new version of Apache Spark and the changes to the Spark APIs to determine any changes that you might need to make to your applications before migration. Use the information at the following links to learn about the changes. Be sure to consider the changes for each in between Apache Spark version. For example, if you are migrating from Apache Spark 2.0.2 to 2.2.0, you need to consider the changes for Apache Spark 2.1.0, 2.1.1, and 2.2.0.
 - For high-level information about new features; changes, removals, and deprecations made to the Apache Spark APIs; performance improvements and known issues, see the following links.
 - If your previous Apache Spark version is 1.5.2, start here:
 - [Spark Release 1.6.0 \(spark.apache.org/releases/spark-release-1-6-0.html\)](http://spark.apache.org/releases/spark-release-1-6-0.html)
 - [Spark Release 1.6.1 \(spark.apache.org/releases/spark-release-1-6-1.html\)](http://spark.apache.org/releases/spark-release-1-6-1.html)

- [Spark Release 1.6.2 \(spark.apache.org/releases/spark-release-1-6-2.html\)](http://spark.apache.org/releases/spark-release-1-6-2.html)
 - [Spark Release 2.0.0 \(spark.apache.org/releases/spark-release-2-0-0.html\)](http://spark.apache.org/releases/spark-release-2-0-0.html)
 - [Spark Release 2.0.1 \(spark.apache.org/releases/spark-release-2-0-1.html\)](http://spark.apache.org/releases/spark-release-2-0-1.html)
 - [Spark Release 2.0.2 \(spark.apache.org/releases/spark-release-2-0-2.html\)](http://spark.apache.org/releases/spark-release-2-0-2.html)
 - If your previous Spark version is 2.0.2, you can start here:
 - [Spark Release 2.1.0 \(spark.apache.org/releases/spark-release-2-1-0.html\)](http://spark.apache.org/releases/spark-release-2-1-0.html)
 - [Spark Release 2.1.1 \(spark.apache.org/releases/spark-release-2-1-1.html\)](http://spark.apache.org/releases/spark-release-2-1-1.html)
 - If your previous Apache Spark version is 2.1.1, you can start here:
 - [Spark Release 2.1.2 \(spark.apache.org/releases/spark-release-2-1-2.html\)](http://spark.apache.org/releases/spark-release-2-1-2.html)
 - [Spark Release 2.1.3 \(spark.apache.org/releases/spark-release-2-1-3.html\)](http://spark.apache.org/releases/spark-release-2-1-3.html)
 - [Spark Release 2.2.0 \(spark.apache.org/releases/spark-release-2-2-0.html\)](http://spark.apache.org/releases/spark-release-2-2-0.html)
 - If your previous Apache Spark version is 2.2.0, you can start here:
 - <https://spark.apache.org/releases/spark-release-2-2-1.html>
 - <https://spark.apache.org/releases/spark-release-2-2-2.html>
 - <https://spark.apache.org/releases/spark-release-2-2-3.html>
 - <https://spark.apache.org/releases/spark-release-2-3-0.html>
 - <https://spark.apache.org/releases/spark-release-2-3-1.html>
 - <https://spark.apache.org/releases/spark-release-2-3-2.html>
 - <https://spark.apache.org/releases/spark-release-2-3-3.html>
 - <https://spark.apache.org/releases/spark-release-2-3-4.html>
 - If you are migrating from Apache Spark version 1.5.2, see [Spark 2.0 deprecations and removals \(https://issues.apache.org/jira/browse/SPARK-11806\)](https://issues.apache.org/jira/browse/SPARK-11806).
 - The Spark SQL and Spark ML projects have additional migration changes for each version of Apache Spark. See the following resources for details.
 - <http://spark.apache.org/docs/2.4.8/sql-programming-guide.html>.
 - <http://spark.apache.org/docs/2.4.8/ml-guide.html>. Note that as of Apache Spark 2.0, the RDD-based APIs in the `spark.mllib` package have entered maintenance mode; consider how this might affect your applications.
 - [Spark MLlib Old Migration Guides \(spark.apache.org/docs/2.0.2/ml-migration-guides.html\)](http://spark.apache.org/docs/2.0.2/ml-migration-guides.html).
 - The following Spark projects have no specific migration steps. However, they might document new behaviors as of the Spark version.
 - <http://spark.apache.org/docs/2.4.8/streaming-programming-guide.html>
 - <http://spark.apache.org/docs/2.4.8/structured-streaming-programming-guide.html>
 - <http://spark.apache.org/docs/2.4.8/graphx-programming-guide.html>
3. Based on your findings from the information in step “2” on page 159, update your applications as needed to work with the new Spark version.
 4. If you are using an older Java level than the one indicated in the RELEASE file, consider updating your Java level.
 5. Ensure that any other open source or third-party software in your environment that interacts with Spark supports the new version of Apache Spark. For example, some versions of Scala Workbench do not work with the new versions of Apache Spark.

Installing the new version of Apache Spark

Install IBM Open Data Analytics for z/OS Spark, FMID HSPK120 and its service updates (PTFs).

For installation guidelines, see *Program Directory for IBM Open Data Analytics for z/OS*.

After installing the new version of Apache Spark

1. Recompile applications that use any of the changed Spark APIs.
2. Consider using the Spark workflow to configure the new release of Spark. If not using the workflow, examine any new Apache Spark configuration options and make necessary changes to your `spark-defaults.conf` and `spark-env.sh` configuration files.

For the current list of configuration options, see <http://spark.apache.org/docs/2.4.8/configuration.html>. A new Apache Spark version might introduce new configuration options as well as deprecate existing ones.

Note: For the contents of the `spark-defaults.conf` and `spark-env.sh` configuration files, you can find IBM-supplied default values in `spark-defaults.conf.template` and `spark-env.sh.template`.

3. If you use the `spark-submit` or `spark-sql` command line interface, you must either invoke them from a writable directory or change your configuration files. For more information, see [“Updating the Apache Spark configuration files”](#) on page 34.

Appendix B. Sample configuration and AT-TLS policy rules for z/OS Spark client authentication

The following examples show the sample configuration settings and AT-TLS policy rules that you can use in your `spark-env.sh` and `spark-defaults.conf` (both located in the `SPARK_CONF_DIR` directory) and `TCPIP-TTLS.policy` AT-TLS policy file, under each of the z/OS Spark client authentication models. They assume the network port configurations as shown in [Table 15 on page 163](#), and you should modify them to values that are suitable for your environment.

