Cloudera Flow Management 4.13.0

Cloudera Flow Management Migration Tool

Date published: 2019-06-26
Date modified: 2026-10-21

CLOUDERA

https://docs.cloudera.com/

Legal Notice

© Cloudera Inc. 2026. All rights reserved.

The documentation is and contains Cloudera proprietary information protected by copyright and other intellectual property rights. No license under copyright or any other intellectual property right is granted herein.

Unless otherwise noted, scripts and sample code are licensed under the Apache License, Version 2.0.

Copyright information for Cloudera software may be found within the documentation accompanying each component in a particular release.

Cloudera software includes software from various open source or other third party projects, and may be released under the Apache Software License 2.0 (“ASLv2”), the Affero General Public License version 3 (AGPLv3), or other license terms. Other software included may be released under the terms of alternative open source licenses. Please review the license and notice files accompanying the software for additional licensing information.

Please visit the Cloudera software product page for more information on Cloudera software. For more information on Cloudera support services, please visit either the Support or Sales page. Feel free to contact us directly to discuss your specific needs.

Cloudera reserves the right to change any products at any time, and without notice. Cloudera assumes no responsibility nor liability arising from the use of products, except as expressly agreed to in writing by Cloudera.

Cloudera, Cloudera Altus, HUE, Impala, Cloudera Impala, and other Cloudera marks are registered or unregistered trademarks in the United States and other countries. All other trademarks are the property of their respective owners.

Disclaimer: EXCEPT AS EXPRESSLY PROVIDED IN A WRITTEN AGREEMENT WITH CLOUDERA, CLOUDERA DOES NOT MAKE NOR GIVE ANY REPRESENTATION, WARRANTY, NOR COVENANT OF ANY KIND, WHETHER EXPRESS OR IMPLIED, IN CONNECTION WITH CLOUDERA TECHNOLOGY OR RELATED SUPPORT PROVIDED IN CONNECTION THEREWITH. CLOUDERA DOES NOT WARRANT THAT CLOUDERA PRODUCTS NOR SOFTWARE WILL OPERATE UNINTERRUPTED NOR THAT IT WILL BE FREE FROM DEFECTS NOR ERRORS, THAT IT WILL PROTECT YOUR DATA FROM LOSS, CORRUPTION NOR UNAVAILABILITY, NOR THAT IT WILL MEET ALL OF CUSTOMER’S BUSINESS REQUIREMENTS. WITHOUT LIMITING THE FOREGOING, AND TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, CLOUDERA EXPRESSLY DISCLAIMS ANY AND ALL IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, QUALITY, NON-INFRINGEMENT, TITLE, AND FITNESS FOR A PARTICULAR PURPOSE AND ANY REPRESENTATION, WARRANTY, OR COVENANT BASED ON COURSE OF DEALING OR USAGE IN TRADE.

Contents

Migrating from Cloudera Flow Management 2.1.7 to 4.13.0 5

Cloudera Flow Migration Tool release notes 5
- What's new 5
- Support Matrix 5
- System Requirements 6
- Download location 6

Cloudera Flow Migration Tool overview 7
- Key features 8
- Migration stages 8

Recommended migration workflow 10

Migration best practices 11

Migrating a data flow using the Cloudera Flow Migration Tool 12
- Preparing for migration 12
  - Setting up your environment 12
  - Configuring the Cloudera Flow Migration Tool 14
- Migrating flows versioned with NiFi Registry [Technical Preview] 16
- Migrating a flow using flow.json as input 34
  - Migrating templates using flow.json as input 34
  - Migrating variables using flow.json as input 39
  - Migrating components using flow.json as input 45
  - Using migrate-all with flow.json as input 53
  - Handling file formats missing sensitive property values 55
- Migrating a flow using flow.json.gz as input 57
  - Migrating templates using flow.json.gz as input 57
  - Migrating variables using flow.json.gz as input 61
  - Migrating components using flow.json.gz as input 67
  - Using migrate-all with flow.json.gz as input 75
  - Handling file formats missing sensitive property values 77
- Migrating a flow using flow definition JSON as input 79
  - Migrating templates using flow definition JSON as input 79
  - Migrating variables using flow definition JSON as input 79
  - Migrating components using flow definition JSON as input 83
  - Using migrate-all with flow definition JSON as input 88
- Migrating a flow using a directory with flow definition JSON files as input 90
  - Migrating variables using a directory with flow definition JSON files as input 90
  - Migrating components using a directory with flow definition JSON files as input 97
  - Using migrate-all using a directory with flow definition JSON files as input 106
- Migrating a flow using template.xml as input 107
  - Migrating templates using template.xml as input 107

Migrating variables using template.xml as input....112
Migrating components using a template with variables....115
Migrating components using a template without variables....120
Using migrate-all with template.xml as input....124
Migrating a flow using a directory with template.xml files as input....124
Migrating templates using a directory with template.xml files as input....124
Migrating variables using a directory with template.xml files as input....134
Migrating components using a directory with template files as input....141
Using migrate-all using a directory with template.xml files as input....150

Cloudera Flow Migration Tool Reference....150
Commands....150
migrate-templates....150
migrate-variables....152
migrate-components....153
migrate-all....155
migrate-registry....155
upgrade-flow....158
export-registry-result....159
cleanup-working-registry....161
help....161
Activity Log....162

Troubleshooting flow migration issues....164

Migrating from Cloudera Flow Management 2.1.7 to 4.13.0

Moving from Cloudera Flow Management 2.1.7, which is based on Apache NiFi 1, to Cloudera Flow Management 4.13.0 which is based on Apache NiFi 2, requires a migration. You cannot perform an in-place or rolling upgrade between these two versions. You must deploy a new Cloudera Flow Management 4.12.0 environment and migrate your existing data flows to it.

The Cloudera Flow Migration Tool supports this transition by applying predefined transformations to make NiFi 1 flows compatible with NiFi 2. It automates many complex and repetitive changes, but some components and configurations might still require manual updates and validation.

The migration documentation explains how to plan and perform the migration using the Cloudera Flow Migration Tool. It also provides best practices, reference information, and guidance for troubleshooting migration issues.

Cloudera Flow Migration Tool release notes

Learn about the new features, known issues, fixed issues, limitations, and unsupported features in the latest release of the Cloudera Flow Migration Tool.

What's new

The Cloudera Flow Migration Tool helps you transition to NiFi 2 more efficiently by simplifying and accelerating the migration of flows from Cloudera Flow Management 2.x (powered by NiFi 1) to Cloudera Flow Management 4.x (powered by NiFi 2).

The latest version available for on premises migration scenarios is **Cloudera Flow Migration Tool 7.0.2**. It **supports migrations from Cloudera Flow Management 2.1.7.4000 (including hotfixes) and 2.1.7.3000 (including hotfixes) to 4.12.0 (including hotfixes)**, in compliance with the Apache NiFi 1.28.1-to-2.6.0 General Availability (GA) ruleset.

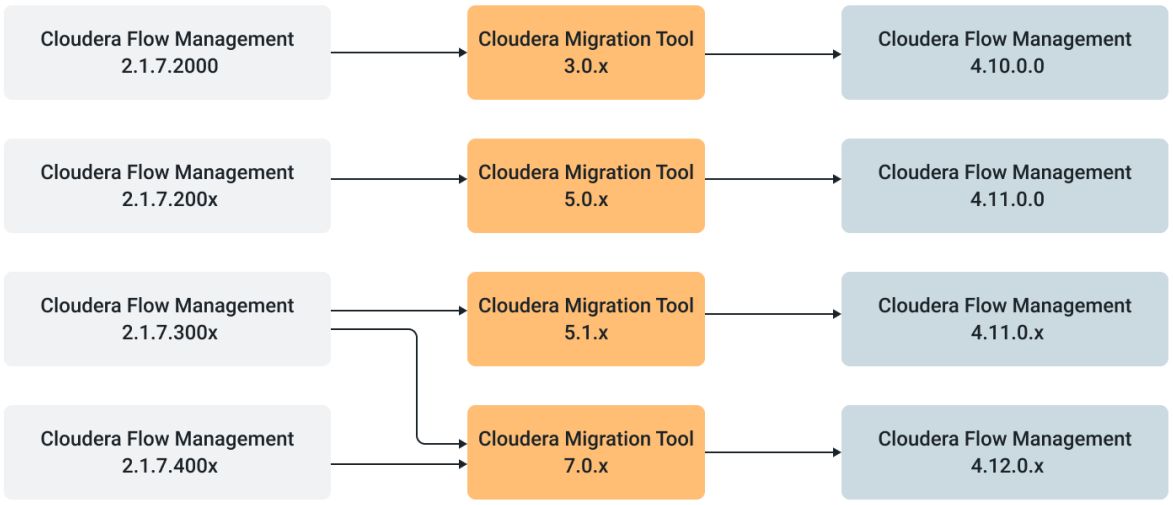
Latest enhancements

- Added support for migrations from the Cloudera Flow Management 2.1.7.4000 Service Pack.
- Fixed an issue where the Migration Tool could report a Circular references in Parameter Contexts not allowed error when migrating process groups that use variables inherited through multiple levels of parent process groups.

Support Matrix

Before starting a migration with the Cloudera Flow Migration Tool, review the supported source and target version requirements to ensure a successful migration.

The following diagram provides an overview of the supported migration paths and the required Migration Tool versions for Cloudera Flow Management on premises.



The table below provides detailed information about the supported migration paths and the latest Migration Tool versions for these paths.

Source Cloudera Flow Management version	Target Cloudera Flow Management version	Latest Cloudera Flow Migration Tool version
2.1.7.2000	4.10.0.0	3.0.3
2.1.7.2000 + 2.1.7.2002 - 2007	4.11.0.0	5.0.6
2.1.7.300x	4.11.0.x	5.1.2
2.1.7.300x + 2.1.7.400x	4.12.0.x	7.0.2

Note:

Cloudera Flow Management 2.1.7.0 and 2.1.7.1000 do not support migration. If you are using one of these versions, first upgrade to a supported source version listed in the table.

Using unsupported version combinations may lead to incorrect migration results or process failures.

Important:

Use only the supported source, target, and Migration Tool version combinations listed in the table, and ensure that all system prerequisites are met before starting the migration.

The migration process does **not** require the source or target Cloudera Flow Management instances to be running. Only the NAR files for the respective NiFi versions are required to complete the process.

System Requirements

Before using the Cloudera Flow Migration Tool, ensure that your system meets the following requirements.

Requirement	Details
Operating system	Linux, macOS
Java version	Java 21
Disk space	At least 25 GB for NiFi distributions and working directories

Download location

You can download the Cloudera Flow Migration Tool software artifacts from the Cloudera Archive.

To access the artifacts, you must have:

- An active Cloudera subscription agreement
- A license key file
- The required authentication credentials (username and password)

Download the version of the tool that matches your requirements. For the exact download URLs, see Cloudera Flow Migration Tool download links.

Cloudera Flow Migration Tool overview

The Cloudera Flow Migration Tool enables the semi-automatic migration of flows from Cloudera Flow Management 2.1.7.4000 powered by NiFi 1 to Cloudera Flow Management 4.12.0 powered by NiFi 2, using predefined transformation rules and logic. It supports a clear, step-by-step approach for migrating your data flows.

When using the Cloudera Flow Migration Tool, you can issue various commands that enable partial or complete migration of the incoming data flow. These commands are customizable through specific arguments to run either individual steps or the full migration process. The Migration Tool stops running after the command is completed and can be run again with the same or different commands, as needed. This modular design allows you to customize the migration workflow according to your needs.

Running Migration Tool commands does not modify the source data flow or the associated NiFi instances. Repeating the same command provides identical results, ensuring predictability. Exceptions are the generation of unique IDs, which may vary across runs, and potential differences in component order within the serialized flow.json file. While file-based comparisons may show variations, the functional outcome remains identical.

Different Migration Tool commands apply specific transformation logic to the input data flow. Each command must be configured appropriately to achieve the desired outcome. For comprehensive details on each command's functionality, as well as the arguments and parameters available for configuration, see the *Cloudera Flow Migration Tool Command Reference* .

The Migration Tool works with flow.json as the input source. From this source, the tool can perform the following actions:

- **Transforming variables to parameter contexts**

  It translates variables into parameter contexts for improved organization and compatibility with NiFi 2.
- **Converting templates**

  It extracts and converts templates into separate flow_definition.json files.
- **Updating components**

  It updates components to align with NiFi 2 requirements wherever possible, potentially applying broader modifications when needed.

The Migration Tool generates various output files, which are saved in the directory specified by the --outputDirectory argument. The results are organized into subdirectories based on the migration stage: sourceVersion for Stage 1 and targetVersion for Stage 2.

In addition to migrating the flow.json, the Migration Tool can also migrate registered flows used by the flow under migration. It does not migrate the entire content of the associated registries, but only the relevant snapshots (flow versions saved to the NiFi Registry) referenced by the flow under migration. Registry migration is not a separate process; it is an extension of flow migration. It is applied only to the required components and follows the same staged approach and predefined migration strategy as flow migration. For more information, see Migrating flows versioned with NiFi Registry [Technical Preview] on page 16.

Related Information

Cloudera Flow Migration Tool Command Reference

Key features

The Cloudera Flow Migration Tool provides several features to support the migration of flows in smaller, manageable chunks, enhancing control and validation.

Reusability

Since the Migration Tool does not modify the input files and produces deterministic results (except for unique identifiers of newly generated components), you can rerun the migration on the same input multiple times. This can be useful if adjustments are needed before or after migration and the results need to be compared.

Activity log

The Activity Log is an important tool for supervising changes in the flow. It lists every modification and provides the reasoning (change-info) behind them.

Stages

The Migration Tool allows you to handle manual migration steps while continuing to use the source NiFi version for most of the migration process. This approach allows you to apply a high number of expected changes without encountering version-related differences, making validation and tracking modifications easier.

Separable commands

Flow migration steps such as template migration can be run separately, allowing you to work on smaller, more manageable parts of the flow.

Process groups

Migration can be scoped to specific process groups by setting the Process Group ID argument. This limits transformations to the specified group and its children, enabling targeted migrations.

Iterative migration (Loopback)

The main output of the Migration Tool, typically a flow.json file, can be used as input for subsequent migrations. This iterative approach helps identify and address additional issues after manual adjustments.

Registry migration

The Migration Tool is capable to support migrating flows which utilise versioned flows from NiFi Registries. By an extended migration strategy, MT helps with migrating the referenced versioned flows, even from multiple source registries and supports creating Registry exports might be simply imported onto the target NiFi Registries. For more information, see Migrating flows versioned with NiFi Registry [Technical Preview] on page 16.

Migration stages

Migration is performed in two sequential stages (referred to as “Stage 1” and “Stage 2”) ensuring a structured transition from source to target compatibility. A complete migration workflow consists of a Stage 1 migration of the incoming flow and a consecutive Stage 2 migration of the result of the Stage 1 migration. The final output of Stage 2 is the completed migration result: a fully migrated and NiFi 2 compatible flow.

Stage 1: Source compatibility

- Processes the incoming NiFi flow (input flow.json) to create a partially updated flow as the output, compatible with NiFi 1.



Note: This intermediate state is meant only for manual adjustments in preparation for Stage 2. Do not use a partially migrated flow for production or business operations.

- Running Stage 1 commands for a template, variable, component, or using the aggregate command for all steps results in transformations compatible with the original version.
- Generates a list of manual changes needed for further adjustments.
- Requires iterative execution after applying manual changes until the flow reaches stability (no further modifications are required).

Stage 2: Target compatibility

- Takes the output of Stage 1 and processes it as input to produce a flow compatible with NiFi 2.
- Applies transformations that go beyond the source NiFi version's capabilities to meet NiFi 2 standards.
- Ensures that the final output is a fully migrated and compliant data flow.

Both migration stages consist of multiple steps that can be executed individually or as a whole. For best results, Cloudera recommends completing all steps within a stage before proceeding to the next one to ensure consistency and avoid potential issues.

Migration steps are present in both stages but involve different transformations and serve different purposes:

- In **Stage 1**, component migration applies changes compatible with the source version.
- In **Stage 2**, component migration applies changes required by the target version.



Important: Running only a Stage 1 migration does not complete the full migration process and may produce incorrect or incomplete results.

NiFi Registry migration follows the same Stage 1 and Stage 2 sequencing for the overall effort, but adds a working NiFi Registry, registry dumps, and commands such as migrate-registry and export-registry-result. For more information, see Migrating flows versioned with NiFi Registry [Technical Preview] on page 16.

Limitations and manual steps

Some aspects of the migration are not handled automatically by the Migration Tool and require manual intervention.

Deprecated components

Certain deprecated component types lack replacements in the target NiFi version or cannot be algorithmically replaced. These components are not migrated automatically when using the Migration Tool and need to be updated manually. In most cases, the Migration Tool provides information on the expected manual changes.

Custom components

Custom components, such as processors not included in the Cloudera Flow Management version in use, are not supported. The definitions of such components, including attributes and bundle versions, are preserved without modification. You must manually update these components.



Note:

As NiFi 1 and NiFi 2 component APIs are incompatible, custom implementations also need to be replaced.

Ghost components in templates

Components saved as ghost components within templates are not migrated during the template migration process.

Variables, dynamic properties, and flow file attributes

Overlapping names among variables, dynamic properties, and flow file attributes are not migrated automatically. You must manually resolve these conflicts and update the elements to ensure compatibility.

Recommended migration workflow

Cloudera recommends dividing the process into manageable phases and validating the results at each step. Below is the suggested migration workflow to ensure a smooth and successful transition.

Stage 1 migration

1. Template migration
   a. Migrate templates for Stage 1.
   b. Validate the results.
   c. Address any issues and manual change requests.
2. Variable migration
   a. Migrate variables to parameter contexts for Stage 1.
   b. Validate the results.
   c. Address any issues and manual change requests.
3. Component migration
   a. Migrate components in smaller batches by specifying Process Group IDs for Stage 1.
   b. Validate the results for each batch.
   c. Address any issues and manual change requests.
   d. Repeat iteratively until no further changes are required.
4. Management-level component migration
   a. Perform component migration without specifying a Process Group ID to ensure management-level components such as Controller Services are migrated.
   b. Validate the changes.
   c. Address any issues and manual change requests.
5. Final validation
   a. When all Stage 1 steps are complete, validate the entire flow in a NiFi 1 instance for final validation.
   b. After this point, the flow will no longer be compatible with the source version.

Stage 2 migration: Target version compatibility

- Run the migration commands without specifying the Stage argument to process both stages in sequence and address any remaining compatibility issues with NiFi 2.

  Note:

  This runs both Stage 1 and Stage 2, but if previous recommendations were followed, Stage 1 has no impact on the flow. While Stage 1 commands should no longer modify the flow, some Stage 2 commands depend on Stage 1 outputs as input (for example for template migration).
- Follow the same steps as in Stage 1: Migrate templates # Migrate variables # Migrate components, while validating the results and addressing manual adjustment needs.
- Stage 2 migration is not iterative in a sense that the output of Stage 2 cannot be reintroduced into the migration process. To ensure a seamless workflow, fully complete Stage 1 before starting Stage 2, and run Stage 2 only once. The final output will reflect all transformations applied by the Migration Tool. Resolving manual change requests will not trigger further migration events. Any modifications beyond this point are considered business logic adjustments and fall outside the tool's scope.

For detailed instructions and examples of the end-to-end migration process, see *Migrating a data flow*.

Related Information

Migrating a data flow

Migration best practices

Migrating complex data flows requires careful planning, validation, and following a structured strategy to ensure success. While the Cloudera Flow Migration Tool automates many aspects of the process, manual oversight and adjustments in the migration process are essential for achieving accurate results. Additionally, some expectations from NiFi 2 cannot be fully addressed through automation alone.

To achieve a successful migration, Cloudera recommends a systematic approach combining the tool's features with manual intervention and careful validation.

Follow an iterative refinement process in Stage 1!

Stage 1 migration automatically converts the flow, applying transformations while maintaining compatibility with NiFi 1, and a list of manual changes is generated. You have to manually edit your flow to fix the requested changes. After making these manual edits, the updated flow should be used as input for the next iteration of Stage 1. Repeat this process on the modified flow iteratively until the flow reaches stability, meaning that no automatic changes are introduced during conversion (there are no change entries in the Activity Log) and there are no manual change requests in the Activity Log that would require attention in Stage 1.



Important:

If a manual change request can be resolved using NiFi 1 features, it is best to address it during Stage 1 migration iterations. However, if the resolution requires a NiFi 2 capability, you may defer it until Stage 2. While migration can proceed with unresolved manual change requests, Cloudera recommends resolving them as early as possible. Only postpone fixes if a NiFi 2 feature is required.

At this point, Stage 1 can be considered fully completed with all issues resolved, allowing you to proceed to Stage 2.



Note:

If you make any manual adjustments to your flow after finishing Stage 1 migration, rerun Stage 1 before proceeding to Stage 2.

Partition the migration process for better control!

Migrating complex data flows requires a more structured approach. Cloudera advises not to migrate the entire flow in a single Stage 1 migration. Instead, partition the migration and focus on one part of the flow at a time to simplify troubleshooting and applying manual changes. You can use the following methods:

- Run the migration in smaller steps, starting with templates, followed by variables, and then components.
- Run the migration by process group.

Both approaches allow you to focus on smaller, more manageable sections of the flow. For larger flows, you can also combine these two methods. The migration logic you choose should be tailored to your specific needs and flow structure. Keep in mind that multiple levels of partitioning may require additional coordination, so it is important to select a level of granularity that provides more benefit than added complexity.

Validate after every iteration!

Load the modified flow into a NiFi instance matching the version of the stage you are on. Confirm functionality and resolve any manual validation requests from the Activity Log.

Always review the Activity Log!

You can use the Activity log to understand the rationale behind changes and identify what manual adjustments are needed in the flow.

Run a full migration after each stage to confirm completeness!

As part of your final validation, run a full migration after each stage. This ensures that no part of the flow was overlooked during partitioning.

- At the end of Stage 1, perform a full migration only using the Stage 1 restriction to verify completeness.
- At the end of Stage 2, run a final full migration without restrictions to ensure the entire flow migration is complete.

Clean up after the migration process!

The Migration Tool includes dependencies on both Cloudera Flow Management 2 and Cloudera Flow Management 4. As its purpose is to support migration from Cloudera Flow Management 2, remove the tool and its associated resources after the migration is complete.

1. Delete the directory where the Migration Tool was extracted, including all contents.
2. Open the migration.tool.properties file and identify the configured paths for nar and the working directories.
3. Verify whether these directories are used by any other services.
4. If the directories are not used by other services, delete them.

You can retain and archive the outputs generated by the Migration Tool, such as migrated flows and activity logs. These files do not contain Cloudera Flow Management 2 or 4 artifacts.

Migrating a data flow using the Cloudera Flow Migration Tool

The migration workflow consists of two main parts: preparation and migration. The *Preparing for migration* section walks you through the preparation steps you should take before starting any migrations, and the *Migrating a flow* section provides examples demonstrating how you can run different types of migrations using different input files.

Preparing for migration

While the Cloudera Flow Migration Tool does not require NiFi to be running, certain NiFi-related steps are needed to prepare for a successful migration.

Setting up your environment

Learn about the necessary setup steps, including installing Java, configuring directories, preparing the source NiFi instance, and acquiring sensitive properties.

Before you begin

- Migrating flows to Cloudera Flow Management 4.12.0 requires upgrading to Cloudera Flow Management 2.1.7.4000 to use as your source version.
- Download the Cloudera Flow Migration Tool from the Cloudera Flow Migration Tool Repository.

Procedure

1. Install and set up Java.
   a) Install Java 21 on the machine where you are performing the migration.
   b) Add Java 21 to the PATH environment variable.
2. Download and unpack the Cloudera Flow Migration Tool on the machine where the migration will run.

   For example: /etc/nifi-migration-tool-bin

3. Configure the following directories to support the Migration Tool and its dependencies.
   a) NiFi 1 and 2 library folders:
      - Provide read-only access to the NiFi 1 lib folder or equivalent directory containing all necessary NAR artifacts.

        For example: /etc/nifi1/lib
      - Provide read-only access to the NiFi 2 lib folder or equivalent directory containing all necessary NAR artifacts.

        For example: /etc/nifi2/lib

   b) Work directories:
      - Create two, initially empty directories for the Migration Tool to unpack NiFi 1 and NiFi 2's dependency NAR files.

        For example: /etc/nifi-1-work and /etc/nifi-2-work
      - Ensure the Migration Tool has write access to these directories that will serve as NiFi 1 and 2 working areas.
      - Avoid using these directories for other purposes.

   c) Output directory:
      - Create an empty directory to store migration artifacts generated by the Migration Tool.

