horizontal line

Automating Product Migration Testing

**Intern Project Number: 255**

### Mentored By : Ashen Weerathunga

### Co - Mentored By : Chamath Samarawickrama

### Intern : Jayana Gunaweera

# OVERVIEW

Project Description :

We provide a separate migration client for users to handle the WSO2 Identity Server migration between different product versions. If a user wants to migrate to the latest version from an old WSO2 Identity Server version, they can execute the migration client and follow the migration documentation to complete the migration.

We have to test these migrations every time when we do a new release with different infrastructure combinations such as database and OS. Since we have multiple product versions, executing manual migration testing with all the infrastructure combinations takes lots of time and resources. Therefore we can reduce that overhead by automating the migration client test execution.

Scope:

* Understanding the WSO2 IS migration process and tools
* Evaluate the options to automate the migration testing process with different infrastructure combinations
* Implement the required process with the relevant infrastructure
* Provide the necessary documentation and other resources

# Implementation

The workflow automation is implemented using GitHub Actions, which provides a flexible and customizable framework for continuous integration and deployment.

**Workflow Overview:**

The "Migration Tester" workflow allows users to specify the necessary parameters for testing the migration of WSO2 IS from one version to another. The workflow can be manually triggered using the GitHub Actions workflow\_dispatch event. It collects inputs such as the current version of WSO2 IS, the target migrating version, the database type, the operating system, and the download URLs for the old and new versions.

**Key Steps in the Workflow**

**a. Trigger:**

The workflow can be triggered manually using the GitHub Actions workflow\_dispatch event. This gives users control over when to initiate the migration testing process.

**b. Inputs:**

The workflow defines several input parameters that users can provide when triggering the workflow. These parameters include:

**currentVersion**: Users specify the version of WSO2 IS presently installed in their environment. The options include versions 5.9.0, 5.10.0, 5.11.0, 6.0.0 and 6.1.0

**migratingVersion**: Users specify the version of WSO2 IS they want to migrate to. The available options include versions 5.10.0, 5.11.0, 6.0.0 and 6.1.0

**database**: Users select the database type to be used during migration, with options including "mysql", "mssql", and "postgres".

**os**: Users select the operating system relevant to their environment. Options include "ubuntu-latest" and "macos-latest"

**urlOld**: Users provide the URL to download the old version of WSO2 IS that corresponds to their currentVersion selection.

**urlNew**: Users provide the URL to download the version they want to upgrade WSO2 IS to, based on the migratingVersion specified.

**This workflow file comprises 6 Jobs.These jobs automates the testing process for product migration in the context of WSO2 IS (Identity Server). These jobs are triggered based on specific conditions and are designed to handle different database migrations on different Operating Systems.**

**1.ubuntu-postgres-migration**: Migrates PostgreSQL databases on Ubuntu.

**2.ubuntu-mssql-migration**: Migrates Microsoft SQL Server (MSSQL) databases on Ubuntu.

**3.macos-mssql-migration:** Migrates MSSQL databases on macOS.

**4.macos-postgres-migration**: Migrates PostgreSQL databases on macOS.

**5.ubuntu-mysql-migration:** Migrates MySQL databases on Ubuntu.

**6.macos-mysql-migration**: Migrates MySQL databases on macOS.

**7.validate-migration**: Validates migration processes and performs post-migration testing.

These jobs cover a range of migration tasks, including setup, SQL script execution, and running migration automation scripts. Logs are persisted and uploaded as artifacts for analysis. The "validate-migration" job checks the logs for errors and provides a log report of the migration process.

**Common Steps in above mentioned Jobs:**

**Checkout code**: The job begins by checking out the code repository to access the necessary files for migration testing.

**Execute Migration Automation Script**: A migration automation script specific to the database and operating system is executed. The script handles the migration process and may require inputs such as URLs, versions, credentials, and tokens.

**Persist Logs**: The logs generated during the migration process are saved to a designated location for further reference and analysis. The logs are typically stored in a file named "logs.txt."

**Upload Artifacts**: Artifacts, including the log files, are uploaded for storage and easy access. The artifacts are typically named based on the version, database, and operating system involved in the migration.

**Download Artifacts** (validate-migration): In the "validate-migration" job, artifacts containing logs from previous migration jobs are downloaded for analysis.

These common steps ensure consistency and provide a structured approach to the migration testing process across different database types and operating systems.