Network port configurations

Table 15. Example network port configurations		
Port name	Default port number	Port number in example configuration
Master port	7077	7077
Mast REST port	6066	6066
Worker port	(random)	7177
Block manager port	(random)	7511
External shuffle server	7337	7337
PySpark daemon	(random)	7722
Driver port	(random)	7277
Driver block manager port	(value of <code>spark.blockManager.port</code>)	7611

For more information about configuring the Spark configuration files for z/OS Spark client authentication, see [“Configuring additional authorities and permissions for the Spark cluster” on page 53](#).

For more information about configuring the AT-TLS policies for z/OS Spark client authentication, see [“Defining the AT-TLS policy rules” on page 49](#).

You can find detailed information about the syntax of each AT-TLS policy statements in [“AT-TLS policy statements in z/OS Communications Server: IP Configuration Reference”](#)

- When AT-TLS is the client authentication method.

`spark-defaults.conf`

```
# Set this value to false if you want to disable client authentication on the master port
# The default is true. This option only applies to client deploy mode.
# spark.zos.master.authenticate.method indicates the authentication method to use.
spark.zos.master.authenticate      true

# Method used for client authentication. Valid values are ATTLS (default) and TrustedPartner.
# Only applicable if spark.zos.master.authenticate is enabled. See IzODA Installation
# and Customization Guide for more information on required configuration for each method.
spark.zos.master.authenticate.method  ATTLS

# The REST server does not support client authentication nor application-layer TLS.
# Only enable this once you have adequate security in place for the REST port.
spark.master.rest.enabled          false

spark.driver.blockManager.port 7611
```

```

spark.driver.port 7277
spark.blockManager.port 7511
spark.python.daemon.port 7722

# uncomment and set these if not using default values
# spark.master.rest.port 6066
# spark.shuffle.service.port 7337

# uncomment this to enable external shuffle server
# spark.shuffle.service.enabled true

```

spark-env.sh

```

# uncomment and set this if not using default value
# SPARK_MASTER_PORT=7077
SPARK_WORKER_PORT=7177

```

AT-TLS policy rules

```

###
### AT-TLS POLICY AGENT CONFIGURATION FILE FOR SPARK ON Z/OS
###
#####
### SparkClusterGrp_ATTLS contains the PortRange references
### for the Spark cluster ports that support AT-TLS security.
###
### For detailed usage information and configuration
### instructions of these ports, please refer to the
### IzODA Installation and Customization Guide
###
### When port-binding fails (due to, e.g., port already in
### use), it will retry on sequential ports to a number of times
### equal the value of spark.port.maxRetries (default: 16).
### This value can be configured in spark-defaults.conf.
###
### (The Shuffle Server port does not support port range retry.)
###
### Considering this behavior, we recommend configuring
### a PortRange to account for the retry behavior for
### ports that support port range retry:
###
### port_number - (port_number + spark.port.maxRetries)
#####
PortGroup                               SparkClusterGrp_ATTLS
{
  PortRangeRef                          SparkMaster_ATTLS
  PortRangeRef                          SparkMasterRest_ATTLS
  PortRangeRef                          SparkExtShuffleServer_ATTLS
}
PortRange                               SparkMaster_ATTLS
{
  Port                                  7077-7093
}
PortRange                               SparkMasterRest_ATTLS
{
  Port                                  6066-6082
}
PortRange                               SparkExtShuffleServer_ATTLS
{
  Port                                  7337
}
#####
###
### KeyRing_ATTLS defines the keyring that will be used during
### Spark AT-TLS authentication.
###
#####
TLSKeyRingParms                         KeyRing_ATTLS
{
  Keyring                              SparkRing
}
#####
###
### SparkServer_ATTLS and SparkClient_ATTLS are the rules that work

```



```

## together to encrypt the network traffic among the ports defined
## in the SparkClusterGrp_ATTLS section.
##
#####
TTLSPRule                                     SparkServer_ATTLS
{
  Direction                                     Inbound
  LocalPortGroupRef                             SparkClusterGrp_ATTLS
  TLSGroupActionRef                             GroupAct_TTLS_On
  TTLEnvironmentActionRef                       EnvAct_SparkServer_ATTLS
}
TTLSPRule                                     SparkClient_ATTLS
{
  Direction                                     Outbound
  RemotePortGroupRef                             SparkClusterGrp_ATTLS
  TLSGroupActionRef                             GroupAct_TTLS_On
  TTLEnvironmentActionRef                       EnvAct_SparkClient_ATTLS
}
#####
##
## EnvAct_SparkServer_ATTLS and EnvAct_SparkClient_ATTLS
## establish the environment for the connections that match the
## corresponding TTLSPRules, using the role and keyring specified.
##
#####
TTLEnvironmentAction                         EnvAct_SparkServer_ATTLS
{
  HandshakeRole                                 ServerWithClientAuth
  EnvironmentUserInstance                       0
  TLSKeyRingParmsRef                           KeyRing_ATTLS
  TTLEnvironmentAdvancedParmsRef               EnvAdv_TLS
}
TTLEnvironmentAction                         EnvAct_SparkClient_ATTLS
{
  HandshakeRole                                 Client
  EnvironmentUserInstance                       0
  TLSKeyRingParmsRef                           KeyRing_ATTLS
  TTLEnvironmentAdvancedParmsRef               EnvAdv_TLS
}
#####
##
## GroupAct_TTLS_On is the group action that enables TLS
## security for connections utilizing it.
##
#####
TTLSGroupAction                             GroupAct_TTLS_On
{
  TTLS-enabled                                  On
}
#####
##
## EnvAdv_TLS is an advanced environment parms object
## enforcing the following conditions on the server:
##   - The server can only utilize TLSv1.2 when accepting
##     a connection,
##   - The server must use level 2 client authentication
##     in the TLS handshake.
##
#####
TTLEnvironmentAdvancedParms                 EnvAdv_TLS
{
  ClientAuthType                               SAFCheck
  TLSv1                                         Off
  TLSv1.1                                       Off
  TLSv1.2                                       On
}

```

- When Trusted Partner is the client authentication method.

spark-defaults.conf

```

# Set this value to false if you want to disable client authentication on the master
# port. The default is true. This option only applies to client deploy mode.
# spark.zos.master.authenticate.method indicates the authentication method to use.
spark.zos.master.authenticate               true

# Method used for client authentication. Valid values are ATTLS (default) and TrustedPartner.
# Only applicable if spark.zos.master.authenticate is enabled. See IzODA Installation
# and Customization Guide for more information on required configuration for each method.