        For example: /etc/migration-tool-output
      - Multiple runs of the Migration Tool using the same output directory will overwrite existing files. For example, if a template migration is run before a component migration, template migration files may remain intact, while other files, such as the Activity Log, are overwritten during the second run.
4. Prepare the source NiFi Instance.
   a) Ensure all components are in a valid state, as invalid components may not be migrated correctly.
   b) Stop and offload all processors.
   c) Stop the NiFi instance and ensure all FlowFiles have been processed.
5. Export the flow.json.gz file from the source NiFi 1 instance to the machine performing the migration.

   For example: /etc/flow.json.gz

   Important:

   Disable all components in the source NiFi instance before exporting the flow.json file for migration purposes.

   a) Extract flow.json from flow.json.gz.
   b) Ensure that the Migration Tool has access to this file.
6. Acquire the Sensitive Properties Key and Algorithm.
   a) Obtain the nifi.sensitive.props.key and nifi.sensitive.props.algorithm from the source NiFi 1 instance.

      You will need these when updating the migration.tool.properties configuration file.

      Note:

      You will need to use the same key and algorithm later to load the migrated flow into the NiFi 2 instance.

   b) For deployments in Cloudera on premises or Cloudera on cloud, follow these steps to acquire the sensitive key:

      1. Configure Cloudera Manager to allow API calls to return sensitive values. For more information, see *Disabling Redaction of sensitive information when using the Cloudera Manager API*.

2. Run the following API call:

```
https://[***HOST***].root.comops.site:7183/api/v54/clusters/[***CLUS
TER_NAME***]/services/[***SERVICE_NAME***]?view=EXPORT
```

   Replace [***HOST***], [***CLUSTER_NAME***], and [***SERVICE_NAME***] with the appropriate values for your cluster.
3. Locate the random.nifi.sensitive.props.key value, which serves as the nifi.sensitive.props.key property.
4. Revert Cloudera Manager to its original state to disable sensitive value return.

7. If the source and target instances use different sensitive properties, update the nifi.sensitive.props.key and nifi.sensitive.props.algorithm properties using a copy of the original nifi.properties file.

   For more information, see the *NiFi System Administrator's Guide*.

Related Information

Disabling redaction of sensitive information when using the Cloudera Manager API

NiFi System Administrator's Guide

Configuring the Cloudera Flow Migration Tool

Learn how to set up and customize the Migration Tool properties file for your migration environment.

Procedure

1. Locate migration.tool.properties configuration file in the conf directory of the Migration Tool root.
2. Configure the directory paths by updating the following mandatory properties in the configuration file.

   embedded.nifi.v1.nar.directory

   - Specify the read-only directory containing the NAR files associated with the source NiFi version.
   - It can be a lib folder of an actual NiFi instance with a matching version or a different directory containing all NAR artifacts from a given Cloudera distribution. For example: /etc/nifi1/lib
   - Although the Migration Tool does not need a running NiFi instance and does not modify the contents of this directory, it requires read access to these files.

   embedded.nifi.v1.working.directory

   - Designate an empty workspace for the Migration Tool when it is working with the source NiFi version.
   - It should be initially empty and the Migration Tool must have write access to it, allowing the Migration Tool to perform all necessary operations.
   - It must be separate from the source NiFi instance's working directory and should not be used for any other purposes to avoid potential conflicts.
   - After the migration is complete, it can be cleaned up if needed.

   embedded.nifi.v2.nar.directory

   - Specify the read-only directory containing the NAR files associated with the target NiFi version.
   - It can be a lib folder of an actual NiFi instance with a matching version or a different directory containing the required NAR artifacts.
   - Although the Migration Tool does not need a running NiFi instance and does not modify the contents of this directory, it requires read access to these files.

   embedded.nifi.v2.working.directory

   - Designate an empty workspace for the Migration Tool when it is working with the target NiFi version.
   - It should be initially empty and the Migration Tool must have write access to it, allowing the Migration Tool to perform all necessary operations.

- It must be separate from the target NiFi instance's working directory and should not be used for any other purposes to avoid potential conflicts.
- After the migration is complete, it can be cleaned up if needed.

3. To configure the sensitive properties, update the following mandatory properties in the configuration file.

   embedded.nifi.sensitive.props.key

   - Provide the nifi.sensitive.props.key from the source NiFi instance for handling sensitive properties.
   - This key is used to decrypt sensitive values during migration and re-encrypt them for the target NiFi instance.
   - During migration, the target NiFi must use the same nifi.sensitive.props.key as the source, as configured in the Migration Tool properties.
   - If the target NiFi requires a different sensitive properties key than the source NiFi, update the value only after completing the migration.

   embedded.nifi.sensitive.props.algorithm

   - Provide the nifi.sensitive.props.algorithm from the source NiFi instance required for handling sensitive properties.
   - This algorithm is used to decrypt sensitive values during migration and re-encrypt them for the target NiFi instance.
   - During migration, the target NiFi must use the same nifi.sensitive.props.algorithm as the source, as configured in the Migration Tool properties.
   - If the target NiFi requires a different sensitive properties algorithm than the source NiFi, update the value only after completing the migration.

4. When you use NiFi Registry migration, configure access to the Working Registry.

   The Migration Tool requires access to a Working Registry, which serves as storage for migrated snapshots. This registry must be installed and running separately.

   To enable this, configure the following properties in the migration.tool.properties file:

   registry.working.url

   Specify the location of the Working Registry. This is equivalent to configuring a NiFi Registry Client.

   If the Working Registry is secured (which is the recommended approach), authentication information must be provided, similar to configuring a StandardSSLContextService for a NiFi Registry Client. The required properties correspond to those of the StandardSSLContextService with matching names:

   ```
   registry.working.ssl.keystore.file
   registry.working.ssl.keystore.password
   registry.working.ssl.keystore.type
   registry.working.ssl.truststore.file
   registry.working.ssl.truststore.password
   registry.working.ssl.truststore.type
   registry.working.ssl.key.password
   registry.working.ssl.protocol
   ```

   The only exception is registry.working.timeout.

   registry.working.timeout

   Specify the timeout (in milliseconds) for communication with the Working Registry, including both connection and read operations.

   The default value is 300. If connection issues occur between the Migration Tool and the Working Registry, this value may need to be increased.

For the end-to-end workflow, see Migrating flows versioned with NiFi Registry [Technical Preview] on page 16.

5. Save the modifications of the migration.tool.properties file.

Migrating flows versioned with NiFi Registry [Technical Preview]

You can extend the Cloudera Flow Migration Tool workflow to migrate versioned flows and their NiFi Registry snapshots from NiFi 1 (Cloudera Flow Management 2.x) to NiFi 2 (Cloudera Flow Management 4.x). This process uses a dedicated Working Registry, registry dumps, and additional migration commands.

Registry migration is not a separate process; it builds on the same staged approach as flow migration. The Migration Tool migrates only the snapshots (saved flow versions in NiFi Registry) that are referenced, either directly or as nested registered flows, by the input flow.json or flow.json.gz, rather than entire registry repositories. Key commands used in this process include: migrate-registry, upgrade-flow, export-registry-result, and cleanup-working-registry.

Strategy overview

When migrating without handling Registry, the suggested migration strategy is the same as before. If the migrated flow contains versioned Process Groups, those will be migrated as regular Process Groups. If however, one wants to maintain consistency and handle versioned flows as distinct units, the following approach is suggested.

This approach has multiple merits:

- Helps partitioning the work and share load on different owners
- Multiple checkouts of the same registered flows are registered one time, without introducing inconsistencies and additional work
- Keeps the references consistent
- Resulting a dump which can be load into a NiFi Registry instance with minimal manual work

In order to apply this strategy and benefit from the merits, a different strategy is needed with the usage of both new and existing commands. As an end result you can expect both the flow to be migrated and a set of migrated snapshots ready to be imported into the target registries.

Prerequisites

Properties

Configure registry-related properties in migration.tool.properties as described in Configuring the Cloudera Flow Migration Tool.

Working registry

Stage 1 of registry migration works in an iterative manner not unlike flow migration. For flow migration, the result of a previous execution can be feed back to the Migration Tool after manual changes for further refinement. In case of registry migration, the result of the migration can be used as input for the next execution as well but instead of files, the Migration Tool commits the result of the execution into a NiFi Registry, called working registry. The reasons for this:

- A registry migration might result numerous migrated snapshots, making it hard to coordinate with the owners of the given snapshots
- Due to nested snapshots, migrating a registered flow can have affect on multiple others. Without a central place, these connections would be broken

As of these, the Migration Tool uses the concept of "Working Registry" to store the "intermediate results", for the timeframe of the migration effort. The working Registry must be a NiFi 1 (Cloudera Flow Management 2) based NiFi Registry instance running in a place reachable by the Migration Tool. This instance can be a standalone, completely Cloudera Flow Management independent Registry depending on the company policies. While the bucket names used by Migration Tool are following a specific format, it is suggested to use an instance dedicated to this effort. Also

for logistical reasons it is beneficial to make this instance accessible for teams working on the migration. For further details, see *Strategy*.

Note: The NiFi Registry instance is not part of Migration Tool, will not be installed or administered by the tool in any means.

Registry dumps

The Migration Tool does not connect into running services (except the working registry) and will not collect the registered snapshots directly from the source registry. Instead, "registry dumps" serves as input for the registry migration. For more information, see the Apache NiFi Toolkit Guide — export-all-flows.

It is important to use the version of the NiFi Toolkit associated with the source Cloudera Flow Management version.

The Migration Tool will need the dump of all Registries defined in and used by the input flow. The content of every dumps should be within a separate folder under a shared parent. This parent directory will be set as a value for the relevant property (see below). It is important to mention that the folders containing the dumps must be named as the relevant Registry Client in the NiFi flow. As usual, these input files are not modified by the Migration Tool.

Used flow

The Migration Tool does not migrate all snapshots within the repositories. As most snapshots are not in use and the possible and necessary manual changes would increase the amount of necessary work significantly, only the snapshots will be handled are directly or indirectly (as nested registered flow) referenced by the provided flow (flow.json or flow.json.gz). This incoming flow is also updated during the migration process in order to reflect the changes in the registry snapshots.

External systems

Aside from the working registry, which should be dedicated for the migration process, the Migration Tool still does not connect to external services, included but not limited to NiFi instances or the source Registry of the used dumps. In short: the Migration Tool will not reach out neither the "source" registries and NiFi or the "target" registries and NiFi.

Preparation

Working Registry

In order to have a successful migration, some preparation is needed. Other than exporting the flow.json as previously, further steps are necessary.

First in order, it is necessary for the registry migration to have a storage for migrated flows. Just like with flow migration, the idea is to feed back the results of a previous migration as an input for incremental migration. But due to the number of possible files and to ease the administrational overhead of coordinating manual changes, Migration Tool uses a dedicated NiFi Registry as an intermediate storage: in most cases, Migration Tool will read the subject of the migration from this Registry and write the results back. This Registry is called a Working Registry. The Working Registry is not part of the Migration Tool, but needs to be maintained by the user. Migration Tool merely connects to it using a set of properties defined in the Properties section.

All the intermittent results will be stored within the Working Registry until the last step of the migration. This involves all migrated registries. Also it is important to mention that the Working Registry is also used for resolving manual changes as it is described later in the strategy. Due to this, the instance must be reachable by the personnel responsible to overview the migration and do the manual work with the snapshots.

Registry Dumps

Just like in case of flow migration, the flow.json (or flow.json.gz) must be exported. In the same manner, the content of the Registries the flow connects to must be dumped. This can be done by

using the NiFi Toolkit. The NiFi Toolkit supports the following command can be used for dumping the content of a NiFi Registry:

```
./bin/cli.sh registry export-all-flows -u https://internal-domai
n:18080 --outputDirectory "/my_dir/flow_exports"
```

For more information about this command, see the Apache NiFi Toolkit Guide. Migration Tool expects to have access to the content of all Registries the flow is pointing to using Registry Client. The way to do this is having all dumps within their own directories under a common parent directory. This directory is called "dump directory". Later on, when executing migration steps, this directory will be used as value for command argument. In order to determine the connection between dumps and Registry Clients in the flow, the directory name of the individual dump folders (not the parent) must be the same as the name of the relevant Registry Client.

Migration Tool supports working with NiFi Registry only. While it does not directly connect to the source registry, other implementations might represent details of the registered flow in a different way, not covered by the feature set of the Migration Tool. In case by any reason, multiple Registry Clients are pointing to the same NiFi Registry, all clients have to have the separate dump folder under a dedicated name.



Note: While running the NiFi Toolkit, the tool must have access to the NiFi Registry. This both includes network access and credentials if necessary. All flow information exported from the NiFi Registry will be stored in plain text files.



Note: It is important to use the version of NiFi Toolkit associated with the source NiFi. Otherwise, it is not guaranteed that the dumped snapshots are in the proper format, containing the proper information

Example:

You have a flow.json, containing two Registry Clients: R1 and R2. R1 points to "https://internal-domain:18080" and R2 points to "https://other-internal-domain:18082". These are the only Registry Clients declared in the flow.json. In order to gather the necessary dumps, the following commands needs to be executed (users also keep care of relevant credentials as written in the Toolkit documentation):

```
./bin/cli.sh registry export-all-flows -u https://internal-domai
n:18080 --outputDirectory "/my_dir/flow_exports/R1"

./bin/cli.sh registry export-all-flows -u https://other-internal-
domain:18082 --outputDirectory "/my_dir/flow_exports/R2"
```

In this example, the value of the argument "Export Directory" (see relevant command) will have the value of /my_dir/flow_exports.

Further preparation

Further steps of preparing for the migration are the same as in case of normal flow migration. It is important to highlight however that the flow.json and the collection of dumps serve as "input state" for the migration together. This state will represent the state of the system (NiFi cluster and NiFi Registry all together) from the time of dumping. Further changes in the original will not have an effect. (This is still an offline effort, just like classical migration)



Note: Local changes of the registered flows in the flow.json will be abandoned. During migration, at various points, the content of the versioned process groups in the flow.json are replaced with migrated Registry snapshots.

Bucket and flow names

Bucket and flow names stored in the Working Registry follow specific conventions to ensure they are identifiable. In most cases, the default values for maintaining these conventions work properly, but there may be situations where they clash with the naming of flows and buckets under migration

(see below), making it difficult for the tool to resolve names correctly. For these unlikely cases, a set of properties is provided to override the default settings.

It is important to note that once a migration process has started (that is, after the start step of Migrate Registry Snapshots has been executed), naming has already been applied, and any changes may result in unpredictable outcomes. If naming conflicts are discovered after this point, the Working Registry must be cleaned up and the migration must be restarted.



Note:

The restart step is not sufficient in these cases, as both the cleanup and the migration rely on the values of these settings.

Property Name	Description	Default Value
registry.working.migration.infix	The infix used for under migration buckets which separates the registry name and the original bucket name	under-migration
registry.working.naming.separator	The separator string between different tokens of a concatenated bucket or flow name, like the under migration bucket names within the Working Registry	_
registry.working.flow.multiversion.postfix	The first part of the postfix after the flow name when multiple lines are maintained. The second part of the postfix (after the separator) is the order.	line

Bucket naming strategy

Every bucket created in the Working Registry by the Migration Tool follows this naming convention:

```
{$originalRegistryName}{$separator}{$migrationInfix}{$separator}
{$originalBucketName}
```

Here, $originalRegistryName is the same name used as the directory name for the registry export, and $originalBucketName is the original name of the bucket in the source registry. The $separator and $migrationInfix are defined by the properties listed above.

With the default settings, the naming pattern is:

```
{$originalRegistryName}_under-migration_{$originalBucketName}
```

This can result in bucket names such as R1_under-migration_B1.



Note:

This naming convention allows multiple source registries with identical bucket names to be migrated together.

Flow naming strategy

In most cases, the name of a flow in the Working Registry is identical to the original flow name in the source repository. The exception occurs when the flow.json uses multiple versions of the same flow, which must be migrated separately. In these cases, the original flow is represented by multiple flows in the Working Registry, which must be distinguished from each other.

To differentiate them, a postfix is added using the following format:

```
{$originalFlowName}{$separator}{$multiversionPostfix}{$separator}
{$lineOrder}
```

Here, $separator and $multiversionPostfix are defined by the properties above. $originalFlowName is the flow name from the source repository, and $lineOrder is a numeric value assigned by the Migration Tool to represent the relative order of the flows (for example, flows with higher original version numbers have higher line order values).

With default settings and an original flow named Flow3, the resulting names in the Working Registry would be Flow3_line_1, Flow3_line_5.

Target registries

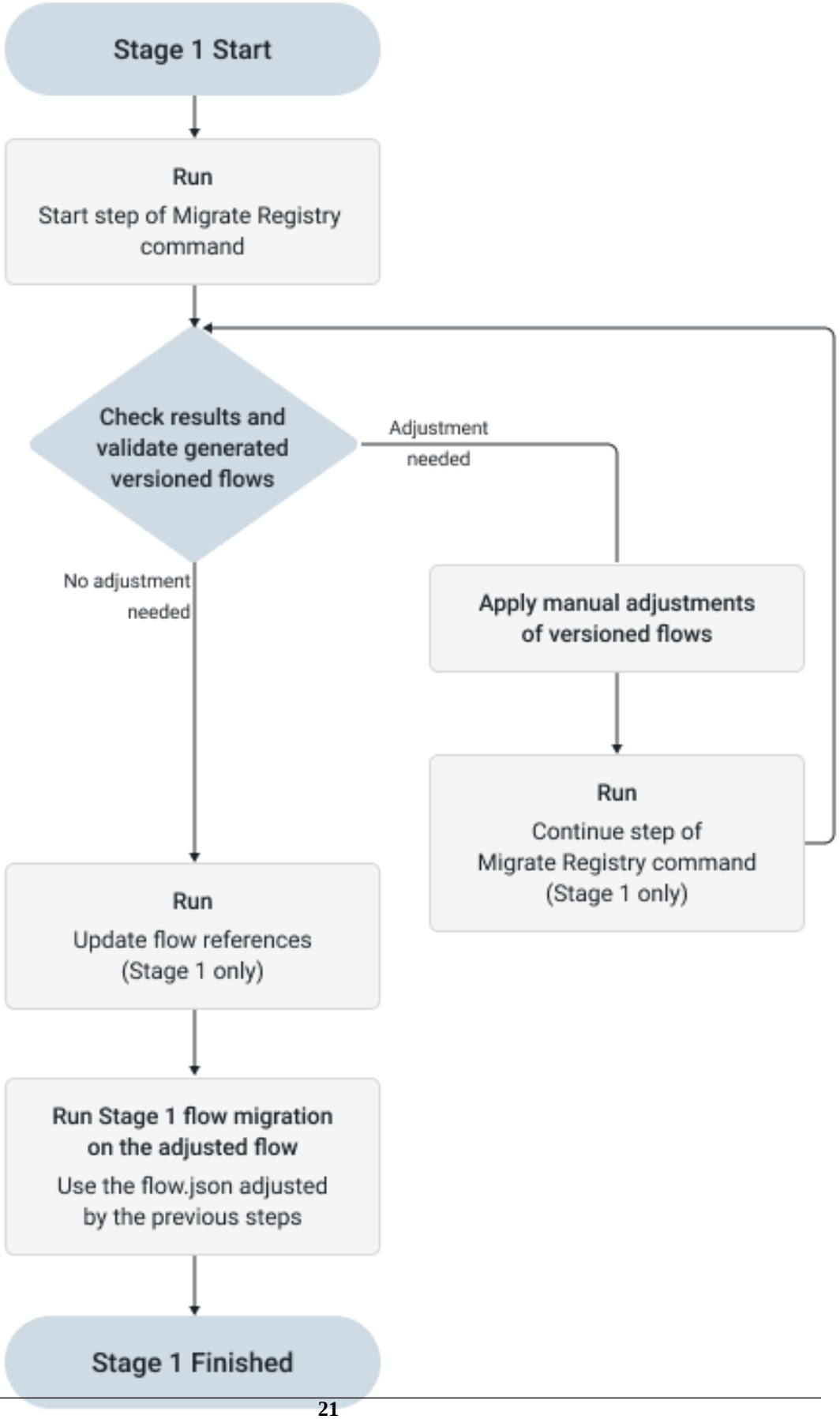
The final result of the registry migration process consists of a fully migrated flow.json and a set of registry exports. During migration, both the references in the flow.json and the references within nested flows point to the Working Registry. This ensures that the snapshots remain loadable and consistent throughout the migration process.

In the final step of the migration, these references are updated to point to the intended target registries. For more information about the required properties, see the preparation steps for Stage 2.

- registry.final.location.{$identifier}.name
- registry.final.location.{$identifier}.url

Stage 1

The following diagram illustrates the recommended workflow for the stage 1 migration of versioned flows with NiFi Registry.


Stage 1 Start
Run
Start step of Migrate Registry command
Check results and validate generated versioned flows
Adjustment needed
No adjustment needed
Apply manual adjustments of versioned flows
Run
Continue step of Migrate Registry command (Stage 1 only)
Run
Update flow references (Stage 1 only)
Run Stage 1 flow migration on the adjusted flow
Use the flow.json adjusted by the previous steps
Stage 1 Finished

Start step

Just like classical flow migration, Stage 1 represents the set of flow changes that keep the flow NiFi 1 compatible. However, unlike flow migration, multiple executions are required to progress through this stage. These executions use the same command with different arguments. Registry migration precedes flow migration, so when the flow is migrated, versioned process groups are substituted with the already migrated ones. The flow.json modified during registry migration should be used as input for subsequent flow migration.

To start registry migration, run the Registry Migration command with the start step. This command performs the initial migration and serves a role similar to the first execution of Component Migration from a strategic perspective. It requires both the flow.json (or .gz) file and the necessary registry dumps. In addition, the Working Registry must be set up, accessible, and empty from a migration perspective. In this context, “empty” means that it does not contain results from an ongoing registry migration.

This state is determined by the Migration Tool based on the existing Registry buckets. At the beginning of execution, the Migration Tool checks for bucket names that match the naming scheme it uses for storing migration results. The format is:

```
{$nameOfTheRegistry}_under-migration_{$bucketName}
```

Here, $nameOfTheRegistry corresponds to the Registry Client name (and the directory name of the dump files for that registry), and $bucketName matches the bucket name in the original registry. This naming convention allows content from multiple source registries to be handled within a single Working Registry. While it is technically possible to store unrelated buckets in the Working Registry, this is not recommended. If the Migration Tool detects buckets that match this naming pattern, it prevents execution of the start step.

When the command is executed with the start step, several actions occur. First, snapshots in the dump(s) are filtered based on usage: only snapshots referenced by the flow.json (either directly or as nested registered flows) are processed. Other snapshots (entire flows or unused versions) are skipped. The selected flows are then subject to component and variable migration, and the results are stored in the Working Registry. The Migration Tool maintains references and updates version information both in the flow.json and in the registered snapshots. This ensures that NiFi can fully load the flow or any registered flow checked out from the Working Registry, including nested snapshots.

Bucket names follow the schema described above. In most cases, flow names remain identical to the original. It is important to note that identifiers are not preserved; from that perspective, these are entirely new (recreated) buckets and flows. The Migration Tool handles these changes within the scope of the flow, but if any external systems rely on these identifiers, they must be updated manually.

Flow names change only when multiple versions of the same registered flow are used (either directly or as nested flows). In such cases, the flow name is postfixed as follows:

```
{$originalFlowName}_line_{$lineOrder}
```

This means that when multiple versions are used, the Working Registry will contain multiple versions of the original registered flow as separate flows (for example, myFlow_line_1, myFlow_line_2). While this is supported by the Migration Tool, it introduces a risk of divergence because these versions are handled separately during migration. As a best practice, align all usages to the latest version before migration whenever possible. These flows function as separate flows during migration and are migrated independently. The lineOrder does not represent the original version number; it is a sequential number assigned by the Migration Tool. The only guarantee is that a higher line order corresponds to a higher original version. These lines are merged into a single flow at the end of the migration.

From this point onward, registry migration becomes an ongoing process that primarily reads from and writes to the Working Registry. Other input files, such as flow.json and the registry dumps, are required at later stages, but most operations rely on iterating over the Working Registry content. Because of this, the start step cannot be executed again once migration has begun, as the Working Registry will already contain buckets that match the migration naming scheme. Such attempts will fail without effect. If the migration must be restarted, use the restart step (with the same migration.tool.properties). Be aware that this command removes all content from the Working Registry under buckets whose names match the “under migration” naming pattern.

It is generally considered best practice to use the Working Registry as a dedicated environment and avoid storing unrelated flows in it. In rare cases where this is not possible, or when bucket or flow names conflict with the migration naming scheme or multi-line flow naming, you can override the default infixes and postfixes using properties defined in the Properties section.