**Other Steps in above mentioned Jobs:**

**1."ubuntu-postgres-migration" Job**

**Setup Docker environment**

* The job sets up the Docker environment required for running the migration tests specifically on the Ubuntu operating system.
* It performs the following actions:
* Retrieves the ID of the "Migration Tester" workflow from the GitHub API using curl and jq commands.
* Updates and upgrades packages and installs necessary dependencies.
* Adds the Docker GPG key and adds the Docker repository.
* Updates the packages again and installs Docker CE.

**Run Docker command**

This step executes various Docker commands to manage the PostgreSQL container and perform database-related actions specific to the migration process.

The actions performed include:

* Creating the PostgreSQL container if it doesn't already exist.
* Starting the PostgreSQL server if it's not running.
* Creating a new database named 'testdb'.
* Granting all privileges to the 'postgres' user.
* Changing the password for the 'postgres' role.
* Copying SQL files to the PostgreSQL container and executing them to create and populate the necessary tables and data.

**2."ubuntu-mysql-migration" Job**

**Conditional Execution**

The "ubuntu-mysql-migration" job is executed only if the specified conditions are met. It checks if the database selected is "mysql" and the operating system is "ubuntu-latest" using the if expression.

**Checkout code**

This step uses the actions/checkout@v2 action to fetch the code repository for further processing.

**3."ubuntu-mssql-migration" Job**

**Create MS SQL Database**

In this step, the MS SQL Server database is created.

* The required package, "mssql-tools," is installed using the apt-get command.
* The sqlcmd utility is used to connect to the SQL Server and execute a SQL query to create a database named "testdb."

**Copy SQL files to MSSQL container**

In this step, SQL script files for various components of the migration process are copied to the MS SQL Server container.

* The docker cp command is used to copy the SQL files from the local repository to the running MS SQL Server container.
* The docker exec command is then used to execute the SQL scripts inside the container, importing the data into the "testdb" database.

**4. "macos-mysql-migration" Job**

**Conditional Execution**

The "macos-mysql-migration" job is executed only if the specified conditions are met. It checks if the database selected is "mysql" and the operating system is "macos-latest" using the if expression.

**Checkout code**

This step uses the actions/checkout@v2 action to fetch the code repository for further processing.

**5."macos-postgres-migration" Job**

**Set up PostgreSQL**

This step sets up PostgreSQL on the macOS machine.

* It uses the Homebrew package manager to install PostgreSQL and initializes the database.
* The PostgreSQL server is started, and a user and database are created.
* Privileges on the database are granted to the user.

**Execute Postgres SQL scripts**

In this step, several Postgres SQL scripts are executed on the "testdb" database.

Each script file is executed using the psql command, passing the script file as input.

**Verify script execution**

This step verifies the execution of the SQL scripts by running a command to show the server version of PostgreSQL.

The psql command is used to execute the SQL query.

**6."macos-mssql-migration" Job**

**Install MS SQL Server Components**

* The job uses the "potatoqualitee/mssqlsuite" action to install MS SQL Server components on the macOS machine.
* The components being installed include the SQL Engine, SQL Client, SQLPackage, and LocalDB.
* The version specified is 2019, and a strong password is set for the SA (System Administrator) account.

**Create MS SQL Database**

* This step creates a database named "testdb" on the MS SQL Server.
* The sqlcmd command is used to execute the SQL query that creates the database.
* The SA username and password are provided for authentication.

**Run query**

In this step, a query is executed on the "testdb" database to retrieve the SQL Server version information.

* The sqlcmd command is used to execute the SQL query.
* The SA username and password are provided for authentication.

**7. "validate-migration" Job**

The "validate-migration" job is part of the "Migration Tester" workflow and is responsible for validating the migration process. This job checks the status of previous migration jobs and performs post-migration testing. Here are the key steps involved in the "validate-migration" job, as defined in the provided YAML file:

**Step 1: Define Dependencies**

The job specifies a list of dependencies that need to be completed before executing the "validate-migration" job.

* The dependencies include the migration jobs for different databases and operating systems, such as "ubuntu-postgres-migration," "ubuntu-mssql-migration," "macos-mssql-migration," "macos-postgres-migration," "ubuntu-mysql-migration," and "macos-mysql-migration."
* These dependencies ensure that the migration jobs are executed first before validating the migration.