```

```

spark.zos.master.authenticate.method TrustedPartner

# The REST server does not support client authentication nor application-layer TLS.
# Only enable this once you have adequate security in place for the REST port.
spark.master.rest.enabled false

spark.driver.blockManager.port 7611
spark.driver.port 7277
spark.blockManager.port 7511
spark.python.daemon.port 7722

# uncomment and set these if not using default values
# spark.master.rest.port 6066
# spark.shuffle.service.port 7337

# uncomment this to enable external shuffle server
# spark.shuffle.service.enabled true

```

spark-env.sh

```

" uncomment and set this if not using default value
" SPARK_MASTER_PORT=7077
SPARK_WORKER_PORT=7177

```

Trusted Partner policy rules

```

###
### TRUSTED PARTNER POLICY AGENT CONFIGURATION FILE FOR SPARK ON Z/OS
###
#####
### SparkClusterGrp_TP contains the PortRange references
### for the Spark cluster ports that support Trusted Partner
### security.
###
### For detailed usage information and configuration
### instructions of these ports, please refer to the
### IzODA Installation and Customization Guide
###
### When port-binding fails (due to, e.g., port already in
### use), it will retry on sequential ports to a number of times
### equal the value of spark.port.maxRetries (default: 16).
### This value can be configured in spark-defaults.conf.
###
### (The Shuffle Server port does not support port range retry.)
###
### Considering this behavior, we recommend configuring
### a PortRange to account for the retry behavior for
### ports that support port range retry:
###
### port_number - (port_number + spark.port.maxRetries)
#####
PortGroup SparkClusterGrp_TP
{
  PortRangeRef SparkMaster_TP
}
PortRange SparkMaster_TP
{
  Port 7077-7093
}
#####
###
### KeyRing_TP defines the keyring that will be used during
### Spark Trusted Partner authentication.
###
#####
TLSKeyRingParms KeyRing_TP
{
  Keyring SparkRingTP
}
#####
###
### SparkServer_TP and SparkClient_TP are the rules that work
### together to encrypt the network traffic among the ports defined
### in the SparkClusterGrp_TP section.
###
#####

```

```

TTLSPolicyRule
{
  Direction                Inbound
  LocalPortGroupRef         SparkClusterGrp_TP
  TLSGroupActionRef         GroupAct_TTLS_On
  TLSEnvironmentActionRef   EnvAct_SparkServer_TP
}
TTLSPolicyRule
{
  Direction                Outbound
  RemotePortGroupRef        SparkClusterGrp_TP
  TLSGroupActionRef         GroupAct_TTLS_On
  TLSEnvironmentActionRef   EnvAct_SparkClient_TP
}
#####
##
## EnvAct_SparkServer_TP and EnvAct_SparkClient_TP
## establish the environment for the connections that match the
## corresponding TTLSPolicyRules, using the role and keyring specified.
##
#####
TTLSEnvironmentAction      EnvAct_SparkServer_TP
{
  HandshakeRole            Server
  EnvironmentUserInstance   0
  TLSKeyRingParmsRef        KeyRing_TP
  TLSEnvironmentAdvancedParmsRef EnvAdv_TLS_TP
}
TTLSEnvironmentAction      EnvAct_SparkClient_TP
{
  HandshakeRole            Client
  EnvironmentUserInstance   0
  TLSKeyRingParms          {
    Keyring                  *AUTH*/*
  }
  TLSEnvironmentAdvancedParmsRef EnvAdv_TLS_TP
}
#####
##
## GroupAct_TTLS_On is the group action that enables TLS
## security for connections utilizing it.
##
#####
TTLSTLSGroupAction         GroupAct_TTLS_On
{
  TTLS-enabled              On
}
#####
##
## EnvAdv_TLS_TP is an advanced environment parms object
## enforcing the following conditions on the server:
##   - The server can only utilize TLSv1.2 when accepting
##     a connection.
##
#####
TTLSEnvironmentAdvancedParms EnvAdv_TLS_TP
{
  TLSv1                    Off
  TLSv1.1                  Off
  TLSv1.2                  On
}

```


Appendix C. Sample z/OS IzODA Livy AT-TLS policy rules

The following example shows the sample AT-TLS policies that you can use in you TCPIP_TTLS.policy AT-TLS policy file for z/OS IzODA Livy. This is meant to be used in conjunction with sample AT-TLS policies shown in [Appendix B, “Sample configuration and AT-TLS policy rules for z/OS Spark client authentication,” on page 163](#) when AT-TLS is used as the Spark client authentication method..

For more information about AT-TLS policies, see [Chapter 8, “z/OS IzODA Livy Installation and Customization,” on page 105](#). You can find detailed information about the syntax of each AT-TLS policy statement in “AT-TLS policy statements” in [z/OS Communications Server: IP Configuration Reference](#).

AT-TLS policies when AT-TLS is used as the Spark client authentication method

```
#####
###
### LivyServer_ATTLS is the port the Livy Server listens
### on for connections from external clients. This port
### supports AT-TLS security.
###
### The Livy Server port does not support port range retry.
###
### By default, livy-server binds to the port specified on
### livy.server.port in $LIVY_CONF_DIR/livy.conf.
###
#####
PortRange                      LivyServer_ATTLS
{
Port                          8998
}
#####
###
### KeyRing_Livy defines the keyring that will be used during
### Livy Server AT-TLS authentication.
###
#####
TTLSKeyRingParms              KeyRing_Livy
{
Keyring                       LivyRing
}
#####
###
### LivyServer_ATTLS and LivyClient_ATTLS are the rules that
### encrypt network traffic going into and out of the Livy Server
### port.
###
### The LivyClient_ATTLS rule is necessary only if you are going
### to submit jobs to the Livy Server from either the same LPAR or
### the same Sysplex. It is not needed if you will be only using
### external HTTP clients.
###
#####
TTLSRule                      LivyServer_ATTLS
{
Direction                    Inbound
LocalPortRangeRef             LivyServer_ATTLS
TTLSGroupActionRef            GroupAct_TTLS_On
TTLSEnvironmentActionRef       EnvAct_LivyServer_ATTLS
}
TTLSRule                      LivyClient_ATTLS
{
Direction                    Outbound
RemotePortRangeRef            LivyServer_ATTLS
TTLSGroupActionRef            GroupAct_TTLS_On
TTLSEnvironmentActionRef       EnvAct_LivyClient_ATTLS
}
#####
###
### EnvAct_LivyServer_ATTLS and EnvAct_LivyClient_ATTLS
### establish the environment for the connections that match the
### corresponding TTLSRules, using the role and keyring specified.
```

```

###
### Remove the EnvAct_LivyClient_ATTLS section if the corresponding
### TTLSRule is not present.
###
#####
TTLSEnvironmentAction      EnvAct_LivyServer_ATTLS
{
  HandshakeRole            ServerWithClientAuth
  EnvironmentUserInstance  0
  TTLSKeyRingParmsRef      KeyRing_Livy
  TTLSEnvironmentAdvancedParmsRef EnvAdv_TLS
}
TTLSEnvironmentAction      EnvAct_LivyClient_ATTLS
{
  HandshakeRole            Client
  EnvironmentUserInstance  0
  TTLSKeyRingParmsRef      KeyRing_Livy
  TTLSEnvironmentAdvancedParmsRef EnvAdv_TLS
}

```

Appendix D. Memory and CPU configuration options

You can configure a variety of memory and CPU options within Apache Spark, IBM Java, and z/OS.