Changing these properties during an ongoing migration can result in unpredictable behavior, as the Migration Tool relies on these naming conventions at multiple stages of the process.

Example:

As part of the preparation for migration, export the flow.json file and create dumps from all NiFi Registries used by the flow. These dumps contain files generated by the NiFi Toolkit, for example:

```
/my_dir/flow_exports/R1/toolkit_registry_export_all_B1_Flow1_1
/my_dir/flow_exports/R1/toolkit_registry_export_all_B1_Flow1_2
/my_dir/flow_exports/R1/toolkit_registry_export_all_B1_Flow1_3
/my_dir/flow_exports/R1/toolkit_registry_export_all_B1_Flow2_1
/my_dir/flow_exports/R1/toolkit_registry_export_all_B2_Flow3_1
/my_dir/flow_exports/R2/toolkit_registry_export_all_B1_Flow4_1
/my_dir/flow_exports/R2/toolkit_registry_export_all_B1_Flow4_2
```

These files represent the contents of two registries, R1 and R2, each in its own directory. In this example, four flows are present:

- 3 versions of Flow1 in bucket B1 of registry R1
- 1 version of Flow2 in bucket B1 of registry R1
- 1 version of Flow3 in bucket B2 of registry R1
- 2 versions of Flow4 in bucket B1 of registry R2

  (this B1 bucket, although it has the same name as in R1, is a different bucket)

The accompanying flow.json uses only the following snapshots:

- Versions 2 and 3 of Flow1
- Version 1 of Flow3
- Version 2 of Flow4

Other versions are not used, either directly in the flow or indirectly through nested references.

With this understanding of the input, the following command starts the migration process:

```
./migration.sh nifi migrate-registry -s -i /my_dir/flow.json -ed /my_dir/flo
w_exports -od /my_dir/out -sco
```

In many ways, this command is similar to migrate-components: it uses -i to define the input (the flow), -od to specify the output directory, and -sco to limit execution to Stage 1. In addition, -ed specifies the directory containing the registry dumps, and the -s argument indicates that the command should run in start mode.

A successful run produces several types of output:

- **flow.json**

  The flow is not yet migrated at this stage, but all registered flow references and their contents are replaced with the newly migrated snapshots from the Working Registry. The properties of all NiFi Registry Clients are also updated to point to the Working Registry. This ensures that when the flow is loaded into a NiFi instance, the version status indicator reflects the actual state of the registered flows. The output flow.json overwrites any existing file in the output directory.
- **Activity logs**

  The command generates one general activity log and one log for each migrated registry snapshot. These logs follow the standard activity log format but are named to distinguish them. Snapshot-specific logs use the following naming convention: {$workingRegistryBucketName}_{$workingRegistryFlowName}-activity-log.json

Activity logs overwrite results from previous executions.

- **Working Registry content**

  All snapshots used by the flow are migrated and written to the Working Registry according to the naming conventions described earlier.

  As a result, the Working Registry contains the following entities:

  - Bucket: R1_under-migration_B1
    - Flow: Flow1_line1, version 1
    - Flow: Flow1_line2, version 1
  - Bucket: R1_under-migration_B2
    - Flow: Flow3, version 1
  - Bucket: R2_under-migration_B1
    - Flow: Flow4, version 1

  Note: For a detailed description of how bucket and flow names are generated, see *Properties*.

  As shown, only actively used snapshots are migrated. This approach avoids unnecessary work on unused flows and reduces the risk of divergence. At a later stage, it is possible to preserve historical versions for archival purposes. These historical snapshots remain unchanged and follow NiFi 1 conventions. While NiFi Registry supports this, NiFi 2 cannot safely use these historical versions.

  It is also important to note that the Working Registry does not preserve the original version numbers. Instead, versioning starts from 1 for migrated flows. These internal version numbers are used only during migration and are discarded at the end of the process.

  Although most flows have a single version after the initial start step, this is not always the case. When flows contain nested references, multiple versions can appear immediately after the initial migration. This occurs because the tool updates references between registered flows.

  For example, if Flow3 contains Flow1_line2 as a nested flow, the Working Registry would look like this:

  - Bucket: R1_under-migration_B1
    - Flow: Flow1_line1, version 1
    - Flow: Flow1_line2, version 1
  - Bucket: R1_under-migration_B2
    - Flow: Flow3, versions 1 and 2
  - Bucket: R2_under-migration_B1
    - Flow: Flow4, version 1

  In this case, Flow3 has an additional version. Accumulating versions during migration is expected and part of the normal process. For migration purposes, the latest version is always used for further processing.

  Note: The Migration Tool supports nested flows across different registries, but only for flows that are included in the provided registry dumps.

Restart step

In some cases, it is satisfactory to restart the migration process from the beginning. For these cases, to simplify the cleanup, a restart step can be run. This is identical to the start step in every regard, except that instead of checking the buckets of the Working Registry, the 'restart' step deletes buckets under migration before executing the migration.

```
./migration.sh nifi migrate-registry -res -i /my_dir/flow.json -ed /my_dir/f
low_exports -od /my_dir/out -sco
```

Note: While this step is included in the migration strategy, it is not part of the normal migration process in most cases.

Manual resolution

Manual resolution of situations cannot be migrated automatically is part of the registry migration too. Contrary to the flow migration, it comes in the form of validating and resolving a high number of independent flows. In order to support this circumstance and support possibly multiple owner teams to work on their related flows, the Registry migration utilises the Working Registry. For every managed flow, the expected way to add manual changes to the registered flows under migration, it is expected to be checked out to a NiFi instance and when the work is finished, to be committed into the Working Registry as a new version of the flow. The Migration Tool will always use the last version from every flow. Because of this, there is no limitation on the number of versions related to manual changes and there is no connection between the versioning of the individual registered flow. While this is the first point in the migration strategy where manual resolution might be necessary, this can happen after any command execution. As registered flows are independent (except referencing to each other), it is a likely situation that some registered flows need manual adjustment after a migration command and others do not.

Example

In this example the initial migration of Flow3 signed a necessary manual change. In order to resolve this the owner of the flow can load the last version of the flow from the Working Registry into the canvas of the NiFi instance dedicated for validating and managing the migration effort (presumably the development environment), apply the changes and then save the flow back into the Working Registry. Later on, during validation of the registered flow an other change has been applied to the flow, resulting in a further snapshot in the Working Registry. As a result, at this point, the following versions of Flow3 exist in the Working Registry:

1. Version 1: Created by the initial migration at the 'start' step
2. Version 2: Also created by the 'start' step in order to update the reference
3. Version 3: The resolution of the manual change
4. Version 4: Follow up changes by owner during validation

The Migration Tool will always use the last version to work with, thus in this example, any further migration will happen in the version 4 snapshot, resulting in a version 5, and so on.

Note: When a new snapshot version is created as part of a migration command, the references in other registered flows managed by the Working Registry will be updated to this newer version. With manual changes this update will not happen. (The next registry-migration execution will execute this update regardless of the number of versions were created between the two runs).

Continue step

After the initial migration of the snapshots happens and the working items are stored in the Working Registry, further refinements might be needed in an iterative manner, as in the case of flow migration. Just like in that case, the result of the previous migration or manual adjustment should be fed back to the Migration Tool for further refinement. Again, just like in the case of flow migration, this should continue until no changes or manual change requests appear in the output. An important change compared to flow migration is that, in the case of Registry migration, it is not necessary to specify the input using command arguments. In the case of Registry migration, after the initial 'start' step, the Migration Tool always works from and to the Working Registry specified by the properties file.

Example

Continuing with the example from before, the following command can be executed in order to apply further migration to the whole content of the Working Registry:

```
./migration.sh nifi migrate-registry -c -od /my_dir/out -sco
```

As seen, this command is similar to the 'start' step but lacks some of the arguments: because the 'continue' step works based on the content of the Working Registry, no input specification is needed. Likewise, the content of the Registry dumps is not necessary at this step. You still need to define the output directory, however, as while the main result of the migration (migrated snapshots) is committed to the Working Registry, the activity logs describing the transformations will be created within the specified directory.

While the 'start' step needs to migrate all registered flows within all buckets, the 'continue' step does not require working with the whole set of registered flows at all times: different flow-owner teams might work at different paces and schedules, so, to support this, it is possible to specify a subset of flows to be migrated. This specification can happen at multiple levels:

- You can specify a given Registry using the registry argument. Only flows originally from the specified Registry will be included in the migration.
- After you specify a Registry, it is possible to specify a Bucket using the registryBucket argument. Only flows originally coming from the specified Registry and Bucket will be included in the migration.
- After you specify a Bucket, it is possible to specify a Flow using the registryFlow argument. Only the flow coming from the specified Registry and Bucket, named as the specified flow, will be migrated.

The previous example executed a migration on all flows managed by the Working Registry. With the addition of some of the arguments above, we can narrow down the scope to Flow1 and Flow2:

```
./migration.sh nifi migrate-registry -c -od /my_dir/out --registry R1 --regi
stryBucket B1 -sco
```

In case we want to limit the scope even further, we can use the third relevant argument, registryFlow. The following example will migrate Flow2 only:

```
./migration.sh nifi migrate-registry -c -od /my_dir/out --registry R1 --regi
stryBucket B1 --registryFlow Flow2 -sco
```



Note: While the registry and registryBucket arguments work based on the original naming from the Registry dumps, the registryFlow argument works with the actual name of the flow, which means that if the original flow was split into multiple lines, the name with the line order needs to be specified here.



Note: If the specification is too strict, it is possible that no registered flow will be migrated.

When the scope is limited, no registered flow outside the specified boundaries is updated. In practice, this means that no new snapshot will be created for those flows, including snapshots for updating reference versions. This will not directly affect later migrations, as nested flows do not have a direct effect on the migration of the actual subject, but this requires attention during manual steps, where, if not set by the user, the referenced flows might not be up to date.

Example

Flow3 has a nested flow that references Flow1. We use the following arguments: registry R1 and registryBucket B1. This will result in a migration where Flow1 (the nested flow in this scenario) will be part of the migration, thus a new version will be created, while Flow3 will be excluded. As a result of this migration, Flow3 will point to a now obsolete version of Flow1. This is deliberate and supports scenarios where work is split between different teams and registered flows are migrated by different people at their own pace. While there are dependencies between flows, a change in a dependency does not automatically trigger a change in the dependent flow, as it might have been modified but not yet committed. The NiFi instance used for this purpose will indicate that a new version is available, and the owners of Flow3 can manually upgrade the reference as it fits their workflow. Alternatively, Upgrade Flow References can be used.

After execution, the usual manual step is expected: validating the result and executing the required manual changes are user responsibilities. To support this, as in the case of the 'start' step, there is a separate activity log for every migrated registered flow, making it easy for the owner of the given registered flow to validate the changes. When activity logs for all registered flows managed by the Working Registry show that there are no changes and no expected manual changes, the Stage 1 migration of the registered flows is considered complete, and the migration process can continue with the next step.

Upgrade flow references

Optionally, before proceeding with the migration of the flow.jso (or at any appropriate time), you can update the references in the flow.json. By executing the upgrade-flow command, the Migration Tool replaces all registered flows with the latest snapshots available in the Working Registry.

This process does not update the references between registered flows. As a result, if those references are outdated, the flow will continue to reflect that state.

To perform the flow upgrade, run the following command:

```
./migration.sh nifi upgrade-flow -i /my_dir/flow.json -od /my_dir/out/out -s
co
```

Migration of the flow.json



Note:

As previous steps applied changes to the flow.json, the updated file should be used here. As in the original migration strategy, the flow generated by the last run of the Migration Tool should always be used.

After the Stage 1 migration of the registry snapshots, normal flow migration can be executed. The majority of the flow migration follows the existing strategy: an iterative approach where the results of the previous execution are fed back into the tool, with manual corrections applied between iterations. The crucial difference between the two approaches is that when versioned process groups are migrated as part of the registry, the Migration Tool should skip these process groups during flow migration. To achieve this, flow migration commands must be executed using the --registryAware argument in addition to the usual arguments, as shown in the following example:

```
./migration.sh nifi migrate-components -i /my_dir/flow.json -od /my_dir/out/
out -sco --registryAware
```

When the Migration Tool detects this argument, it skips all versioned process groups. It assumes that these process groups have already been migrated during previous steps. Since the registered flows are already migrated (Stage 1 only), this allows the personnel responsible for flow migration to better understand how earlier changes affect the flow and what additional adjustments are required to keep the flow aligned with the registry changes.

In most cases, registry-aware mode is essential to avoid introducing changes into already migrated and validated registered flows, which may have been checked out multiple times, potentially leading to inconsistencies. In addition, registry-aware mode has further implications:

External controller services might be replaced or modified during flow migration, but these changes are not reflected in the checked-out registered flows or in the snapshots stored in the Working Registry. In practice, this means that an external controller service might no longer exist or might exist in a different form, resulting in broken references within the registered flows. These issues must be resolved manually, depending on the nature of the changes and business requirements. The Migration Tool cannot resolve this automatically, but to help identify such cases, it generates activity log entries when a flow is migrated in registry-aware mode and skipped registered flows still reference external services. These appear as activity log events such as:

```
{
 "sequence" : 4,
 "type" : "manual-validation-request",
 "subject" : "7be1831f-019d-1000-02ab-09481049764a",
 "message" : "Please check if the external service references are still w
orking after the migration",
 "context" : {
```

```
    "externalServices" : "5d5ec272-5d91-382e-96b2-9491c1809213"
  }
}
```

Furthermore, a registered flow has visibility into the variables of its containing process groups. It is assumed that by the time the flow is migrated, the registered flows have already been migrated and stored in the Working Registry. Since registered flows in the Working Registry have already undergone component and variable migration, the variables within those flows are converted to parameters according to the rules of variable migration.

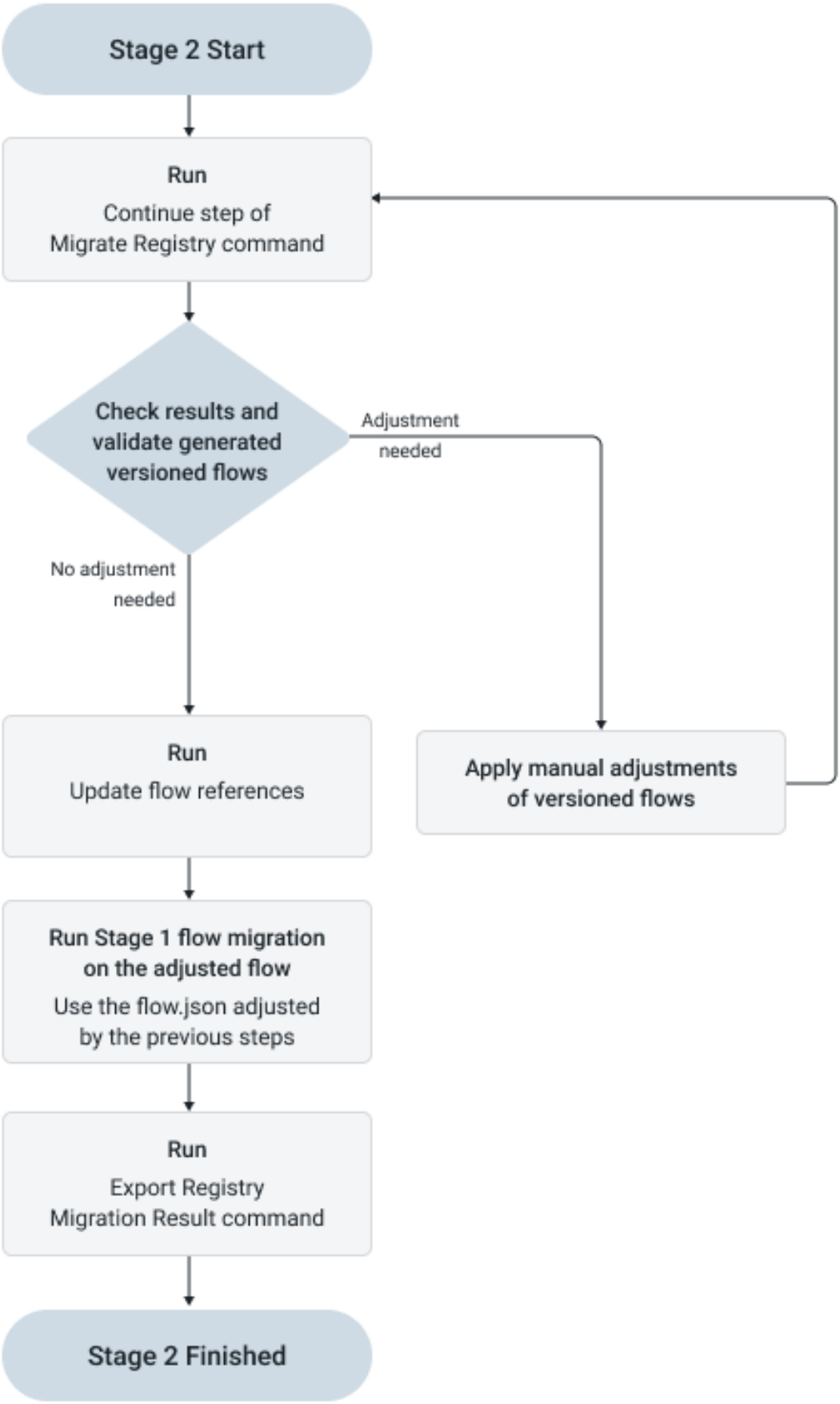


Note: Because this approach treats registered flows separately from the main flow, variable instances within registered flows are converted into parameters in parameter contexts that are independent from similarly named parameters created during flow migration. These are not linked, and if their lifecycles need to remain aligned, they must be adjusted manually.

If a registered flow uses a Parameter Context defined outside the registered flow (and incorporated into it before migration), those Parameter Contexts are not modified during migration. As a result, NiFi's existing behavior for resolving such Parameter Contexts remains unchanged after migration.

Stage 2

The following diagram illustrates the recommended workflow for the Stage 2 migration of versioned flows with NiFi Registry.


Stage 2 Start
Run
Continue step of
Migrate Registry command
Check results and
validate generated
versioned flows
Adjustment
needed
No adjustment
needed
Apply manual adjustments
of versioned flows
Run
Update flow references
Run Stage 1 flow migration
on the adjusted flow
Use the flow.json adjusted
by the previous steps
Run
Export Registry
Migration Result command
Stage 2 Finished

Just like in normal flow migration, Stage 1 follows an iterative approach, while Stage 2 cannot be executed again on a set of artifacts that have already been processed. To successfully migrate the registries and the flow, however, some additional steps are necessary.

Preparation

Running Stage 2 requires further preparation. This step can be completed during the initial preparation, but considering the potential length of the process, it is possible that some of the necessary information is not available at that time. At this point, the Migration Tool must be prepared to export the required information from the Working Registry and apply the necessary transformations to bring the snapshots to their final state.

To achieve this, additional properties must be provided as follows: for every registry dump that was migrated during the previous steps, the following properties must be defined in the migration.tool.properties file:

```
registry.final.location.{$identifier}.name
registry.final.location.{$identifier}.url
```

Here, the identifier is a property-format-compatible string that links the two parameters. The name must match the directory name of the corresponding registry dump (and the related Registry Client), and the url should be the URL of the target NiFi Registry. Using this information, when the Migration Tool writes out the results of the final step and creates an importable dump, it ensures that nested flow references point to the correct target URL.

Example

In the previous examples, https://internal-domain:18080 and https://other-internal-domain:18082 were used as the URLs for the R1 and R2 registries, respectively. These are the URLs of the source registries, from which you want to migrate. The target registries are installed separately at different URLs, and you want the internal references (nested flows) to point to the correct URLs. To achieve this, you need to set the target URL for both registries. This can be done as follows:

```
registry.final.location.firstRegisrty.name=R1
registry.final.location.firstRegistry.url=https://new-internal-domain:18087
registry.final.location.secondRegistry.name=R2
registry.final.location.secondRegistry.url=https://new-other-internal-domai
n:18088
```

As a result of this configuration, when further steps are executed and the final output of the registry migration is created, the storage location (the attribute that connects the nested flow to the referring one) of the reference will be set based on this mapping. Thus, if a registered flow has a nested flow from the R2 registry, the storage location will contain https://new-other-internal-domain:18088.

It is possible to use the same target URL for multiple registries. This can be useful if, in the Cloudera Flow Management 2 environment, multiple registries served the NiFi cluster, but in the Cloudera Flow Management 4 environment, a smaller number or even a single registry backs the NiFi cluster. While this is a fully supported scenario, there are some considerations to keep in mind:

- Regardless of URL reuse, all property pairs must be fully specified.
- The Migration Tool generates independent, import-ready registry dumps and does not handle overlapping flow names. If multiple registries contain flows with the same (original) name under the same (original) bucket name, importing them into the same target registry will fail.

In this scenario, the following properties would be expected:

```
registry.final.location.firstRegisrty.name=R1
registry.final.location.firstRegistry.url=https://new-internal-domain:18087
registry.final.location.secondRegistry.name=R2
registry.final.location.secondRegistry.url=https://new-internal-domain:18087
```

If the R2 registry has a flow named Flow1 in bucket B1 (and the R1 registry also has a flow with the same name in a bucket with the same name), both dumps will contain a Flow1 under the same bucket name (B1), preventing successful import into a single target registry.

Note: While this is a limitation of the target environment, the Working Registry can handle this situation. Because each bucket is prefixed with the name of the source registry, there are no naming conflicts. In this example, the Working Registry would contain separate Flow1 instances under the buckets R1_under-migration_B1 and R2_under-migration_B1, handled separately.

Migrateing the Registry

At this point, the final migration of the registry snapshots can be executed. Like in the case of flow migration, this execution applies changes that break Cloudera Flow Management 2 compatibility, and the result is intended to be used with Cloudera Flow Management 4. During this step, the Migration Tool loads all managed snapshots from the Working Registry, applies the necessary transformations, and writes the results back into the Working Registry. From this point, the resulting snapshots cannot be loaded into Cloudera Flow Management 2.

Just as before, all flows have an accompanying activity log containing all necessary information about the migration relevant to that particular registered flow. No partitioning is allowed for this command using the registry, registryBucket, or registryFlow arguments; the migration is applied to the entire set of flows managed by the Working Registry.



Note: As Stage 1 always runs before Stage 2, if there are obsolete nested references within the registries, or if manual changes have generated further migration events that have not yet been applied, new versions may be added before the Migration Tool starts Stage 2. This situation should be avoided, as Stage 1 changes may be applied together with Stage 2, which prevents resolving them within Stage 1.

```
./migration.sh nifi migrate-registry -c -od /my_dir/out
```

Only the continue step is valid for execution involving Stage 2, and the filtering arguments (registry, registryBucket, and registryFlow) must not be used. The flow.json is not updated during this step; its content is still considered to be Stage 1 compatible.

Manual resolution

After upgrading the registered flows to be compatible with Cloudera Flow Management 4, further manual changes and adjustments might be necessary. The approach is the same as before, except that the snapshots now require Cloudera Flow Management 4 to be loaded. The activity logs generated as a result of the last command serve as a guideline for the expected and possible functional changes.

In addition, it is important to validate the registered flows from a business perspective. While most Cloudera Flow Management 4 components behave identically or similarly to their Cloudera Flow Management 2 counterparts, and the same applies to changes introduced by the migration, the similarity is not 100%, so identical behavior cannot be guaranteed without human oversight.

Migrating the flow.json

Stage 2 migration of the flow (variables, templates, and components) proceeds in the same way as usual, except for the use of the registryAware argument. As in Stage 1, this primarily serves as a flag to skip versioned Process Groups in the flow.

In addition, if the user executes all commands together using the shorthand migrate-all, an additional command, Upgrade Flow References, is executed. If the commands are executed one by one, this command must be executed manually. As usual, manual adjustment of the flow takes place at this stage.

Furthermore, because the Upgrade Flow References step (see below) runs before manual adjustments (either as part of migrate-all or through manual execution), the flow should contain the latest version of all registered snapshots checked out from the Working Registry. This allows further adjustments to be made to the registered flows within the context of the full flow. These changes must be committed to the Working Registry.

While this is a useful stage for resolving discrepancies between the flow and the registered flows, it is important to keep in mind that a given registered flow may be used in multiple places. Also note that, as this is a manual step, no references (including both references directly in the flow.json and nested flow references) are updated automatically.

Upgrading flow references

The migrate-registry command executed previously handled the Stage 2 changes for the snapshots but did not update the state of the flow.json. This command is responsible for connecting the two and checking out the newest versions of the registered flows from the Working Registry.

