IBM Platform Symphony RFC 3719 Readme File

R-Platform Symphony integration

This feature enables you to run your R scripts on Platform Symphony. You can utilize the power of Platform Symphony directly within R, and submit workload to Platform Symphony. Benefits of using this feature include support for running the R script on both the local side, and on the Platform Symphony service side. Without this feature, you cannot run R scripts on the Platform Symphony service side.

Readme file for: IBM® Platform Symphony
Product/Component Release: Symphony 7.1
Update Name: R-Platform Symphony integration
Fix ID: sym-7.1-build242470
Publication date: 1 September 2014
Last modified date: 4 September 2014

1. Scope

Applicability	
Operating system	Linux 64 bit
Symphony version	7.1
Cluster types	This feature applies to a single Platform Symphony cluster or the Platform Symphony Developer Edition.
Other	None
Dependencies	
	To install this feature, you must have: • Platform Symphony 7.1 or Platorm Symphony Developer Edition 7.1 installed. • Open source R installed on the cluster (R-2.14.2 is supported and has been tested).
Limitations	
Limitations	a. The R function set is not thread-safe. Ensure the R APIs in the client script are called in a single thread. b. MapReduce workload is not supported. c. On the client side, the C++ RClient agent process has blocked signals. This is to avoid exiting abnormally when receiving unexpected signals (such as SIGTERM) from the client script. d. The R input function, input parameter list, and output object will be serialized to raw vectors internally. The raw vector size is limited to 2^{31} to 1 bytes.
Known issues	None.

2. Configuration

1) Prerequisites

Open source R must be installed on the client host and all compute hosts. R-2.14.2 is supported and has been tested for this feature. Build 242470 contains the required files for Platform Symphony and Platform Symphony Developer Edition clusters.

2) Installation files

Build 242470 is applicable to Linux 64-bit hosts. The <top_dir> below refers to the top directory in the packages, such as `R-lnx26-lib23-x64-7.1`..
This package includes the following files:

File name	Description
`<top_dir>/cshrc.symphonyR;` `<top_dir>/profile.symphonyR`	Shell files for setting the environment variables on the client side.
`<top_dir>/SymphonyRAPP.7.1.xml`	Application profile for the R-Platform Symphony integration.
`<top_dir>/ RUtils.R`	Internal module for R functions.
`<top_dir>/RClient.R`	The R script wrapper on the client side. It provides a set of R APIs to communicate with the Platform Symphony C++ RClient agent and to submit tasks.
`<top_dir>/RClientAgent`	The Platform Symphony C++ RClient agent binary, which is a client process for Platform Symphony. It communicates with an R script so that you can submit task functions from the client side, and run the task functions run on the service side.
`<top_dir>/RService.tar`	The service package for R- Platform Symphony

File name	Description
	integration.
`<top_dir>/Samples/test_blocking.R`	A sample case demonstrating how to submit blocking workload.
`<top_dir>/Samples/test_non_blocking.R`	A sample case demonstrating how to submit non-blocking workload.

3) Installation procedure

1. Download the `R-lnx26-lib23-x64-7.1.tar.gz` package from the IBM Fix Center.

2. Decompress the package:

```
> tar xzvf R-lnx26-lib23-x64-7.1.tar.gz
```

4) Configuration procedure

Configuration source	Setting	Behavior
Edit `cshrc.symphony` or `profile.symphonyR`	Replace `@SYMPHONY_R_WORK_DIR @` with the absolute path where the top directory is located. For example, if you extracted the installation package to `/opt/`, replace the `@SYMPHONY_R_WORK_DIR@` with `/opt/R-lnx26-lib23-x64-7.1`. Set environment variables to make them take effect on the client side. For example, if SYMPHONY_R_LOG_LEVEL is set to `INFO`, the level of logs on the client side will be set	After sourcing the shell file on the client, SYMPHONY_R_WORK_DIR will point to the directory where the top directory is located. Other environment variable settings will take effect on the client side.

Configuration source	Setting	Behavior
	to `INFO`.	
`SymphonyRAPP.7.1.xml`	If you want to change the name of the application, change the `applicationName` attribute in the Consumer section of the application profile to your desired name. The current `applicationName` value is `SymphonyRApp`.	After registering this profile, the application with this name will be registered. **Note:** If the application name is customized, remember to adjust the `connInfo$appName` setting in `sym.initialize`. Otherwise, if the `connInfo$appName` does not match the application name configured in the application profile, the RClient agent cannot submit tasks to Platform Symphony.
	Change the SessionTypes section of the application profile, including the names of the session types and detailed configuration for the session types.	After registering this profile, the application with the customized session types setting will take effect. **Note:** If the names of session types are customized, remember to adjust the `jobInfo$type` setting in `sym.createJob`. Otherwise, if the `jobInfo$type` does not match any session type name configured in the application profile, the RClient agent cannot submit tasks to Plaform Symphony.