Apache Spark configuration options

There are two major categories of Apache Spark configuration options: Spark properties and environment variables.

Spark properties control most application settings and can be configured separately for each application. You can set these properties in the following ways, in order from highest to lowest precedence:

1. Directly on a SparkConf passed to your SparkContext
2. At run time, as command line options passed to **spark-shell** and **spark-submit**
3. In the `spark-defaults.conf` properties file

In addition, you can configure certain Spark settings through environment variables, which are read from the `$SPARK_CONF_DIR/spark-env.sh` script.

Table 16 on page 171 through Table 19 on page 172 summarize some of the Spark properties and environment variables that control memory and CPU usage by Spark applications.^{1,2}

Table 16. Environment variables that control memory settings

Environment variable	Default value	Description
SPARK_WORKER_MEMORY	Total memory on host system minus 1 GB	Total amount of memory that a worker can give to executors. Note: This value should equal the product of <code>spark.executor.memory</code> times the number of executor instances.
SPARK_DAEMON_MEMORY	1 GB	Amount of memory to allocate to the Spark master and worker daemons.

Table 17. Spark properties that control memory settings

Spark property	Default value	Description
<code>spark.driver.memory</code>	1 GB	Amount of memory to use for the driver process (that is, where SparkContext is initialized). Note: In client mode, this property must not be set through the SparkConf directly in your application. Instead, set this through the --driver-memory command line option or in your default properties file.
<code>spark.driver.maxResultSize</code>	1 GB	Limit of the total size of serialized results of all partitions for each Spark action (for instance, collect). Note: Jobs will fail if the size of the results is above this limit. Having a high limit may cause out-of-memory errors in the driver.

¹ <http://spark.apache.org/docs/2.4.8/configuration.html>

² <http://spark.apache.org/docs/2.4.8/spark-standalone.html>

Table 17. Spark properties that control memory settings (continued)

Spark property	Default value	Description
<code>spark.executor.memory</code>	1 GB	Amount of memory to use per executor process. Note: This sets the heap size of the executor JVM.
<code>spark.memory.fraction</code>	0.6	Fraction of (heap space - 300 MB) used for execution and storage. The lower this value is, the more frequently spills and cached data eviction occur. The purpose of this property is to set aside memory for internal metadata, user data structures, and imprecise size estimation in case of sparse, unusually large records.
<code>spark.memory.storageFraction</code>	0.5	Amount of storage memory that is immune to eviction, expressed as a fraction of the size of the region set aside by <code>spark.memory.fraction</code> . The higher this value is, the less working memory may be available to execution and tasks may spill to disk more often.
<code>spark.memory.offHeap.enabled</code>	false	If set to true, Spark attempts to use off-heap memory for certain operations. If off-heap memory use is enabled, <code>spark.memory.offHeap.size</code> must be positive.
<code>spark.memory.offHeap.size</code>	0	The absolute amount of memory, in bytes, that can be used for off-heap allocation. This setting has no impact on heap memory usage, so if your executors' total memory consumption must fit within some hard limit, be sure to shrink the JVM heap size accordingly. This must be set to a positive value when <code>spark.memory.offHeap.enabled</code> is set to true.

Table 18. Environment variables that control CPU settings

Environment variable	Default value	Description
SPARK_WORKER_CORES	All cores on host system	Number of cores to use for all executors. Note: This value should equal the product of <code>spark.executor.cores</code> times the number of executor instances.

Table 19. Spark properties that control CPU settings

Spark property	Default value	Description
<code>spark.deploy.defaultCores</code>	Infinite	Default number of cores to give to applications if they do not set a <code>spark.cores.max</code> value.
<code>spark.cores.max</code>	(Not set)	Maximum number of cores to give to an application across the entire cluster. If not set, the default is the <code>spark.deploy.defaultCores</code> value.
<code>spark.driver.cores</code>	1	Number of cores to use for the driver process, only in cluster mode.

Table 19. Spark properties that control CPU settings (continued)

Spark property	Default value	Description
<code>spark.executor.cores</code>	All cores on host system	Number of cores to use on each executor. Setting this parameter allows an application to run multiple executors, provided that there are enough cores on the worker (SPARK_WORKER_CORES); otherwise, only one executor per application runs on each worker.

Table 20. Spark properties that affect application and cluster parallelism

Spark property	Default value	Description
<code>spark.zos.master.app.alwaysScheduleApps</code>	false	Use this property to enable the Spark Master to start applications, even when the current applications appear to be using all the CPU and memory. This enables the system to start and then manage the resources across the applications, according to the WLM policy.
<code>spark.zos.maxApplications</code>	5	When <code>spark.zos.master.app.alwaysScheduleApps</code> is set to true, specifies the maximum number of applications that can be scheduled to run concurrently. This value should be set based on resources available according to the current WLM policy.
<code>spark.dynamicAllocation.enabled</code>	false	Specifies the Spark Master can request that the Spark Worker add or remove executors based on the workload and available resources.
<code>spark.shuffle.service.enabled</code>	false	Specifies that the worker should start the Spark Shuffle service, which allows executors to transfer (shuffle) data across executors, as needed. This is required when <code>spark.dynamicAllocation.enabled</code> is true.

For more information about the Apache Spark configuration options, see <http://spark.apache.org/docs/2.4.8/configuration.html>. For more information about tuning the Apache Spark cluster, see <http://spark.apache.org/docs/2.4.8/tuning.html>.

IBM Java configuration options

In addition to the Apache Spark settings, you can use IBM JVM runtime options to manage resources used by Apache Spark applications.

You can set the IBM JVM runtime options in the following places:

IBM_JAVA_OPTIONS

Use this environment variable to set generic JVM options.

SPARK_DAEMON_JAVA_OPTS

Use this environment variable to set additional JVM options for the Apache Spark master and worker daemons.

spark.driver.extraJavaOptions

Use this Apache Spark property to set additional JVM options for the Apache Spark driver process.

spark.executor.extraJavaOptions

Use this Apache Spark property to set additional JVM options for the Apache Spark executor process. You cannot use this option to set Spark properties or heap sizes.

Table 21 on page 174 summarizes some of the IBM JVM runtime options that control resource usage.