Although this is the same command used during Stage 1 and it will inevitably run the Stage 1 part, in this case it will not cause any regression; the result will contain only Stage 2 changes. The following command must be executed:

```
./migration.sh nifi upgrade-flow -i /my_dir/flow.json -od /my_dir/out/out
```

The command still requires a running Working Registry. After execution, the checked-out registered flows in the flow.json will reflect the latest versions from the Working Registry.

Exporting Registry migration result

As a final step of the actual Registry migration process, the artifacts to be imported into the target NiFi Registry and NiFi must be generated. To do this, it is necessary to have the updated version of the flow.json from the last step, a running Working Registry, and the properties defined at the beginning of Stage 2. During this command, the URLs will be replaced with the target (final location) URLs, corresponding to the registry name, in both the flow and the registered flows, and the necessary snapshots will be exported as a registry export. The following command must be executed:

```
./migration.sh nifi export-registry-result -i /my_dir/flow.json -od /my_dir/
out
```

The command replaces URLs in the flow and nested references based on the previously set properties and writes the results as exported registry snapshots into the output directory, under the directory specified by the name of the registry. After this point, the up-to-date versions of the snapshots are no longer in the Working Registry but in the registry exports generated by this command. As a result, further changes made in the Working Registry will not be reflected in the final artifacts unless this command is executed again, generating a new set of registry migration results. A newly generated result is completely independent of any previously generated set.

Thus, instead of pointing to the URL of the Working Registry, these registered flows will point to the target registries defined in the migration.tool.properties. In addition, the URLs in the Registry Clients will be updated based on the same logic. For example, if the final (target) registry URLhttps://new-internal-domain:18087 is bound to the registry named R1 in the properties, the Registry Client with the name R1 will be updated to use that URL.

The version of the exported snapshot is usually 1. There are two exceptions to this. The first is when the Migration Tool is instructed to keep the version history. In this case, all snapshot versions from the original dump (not from the Working Registry; mid-migration versions are not kept in the final result) are added to the output in their original format. These are still non-migrated Cloudera Flow Management 2 snapshots and are not intended for loading into a Cloudera Flow Management 4 instance, but they can be useful for preserving change history within the target NiFi Registry. To achieve this, the following command must be executed:

```
./migration.sh nifi export-registry-result -i /my_dir/flow.json -ed /my_dir/
flow_exports -od /my_dir/out --keepHistory
```

As shown in the command, the keepHistory argument is required for this behavior. In addition, the path to the original registry exports used during the initial phase of the migration must be provided. These historical snapshots are not added to the Working Registry, only to the export. In such cases, the version of the migrated snapshot is not 1; instead, it is assigned the next version in the sequence. For example, if the last version in the history is 17, the migrated Cloudera Flow Management 4–compatible snapshot will have version 18.

The other situation in which the version might differ from 1 is when multiple versions of the original flow are used in the flow or in referencing registered flows, and the Migration Tool “splits” the flow into multiple lines (one for each version used). This splitting exists only to simplify the migration process (although it is generally preferable to resolve these differences before migration begins), and the lines are merged in this final phase. The actual content of the snapshots is not merged; instead, rather than multiple separate flows, a single flow (using the original name, without the “line” postfix) is exported with multiple snapshots, where each snapshot represents the final version of a line. The version number corresponds to the line order; therefore, while the version numbers may differ from the

original, the relative order is preserved. Snapshots from lines with higher version numbers will have higher version numbers in the result.

Using the dumps from the previous example, and considering that the flow.json uses Flow1 (versions 2 and 3) and Flow4 (version 2) directly, and Flow2 (version 1) via Flow4, the following files will be present in the export generated by the command:

```
/my_dir/out/R1/toolkit_registry_export_all_B1_Flow1_1
/my_dir/out/R1/toolkit_registry_export_all_B1_Flow1_2
/my_dir/out/R1/toolkit_registry_export_all_B1_Flow2_1
/my_dir/out/R2/toolkit_registry_export_all_B1_Flow4_1
```

During migration, because multiple versions of Flow1 were used by the flow.json, these versions were represented as separate flows and therefore migrated separately. In the Working Repository, these versions appeared as the flows Flow1_line_1 and Flow1_line_2. As the list of results shows, these were merged back into snapshots under the same flow, Flow1.

Here, toolkit_registry_export_all_B1_Flow1_1 and toolkit_registry_export_all_B1_Flow1_2 are the fully migrated versions of versions 2 and 3 of the original Flow1, respectively (note that intermediate versions from the Working Registry are not kept). Similarly, toolkit_registry_export_all_B1_Flow2_1 is the fully migrated version of version 1 of the original Flow2, and toolkit_registry_export_all_B1_Flow4_1 is the fully migrated version of version 2 of the original Flow4.

These two scenarios (keeping history and having multiple lines) can overlap. In such cases, the migrated snapshots receive the next version numbers sequentially, in the order defined by the line order. When keeping history, the same migration would result in the following exported snapshots:

```
/my_dir/flow_exports/R1/toolkit_registry_export_all_B1_Flow1_1
/my_dir/flow_exports/R1/toolkit_registry_export_all_B1_Flow1_2
/my_dir/flow_exports/R1/toolkit_registry_export_all_B1_Flow1_3
/my_dir/flow_exports/R1/toolkit_registry_export_all_B1_Flow1_4
/my_dir/flow_exports/R1/toolkit_registry_export_all_B1_Flow1_5
/my_dir/flow_exports/R1/toolkit_registry_export_all_B1_Flow2_1
/my_dir/flow_exports/R1/toolkit_registry_export_all_B1_Flow2_2
/my_dir/flow_exports/R1/toolkit_registry_export_all_B2_Flow3_1
/my_dir/flow_exports/R2/toolkit_registry_export_all_B1_Flow4_1
/my_dir/flow_exports/R2/toolkit_registry_export_all_B1_Flow4_2
/my_dir/flow_exports/R2/toolkit_registry_export_all_B1_Flow4_3
```

In this case, toolkit_registry_export_all_B1_Flow1_4 and toolkit_registry_export_all_B1_Flow1_5 are the fully migrated versions of versions 2 and 3 of the original Flow1, respectively; toolkit_registry_export_all_B1_Flow2_2 is the fully migrated version of version 1 of the original Flow2; and toolkit_registry_export_all_B1_Flow4_3 is the fully migrated version of version 2 of the original Flow4. All other snapshots are copied from the original export.

In most cases, the flow is ready to be loaded, but before that, some final steps are necessary.

Closure

In order to have a fully working NiFi cluster (with registered flows), some additional steps are required after the migration. This section assumes that both the target NiFi environment and the NiFi Registries are in place and properly set up.

Importing Registry exports

The exports generated by the Export Registry Migration Result command must be imported into the target registry, one by one. This can be done using the Import All Flows command of the NiFi Toolkit, as shown in the following examples:

```
./bin/cli.sh registry import-all-flows -u http://new-internal-domain:18087 -
-input "/my_dir/flow_exports/R1"
-- and
```

```
./bin/cli.sh registry import-all-flows -u http://new-other-internal-domain:
18088 --input "/my_dir/flow_exports/R2"
```

For more information on this command, see the Apache NiFi Toolkit Guide. Each registry requires a separate import, so R1 and R2 in the example are imported using separate commands with their own specific arguments. If the commands above are executed correctly, the nested references in the registry snapshots will point to the registries into which they are imported.

Continuing the example, Flow4 in R2 has a nested flow, Flow2 from R1. The exported snapshot of Flow4 will have its reference to Flow2 pointing to https://new-internal-domain:18087, based on the configured properties. If Flow2 is imported into a different NiFi Registry instance at another location, the reference will be invalid, preventing NiFi from properly resolving the nested flow when adding it to the canvas.

Note:

NiFi has limited capabilities to resolve URL differences during loading. This should be considered a failsafe rather than a recommended approach. While NiFi may be able to load nested flows correctly in such cases, it is safer to ensure that the URLs match.

Adding flow.json to NiFi

As in migration without registry support, the flow.json must be added to a NiFi instance to be used. As with registry imports, flow references point to specific URLs (through clients, see below) defined by the property files. To keep the flow functioning properly, it is recommended to ensure that these URLs match.

Updating client properties

The Migration Tool keeps the registry URLs up to date by replacing the URL property of the NiFi Registry clients according to the properties file, in sync with the registry references. However, the Migration Tool is not aware of any authentication or authorization information.

To properly connect to the NiFi Registry instances, these settings must be updated by the user according to the requirements of the given registry instance. This can be done by opening the Controller Service and navigating to the Registry Clients tab. By editing the SSL Context Service property of the given Registry Client, the necessary settings can be applied.

Manual validation

During previous steps, manual validation was included at multiple points as part of the migration strategy. While these steps helped prevent different parts of the flow from deviating from the desired results, it is strongly recommended to validate the entire flow in the target environment. This includes, but is not limited to, verifying that NiFi can establish connections with the NiFi Registries used and that the registered flows point to the correct versions.

Migrating a flow using flow.json as input

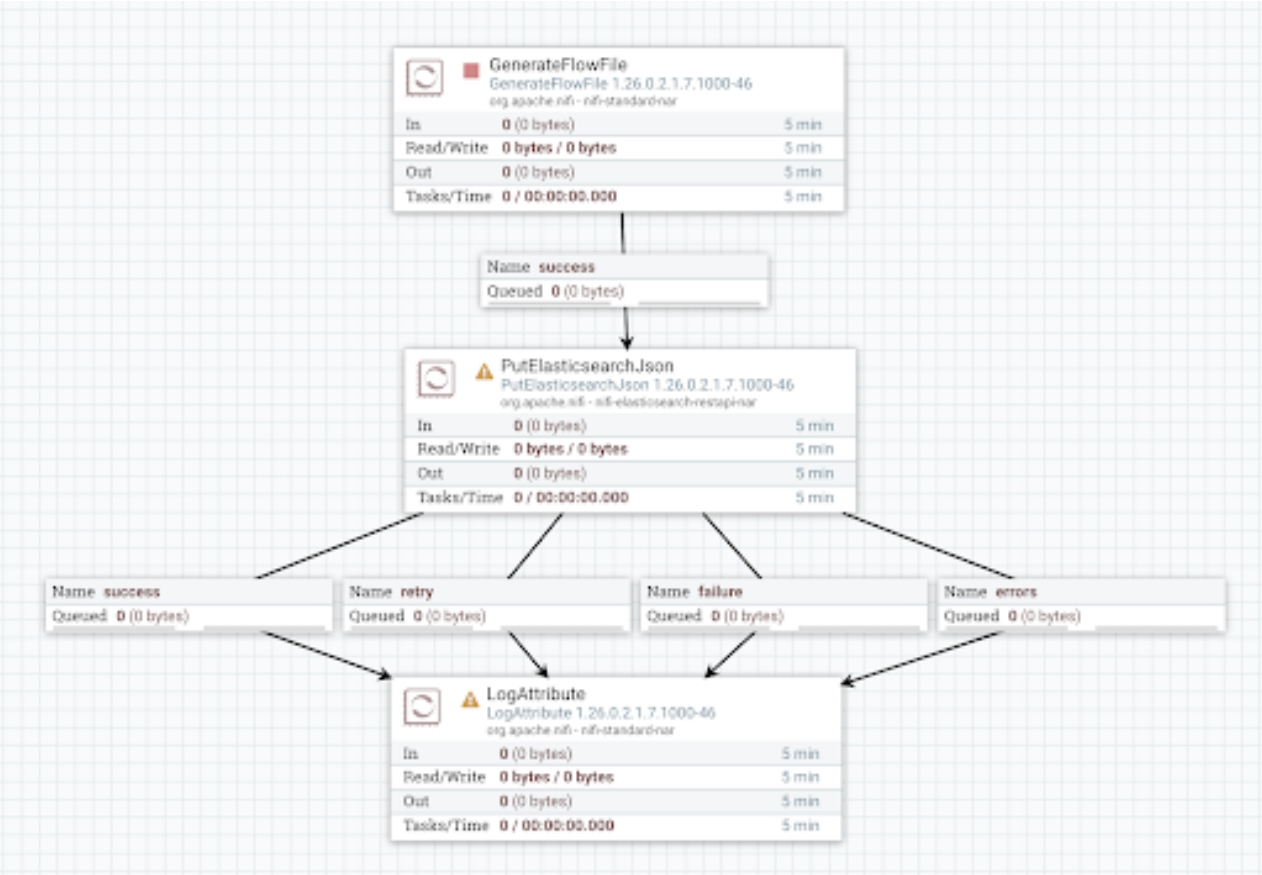
This section provides step-by-step examples of how to run different migrations with the Cloudera Flow Migration Tool using a **flow.json** file as input.

Migrating templates using flow.json as input

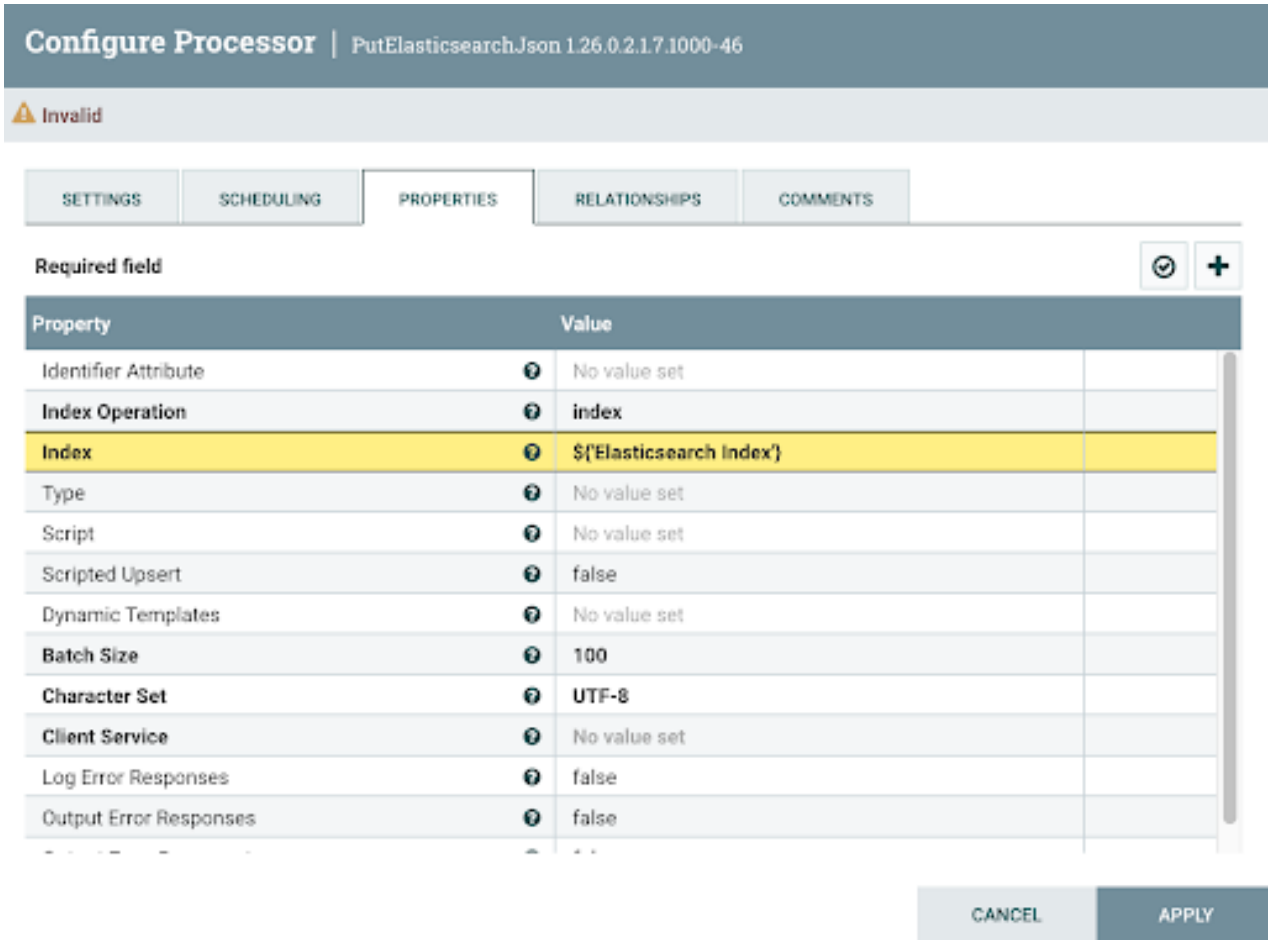
Learn how to use the Cloudera Flow Migration Tool to extract and transform templates for compatibility with NiFi 2 using a flow.json file as input. NiFi 2 does not support templates, so this step is required to ensure compatibility with the target version.

Example flow for migrating a template

You have a flow that has a template named **elastic_template**, which was created from a process group called **elastic**. This process group contains the following simple flow:



The process group has a variable, **Elasticsearch Index**, referenced in the Index property of the PutElasticsearchJson processor.



Your goal is to migrate this template to NiFi 2 used in Cloudera Flow Management 4.12.0.

Before you begin

1. Stop all processors and disable all controller services in NiFi.
2. Stop NiFi.
3. Copy the flow.json.gz file from NiFi's conf directory to the Migration Tool's input folder (/etc/migration-tool-input).
4. Unzip the file to obtain flow.json.

Procedure

1. Run Stage 1 template migration using the following command.

```
bin/migration.sh nifi migrate-templates \
-i /etc/migration-tool-input/flow.json \
-od /etc/migration-tool-output/templates \
-sco
```

This generates a sourceVersion folder that contains the output files of the migration.

```
templates
### sourceVersion
    ### NiFi_Flow_ad3a86a6-0194-1000-78d9-6374298d9a0c
    #   ### elastic_template.json
    #   ### elastic_template.xml
    ### activity_log.json
```

The folder name NiFi_Flow_ad3a86a6-0194-1000-78d9-6374298d9a0c indicates that the exported template belongs to the root process group, which is named NiFi Flow in NiFi. The unique ID for this group is ad3a86a6-0194-1000-78d9-6374298d9a0c.

elastic_template.json

- Flow definition containing the contents of the original template
- Equivalent to manually converting a template to a flow definition in NiFi 1 by:
  a. Instantiating the template on the canvas.
  b. Right-clicking the process group.
  c. Selecting Download flow definition without external services.
- Modified by the Migration Tool to ensure compatibility:
  - Variables (not supported in NiFi 2) converted into parameters
  - Parameter context created to hold the new parameters
  - Processors updated to reference parameters instead of variables
- This is still a NiFi 1 flow definition, so do not load this file directly into NiFi 2. Use an empty NiFi 1 instance to review the modification made by the tool. The flow will appear unchanged, but variables will be replaced with parameters of the same name, which are now referenced by the processor instead.
- Changes in the syntax: Variables were previously referenced using ${}, whereas parameters are now referenced using #{}.
  - **NiFi 1 variables**: ${variable_name}
  - **NiFi 2 parameters**: #{parameter_name}

elastic_template.xml

- Original template exported from NiFi 1
- Identical to downloading it from the Templates menu in NiFi 1
- Compatible only with NiFi 1 as NiFi 2 does not support templates and cannot parse XML template files

activity_log.json

- Log of all actions performed during this stage of the migration
- The following were changes made during the template migration:
  - A new parameter context was created with a new parameter in it.
  - The parameter context was assigned to the process group.
  - The PutElasticsearchJson processor was updated to reference the new parameter.
- Components are referenced by a unique ID, not by name.

2. Validate the Stage 1 template migration output.

a) Check the new flow definition in a NiFi 1 instance to verify that the flow matches your expectations.

1. Start your NiFi 1 instance.
2. Create a new process group.
3. Load elastic_template.json to NiFi 1.
4. Confirm that variables were correctly converted to parameters.

b) Review activity_log.json and address any manual-change-requests or manual-validation-requests.

c) If there are manual-change-requests or manual-validation-requests to handle, follow these steps:

1. Start a NiFi 1 instance.
2. Load the sourceVersion/<template_name>.json flow definition by creating a new process group and uploading the JSON file.
3. Make the modifications on the NiFi canvas, indicated by the manual-change-requests or manual-validation-requests in the sourceVersion/activity_log.json.

4. Stop the NiFi instance.
5. Fetch the flow.json.gz file from the NiFi conf directory.
6. Unzip it to produce a flow.json file.
7. Perform variable migration and then component migration on this flow.json file. For instructions, see the example scenario for variable and component migration.

d) If no additional changes are required, proceed to Stage 2.

3. Run full template migration (Stage 1 and 2) using the following command.

```
bin/migration.sh nifi migrate-templates \
-i /etc/migration-tool-input/flow.json \
-od /etc/migration-tool-output/templates
```

The contents of the previously generated sourceVersion folder will be overwritten, but this is not a concern as the tool generates the same sourceVersion output as before. Additionally, a targetVersion directory is created, containing the output files of the Stage 2 part of the migration.

```
targetVersion
### NiFi_Flow_ad3a86a6-0194-1000-78d9-6374298d9a0c
#   ### elastic_template.json
### activity_log.json
```

elastic_template.json

- NiFi 2-compatible flow definition
- It contains the contents of the original template converted into an exported process group.
- This version is compatible with NiFi 2, but no longer supports NiFi 1.



Note:

An elastic_template.xml is not generated as NiFi 2 does not support XML templates.

activity_log.json

- List of all actions performed during this stage of the migration.

4. Validate the Stage 2 template migration output.

a) Check the new flow definition in a NiFi 2 instance to verify that the flow matches your expectations.

1. Start your NiFi 2 instance.
2. Create a new process group.
3. Load elastic_template.json into a NiFi 2 instance.

   The new process group will be called **elastic_template** and will contain another process group named **elastic**, matching the name of the process group that was converted into a template in NiFi 1 before you started the migration.
4. Verify parameter replacements and processor updates.

b) Review activity_log.json and address any manual-change-requests or manual-validation-requests.

In this case, you can see the following manual-change-request:

```
{
 "sequence" : 4,
 "type" : "manual-change-request",
 "subject" : "a1468d69-3f30-3f83-a7c1-91dad09890f7",
 "message" : "New relationship [original] has been added, it has to be
 connected to a downstream processor or terminated.",
 "context" : {
   "rule" : "6501a693-b524-4d1a-b11e-69747c2651e8"
}
```

1. Open the targetVersion/elastic_template.json file and search for the "subject" ID, a1468d69-3f30-3f83-a7c1-91dad09890f7. This ID refers to the PutElasticsearchJson processor.
2. Go to the NiFi 2 canvas and check the processor. You will find that it has an unbound “original” relationship that needs to be connected to a downstream component.
3. Make the required change manually on the canvas.
4. Once done, export the process group. This exported process group is now a fully NiFi 2-compatible version of the original template.
5. Save the file.

Results

You have finished the template migration process.

Migrating variables using flow.json as input

Learn how to use the Cloudera Flow Migration Tool to convert variables to parameters and parameter contexts using a flow.json file as input.

Example flow for migrating variables

The following NiFi flow is used to demonstrate both variable and component migration. It illustrates how variables are used within process groups and how they are referenced by individual components in the NiFi flow.