Configuration source	Setting	Behavior
	If R is not installed under `/usr/local/bin/`, change the path of R in <env name="PATH"> under **Service > osTypes > osType>**, . The current PATH value is `/usr/local/bin:/bin`. For example, you installed R under `/opt/R/bin/`, change the PATH value to `/opt/R/bin:/bin.`	After registering this profile, the new PATH value will take effect.
	If the **sh** command (used to spawn R processes on the service side) is not installed in `/bin/`, change the path of **sh** in <env name="PATH"> under **Service > osTypes > osType.** The current PATH value is `/usr/local/bin:/bin`. For example, if **sh** is installed under `/usr/bin/`, change the PATH value to `/opt/R/bin:/usr/bin`.	After registering this profile, the new PATH value will take effect.
	Set environment variables in <env name="@ENVNAME@"> under **Service > osTypes > osType** to make them take effect on the service side. For example, if <env name="SYMPHONY_R_LOG_LEVEL">INFO</env> is set, the level of logs on the service side will be set to `INFO`.	After registering this profile, the new environment variable settings will take effect on the service side.

5) Verification procedure

To verify the installation and configuration on Platform Symphony:

1. Set up a cluster with only one host and source the `$EGO_TOP/profile.platform` in the cluster.
2. Go to the top directory of the decompressed package:
```
> cd <top_dir>
```
3. Edit `cshrc.symphonyR` or `profile.symphonyR` to replace the `@SYMPHONY_R_WORK_DIR@` with the absolute path where the top directory is located.
4. Source `cshrc.symphonyR` or `profile.symphonyR`.
5. Create the consumer `/SymphonyRApp` in Plaform Management Console.
6. Deploy the service package:
```
> soamdeploy add SymphonyRService -p RService.tar -c "/SymphonyRApp"
```
7. Register the application:
```
> soamreg SymphonyRAPP.7.1.xml
```
8. Go to the samples directory to test the workload:
```
> cd Samples
```
9. Run the sample `test_blocking.R` to test the blocking workload:
```
> Rscript test_blocking.R
```

   The following is an example successful output of running this sample:
```
Succeeded to initialize the Symphony framework.
Succeeded to create the Symphony job <SymphonyR>. The job id is <202>.
Succeeded to submit blocking task workload.
Task <1>: 5
Succeeded to close job: <SymphonyR>.
Succeeded to uninitialize the Symphony framework.
```
10. Run the sample `test_non_blocking.R` to test the non-blocking workload:
```
> Rscript test_non_blocking.R
```

   The following is an example successful output of running this sample:
```
Succeeded to initialize the Symphony framework.
Succeeded to create the Symphony job <SymphonyR>. The job id is <203>.
Succeeded to send task: <1>.
Succeeded to send task: <2>.
Succeeded to retrieve outputs:
Task <1>: 102 123 104 125 106 127 108 129 110 131 112 133 114 135 116 137
118 139 120 141
Task <2>: 103 124 105 126 107 128 109 130 111 132 113 134 115 136 117 138
```

```
119 140 121 142
Succeeded to close job: <SymphonyR>.
Succeeded to uninitialize the Symphony framework.
```

3. Usage

1) How this feature works

This feature allows you to use the power of Platform Symphony directly within R and to submit workload to Platform Symphony. It provides a new set of function calls for R, so that you can send R functions along with its input as workload to Platform Symphony, and to gather the results from the grid.

2) Environment variables and log files

The following are the environment variables for R-Platform Symphony integration. Set the environment variables on the client side to make them take effect in the processes running on the client side. For the environment variables to take effect on the service side, specify the environment variables in the application profile.

- **SYMPHONY_R_WORK_DIR (**applicable only on the client side**):**
  The working directory path of the R- Platform Symphony integration package. Specify the absolute path where the top directory of the package is located.