Table 21. IBM JVM runtime options that control resource usage

Option	Default value	Description
-Xmx	Half of the available memory, with a minimum of 16 MB and a maximum of 512 MB	Maximum heap size. Note: Set Spark JVM heap sizes via Spark properties (such as <code>spark.executor.memory</code>), not via Java options.
-Xms	4 MB	Initial heap size.
-Xss	1024 KB	Maximum stack size.
-Xcompressdrefs	Enabled by default when the value of the -Xmx option is less than or equal to 57 GB	Enables the use of compressed references.
-Xlp	1M pageable pages, when available, are the default size for the object heap and the code cache. If -Xlp is specified without a size, 1M non-pageable is the default for the object heap.	Requests that the JVM allocate the Java object heap using large page sizes (1M or 2G). Instead of -Xlp , you can use -Xlp:codecache and -Xlp:objectcache to set the JIT code cache and object heap separately. Note: There is a limit to the number of large pages that are available on a z/OS system. Consult your system administrator before changing this setting.
-Xmn	(Not set)	Sets the initial and maximum size of the new area to the specified value when using the -Xgcpolicy:gencon option.

For more information about IBM Java on z/OS, see IBM SDK, Java Technology Edition z/OS User Guide (www.ibm.com/support/knowledgecenter/SSYKE2_8.0.0/pdf/en/sdkandruntimetimeguide.zos.80_8.0.pdf).

z/OS configuration parameters

The z/OS operating system provides additional configuration options that allow the system administrator to ensure that no workloads use more resources than they are allowed. Be sure that any application-level configuration does not conflict with the z/OS system settings. For example, the executor JVM will not start if you set `spark.executor.memory=4G` but the MEMLIMIT parameter for the user ID that runs the executor is set to 2G.

Typically, you can modify these settings in the following ways (in order from highest to lowest precedence):

1. Use the ALTUSER RACF command to modify the resource settings in the OMVS segment of the security profile for the user ID under which Apache Spark runs.
2. Use the IEFUSI exit, which receives control before each job step starts.
3. Set the system-wide defaults in the appropriate SYS1.PARMLIB member.

Table 22 on page 175 summarizes some of the z/OS configuration options that might be relevant to your Open Data Analytics for z/OS environment.

Table 22. IBM z/OS configuration parameters

Parameter	Default value	Description	Where to set
MEMLIMIT	2 GB	<p>Amount of virtual storage above the bar that an address space is allowed to use.</p> <p>Note: Set the MEMLIMIT for the Spark user ID to the largest JVM heap size (executor memory size) plus the amount of native memory that Spark processing might require (typically, at least 2 GB).</p>	<ul style="list-style-type: none"> • MEMLIMIT parameter in the SMFPRMxx parmlib member (sets the system-wide default) • MEMLIMIT parameter on JCL JOB or EXEC statements • IEFUSI exit • MEMLIMIT variable in the OMVS security profile segment
MAXMMAPAREA	40960	<p>Maximum amount of data space storage, in pages, that can be allocated for memory mapping of z/OS UNIX files.</p>	<ul style="list-style-type: none"> • MAXMMAPAREA parameter in the BPXPRMxx parmlib member (sets the system-wide default) • MMAPAREAMAX variable in the OMVS security profile segment
MAXASSIZE	200 MB	<p>Maximum address space size for a new process.</p> <p>Note: The JVM will fail to start if the MAXASSIZE for the Spark user ID is too small. The recommended minimum for Java is 2,147,483,647 bytes.</p>	<ul style="list-style-type: none"> • MAXASSIZE parameter in the BPXPRMxx parmlib member (sets the system-wide default) • IEFUSI exit • ASSIZEMAX variable in the OMVS security profile segment
MAXCPU TIME	1000	<p>Maximum CPU time, in seconds, that a process can use.</p>	<ul style="list-style-type: none"> • MAXCPU TIME parameter in the BPXPRMxx parmlib member (sets the system-wide default) • CPU TIME MAX variable in the OMVS security profile segment
MAXPROCESSOR	25	<p>Maximum number of concurrently active processes for a single z/OS UNIX user ID.</p>	<ul style="list-style-type: none"> • MAXPROCUSER parameter in the BPXPRMxx parmlib member (sets the system-wide default) • PROCUSERMAX variable in the OMVS security profile segment

For more information about z/OS parmlib members, see *z/OS MVS Initialization and Tuning Reference*.

Appendix E. Spark properties specific to the z/OS environment

Table 23 on page 177 lists the Spark properties that are specific to Spark running on z/OS systems as part of IBM Open Data Analytics for z/OS.

You set these properties in the `spark-defaults.conf` file.

Table 23. Spark properties specific to the z/OS environment

Spark property	Default value	Description
<code>spark.zos.master.authenticate</code>	<code>true</code>	Specifies whether Spark authenticates connections to the Spark master port. The authentication method that is used is indicated by <code>spark.zos.master.authenticate.method</code> .
<code>spark.zos.master.authenticate.method</code>	<code>ATTLS</code>	Specifies the method that Spark is to use to authenticate connections to the Spark master port. The value, <code>ATTLS</code> , requires AT-TLS with client authentication. The value, <code>TrustedPartner</code> , requires that all connections are internal. This property is ignored if <code>spark.zos.master.authenticate=false</code> is specified.
<code>spark.zos.cluster.instanceNumber</code>		Specifies the instance number of the Spark cluster.
<code>spark.zos.driver.jobname.prefix</code>	<code>ODASD</code>	Specifies the job name prefix for drivers in cluster deploy mode. The suffix will be as many digits of the corresponding driver instance number as can fit to form an 8-character job name, starting from the last (rightmost) character of the driver instance number. For example, if you specify <code>spark.zos.driver.jobname.prefix=SPARKD</code> , the job name of the driver with a driver instance number of 0001 will be <code>SPARKD01</code> . The <code>spark-defaults.conf</code> file contains a default setting of <code>spark.zos.driver.jobname.prefix=ODASD</code> . Note that this job name prefix applies only to drivers in cluster deploy mode. For client deploy mode, you can continue to use the <code>_BPX_JOBNAME</code> environment variable to set the job name of the driver.