The flow consists of two process groups:

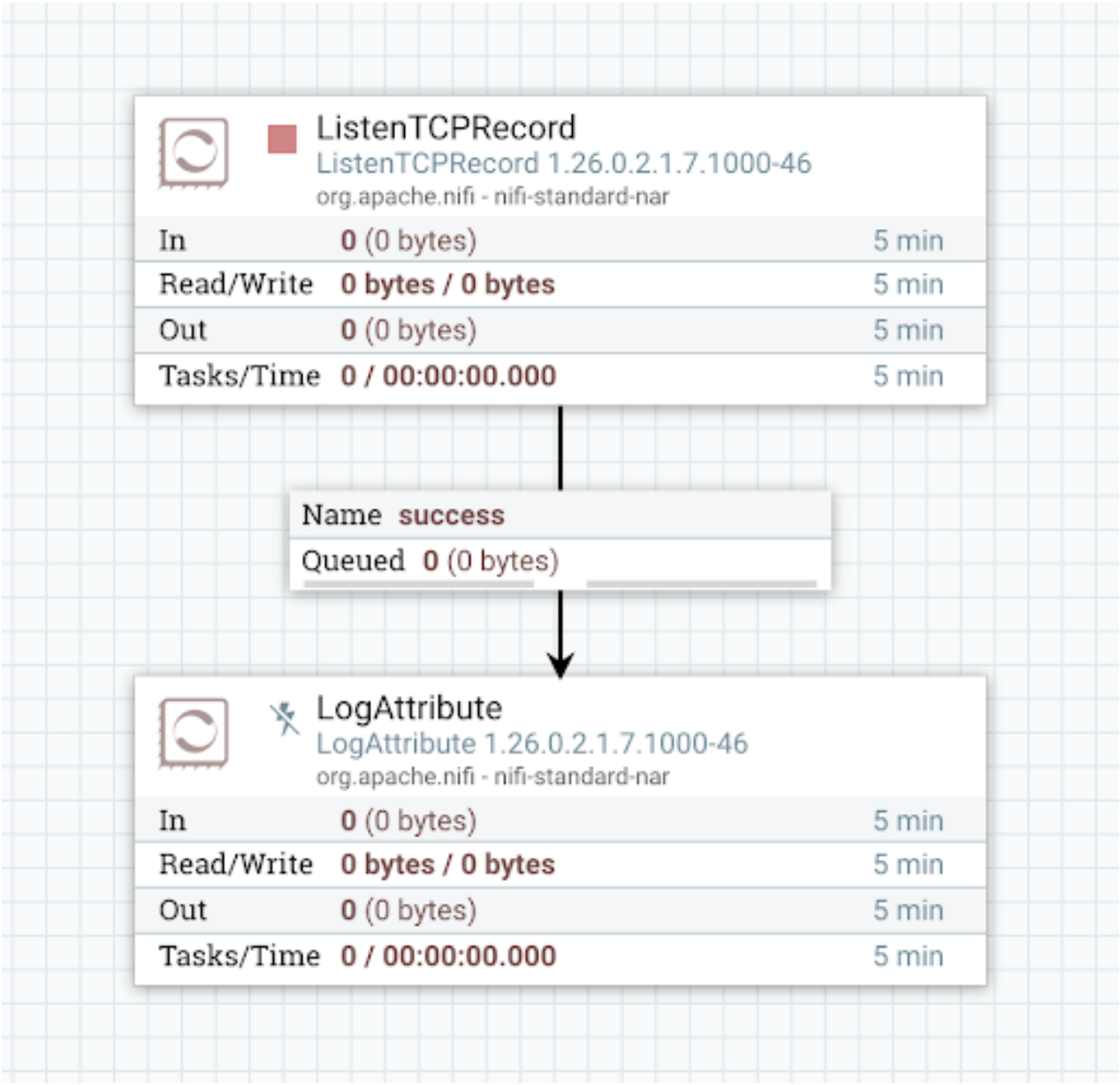
- **TCP Listener** (ID: b41940d7-0194-1000-42fc-458834630567)
- **Elastic** (ID: b42881c7-0194-1000-3cdf-1bd453a0ed0f)

At the root level, the flow is structured as follows:

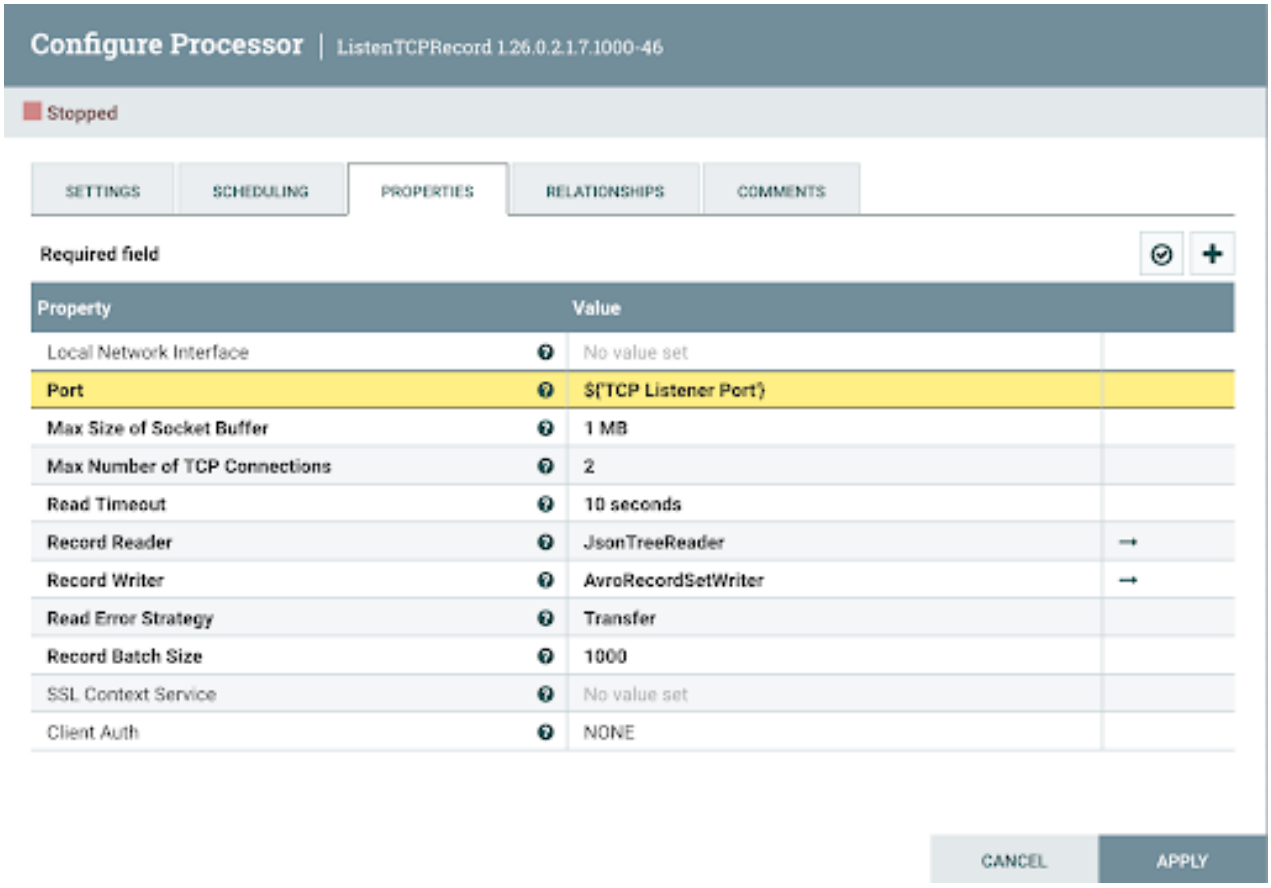


TCP Listener Process Group

This process group contains the following simple flow:

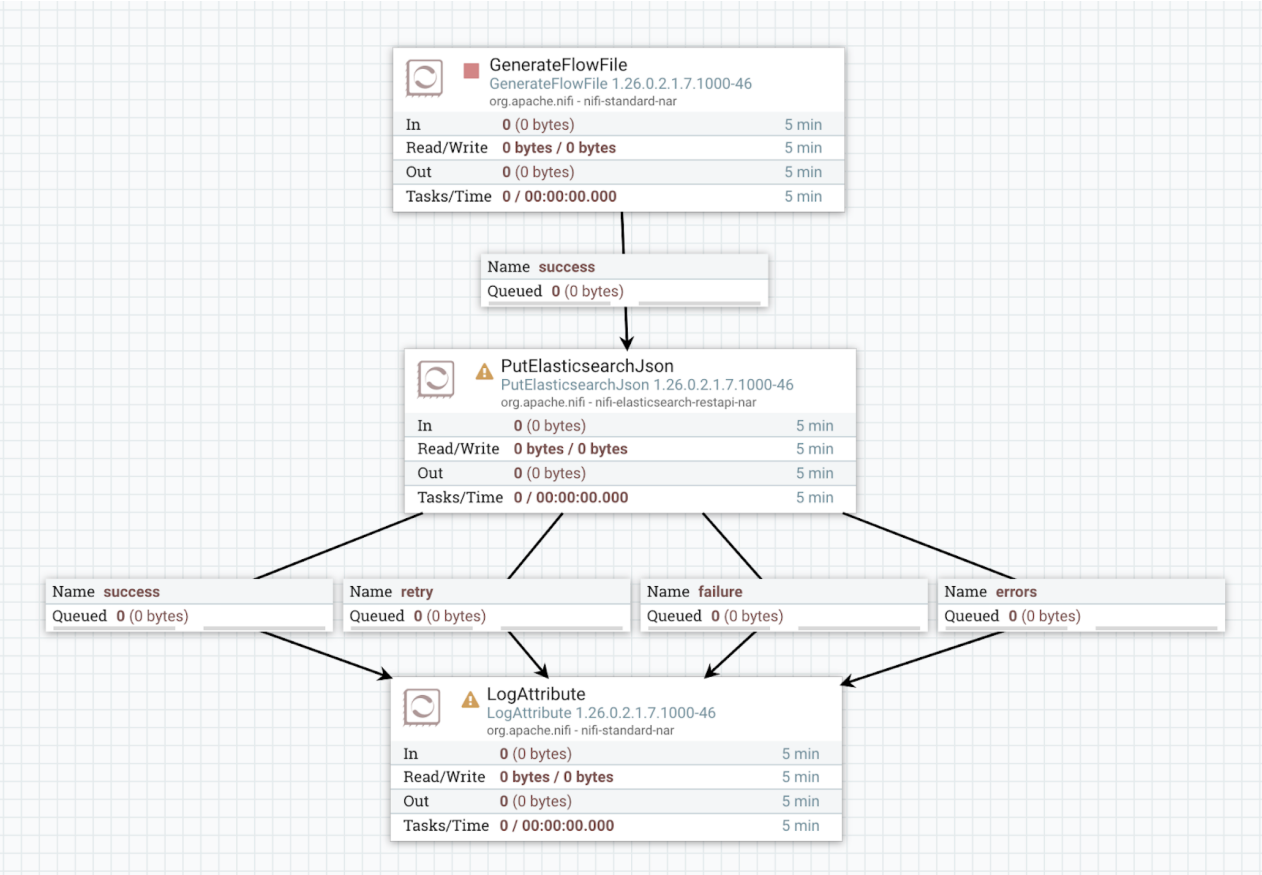


The process group defines a variable called **TCP Listener Port**, which is referenced by the ListenTCPRecord processor.

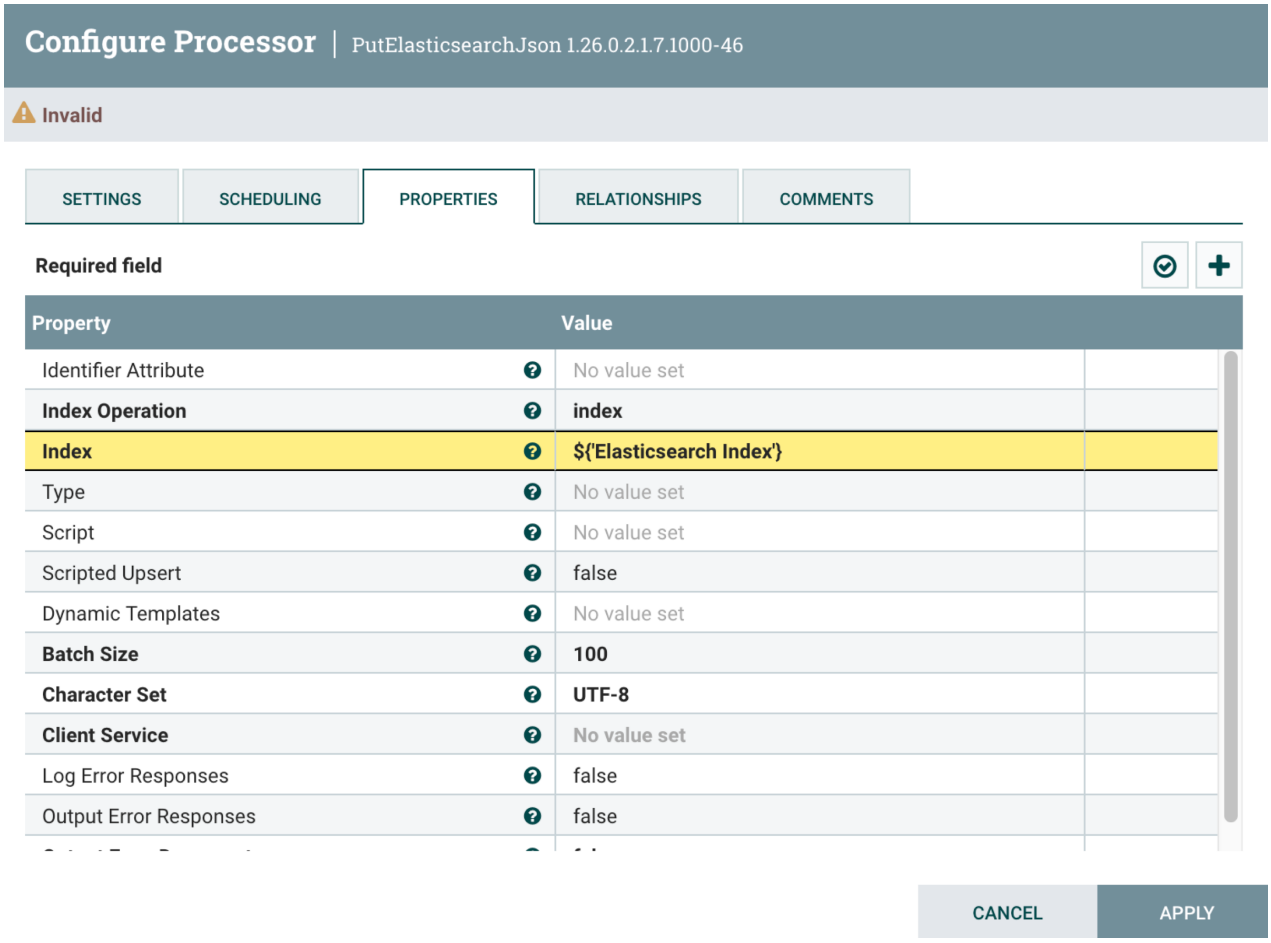


Elastic Process Group

This process group also contains a simple flow:



The process group defines a variable called **Elasticsearch Index**. This variable is referenced in the PutElasticsearchJson processor's Index property.



Your goal is to migrate the variables used in Cloudera Flow Management 2.1.7.4000 powered by NiFi 1 to parameters used in Cloudera Flow Management 4.12.0 powered by NiFi 2.

Note: NiFi 2 supports only parameters, not variables, so variable migration is required to ensure compatibility with the target version.

The example guides you through variable migration one process group at a time, simplifying the process and maintaining a clear activity log. While this example flow is simple, the step-by-step approach shows how this method improves clarity for more complex migrations. In real-world scenarios, you have to define a migration strategy based on your flow's structure, individual process group migration may not always be necessary. Migrating a parent process group automatically applies the changes recursively to its descendants.

Before you begin

1. Stop all processors and disable all controller services in NiFi.
2. Stop NiFi.
3. Copy the flow.json.gz file from NiFi's conf directory to the Migration Tool's input folder (/etc/migration-tool-input).
4. Unzip the file to obtain flow.json.

Procedure

1. Run Stage 1 variable migration on the TCP Listener process group using the following command.

```
bin/migration.sh nifi migrate-variables \
-i /etc/migration-tool-input/flow.json \
-od /etc/migration-tool-output/variables \
```

```
-pgid <process_group_id> \
--sourceCompatibleOutput
```

This generates a sourceVersion folder that contains the output files of the Stage 1 migration.

```
variables
### sourceVersion
    ### activity_log.json
    ### migrated_flow.json
```

activity_log.json

- Log of all actions performed during this stage of the migration
- The following were changes made during the template migration:
  - A new parameter context was created with a new parameter called **TCP Listener Port**, which replaces the corresponding variable.
  - The **TCP Listener Port** variable was removed.

migrated_flow.json

- A modified NiFi 1 flow, which is not compatible with NiFi 2 yet.
- It contains everything the original flow did, except the **TCP Listener** process group now references a parameter instead of the removed variable.

2. Validate the Stage 1 variable migration output for the **TCP Listener** process group.
   a) Load the migrated_flow.json into a NiFi 1 instance and check the flow.
      1. Ensure NiFi is not running.
      2. Go to NiFi's conf directory and back up the flow.json.gz and flow.xml.gz files.
      3. Delete the original flow.json.gz and flow.xml.gz files.
      4. Rename the migrated_flow.json to flow.json and compress it using gzip to create flow.json.gz.
      5. Copy the newly created flow.json.gz file to NiFi's conf directory.
      6. Start NiFi and check the flow.
   b) Review the activity_log.json file and check for any manual-change-requests or manual-validation-requests. If none are present, proceed to step 3 to run Stage 1 variable migration on the **Elastic** process group.
      1. If manual changes are necessary, load the migrated_flow.json to NiFi, update and validate it.
      2. Once the flow is validated and meets expectations, continue with the next step using the new flow.json.gz file (unzipped to produce flow.json).
3. Run Stage 1 variable migration on the **Elastic** process group.
   a) Move migrated_flow.json from Step 2 into the input folder (/etc/migration-tool-input) and rename it to flow.json for clarity.
   b) Make a backup of the output folder (/etc/migration-tool-output/variables) before running the next migration step.

      Note: This step is crucial because the same output folder is used as in the previous migration, and its contents will be overwritten. To retain a record of changes made at each stage, it is better to create a backup.
   c) Run Stage 1 variable migration on the **Elastic** process group using the following command.

```
bin/migration.sh nifi migrate-variables \
-i /etc/migration-tool-input/flow.json \
-od /etc/migration-tool-output/variables \
-pgid b42881c7-0194-1000-3cdf-1bd453a0ed0f \
--sourceCompatibleOutput
```

This generates a sourceVersion folder that contains the two output files of the Stage 1 migration.

activity_log.json

- Log of all actions performed during this stage of the migration
- The following were changes made during the template migration:
  - A new parameter context was created with a new parameter called **Elasticsearch Index**, which replaces the corresponding variable of the same name.
  - The new parameter is referenced from the PutElasticsearchJson processor.
  - The **Elasticsearch Index** variable was removed.

migrated_flow.json

- A modified NiFi 1 flow, which is not compatible with NiFi 2 yet.
- It contains everything the original flow did, but the **Elastic** process group now references a parameter instead of the removed variable.

4. Validate the Stage 1 variable migration output for the **Elastic** process group.
   a) Load the migrated_flow.json into a NiFi 1 instance and check the flow.
   b) Review the activity_log.json file and check for any manual-change-requests or manual-validation-requests. If none are present, proceed to the next step.
   c) If manual changes are necessary, update the migrated_flow.json on the NiFi canvas after loading it. Once the flow is validated and meets expectations, continue with the next step.

   At this stage, both process groups no longer contain variables and use parameters instead. If the flow meets your expectations, you can either run a full variable migration to validate your flow in NiFi 2 or proceed with migrating the components.
5. Run full variable migration (Stage 1 and 2) using the following command.

```
bin/migration.sh nifi migrate-variables \
-i /etc/migration-tool-input/flow.json \
-od /etc/migration-tool-output/variables
```

   The contents of the previously generated sourceVersion folder will be overwritten, but this is not a concern as the tool generates the same sourceVersion output as before.

   Additionally, a targetVersion directory is created, containing the output files of the Stage 2 part of the migration.

   migrated_flow.json

   - NiFi 2-compatible flow in terms of variables
   - You can load it into NiFi 2 and validate it

   activity_log.json

   - List of all actions performed during this stage of the migration.

   Stage 2 migration allows you to validate the result of the variable migration process in NiFi 2. However, component migration should not be performed on the targetVersion/migrated_flow.json because the input of component migration must be a NiFi 1 flow.
6. Proceed with component migration using the migrated_flow.json output from step 3.

Migrating components using flow.json as input

Learn how to use the Cloudera Flow Migration Tool to migrate components to parameters and parameter contexts using a flow.json file as input.

Example flow for migrating components

The following NiFi flow is used to demonstrate both variable and component migration.

It consists of two process groups:

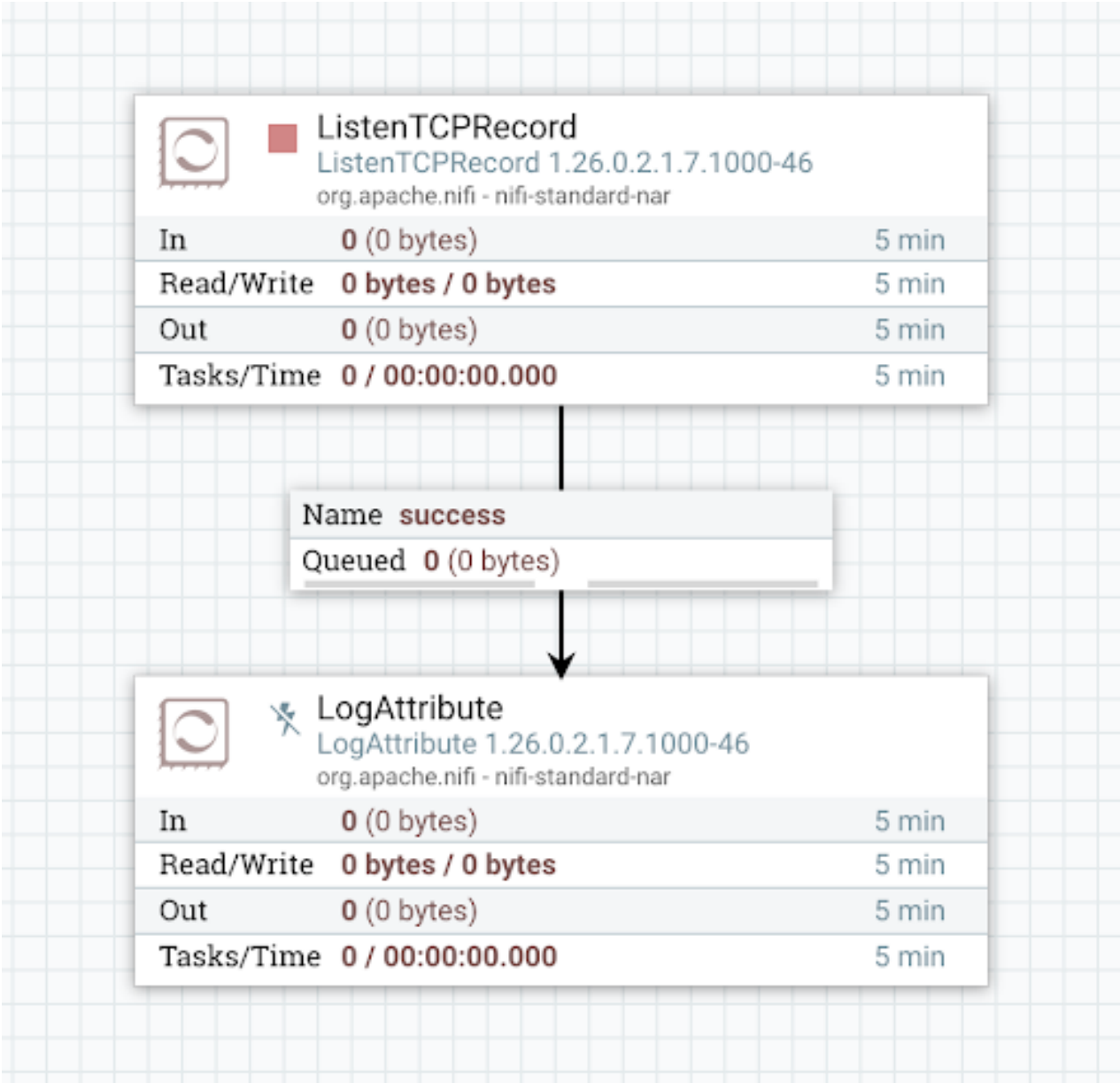
- **TCP Listener** (ID: b41940d7-0194-1000-42fc-458834630567)
- **Elastic** (ID: b42881c7-0194-1000-3cdf-1bd453a0ed0f)

At the root level, the flow is structured as follows:

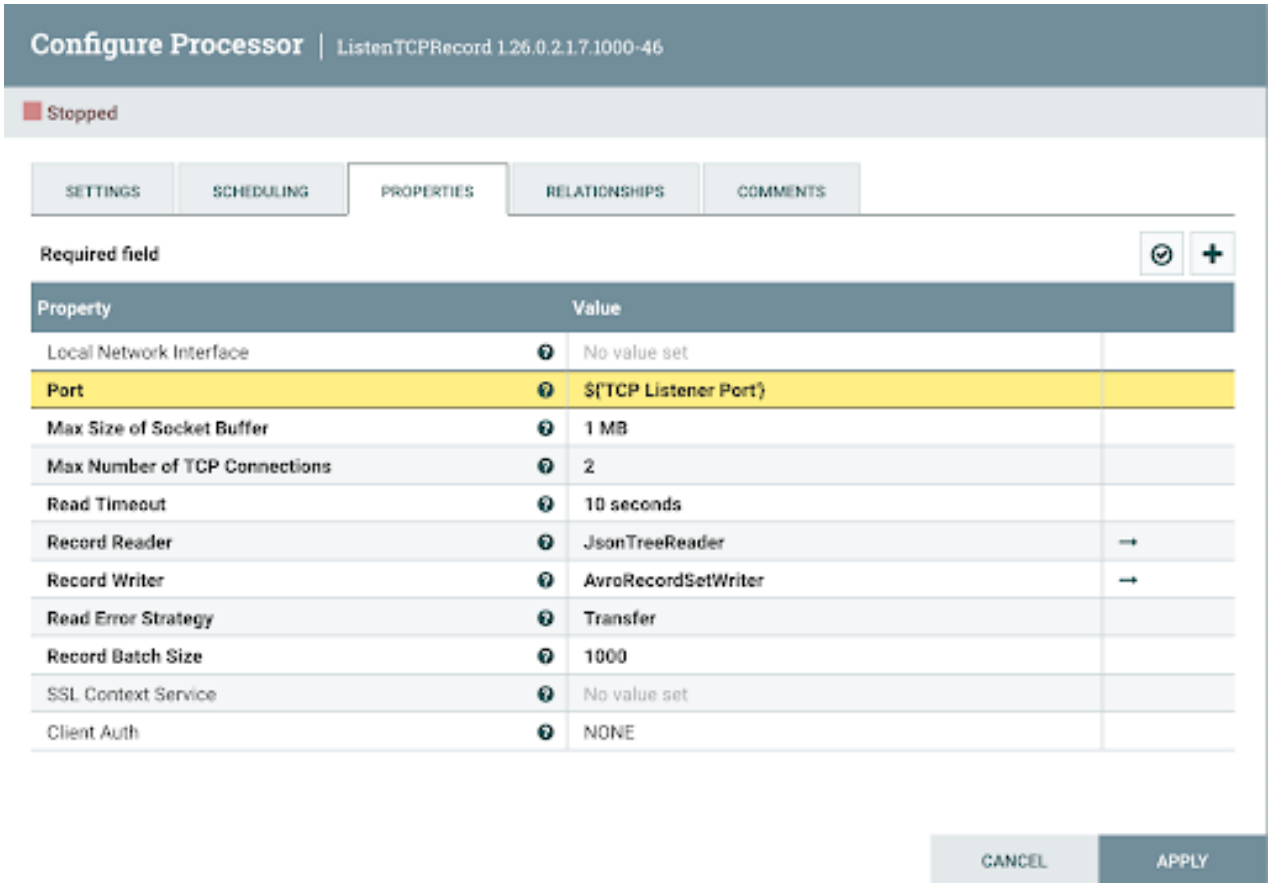


TCP Listener Process Group

This process group contains the following simple flow:

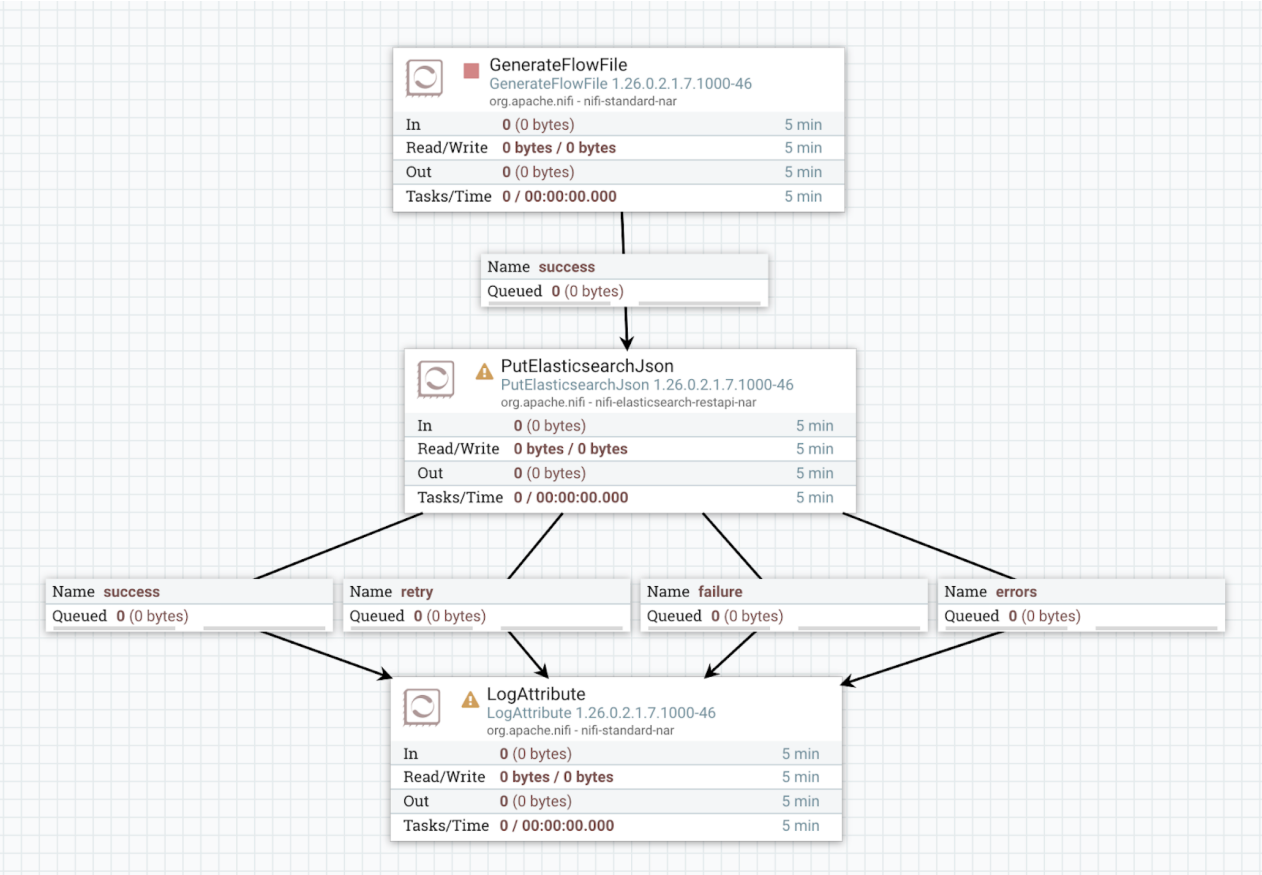


The process group defines a variable called **TCP Listener Port**, which is referenced by the ListenTCPRecord processor.

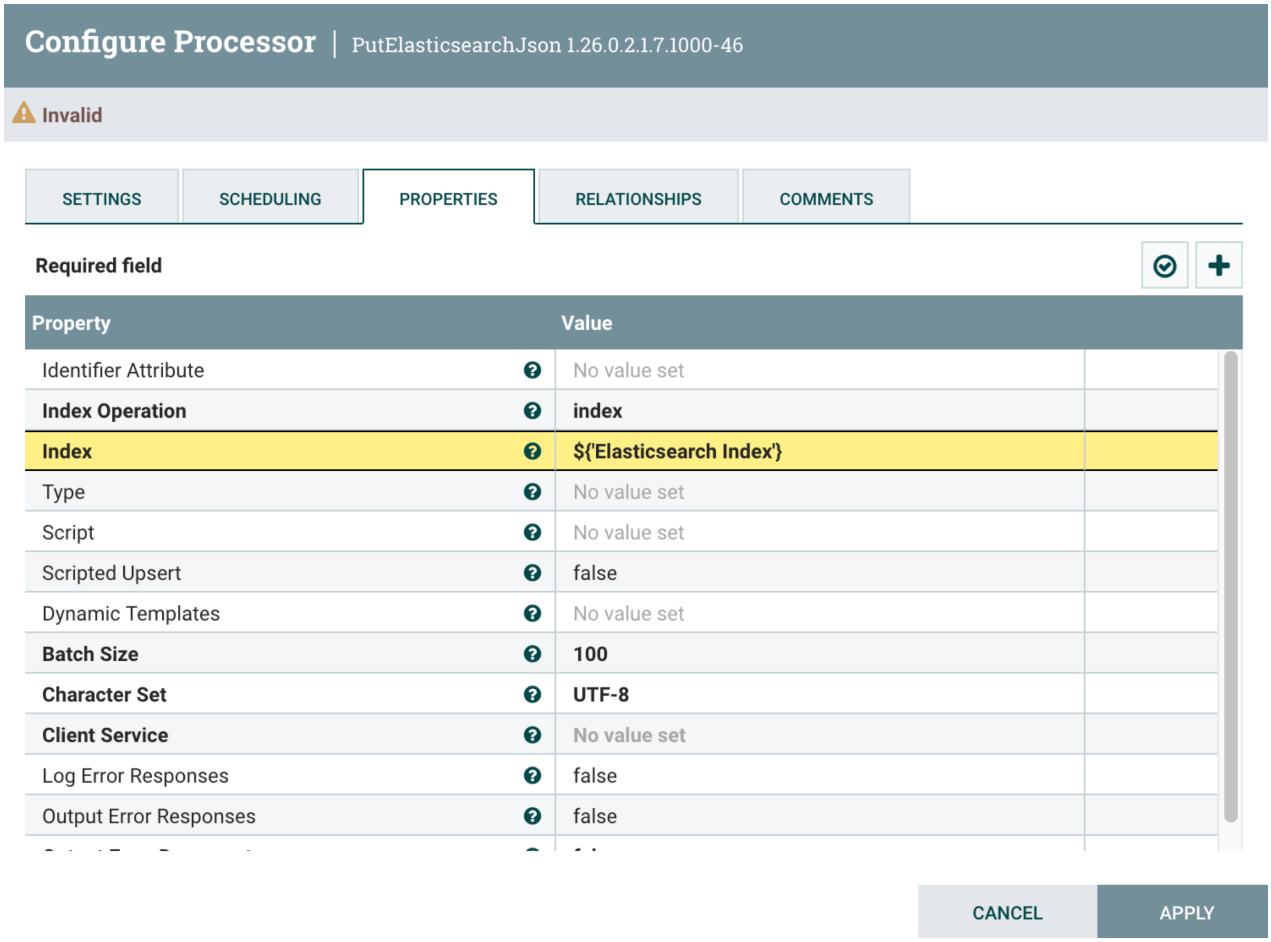


Elastic Process Group

This process group also contains a simple flow:



The process group defines a variable called **Elasticsearch Index**. This variable is referenced in the PutElasticsearchJson processor's Index property.



Your goal is to migrate the components used in Cloudera Flow Management 2.1.7.4000 powered by NiFi 1 to NiFi 2-compatible components used in Cloudera Flow Management 4.12.0 powered by NiFi 2.

The example guides you through component migration one process group at a time, simplifying the process and maintaining a clear activity log. While this example flow is simple, the step-by-step approach shows how this method improves clarity for more complex migrations. In real-world scenarios, you have to define a migration strategy based on your flow's structure, individual process group migration may not always be necessary. Migrating a parent process group automatically applies the changes recursively to its descendants.

Procedure

1. Move the migrated_flow.json file, the output of variable migration from Step 3, into the input folder (/etc/migration-tool-input) and rename it to flow.json for clarity.

   This is a NiFi-1 compatible flow that no longer contains variables.
2. Run Stage 1 component migration on the **TCP Listener** process group using the following command.

   ```
   bin/migration.sh nifi migrate-components \
   -i /etc/migration-tool-input/flow.json \
   -od /etc/migration-tool-output/components \
   -pgid b41940d7-0194-1000-42fc-458834630567 \
   --sourceCompatibleOutput
   ```

   This generates a sourceVersion folder that contains the output files of the Stage 1 migration.

   ```
   components
   ### sourceVersion
       ### activity_log.json
   ```

```
    ### migrated_flow.json
```

activity_log.json

- The log describes all the actions that were performed for this stage of the process group migration.
- Log of all actions performed during this stage of the process group migration.

migrated_flow.json

- A modified NiFi 1 flow, which is not compatible with NiFi 2 yet.
- It contains everything the original flow did, except the **TCP Listener** process group was modified as described in the activity log.

3. Validate the Stage 1 component migration output for the **TCP Listener** process group.
   a) Load the migrated_flow.json into a NiFi 1 instance and check the flow.
   b) Review the activity_log.json file and check for any manual-change-requests or manual-validation-requests. If none are present, proceed to the next step.

      In this example, you can see the following information in the activity log:

      ```
      {
       "sequence" : 2,
       "type" : "change-info",
       "subject" : "b41966ad-0194-1000-a08d-a92489457356",
       "message" : "Component [org.apache.nifi.processors.standard.ListenTCPRecord] has been deprecated (NIFI-13509)",
       "context" : {
         "rule" : "36227b60-75f0-40dc-8caf-a2ec577aa54c"
      }
      ```

      Search for the value of the “subject” element in NiFi’s search box.

      You are directed to the ListenTCPRecord processor. No manual modifications are needed at this stage. However, it is important to note that the ListenTCPRecord processor is deprecated and is not available in NiFi 2. Once the full component migration (Stage 1 and Stage 2) is complete, instructions will be provided on how to handle this deprecation.
   c) If manual changes are necessary, update the migrated_flow.json on the NiFi canvas after loading it. Once the flow is validated and meets expectations, continue with the next step.
4. Run Stage 1 component migration on the **Elastic** process group.
   a) Move migrated_flow.json from Step 2 into the input folder (/etc/migration-tool-input) and rename it to flow.json for clarity.
   b) Make a backup of the output folder (/etc/migration-tool-output/components) before running the next migration step.

      Note:

      This step is crucial because the same output folder is used as in the previous migration, and its contents will be overwritten. To retain a record of changes made at each stage, it is better to create a backup.
   c) Run Stage 1 component migration on the **Elastic** process group using the following command.

      ```
      bin/migration.sh nifi migrate-components \
      -i /etc/migration-tool-input/flow.json \
      -od /etc/migration-tool-output/components \
      -pgid b42881c7-0194-1000-3cdf-1bd453a0ed0f \
      --sourceCompatibleOutput
      ```

      This generates a sourceVersion folder that contains the two output files of the Stage 1 migration.

      activity_log.json

      - Log of all actions performed during this stage of the migration.

migrated_flow.json

- A modified NiFi 1 flow, which is not compatible with NiFi 2 yet.
- It contains everything the original flow did, except the Elastic process group was modified with the actions described in the activity log.

5. Validate the Stage 1 component migration output for the **Elastic** process group.
   a) Load the migrated_flow.json into a NiFi 1 instance and check the flow.
   b) Review the activity_log.json file.

      It contains a change-info entry that informs you of changes to a NiFi 2 processor, identified by the subject ID.

```
{
 "sequence" : 2,
 "type" : "change-info",
 "subject" : "b428aad6-0194-1000-d73d-9f2332a59f04",
 "message" : "Property [Max JSON Field String Length] has been added (N
IFI-12343); Property [put-es-json-not_found-is-error] has been renamed t
o [put-es-not_found-is-error] (NIFI-12255); Property [put-es-json-error-
documents] has been removed (NIFI-12255); Relationships [success] has be
en renamed to [original]; Relationship [successful] has been added (NIFI
-12255);",
 "context" : {
   "rule" : "6501a693-b524-4d1a-b11e-69747c2651e8"
}
```

      To identify the processor affected by these changes, search for the value of the “subject” element in NiFi’s search box.

      You are directed to the PutElasticsearchJson processor. No manual modifications are needed at this stage. However, it is important to note that changes will be applied to the PutElasticsearchJson processor during Stage 2 of the migration. Once the full component migration (Stage 1 and Stage 2) is complete, instructions will be provided on how to handle this change.

   At this stage, you have completed Stage 1 of both variable and component migration for the process groups in your flow. After reviewing the logs, you confirmed that no manual changes were needed. You can proceed with a full component migration using the migrated_flow.json from Step 4.

6. Run full component migration (Stage 1 and 2).
   a) Move migrated_flow.json from Step 4 into the input folder (/etc/migration-tool-input) and rename it to flow.json for clarity.
   b) Make a backup of the output folder (/etc/migration-tool-output/components) before running the next migration step.

      Note:

      This step is crucial because the same output folder is used as in the previous migration, and its contents will be overwritten. To retain a record of changes made at each stage, it is better to create a backup.

   c) Run the full component migration using the following command.

```
bin/migration.sh nifi migrate-components \
-i /etc/migration-tool-input/flow.json \
-od /etc/migration-tool-output/components
```

      This generates the following output:

```
components
### sourceVersion
#   ### activity_log.json
#   ### migrated_flow.json
### targetVersion
    ### activity_log.json
```

```
    ### migrated_flow.json
```

The contents of the previously generated sourceVersion folder will be overwritten. The contents represent the NiFi 1-compatible version of the migrated flow. Since the root process group only contained the two process groups on which you already performed Stage 1 component migration, the activity log will only include the same entries as those in Steps 2 and 4.

Additionally, a targetVersion directory is created, containing the output files of the Stage 2 part of the migration:

migrated_flow.json

- NiFi 2-compatible flow in terms of variables
- You can load it into NiFi 2 and validate it

activity_log.json

- List of all actions performed during this stage of the migration, and any manual steps that you need to perform.

7. Validate the Stage 2 component migration output.
   a) Load the targetVersion/migrated_flow.json into a NiFi 2 instance and check the flow.
   b) Review the targetVersion/activity_log.json file.

   In this example, you can see the following manual-change-request entry.

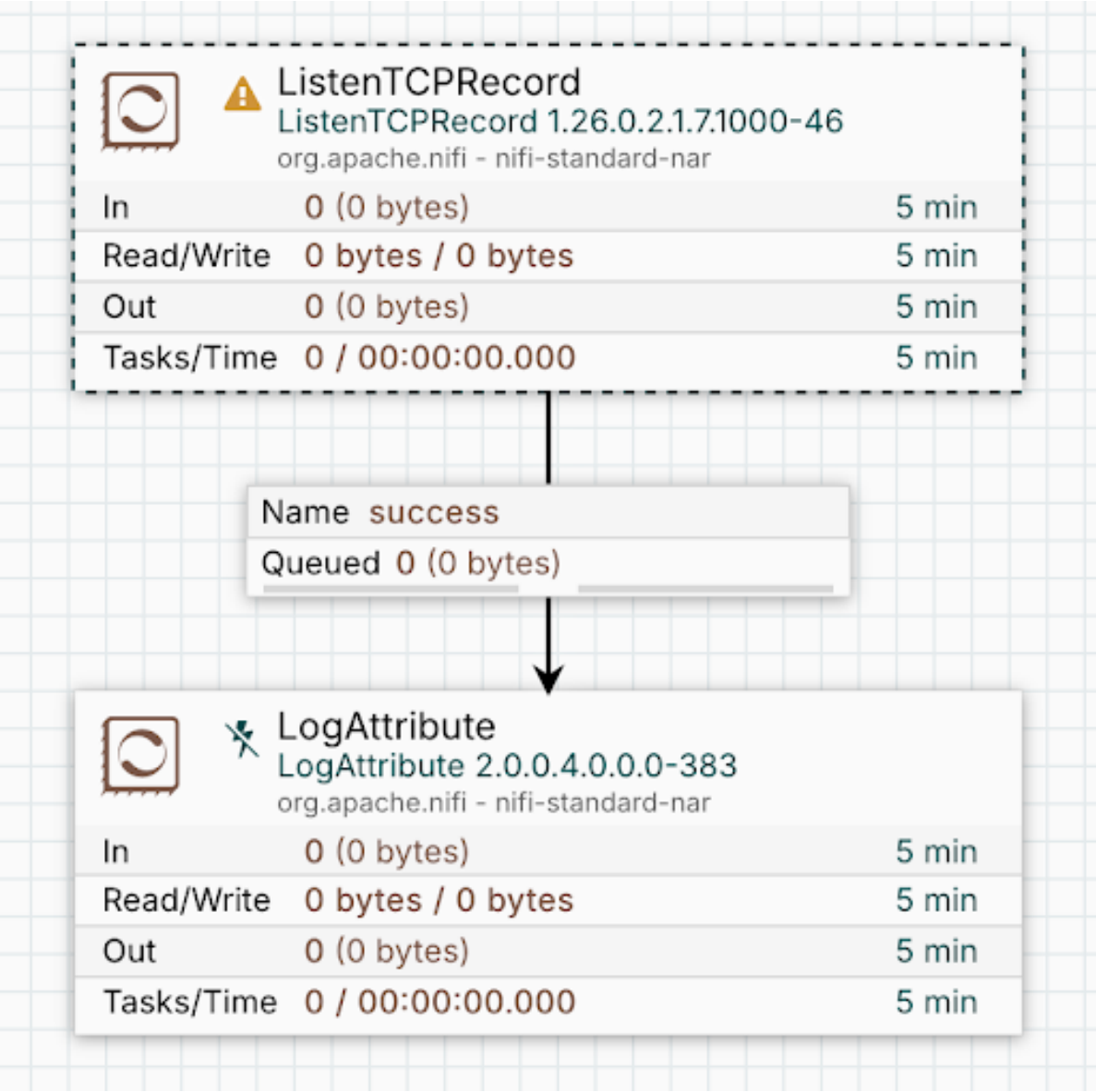
```
{
 "sequence" : 3,
 "type" : "manual-change-request",
 "subject" : "b428aad6-0194-1000-d73d-9f2332a59f04",
 "message" : "New relationship [original] has been added, it has to be
 connected to a downstream processor or terminated.",
 "context" : {
   "rule" : "6501a693-b524-4d1a-b11e-69747c2651e8"
}
```

To identify the processor affected by these changes, search for the value of the “subject” element in NiFi’s search box. It refers to the PutElasticsearchJson processor. You will find an unbound "original" relationship that needs to be connected to a downstream processor or terminated. Make the necessary modifications manually. Once completed, this change request is resolved.

You can see another manual-change-request entry in the Activity Log.

```
{
 "sequence" : 6,
 "type" : "manual-change-request",
 "subject" : "b41966ad-0194-1000-a08d-a92489457356",
 "message" : "The component has been deprecated. It is suggested to r
eplace it with a combination of [org.apache.nifi.processors.standard.Lis
tenTCP] and [org.apache.nifi.processors.standard.ConvertRecord] processo
rs",
 "context" : {
   "rule" : "36227b60-75f0-40dc-8caf-a2ec577aa54c"
}
```

To identify the processor affected by these changes, search for the value of the “subject” element in NiFi’s search box. It refers to the ListenTCPRecord processor. Open the TCP Listener process group. You can see that the processor is marked with dashed borders and its version number still shows that of the NiFi 1 instance. It means that it is a "ghost processor", not available in NiFi 2. The log message suggests replacing it with a combination of ListenTCP and ConvertRecord processors.



Instantiate and configure the ListenTCP and ConvertRecord processors, connect them appropriately, and then remove the ghost processor.

Besides these changes there are no further manual-change-requests or manual-validation-requests in the activity log.

8. Wrap up the migration.
   a) Review the completed flow to ensure it meets your use cases and that all necessary changes have been made.
   b) Enable the controller services associated with the flow in the NiFi 2 instance.
   c) Once the controller services are enabled, start the processors in your migrated NiFi 2 flow.

Using migrate-all with flow.json as input

Learn how to use the migrate-all command to migrate templates, variables, and components in a single operation using a flow.json file as input.

This command combines the functionality of the migrate-templates, migrate-variables, and migrate-components commands for ease and convenience. While migrate-all simplifies the migration process, for larger flows, it is usually better to run the three migration commands separately. You may also want to migrate process groups individually to ensure activity logs and manual validation remain manageable.

The migrate-all command can be useful in the following scenarios:

- The flow you want to migrate is simple and of manageable size.
- The flow is large and complex, and you need an initial high-level overview of the migration outcome before performing a step-by-step migration. Running migrate-all provides a preview of the final migrated flow. The Activity Log shows you the actions taken during migration and helps assess the required manual intervention by listing manual-change-requests and manual-validation-requests.

Migrating templates, variables, and components together using migrate-all with flow.json as input

Before you begin

1. Offload all flowfiles from NiFi.
2. Stop all processors and disable all controller services in NiFi.
3. Stop NiFi.
4. Copy the flow.json.gz file from NiFi's conf directory to the Migration Tool's input folder (/etc/migration-tool-input).
5. Unzip the file to obtain flow.json.

Procedure

1. Run the following command to migrate the flow.

```
bin/migration.sh nifi migrate-all \
-i /etc/migration-tool-input/flow.json \
-od /etc/migration-tool-output
```

   This generates the following output:

```
sourceVersion
### NiFi_Flow_b4175f57-0194-1000-8470-9251a24519b4
#   ### elastic_template.json
#   ### elastic_template.xml
### activity_log.json
### migrated_flow.json

targetVersion
### NiFi_Flow_b4175f57-0194-1000-8470-9251a24519b4
#   ### elastic_template.json
### activity_log.json
### migrated_flow.json
```

2. Review the migration output.
   - The sourceVersion directory contains the NiFi 1-compatible version of the flow, including the exported template and its process group counterpart, along with the Activity Log.
   - The targetVersion directory contains the NiFi 2-compatible version of the flow and its corresponding Activity Log.
3. Address issues in NiFi 1 (if needed).