- **SYMPHONY_R_LOG_STDIO** (applicable only on the client side**):**
  If it is set to `YES` or `yes`, the log messages in the RClient agent process will be printed to stdio, instead of to a log file. Otherwise, the log messages will be printed into the `RClientAgent.<host_name>.log` file in the current directory on the local host. By default, this environment variable is not set.

  Note: If the log messages in the RClient agent process are printed to stdio, which is duplicated with the R client process, there is a possibility of message overlapping between the RClient agent process and R client process.

- **SYMPHONY_R_LOG_LEVEL** (applicable to both the client and service side)**:**
  The level of details for the log files. The valid values are `DISABLE`, `ERROR`, `WARN`, `INFO`, `DEBUG`, or `ALL`. The default value is `INFO`.

- **Log files created on the client and service side:**
  On the client side, if SYMPHONY_R_LOG_STDIO is not set to `YES` or `yes`, then the log messages in the RClient agent process will be printed into the `RClientAgent.<host_name>.log` file in the current directory on the local host.

  On the service side, the log messages of the C++ R service process will be printed into the `RServiceContainer.<host_name>.<appname>.<index>.log` file (for example, `RServiceContainer.rh63n2.SymphonyRApp.1.log`). The logs of the RService agent (R process on service side) will be printed into the `RServiceAgent.<hostname>.<appname>.<index>.log` file (for example, `RServiceAgent.rh63n2.SymphonyRApp.1.log`). All the logs on service side will be created in the `${SOAM_HOME}/logs` directory.

3) Plaform Symphony R function set

This feature provides a new set of R function calls, so that you can submit both blocking and non-blocking workload to Platform Symphony.

- **sym.initialize (connInfo = NULL);**
  This function is used to start the Platform Symphony C++ client agent, and to initialize the Plaform Symphony framework.
  connInfo: Information used to create the Platform Symphony connection. The default value is `NULL`, which means the default values for `connInfo$appName`, `connInfo$userName`, and `connInfo$passWord` will be used.
  connInfo$appName: The application name. If set to `NULL`, the default value (`SymphonyRApp`) will be used.
  connInfo$userName and ***connInfo$passWord****:* The application credentials. If set to `NULL`, the default values `Guest` and `Guest` will be used.

  Returns*:* The return code to indicate whether the API is successful.

- **sym.uninitialize();**

  This function is used to deinitialize the Platform Symphony framework and to terminate the Platform Symphony C++ RClient agent.

  ***Returns**:* The return code to indicate whether the API is successful.

- **sym.createJob(jobName, jobInfo = NULL, libraries = NULL);**

  This function is used to create a Platform Symphony session.
  jobName: The Platform Symphony session name.
  jobInfo: Information used to create a Platform Symphony session. The default value is `NULL`, which means the default values for `jobInfo$type` and `jobInfo$compression` will be used.
  jobInfo$type: The Platform Symphony session type. If set to `NULL`, the default value (`RecoverableAllHistoricalData`) will be used.
  jobInfo$compression: Indicates whether to enable compression of data transferred between the Platform Symphony client and the Platform Symphony service. The valid values are `TRUE`, `true`, `FALSE`, and `false`. If set to `NULL`, or if values are not valid, the default value (`TRUE`) will be used.
  libraries: The third party libraries needed to be loaded into R when executing the script. If you use third party libararies, ensure you have them installed in R on all relevant compute hosts.

  ***Returns**:* The created job ID.

- **sym.applyNonBlocking (jobName, func, params);**

  This function is used to submit the non-blocking workload into Platform Symphony.
  jobName: The Platform Symphony session name.
  func: The R function that will be executed in the R process on compute hosts.
  params: The input parameters for the R function.

  ***Returns**:* The created task ID.

- **sym.getResultsNonBlocking (jobName);**

  This function is used to retrieve output for the submitted non-blocking workload.
  jobName: The Platform Symphony session name.

  ***Returns**:* The result of R non-blocking workload.

- ***sym.applyBlocking (jobName, func, params);***

This function is used to submit the blocking workload into Platform Symphony.

jobName: The Platform Symphony session name.

func: The R function that will be executed in the R process on compute hosts.

params: The input parameters for the R function.

Returns: The result of R blocking workload.

- **sym.closeJob (jobName);**

  This function is used to close a Platform Symphony session.

  jobName: The Plaform Symphony session name.

  Returns: The return code to indicate whether the API is successful.

4. Troubleshooting

1. Log files

Both the processes on the client and service side have a logging mechanism. Refer to section 3.2 “Environment variables and log files” for details.

5. Copyright and trademark information