Table 23. Spark properties specific to the z/OS environment (continued)

Spark property	Default value	Description
<code>spark.zos.driver.jobname.template</code>		<p>Specifies the job name template for drivers in cluster deploy mode. The template property provides further customization when you are generating job names for the driver. The template uses variables that can be substituted for several pieces of information about the work that the driver is running. These variables include <i>cluster</i> and <i>driver</i>.</p> <p>For example, ODA<<i>cluster:1</i>><<i>driver:2</i>> where cluster number is 6 and driver number is 0312, would yield ODA612.</p>
<code>spark.zos.executor.jobname.prefix</code>	ODASX	<p>Specifies the job name prefix for executors. The suffix will be as many digits of the corresponding application instance number as can fit to form an 8-character job name, starting from the last (rightmost) character of the application instance number. All executors for the same application will have the same job name.</p> <p>For example, if you specify <code>spark.zos.executor.jobname.prefix=SPARKX</code>, the job names of the executors for application instance number 0001 will be SPARKX01.</p> <p>The <code>spark-defaults.conf</code> file contains a default setting of <code>spark.zos.executor.jobname.prefix=ODASX</code>. If no executor prefix is specified in <code>spark-defaults.conf</code>, the executor job names follow the z/OS UNIX default (user ID with a numeric suffix).</p>
<code>spark.zos.executor.jobname.template</code>		<p>Specifies the job name template for executors. The template property provides further customization when you are generating job names for executors. The template uses variables that can be substituted for several pieces of information about the work that the executor is running. These variables include <i>cluster</i>, <i>application</i>, and <i>executor</i>.</p> <p>For example, ODA<<i>cluster:1</i>><<i>application:2</i>><<i>executor:2</i>> where cluster number is 2, application number is 0312, and executor is 0000, would yield ODA21200.</p>

Table 23. Spark properties specific to the z/OS environment (continued)

Spark property	Default value	Description
<code>spark.zos.environment.verify</code>	<code>true</code>	<p>Specifies whether Spark should perform environment verification during master and worker daemon initialization. If enabled, Spark verifies that the daemons have the proper authority to perform the necessary functions. The daemons are terminated if they fail the verification. The results of the verification can be found in the daemon logs.</p> <p>This property requires the Spark cluster user ID to have READ access to the BPX.SERVER FACILITY class profile, unless client authentication is not enabled (<code>spark.zos.master.authenticate</code> is false). The configuration instructions for client authentication describe this in detail.</p>
<code>spark.python.daemon.port</code>		<p>Specifies the port for the PySpark daemon to listen on. This property is only applicable if you are using PySpark. The value must be between 1024 - 65535 (inclusive), or 0 to specify a random port. The number of retries is determined by <code>spark.port.maxRetries</code>. For Spark 2.2.0, APAR PI98042 is required to use this property.</p>
<code>spark.zos.master.app.alwaysScheduleApps</code>	<code>False</code>	<p>Specifies whether Spark should start applications when resources such as CPU and memory appear to be all in use. This enables the system to manage the applications according to the policies established by WLM.</p>
<code>spark.zos.maxApplications</code>	<code>5</code>	<p>When <code>spark.zos.master.app.alwaysScheduleApps</code> is set to true, specifies the maximum number of applications that can be scheduled to run concurrently. This value should be set based on resources available according to the current WLM policy.</p>

Appendix F. Data sets

The following table lists the data sets that installation member INSTPAC creates.

The installation data sets have the format *hlq.data_set_name_suffix*, where *hlq* is the high level qualifier and *data_set_name_suffix* is as listed in the table.

Any data set prefixed with the subsystem ID (*SSID*) is a data set created during post-installation to allow user modification. These post-installation data sets have the format *hlq.SSID.data_set_name_suffix*.

Table 24. Data sets created by INSTPAC							
Data set name suffix	SMP/E data type	SMP/E data definition	Data set organization	Record format	Logical record length	Block size	tracks
AAZKCNTL	USER2	ASDBCNTL	PO-E	FB	80	32720	180
AAZKDBRM	USER1	ASDBDBRM	PO-E	FB	80	32720	300
AAZKEXEC	EXEC	ASDBEXEC	PO-E	FB	80	32720	450
AAZKHTML	TEXT	ASDBHTML	PO-E	VB	32765	32760	15
AAZKHTX	PRODXML	ASDBHTX	PO-E	VB	19036	19040	15
AAZKJLOD	PROGRAM	ASDBJLOD	PO-E	U	0	6000	15
AAZKLIST	UTOUT	ASDBLIST	PO-E	FBA	133	32718	45
AAZKMAP	DATA	ASDBMAP	PO-E	FB	1024	31744	15
AAZKMLIB	MSG	ASDBMLIB	PO-E	FB	80	32720	15
AAZKMOD	MOD	ASDBMOD	PO-E	U	0	6144	1800
AAZKOBJ	USER3	ASDBOBJ	PO-E	FB	80	32720	1275
AAZKPLIB	PNL	ASDBPLIB	PO-E	FB	80	32720	105
AAZKRPC	PROGRAM	ASDBRPC	PO-E	U	0	6000	750
AAZKSAMP	SAMP	ASDBSAMP	PO-E	FB	80	32720	270
AAZKSLIB	SKL	ASDBSLIB	PO-E	FB	80	32720	15
AAZKSWI	DATA	ASDBSWI	PO-E	FB	80	32720	30
AAZKTLIB	TBL	ASDBTLIB	PO-E	FB	80	32720	15
AAZKXATH	EXEC	ASDBXATH	PO-E	FB	80	32720	15
AAZKXCMD	EXEC	ASDBXCMD	PO-E	FB	80	32720	15
AAZKXEXC	EXEC	ASDBXEXC	PO-E	FB	80	32720	15
AAZKXGLV	EXEC	ASDBXGLV	PO-E	FB	80	32720	15
AAZKXPUB	EXEC	ASDBXPUB	PO-E	FB	80	32720	15
AAZKXRPC	EXEC	ASDBXRPC	PO-E	FB	80	32720	15
AAZKXSQL	EXEC	ASDBXSQL	PO-E	FB	80	32720	15
AAZKXTOD	EXEC	ASDBXTOD	PO-E	FB	80	32720	15
AAZKXVTB	EXEC	ADV SXVTB	PO-E	FB	80	32720	15