**Step 2: Conditional Execution**

The "validate-migration" job is configured to execute unconditionally using the if expression with the value "always()".

This ensures that the job runs regardless of the conditions specified in the workflow.

**Step 3: Checkout code**

This step uses the actions/checkout@v2 action to fetch the code repository for further processing.

**Step 4: Download Artifacts**

The job downloads the artifacts generated by the migration jobs.

* It uses the actions/download-artifact@v2 action to retrieve the artifacts.
* The artifacts include the logs generated during the migration process.
* The downloaded artifacts are stored in the designated directory within the workspace.

**Step 5: Post Migration Testing**

In this step, post-migration testing is performed on the migrated system.

* The logs from the downloaded artifacts folder are accessed and processed line by line.
* Each line is checked for the presence of error messages by using a pattern matching condition.
* If an error is detected (indicated by the presence of "ERROR," "error," or "Error"), the line is printed in red color to highlight the error.
* Otherwise, the line is printed as is.

The "validate-migration" job ensures that the migration process is validated after the completion of the migration jobs for different databases and operating systems. It retrieves the logs from the previous migration jobs and performs post-migration testing by analyzing the logs for any error messages.

**Main Migration Shell Scripts**  
  
Introduction

2 shell scripts called [**migration-script-ubuntu.sh**](https://github.com/JayanaGunaweera01/product-is/blob/JayanaGunaweera01-migration-tester-test/.github/migration-tester/migration-automation/ubuntu-os/migration-script-ubuntu.sh)and [**migration-script-mac.sh**](https://github.com/JayanaGunaweera01/product-is/blob/JayanaGunaweera01-migration-tester-test/.github/migration-tester/migration-automation/mac-os/migration-script-mac.sh) have been written to automate product migration testing from one version of a product to another in a Linux (Ubuntu) or a Macos environment. These scripts focus on migrating databases such as MySQL, MSSQL, or PostgreSQL using GitHub Actions.Both scripts follow the same procedure but all the steps in doing a manual migration and migration testing are automated here.

The script begins by defining color variables for enhanced output formatting.

The system is updated using the apt-get command to ensure the latest packages are available.

The script navigates to the relevant directory for the migration automation process.

**Input Variables**

The script retrieves input values from the workflow dispatch, such as URLs, versions, databases, operating systems, and credentials.

These inputs are used throughout the script to customize the migration process.

**Pre-migration Steps**

The script performs various operations to set up the migration environment before starting the migration process.

The env.sh file is modified to replace placeholders with actual version numbers.

Java is installed using the apt-get command.

The IS\_HOME\_OLD directory is created to store the existing WSO2IS installation.

The required WSO2IS zip file is downloaded based on the provided version using Google Drive API and an access token.

The downloaded zip file is extracted into the IS\_HOME\_OLD directory.

Update IS Packs

The script updates the IS packs using the update-pack.sh script, which takes email and password parameters.

Deployment Configuration

The script modifies the deployment.toml file based on the provided database, operating system, and versions.

If the selected database is MySQL, additional setup is performed using the setup-mysql-ubuntu.sh script.

JAR Files and Data Population

The script copies the necessary JAR files based on the database and operating system.

The data population script (automated-data-population-and-validation-script-ubuntu.sh) is executed to populate data in the database, create users, tenants, user stores, and generate OAuth tokens.

Stopping the Existing WSO2IS

The script stops the running WSO2IS instance using the stop-server.sh script.

New WSO2IS Installation

The IS\_HOME\_NEW directory is created to store the new WSO2IS installation.

The latest WSO2IS zip file is downloaded using Google Drive API and an access token.

The downloaded zip file is extracted into the IS\_HOME\_NEW directory.

**Migration Client Setup**

The migration client is downloaded and extracted from the provided zip file.

The migration client and resources are copied to the appropriate directories in the new WSO2IS installation.

Configuration Changes

The migration-config.yaml file is modified to include the current and migrating versions and the operating system.

**Copying Files**

Various files, such as JAR files, userstores, and tenants, are copied from the old WSO2IS installation to the new one.

Check if all files are copied successfully.

Change the deployment.toml file in the newIS version based on the provided parameters.

Execute consent management DB scripts for IS 5.11.0 - MySQL if the migrating version and database match the specified conditions.