<i>Table 24. Data sets created by INSTPAC (continued)</i>							
Data set name suffix	SMP/E data type	SMP/E data definition	Data set organi- zation	Record format	Logical record length	Block size	tracks
SAZKXATH	EXEC	SSDBXATH	PO-E	FB	80	32720	15
SAZKCLOD	MOD	SSDBCLOD	PO-E	U	0	6000	270
SAZKXCMD	EXEC	SSDBXCMD	PO-E	FB	80	32720	15
SAZKCNTL	USER2	SDBCNTL	PO-E	FB	80	32720	180
SAZKDBRM	USER1	SSDBDBRMX	PO-E	FB	80	32720	15
SAZKEEXEC	EXEC	SSDBXEXEC	PO-E	FB	80	32720	15
SAZKEEXEC	EXEC	SSDBEXEC	PO-E	FB	80	32720	450
SAZKXGLV	EXEC	SSDBXGLV	PO-E	FB	80 3	2720	15
SAZKHTML	TEXT	SSDBHTML	PO-E	VB	32756	32760	15
SAZKHTX	PRODXML	SSDBHTX	PO-E	VB	19036	19040	15
SAZKINST	SMP/E does not maintain this data set.		PO-E	FB	80	32720	18
SAZKJLOD	PROGRAM	SSDBJLOD	PO-E	U	0	6000	15
SAZKLIST	UTOUT	SSDBLIST	PO-E	FBA	133	32718	45
SAZKLOAD	MOD	SSDBLOAD	PO	U	0	6000	4500
SAZKMAP	DATA	SSDBMAP	PO-E	FB	1024	31744	15
SAZKOBJ	USER3	SSDBOBJ	PO-E	FB	80	32720	1125
SAZKXPUB	EXEC	SSDBXPUB	PO-E	FB	80	32720	15
SAZKXRPC	EXEC	SSDBXRPC	PO-E	FB	80	32720	15
SAZKRPC	PROGRAM	SSDBRPC	PO-E	U	0	6000	375
SAZKSAMP	SAMP	SSDBSAMP	PO-E	FB	80	32720	300
SAZKMDL1	SMP/E does not maintain this data set.	SDBLMOD	PO-E	U	0	6000	18
SAZKMDL2	SMP/E does not maintain this data set.	SDBLMOD2	PO-E	U	0	6000	18
SAZKMENU	MSG	SHDWMLIB	PO-E	FB	80	32720	15
SAZKPENU	PNL	SHDWPLIB	PO-E	FB	80	32720	210
SAZKSLIB	SKL	SHDWSLIB	PO-E	FB	80	32720	15
SAZKTENU	TBL	SHDWTLIB	PO-E	FB	80	32720	15
SAZKXSQL	EXEC	SSDBXSQL	PO-E	FB	80	32720	15
SAZKXWWW	DATA	SSDBSWI	PO-E	FB	80	32720	30

<i>Table 24. Data sets created by INSTPAC (continued)</i>							
Data set name suffix	SMP/E data type	SMP/E data definition	Data set organi- zation	Record format	Logical record length	Block size	tracks
SWI.OBJ1	SMP/E does not maintain this data set.		PO-E	VB	4092	12288	30
SAZKXTOD	EXEC	SSDBXTOD	PO-E	FB	80	32720	15

Note: SMP/E does not maintain this data set.

Appendix G. Restrictions

The following restrictions currently exist for IBM Open Data Analytics for z/OS Version 1.1.0:

- The R programming language is not supported.
- Hive on Spark is not supported.
- Running Spark on YARN or Mesos is not supported.
- Apache Thrift server is not supported.
- The use of multiple workers is not supported.
- Mixed-endian environments in client deploy mode are not supported.

Appendix H. Apache Spark in a mixed-endian environment

IBM Open Data Analytics for z/OS uses Apache Spark 3.2.0 which does not support mixed-endian environments in client deploy mode.

A mixed-endian environment exists if part of your Apache Spark setup, typically the worker, runs on an IBM Z platform (big-endian), and another part, typically the driver, runs on a different platform (little-endian). The Spark driver is the process that hosts your Spark application, and it can run as an independent process or inside a third-party product, such as Scala Workbench (including Jupyter Notebook) or Spark Job Server. However, there are third-party solutions that address this problem and allow Spark applications to run in a mixed-endian environment. See the following alternatives for more details.

If you currently run Apache Spark in a mixed-endian environment and deploy the driver in client mode, consider using one of the following alternatives:

- Deploy the driver in cluster mode. A cluster-mode deployment launches the driver inside the Spark cluster. The default for Apache Spark 3.2.0 is client mode, which launches the driver outside the cluster. Note that Scala Workbench does not support cluster-mode deployment.

For more information about cluster deploy mode, see .

- Consider migrating to a solution that supports Spark in a mixed-endian environment. For instance, Jupyter Notebook Extension to Kernel Gateway (NB2KG) running on an x86 platform can connect to Apache Spark on z/OS through the Jupyter Kernel Gateway to Apache Toree on z/OS (KG2ATz).
- If you use a third-party products to host your Spark driver and the product runs on the Linux® platform, consider migrating it to a Linux on z Systems® platform.
- Start your application from a z/OS system, as follows:
 1. Assemble application and package dependencies into a self-contained JAR file using a build management tool, such as SBT or Apache Maven. For more information about self-contained applications, see [Quick Start: Self-Contained Applications \(spark.apache.org/docs/latest/quick-start.html#self-contained-applications\)](https://spark.apache.org/docs/latest/quick-start.html#self-contained-applications).
 2. Transfer the JAR file to the z/OS UNIX System Services (z/OS UNIX) environment on which your Apache Spark cluster resides, if it is not already there. For information about transferring files to the z/OS UNIX environment, see "File transfer directly to or from z/OS UNIX" in *z/OS UNIX System Services User's Guide*.
 3. Perform **spark-submit** from the z/OS system on which your Apache Spark cluster resides in one of the following ways:
 - Remotely connect to the z/OS UNIX environment and run the **spark-submit** script from a z/OS UNIX interface, such as the standard shell or the ISPF interface.
 - Use FTP or Java to submit a batch job that invokes the **spark-submit** script.
 - For information about submitting a job via FTP, see "Interfacing with JES" in *z/OS Communications Server: IP User's Guide and Commands*.
 - For information about submitting a job via Java, see [Submit batch jobs from Java on z/OS \(https://www.ibm.com/developerworks/systems/library/es-batch-zos.html\)](https://www.ibm.com/developerworks/systems/library/es-batch-zos.html).

Accessibility

The publications for IBM Open Data Analytics for z/OS are available in IBM Knowledge Center and Adobe Portable Document Format (PDF) and comply with accessibility standards. If you experience difficulties when you use any of the information, notify IBM through one of the comment methods described in [“How to send your comments to IBM” on page xiii](#).

Notices

This information was developed for products and services offered in the U.S.A. or elsewhere.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

Site Counsel
IBM Corporation
2455 South Road
Poughkeepsie, NY 12601-5400
USA

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

COPYRIGHT LICENSE:

This information might contain sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Trademarks

IBM, the IBM logo, and [ibm.com](http://www.ibm.com)® are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" (www.ibm.com/legal/copytrade.shtml).

Rocket is a registered trademark of Rocket Software, Inc.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle, its affiliates, or both.

UNIX is a registered trademark of The Open Group in the United States, other countries, or both.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Index

A

- accessibility
 - contact IBM [189](#)
- ACF2 [87](#)
- Anaconda [97](#)
- Apache Spark
 - configuring directory structure for [32](#)
 - configuring memory and CPU options for [65](#)
 - configuring workload management (WLM) for [73](#)
 - in mixed-endian environments [187](#)
 - migrating to [159](#)
 - sample configuration files [163](#)
- APF (Authorized Program Facility)
 - LOAD library [88](#)
- AT-TLS policy
 - sample for z/OS IzODA Livy [169](#)
- authentication, configuring [41](#)
- automation
 - starting started tasks [64](#)
- AZKDFDIV member [86](#)
- AZKSIN00 member [89](#)

B

- bash, level [26](#)

C

- certificates, configuring [43](#)
- checklist, customization [85](#)
- client authentication, configuring [41](#)
- code page [88](#)
- commands for monitoring workload [149](#)
- configuration checker [118](#)
- configuration files
 - samples for Apache Spark
 - spark-defaults.conf [163](#)
 - spark-env.sh [163](#), [164](#)
- configuration options, memory and CPU [171](#)
- configuring
 - APF-authorizing [88](#)
 - ISPF client [91](#)
 - security authorizations [87](#)
 - server component [85](#)
 - server file [89](#)
 - Spark workflow [16](#)
 - started task JCL [90](#)
 - web interfaces [140](#)
 - Workload Manager (WLM) [87](#)
- copying
 - target libraries [88](#)
- CPU configuration options [171](#)
- creating
 - SWIOBJ data set [86](#)
 - system data sets [86](#)

- customization
 - started tasks [60](#)
 - verification [117](#)
- customization checklist [85](#)
- customizing
 - additional authorities and permissions [53](#), [56](#)
 - configuring Apache Spark directory structure
 - creating configuration directory [33](#)
 - creating working directories [35](#)
 - updating configuration files [34](#)
 - configuring Apache Spark memory and CPU options [65](#)
 - configuring certificates and key rings [43](#)
 - configuring client authentication [41](#)
 - configuring Java [57](#)
 - configuring Policy Agent [46](#)
 - configuring TCP/IP for AT-TLS [49](#)
 - defining AT-TLS policy rules [49](#)
 - defining security for Policy Agent [47](#)
 - Policy Agent configuration files [48](#)
 - setting up a user ID [28](#)
 - starting Policy Agent [53](#)
 - stopping Policy Agent [53](#)
 - verifying env command path [32](#)
 - verifying Java and bash [26](#)
 - verifying z/OS UNIX [28](#)

D

- Data Service server
 - configuring [85](#)
 - naming conventions [86](#)
 - starting [90](#)
 - stopping [90](#)
- Data Service Studio
 - download [95](#)
 - installing [95](#)
- data sets
 - created by installation [181](#)
- Data Studio
 - verifying installation [96](#)
- DBCS [88](#)
- default subsystem name [89](#)
- delimited data, configuring [93](#)
- digital certificates [43](#)
- directory structure, configuring for Apache Spark [32](#)
- double-byte character set [88](#)
- Downloading
 - studio component [95](#)

E

- event log directory [142](#)

F

- feedback [xiii](#)

G

generation data set, configuring access [91](#)
Global Registry log stream, creating [89](#)

H

history service [142](#)

I

IBM Health Checker for z/OS [151](#)

IBM Java, configuring [57](#)

installation

- data sets [181](#)
- installation user ID [5](#)
- planning checklist [6](#)
- planning for [5](#)
- required resources [6](#)
- required skills [5](#)
- requisite products [6](#)
- time requirements [5](#)
- verification [117](#)

installing

- JDBC Gateway [99](#)
- studio component [95](#)

ISPF client [91](#)

J

Java, level [26](#)

JDBC Gateway

- installing [99](#)
- starting the administrative console [102](#)
- starting the server [101](#)

K

key rings [43](#)

L

LOAD library [88](#)

log files [143](#)

M

memory and CPU options, configuring for Apache Spark [65](#)

memory configuration options [171](#)

mixed-endian environments [187](#)

monitoring, resource, *See* resource monitoring

N

naming conventions

- Data Service server [86](#)

network ports [37](#), [63](#)

networking

- configuration [37](#), [63](#)

Notices [191](#)

P

performance, system [73](#)

planning for installation

- checklist [6](#)
- installation user ID [5](#)
- product overview [5](#)
- required resources [6](#)
- required skills [5](#)
- requisite products [6](#)
- time requirements [5](#)

Policy Agent

- starting [53](#)

Policy Agent, configuration files [48](#)

Policy Agent, configuring [46](#)

Policy Agent, security for [47](#)

ports, network [37](#), [63](#)

preparation

- customizing [15](#)

product verification [123](#)

PTF level, considerations for updating [159](#)

Python [97](#)

R

RACF [87](#)

Resource Measurement Facility (RMF), z/OS [144](#)

resource monitoring

- history service [142](#)
- IBM Health Checker for z/OS [151](#)
- Spark log files [143](#)
- using z/OS RMF
 - Monitor III reports [144](#)
 - Postprocessor [148](#)
- web interfaces [137](#)
- z/OS and z/OS UNIX commands [149](#)

restrictions [185](#)

S

securing

- Spark web interfaces [141](#)

security

- configuring client authentication [41](#)

security authorizations

- ACF2 [87](#)
- RACF [87](#)
- Top Secret Security [87](#)

sending to IBM

- reader comments [xiii](#)

server event facility (SEF)

- configure delimited data [93](#)
- configure GDG access [91](#)

server file [89](#)

SMP/E [88](#)

Spark properties, z/OS-specific [177](#)

Spark workflow

- upgrading [20](#), [22](#)

started task JCL [90](#)

started tasks

- automation [64](#)
- customization [60](#)

- started tasks (*continued*)
 - set and export environment variables [62](#)
- summary of changes for IBM Open Data Analytics for z/OS
- Installation and Customization Guide [xv](#)
- system data sets
 - creating [86](#)
- system performance [73](#)

T

- target libraries [88](#)
- TCP/IP, configuring for AT-TLS [49](#)
- Top Secret Security [87](#)

U

- updating to latest PTF level, considerations [159](#)
- user interface (UI), web [137](#), [140](#)
- using
 - configuration checker [118](#)
 - IzODA configuration checker [118](#)

V

- verification
 - configuration checker [118](#)
 - customization [117](#)
 - installation [117](#)
- verifying [121](#)
- verifying installation
 - Data Studio [96](#)

W

- web user interface (UI) [137](#), [140](#)
- workload management (WLM), configuring for Apache Spark [73](#)
- Workload Manager (WLM)
 - configuring [87](#)
- workloads [73](#)

Z

- z/OS commands for workload monitoring [149](#)
- z/OS IzODA Livy
 - sample AT-TLS policy [169](#)
- z/OS UNIX requirements [28](#)



SC27-9033-00