Execute consent management DB scripts for IS 5.11.0 - MSSQL if the migrating version and database match the specified conditions.

Execute consent management DB scripts for IS 5.11.0 - PostgreSQL if the migrating version and database match the specified conditions.

Run the migration client.

Display migration details such as currentVersion, migratingVersion, database, operating system, and time and date.

Start the migration server.

Go to the automation home directory.

Stop the WSO2IS server.

**Post-migration Steps**

Apply a special configuration change if migrating from IS 5.9 by modifying the deployment.toml file.

* If the current version is "5.9.0," this step is included to make a special configuration change in the deployment.toml file.
* The script navigates to the deployment directory and modifies the deployment.toml file by commenting out the line type = "database" and uncommenting the line type = "database\_unique\_id".
* It then prints the content of the modified deployment.toml file.

Start the migrated WSO2 Identity Server (IS) with the migrating version.

Enter login credentials (admin) and access the management console home page.

Go to the data population directory for service provider creation.

**Database-validation**

* Run the the scripts to validate the database

Go back to the automation home directory.

Stop the migrated WSO2IS server.

Display the end message for automating product migration testing.

# Urls used

<https://drive.google.com/u/0/uc?id=1GU32FtPGvvB2WsmQoPnHr5yn1M6ddL-h&amp;amp;export=download&amp;amp;confirm=t&amp;amp;uuid=712b27d8-ea10-4e0b-bbbd-3cde24b1d92e&amp;amp;at=AKKF8vzpFDL5XdIrNRv6KFY0ZvPr:1687251333945&amp;confirm=t&amp;uuid=efe88210-d059-4968-84c6-7a1236bb6ef9&amp;at=AKKF8vxFWIDReSULenUtKrASKULT:1687251490306&confirm=t&uuid=7017f976-902b-4050-a3a6-22b26bb46d88&at=AKKF8vyia4DpEk742C_FTMCmJDE9:1687251582718>

<https://github.com/wso2/product-is/releases/download/v5.10.0/wso2is-5.10.0.zip>

<https://github.com/wso2/product-is/releases/download/v6.0.0-rc2/wso2is-6.0.0-rc2.zip>

<https://github.com/wso2/product-is/releases/download/v6.1.0-rc1/wso2is-6.1.0-rc1.zip>

# ERROR

## Migrating from IS 6-6.1-mysql

[] ERROR {org.wso2.carbon.identity.workflow.impl.internal.WorkflowImplServiceComponent} - Error occured while adding default bps profile. org.wso2.carbon.identity.workflow.impl.WorkflowImplException: Error while decrypting the password for BPEL Profile embeded\_bps

Caused by: org.wso2.carbon.core.util.CryptoException: An error occurred while decrypting data.

at org.wso2.carbon.core.util.CryptoUtil.decrypt(CryptoUtil.java:317)

at org.wso2.carbon.core.util.CryptoUtil.base64DecodeAndDecrypt(CryptoUtil.java:431)

Caused by: org.wso2.carbon.crypto.api.CryptoException: An error occurred while decrypting using the algorithm : 'AES/GCM/NoPadding', and crypto provider : 'org.wso2.carbon.crypto.provider.KeyStoreBasedInternalCryptoProvider'

Caused by: java.security.InvalidKeyException: Key for algorithm RSA not suitable for symmetric enryption.

at org.bouncycastle.jcajce.provider.symmetric.util.BaseBlockCipher.engineInit(Unknown Source)

at org.bouncycastle.jcajce.provider.symmetric.util.BaseBlockCipher.engineInit(Unknown Source)

at java.base/javax.crypto.Cipher.init(Cipher.java:1283)

at java.base/javax.crypto.Cipher.init(Cipher.java:1223)

Reproduced with the suggestion of

“I could reproduce this behavior in IS 5.3.0 to IS 5.11.0 migration when we don't change the default configuration for 'currentEncryptionAlgorithm' mentioned under 'EncryptionAdminFlowMigrator' in migration-config.yaml. As OAEP is not enabled by default in IS 5.3.0, we don't use the algorithm mentioned in 'currentEncryptionAlgorithm' to encrypt data. Hence, if we change the algorithm accordingly the problem reported can be solved.” <https://github.com/wso2/product-is/issues/10573>

I don't execute Bps database scripts when setting up the database.Check if thats the reason.

## Future Developments