   If sourceVersion/activity_log.json contains manual-change-requests, follow these steps:

   a) Load sourceVersion/migrated_flow.json into your NiFi 1 instance.
   b) Apply the required manual changes.
   c) Run the migrate-all command again using the manually modified flow.
4. Validate the migrated flow.
   a) Rename targetVersion/migrated_flow.json to flow.json.
   b) Compress it into flow.json.gz.
   c) Load flow.json.gz into your NiFi 2 instance to inspect the migrated flow.

d) Open targetVersion/activity_log.json and review any manual-change-requests and manual-validation-requests.
e) Apply the required manual modifications to the flow.

Migrating a process group using migrate-all with flow.json as input

For complex flows, you can migrate one process group and on one stage at a time. See below for an example workflow.

Procedure

1. Migrate the TCP Listener process group using the following command.

```
bin/migration.sh nifi migrate-all \
-i /etc/migration-tool-input/flow.json \
-od /etc/migration-tool-output -pgid b41940d7-0194-1000-42fc-458834630567
 \
--sourceCompatibleOutput
```

2. Use the output sourceVersion/migrated_flow.json (the result of the previous command moved to the input folder) as the input for migrating the **Elastic** process group.

```
bin/migration.sh nifi migrate-all \
-i /etc/migration-tool-input/flow.json \
-od /etc/migration-tool-output -pgid b42881c7-0194-1000-3cdf-1bd453a0ed0f
 \
--sourceCompatibleOutput
```

3. Use the output sourceVersion/migrated_flow.json (the result of the previous command moved to the input folder) as the input for migrating the root process group and completing the Stage 2 migration.

```
bin/migration.sh nifi migrate-all \
-i /etc/migration-tool-input/flow.json \
-od /etc/migration-tool-output
```

This produces a NiFi 2-compatible version of the flow.



Important:

Always ensure that the input for a migration command is a NiFi 1 flow and not a NiFi 2 flow.

Handling file formats missing sensitive property values

When migrating NiFi flows, the handling of sensitive properties depends on the file format being used.

- flow.json and flow.json.gz files contain all property values, including sensitive ones, in encrypted form.
- Other formats, such as flow definitions or templates, do not include sensitive property values.

As a result, the Migration Tool has limited capabilities when handling sensitive properties. It cannot determine whether sensitive property values were originally set, and therefore cannot guarantee a complete migration of those values.



Important: Always review the migration results carefully when working with file formats that omit sensitive property values.

Example: GetHDFS processor

For the GetHDFS processor, if a controller service is not used for Kerberos credentials, configuration in NiFi 1 requires:

- Kerberos Principal (not sensitive)
- Exactly one of Kerberos Keytab (not sensitive) or Kerberos Password (sensitive)

Kerberos Password is classified as a sensitive property. In NiFi 2, these properties are replaced by a controller service that manages Kerberos credentials.

Migration Tool behavior

- When working with flow.json:
  - The Migration Tool identifies which properties are populated and migrates them to the appropriate controller service.
  - It also validates the configuration and generates alerts through Manual Change Requests or Manual Validation Requests if inconsistencies exist.
  - For example, it can detect misconfigurations and alert you if all three properties are filled or a Kerberos Principal is set without a Kerberos Keytab or Kerberos Password is. These issues are listed as Manual Change Requests or Manual Validation Requests in the Activity Log.
- When working with flow definitions:

  Because the sensitive Kerberos Password value is missing, the Migration Tool cannot determine if Kerberos Password was originally populated or not. In this case, the original configuration is inferred based on the values of the other properties.

  - If both Kerberos Principal and Kerberos Keytab are filled, Kerberos Password is assumed empty, and the Migration Tool creates a KerberosKeytabUserService.
  - If only Kerberos Principal is populated, Kerberos Password is assumed originally filled, and the Migration Tool creates a KerberosPasswordUserService with an empty Kerberos Pas sword property.
  - If Kerberos Principal is empty, the Migration Tool assumes that Kerberos Password was not set. This typically indicates that either a controller service was already used for Kerberos credentials or Kerberos was not involved at all.

Example: GetAzureQueueStorage processor

For the GetAzureQueueStorage processor, credential configuration requires:

- Storage Account Name (sensitive)
- Exactly one of Storage Account Key or SAS Token (both sensitive)

Since all three properties are sensitive and the sensitive values are missing, the Migration Tool cannot reliably determine which property was populated in the original configuration. In such cases, final configuration decisions must be performed manually.

The Migration Tool relies on the original configuration being valid. If it is not, the tool may make incorrect assumptions, apply unintended changes, and may not be able to alert you about them. The Migration Tool provides minimal alerting when handling missing sensitive properties to prevent excessive noise in the Activity Log.

Post-migration requirements

- Verify all sensitive property configurations for file formats that do not include values.
- Ensure that all required sensitive values are populated in NiFi following import.

Sensitive parameters

Even in file types where sensitive property values are not available, parameter references are maintained, although their actual values may be empty. Similarly, in a complete flow.json,it is valid for parameter values to be empty (whether they are sensitive or not).

This means that the presence of a parameter does not guarantee that it resolves to a non-empty value. NiFi evaluates a component’s property configuration based on the value it resolves to, not merely the presence of a parameter reference.

For example, in the GetAzureQueueStorage processor, it is valid (although somewhat unusual) to configure all three properties (Storage Account Name, Storage Account Key, and SAS Token) with parameter references, even if only Storage Account Name and exactly one of Storage Account Key and SAS Token resolve to a non-empty value.

As a result, the Migration Tool generally does not distinguish between property values defined directly or through parameters when making decisions and assumptions about sensitive properties.

The exception occurs when the Migration Tool cannot make meaningful inferences from the configuration alone. In such cases, it can attempt to infer values based on the presence or absence of parameter references. For example:

- If the GetAzureQueueStorage processor contains parameter references for Storage Account Name and Stor age Account Key, but SAS Token is empty, the Migration Tool can infer that the referenced parameters have values.
- If all three properties contain parameter references, or none of them do, the Migration Tool defers the resolution to you.

Migrating a flow using flow.json.gz as input

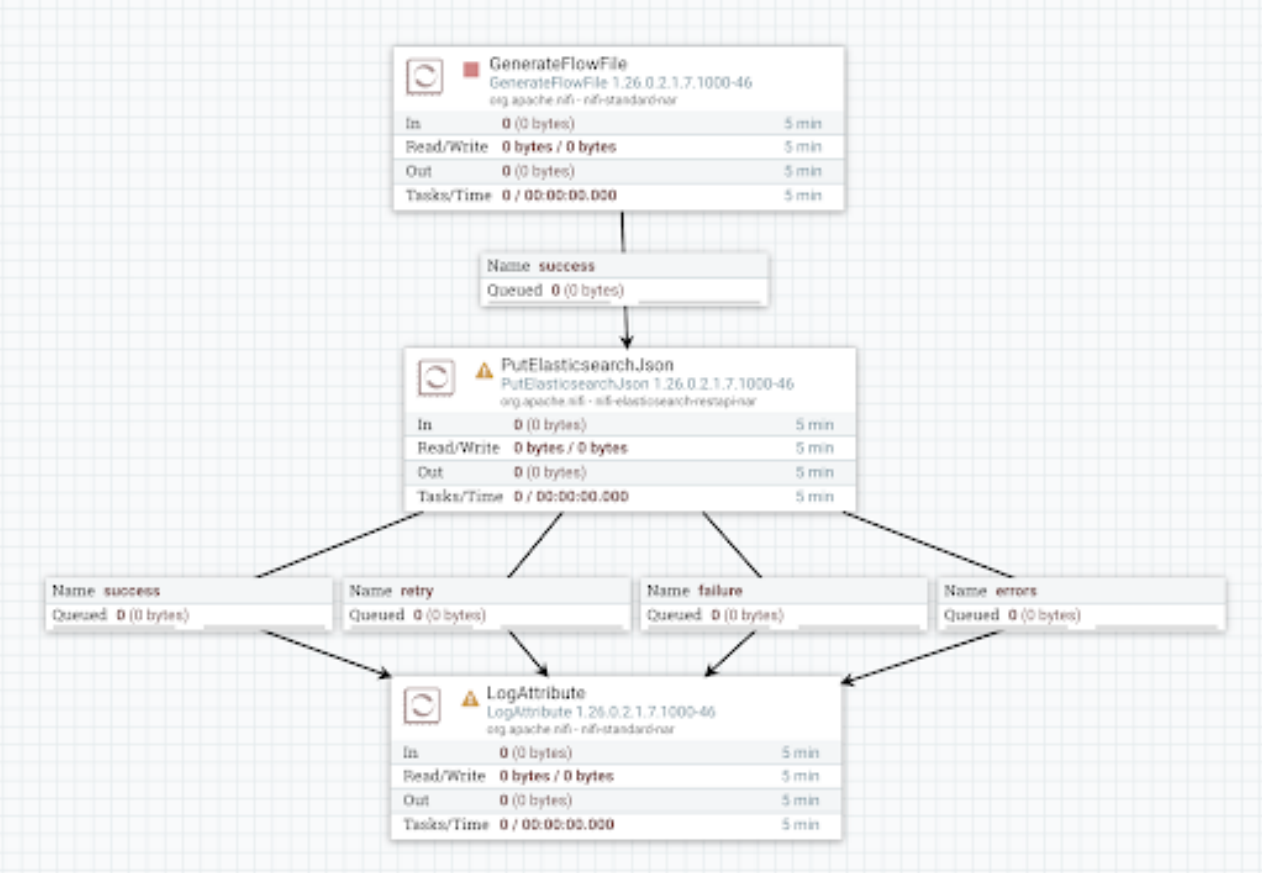
This section provides step-by-step examples of how to run different migrations with the Cloudera Flow Migration Tool using a **flow.json.gz** file as input.

Migrating templates using flow.json.gz as input

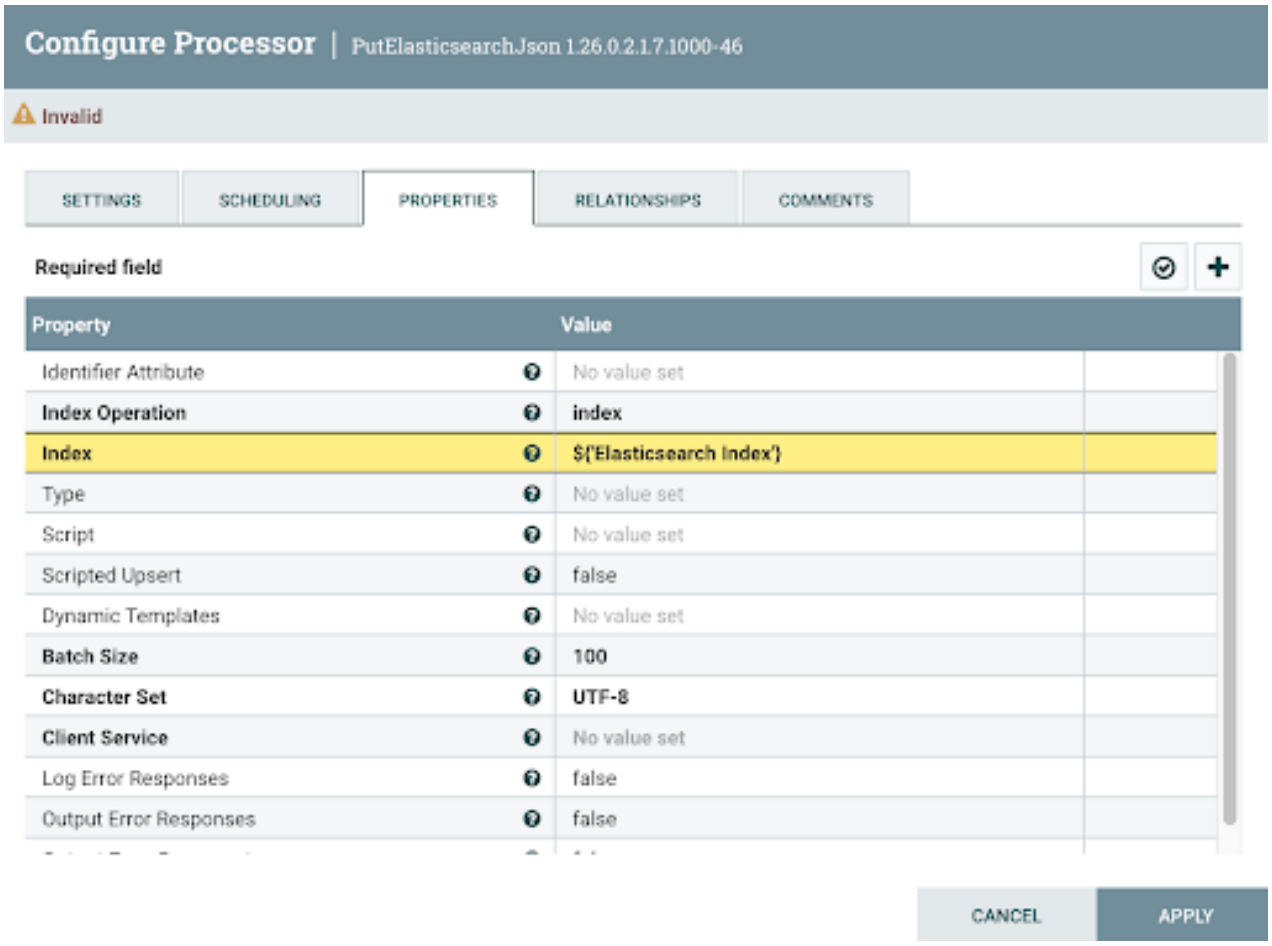
Learn how to use the Cloudera Flow Migration Tool to extract and transform templates for compatibility with NiFi 2 using a flow.json.gz file as input. NiFi 2 does not support templates, so this step is required to ensure compatibility with the target version.

Example flow for migrating a template

You have a flow that has a template named **elastic_template**, which was created from a process group called **elastic**. This process group contains the following simple flow:



The process group has a variable, **Elasticsearch Index**, referenced in the index property of the PutElasticsearchJson processor.



Your goal is to migrate this template to NiFi 2 used in Cloudera Flow Management 4.12.0.

Before you begin

1. Stop all processors and disable all controller services in NiFi.
2. Stop NiFi.
3. Copy the flow.json.gz file from NiFi's conf directory to the Migration Tool's input folder (/etc/migration-tool-input).

Procedure

1. Run Stage 1 template migration using the following command.

```
bin/migration.sh nifi migrate-templates \
-i /etc/migration-tool-input/flow.json.gz \
-od /etc/migration-tool-output/templates \
-sco
```

This generates a sourceVersion folder that contains the output files of the migration.

```
templates
### sourceVersion
    ### NiFi_Flow_ad3a86a6-0194-1000-78d9-6374298d9a0c
    #   ### elastic_template.json
    #   ### elastic_template.xml
```

```
    ### activity_log.json
```

The folder name NiFi_Flow_ad3a86a6-0194-1000-78d9-6374298d9a0c indicates that the exported template belongs to the root process group, which is named NiFi Flow in NiFi. The unique ID for this group is ad3a86a6-0194-1000-78d9-6374298d9a0c.

elastic_template.json

- Flow definition containing the contents of the original template
- Equivalent to manually converting a template to a flow definition in NiFi 1 by:
  a. Instantiating the template on the canvas.
  b. Right-clicking the process group.
  c. Selecting Download flow definition without external services.
- Modified by the Migration Tool to ensure compatibility:
  - Variables (not supported in NiFi 2) converted into parameters
  - Parameter context created to hold the new parameters
  - Processors updated to reference parameters instead of variables
- This is still a NiFi 1 flow definition, so do not load this file directly into NiFi 2. Use an empty NiFi 1 instance to review the modification made by the tool. The flow will appear unchanged, but variables will be replaced with parameters of the same name, which are now referenced by the processor instead.
- Changes in syntax:

  Variables were previously referenced using ${}, whereas parameters are now referenced using #{}.
  - **NiFi 1 variables**: ${variable_name}
  - **NiFi 2 parameters**: #{parameter_name}

elastic_template.xml

- Original template exported from NiFi 1
- Identical to downloading it from the Templates menu in NiFi 1
- Compatible only with NiFi 1 as NiFi 2 does not support templates and cannot parse XML template files

activity_log.json

- Log of all actions performed during this stage of the migration.
- The following were changes made during the template migration:
  - A new parameter context was created with a new parameter in it.
  - The parameter context was assigned to the process group.
  - The PutElasticsearchJson processor was updated to reference the new parameter.
- Components are referenced by a unique ID, not by name.

2. Validate the Stage 1 template migration output.

a) Check the new flow definition in a NiFi 1 instance to verify that the flow matches your expectations.

1. Start your NiFi 1 instance.
2. Create a new process group.
3. Load elastic_template.json to NiFi 1.
4. Confirm that variables were correctly converted to parameters.

b) Review activity_log.json and address any manual-change-requests or manual-validation-requests.

c) If there are manual-change-requests or manual-validation-requests to handle, follow these steps:

1. Start a NiFi 1 instance.
2. Load the sourceVersion/<template_name>.json flow definition by creating a new process group and uploading the JSON file.

3. Make the modifications on the NiFi canvas, indicated by the manual-change-requests or manual-validation-requests in the sourceVersion/activity_log.json.
4. Stop the NiFi instance.
5. Fetch the flow.json.gz file from the NiFi conf directory.
6. Perform variable migration and then component migration on this flow.json.gz file. For instructions, see the example scenario for variable and component migration.

d) If no additional changes are required, proceed to Stage 2.

3. Run full template migration (Stage 1 and 2), using the following command.

```
bin/migration.sh nifi migrate-templates \
-i /etc/migration-tool-input/flow.json.gz \
-od /etc/migration-tool-output/templates
```

The contents of the previously generated sourceVersion folder will be overwritten, but this is not a concern as the tool generates the same sourceVersion output as before. Additionally, a targetVersion directory is created, containing the output files of the Stage 2 part of the migration.

```
targetVersion
### NiFi_Flow_ad3a86a6-0194-1000-78d9-6374298d9a0c
#   ### elastic_template.json
### activity_log.json
```

elastic_template.json

- NiFi 2-compatible flow definition
- Contains the contents of the original template converted into an exported process group.
- This version is compatible with NiFi 2, but no longer supports NiFi 1.



Note:

An elastic_template.xml is not generated as NiFi 2 does not support XML templates.

activity_log.json

- List of all actions performed during this stage of the migration.

4. Validate the Stage 2 template migration output.

a) Check the new flow definition in a NiFi 2 instance to verify that the flow matches your expectations.

1. Start your NiFi 2 instance.
2. Create a new process group.
3. Load elastic_template.json into a NiFi 2 instance.

   The new process group will be called **elastic_template** and will contain another process group named **elastic**, matching the name of the process group that was converted into a template in NiFi 1 before you started the migration.
4. Verify parameter replacements and processor updates.

b) Review activity_log.json and address any manual-change-requests or manual-validation-requests.

In this case, you can see the following manual-change-request:

```
{
 "sequence" : 4,
 "type" : "manual-change-request",
 "subject" : "a1468d69-3f30-3f83-a7c1-91dad09890f7",
 "message" : "New relationship [original] has been added, it has to be
 connected to a downstream processor or terminated.",
 "context" : {
   "rule" : "6501a693-b524-4d1a-b11e-69747c2651e8"
}
```

1. Open the targetVersion/elastic_template.json file and search for the "subject" ID, a1468d69-3f30-3f83-a7c1-91dad09890f7. This ID refers to the PutElasticsearchJson processor.
2. Go to the NiFi 2 canvas and check the processor. You will find that it has an unbound “original” relationship that needs to be connected to a downstream component.
3. Make the required change manually on the canvas.
4. Once done, export the process group. This exported process group is now a fully NiFi 2-compatible version of the original template.
5. Save the file.

Results

You have finished the template migration process.

Migrating variables using flow.json.gz as input

Learn how to use the Cloudera Flow Migration Tool to convert variables to parameters and parameter contexts using a flow.json.gz file as input.

Example flow for migrating variables

The following NiFi flow is used to demonstrate both variable and component migration. It illustrates how variables are used within process groups and how they are referenced by individual components in the NiFi flow.

The flow consists of two process groups:

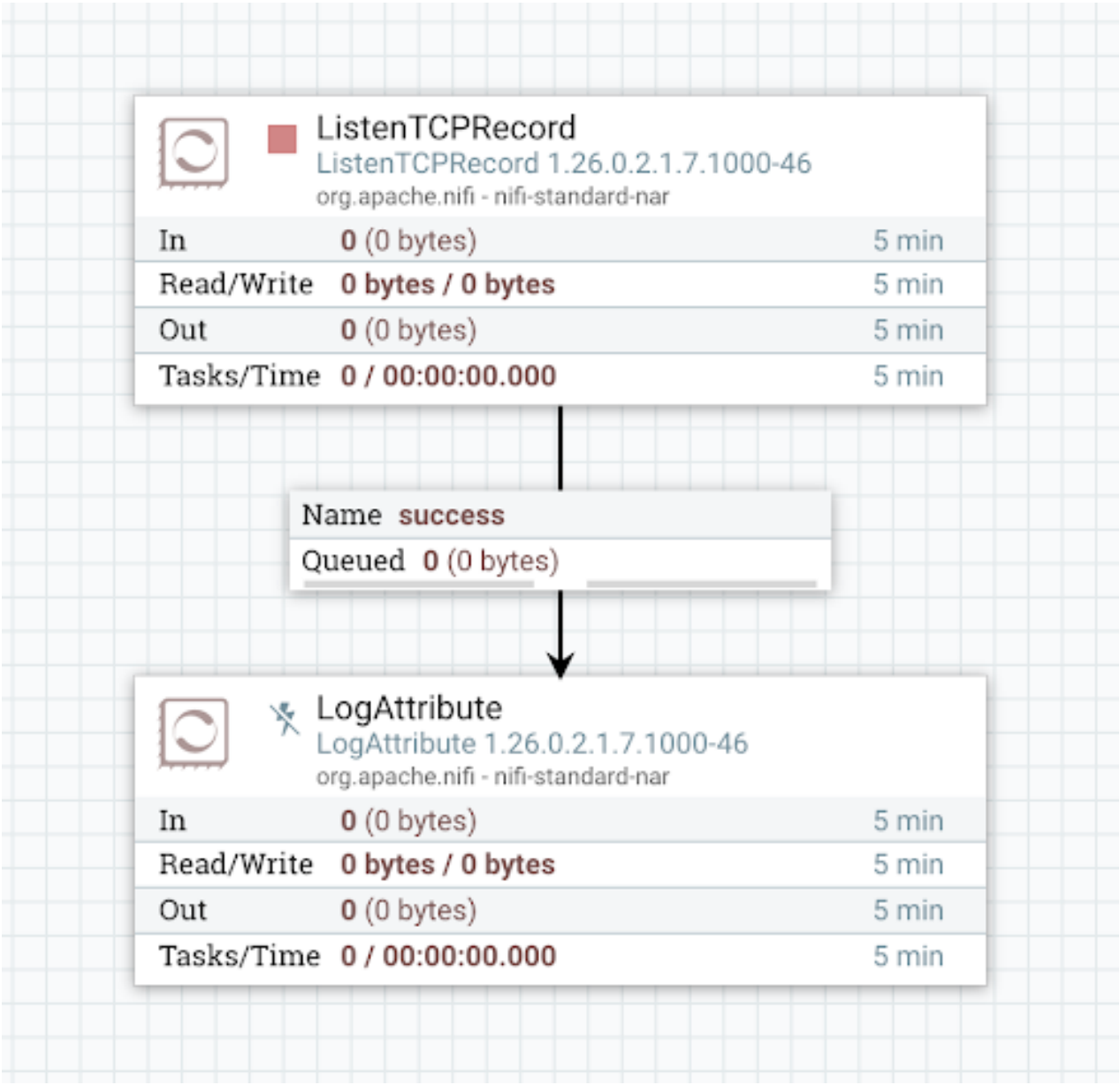
- **TCP Listener** (ID: b41940d7-0194-1000-42fc-458834630567)
- **Elastic** (ID: b42881c7-0194-1000-3cdf-1bd453a0ed0f)

At the root level, the flow is structured as follows:

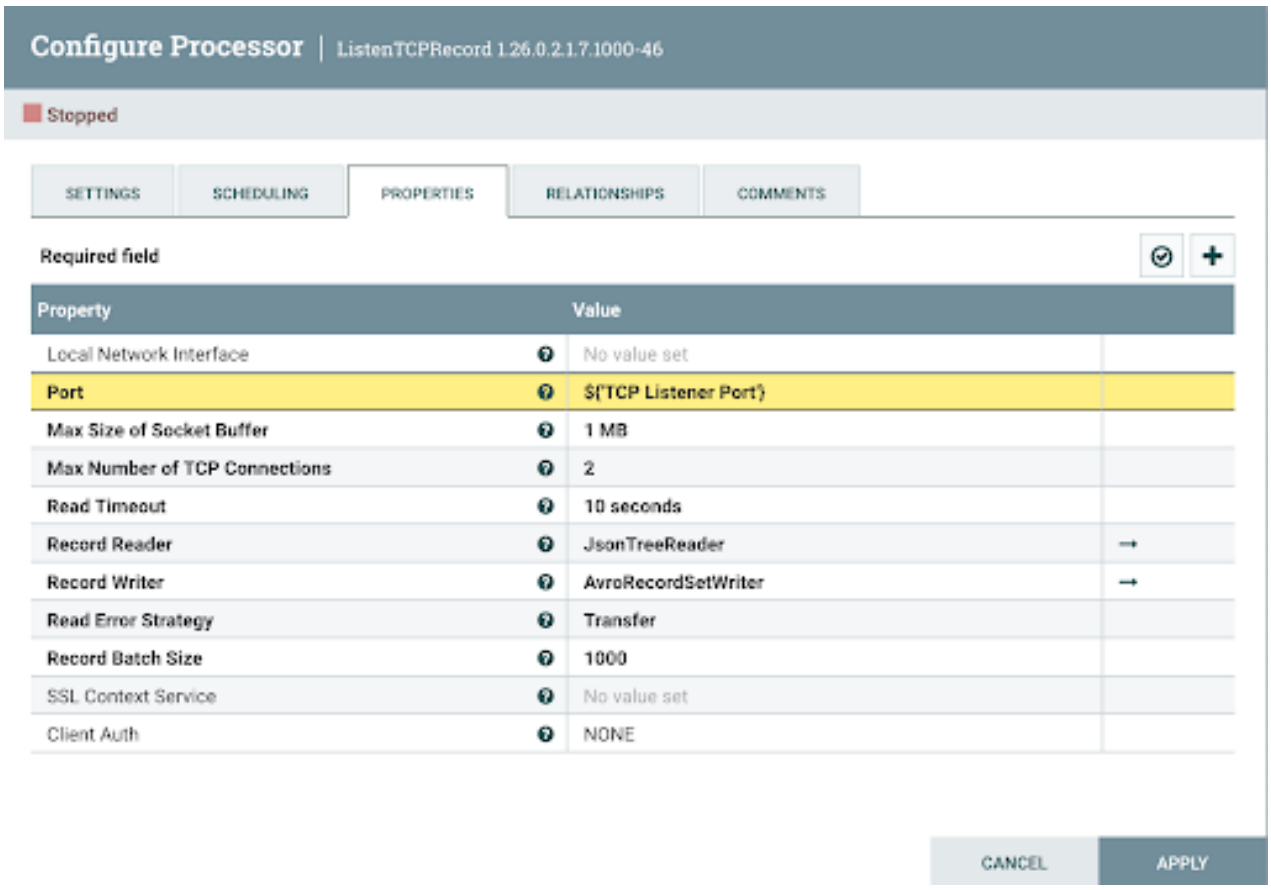


TCP Listener Process Group

This process group contains the following simple flow:

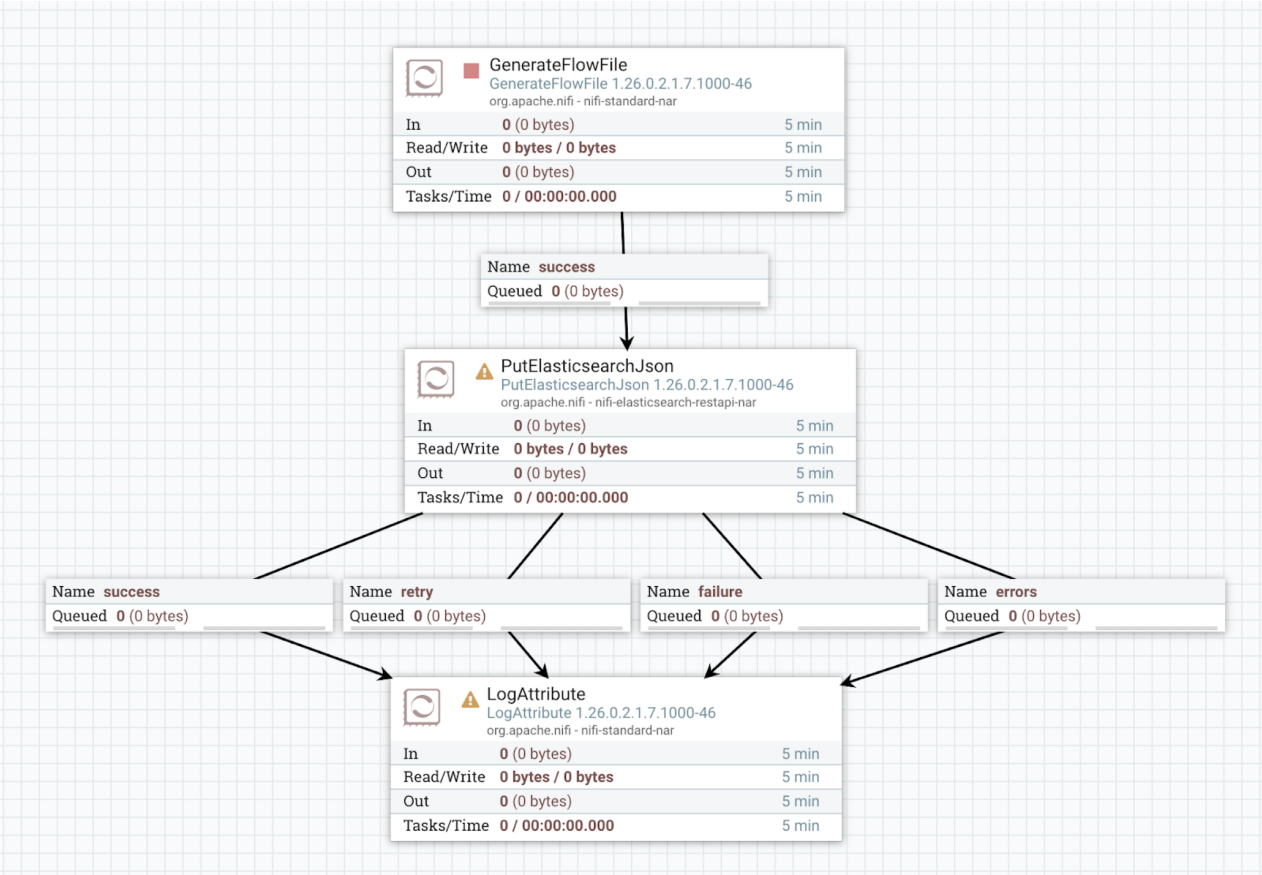


The process group defines a variable called **TCP Listener Port**, which is referenced by the ListenTCPRecord processor.

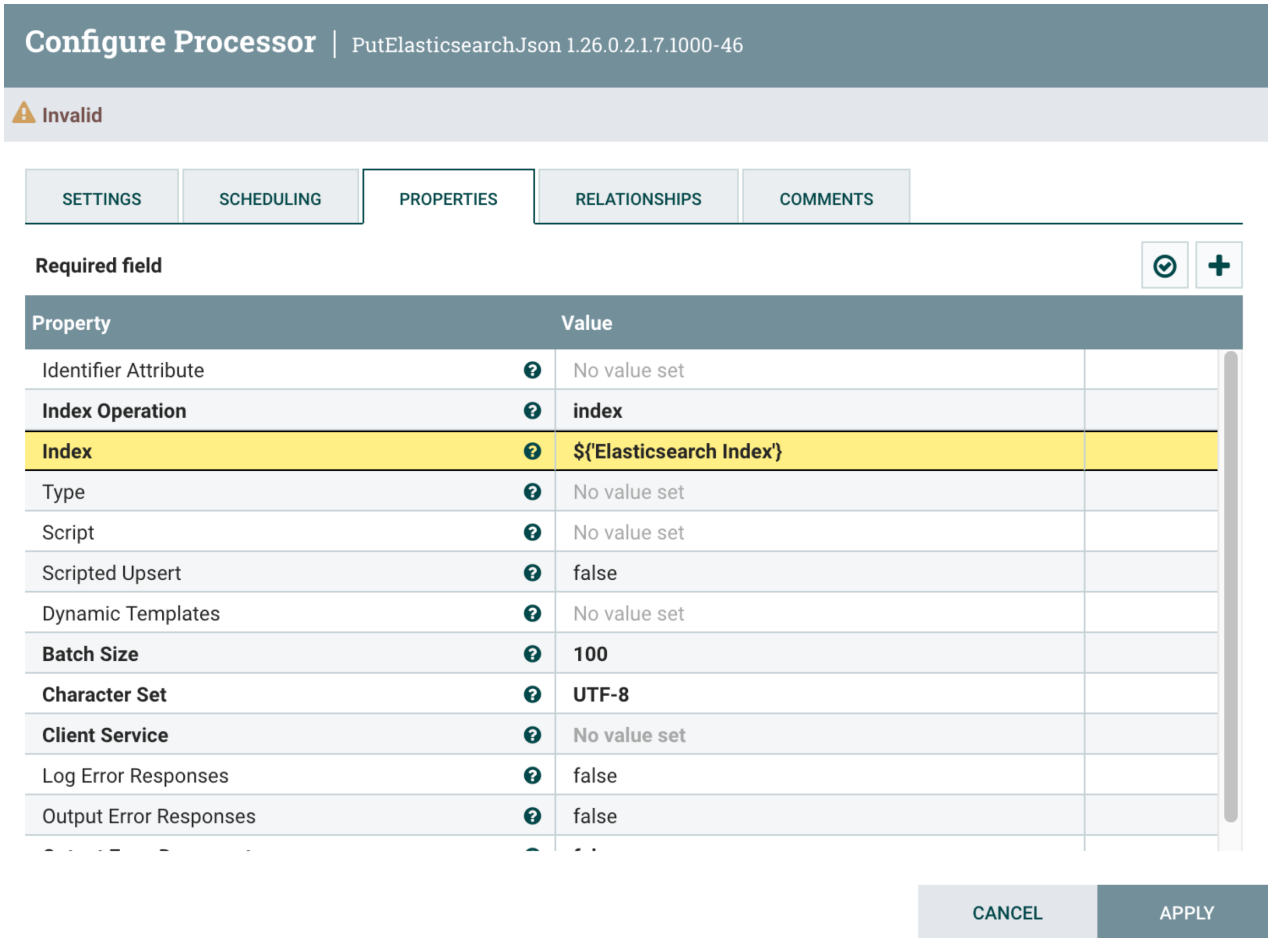


Elastic Process Group

This process group also contains a simple flow:



The process group defines a variable called **Elasticsearch Index**. This variable is referenced in the PutElasticsearchJson processor’s "Index"property.



Your goal is to migrate the variables used in Cloudera Flow Management 2.1.7.4000 powered by NiFi 1 to parameters used in Cloudera Flow Management 4.12.0 powered by NiFi 2.

Note: NiFi 2 supports only parameters, not variables, so variable migration is required to ensure compatibility with the target version.

The example guides you through variable migration one process group at a time, simplifying the process and maintaining a clear activity log. While this example flow is simple, the step-by-step approach shows how this method improves clarity for more complex migrations. In real-world scenarios, you have to define a migration strategy based on your flow's structure, individual process group migration may not always be necessary. Migrating a parent process group automatically applies the changes recursively to its descendants.

Before you begin

1. Stop all processors and disable all controller services in NiFi.
2. Stop NiFi.
3. Copy the flow.json.gz file from NiFi's conf directory to the Migration Tool's input folder (/etc/migration-tool-input).

Procedure

1. Run Stage 1 variable migration on the TCP Listener process group, using the following command.

```
bin/migration.sh nifi migrate-variables \
-i /etc/migration-tool-input/flow.json.gz \
-od /etc/migration-tool-output/variables \
-pgid <process_group_id> \
```

```
--sourceCompatibleOutput
```

This generates a sourceVersion folder that contains the output files of the Stage 1 migration.

```
variables
### sourceVersion
    ### activity_log.json
    ### migrated_flow.json.gz
```

activity_log.json

- Log of all actions performed during this stage of the migration.
- The following were changes made during the template migration:
  - A new parameter context was created with a new parameter called **TCP Listener Port**, which replaces the corresponding variable.
  - The **TCP Listener Port** variable was removed.

migrated_flow.json.gz

- A modified NiFi 1 flow, which is not compatible with NiFi 2 yet.
- It contains everything the original flow did, except the **TCP Listener** process group now references a parameter instead of the removed variable.

2. Validate the Stage 1 variable migration output for the **TCP Listener** process group.
   a) Load the migrated_flow.json.gz into a NiFi 1 instance and check the flow.
      1. Ensure NiFi is not running.
      2. Go to NiFi's conf directory and back up the flow.json.gz and flow.xml.gz files.
      3. Delete the original flow.json.gz and flow.xml.gz files.
      4. Rename the migrated_flow.json.gz to flow.json.gz.
      5. Start NiFi and check the flow.
   b) Review the activity_log.json file and check for any manual-change-requests or manual-validation-requests. If none are present, proceed to running Stage 1 variable migration on the **Elastic** process group.
      1. If manual changes are necessary, load the migrated_flow.json.gz to NiFi, update and validate it.
      2. Once the flow is validated and meets expectations, continue with the next step using the new flow.json.gz file.
3. Run Stage 1 variable migration on the **Elastic** process group.
   a) Move migrated_flow.json.gz from Step 2 into the input folder (/etc/migration-tool-input) and rename it to flow.json.gz for clarity.
   b) Make a backup of the output folder (/etc/migration-tool-output/variables) before running the next migration step.

      Note: This step is crucial because the same output folder is used as in the previous migration, and its contents will be overwritten. To retain a record of changes made at each stage, it is better to create a backup.
   c) Run Stage 1 variable migration on the **Elastic** process group using the following command.

```
bin/migration.sh nifi migrate-variables \
-i /etc/migration-tool-input/flow.json.gz \
-od /etc/migration-tool-output/variables \
-pgid b42881c7-0194-1000-3cdf-1bd453a0ed0f \
--sourceCompatibleOutput
```

This generates a sourceVersion folder that contains the two output files of the Stage 1 migration.

activity_log.json

- Log of all actions performed during this stage of the migration.

- The following were changes made during the template migration:
  - A new parameter context was created with a new parameter called **Elasticsearch Index**, which replaces the corresponding variable of the same name.
  - The new parameter is referenced from the PutElasticsearchJson processor.
  - The **Elasticsearch Index** variable was removed.

migrated_flow.json.gz

- A modified NiFi 1 flow, which is not compatible with NiFi 2 yet.
- It contains everything the original flow did, but the **Elastic** process group now references a parameter instead of the removed variable.

4. Validate the Stage 1 variable migration output for the **Elastic** process group.
   a) Load the migrated_flow.json.gz into a NiFi 1 instance and check the flow.
   b) Review the activity_log.json file and check for any manual-change-requests or manual-validation-requests. If none are present, proceed to the next step.
   c) If manual changes are necessary, update the migrated_flow.json.gz on the NiFi canvas after loading it. Once the flow is validated and meets expectations, continue with the next step.

   At this stage, both process groups no longer contain variables and use parameters instead. If the flow meets your expectations, you can either run a full variable migration to validate your flow in NiFi 2 or proceed with migrating the components.
5. Run full variable migration (Stage 1 and 2) by using the following command.

```
bin/migration.sh nifi migrate-variables \
-i /etc/migration-tool-input/flow.json.gz \
-od /etc/migration-tool-output/variables
```

   The contents of the previously generated sourceVersion folder will be overwritten, but this is not a concern as the tool generates the same sourceVersion output as before.

   Additionally, a targetVersion directory is created, containing the output files of the Stage 2 part of the migration.

   migrated_flow.json.gz

   - NiFi 2-compatible flow in terms of variables
   - You can load it into NiFi 2 and validate it

   activity_log.json

   - List of all actions performed during this stage of the migration.

   Stage 2 migration allows you to validate the result of the variable migration process in NiFi 2. However, component migration should not be performed on the targetVersion/migrated_flow.json.gz because the input of component migration must be a NiFi 1 flow.
6. Proceed with component migration using the migrated_flow.json.gz output from step 3.

Migrating components using flow.json.gz as input

Learn how to use the Cloudera Flow Migration Tool to migrate components to parameters and parameter contexts using a flow.json.gz file as input.

Example flow for migrating components

The following NiFi flow is used to demonstrate both variable and component migration.

It consists of two process groups:

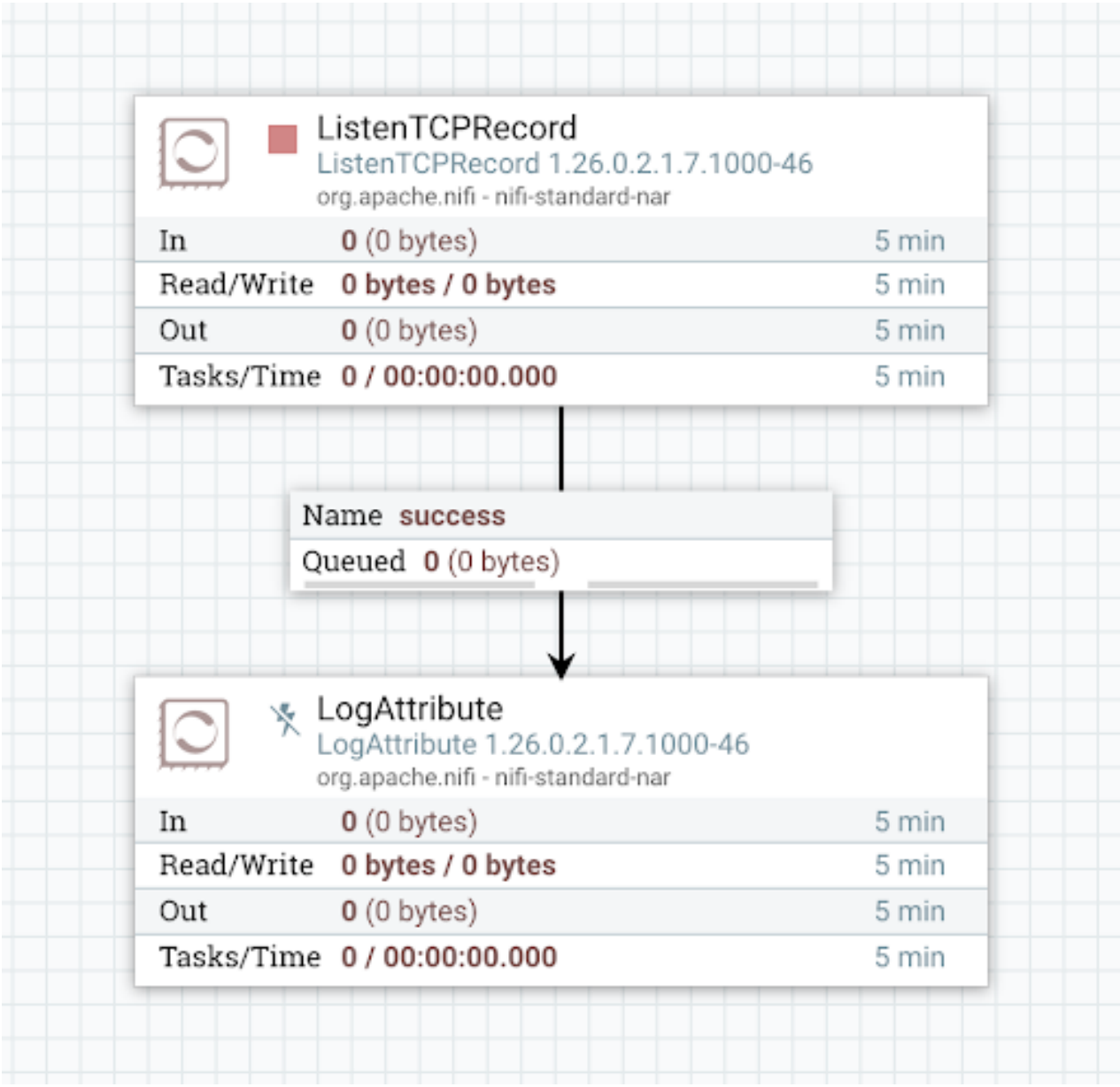
- **TCP Listener** (ID: b41940d7-0194-1000-42fc-458834630567)
- **Elastic** (ID: b42881c7-0194-1000-3cdf-1bd453a0ed0f)

At the root level, the flow is structured as follows:

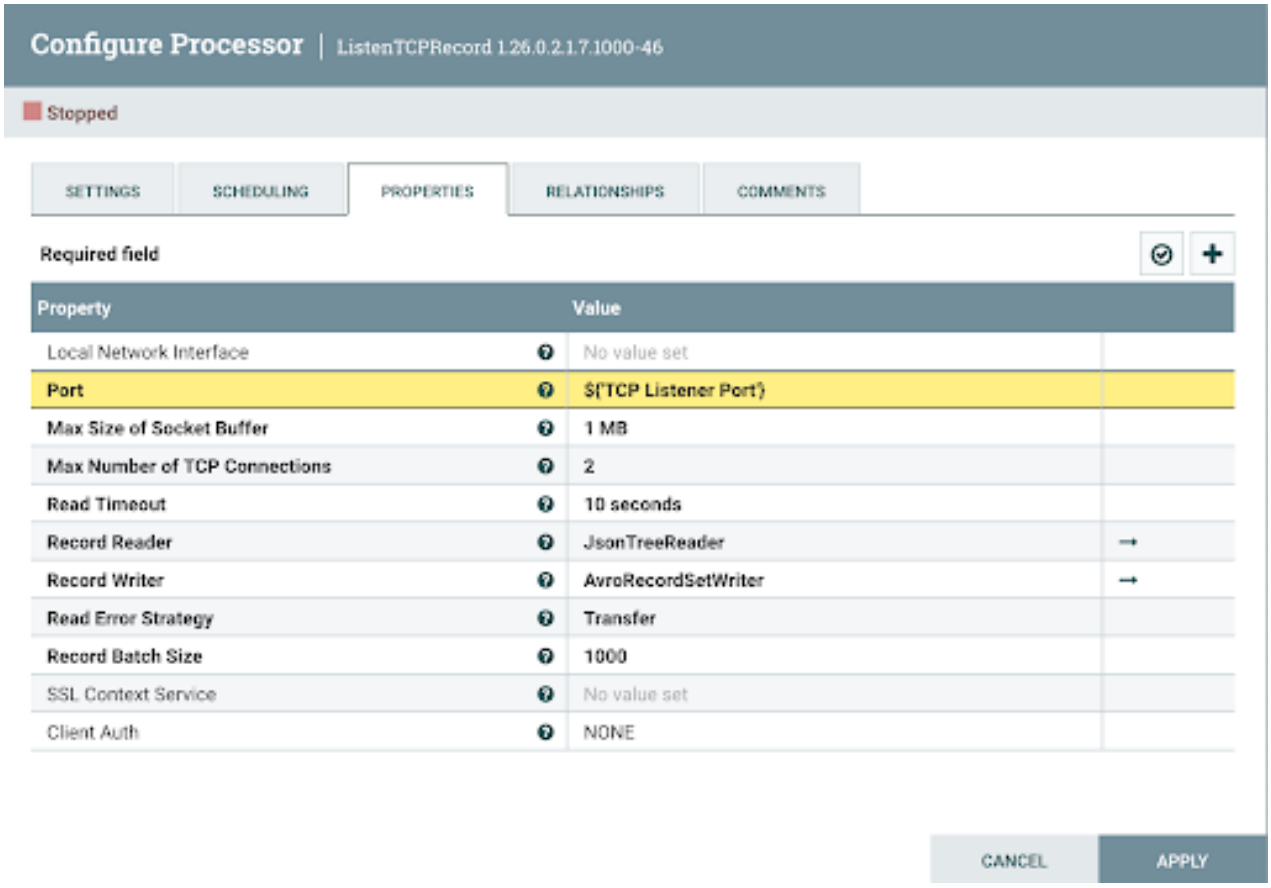


TCP Listener Process Group

This process group contains the following simple flow:

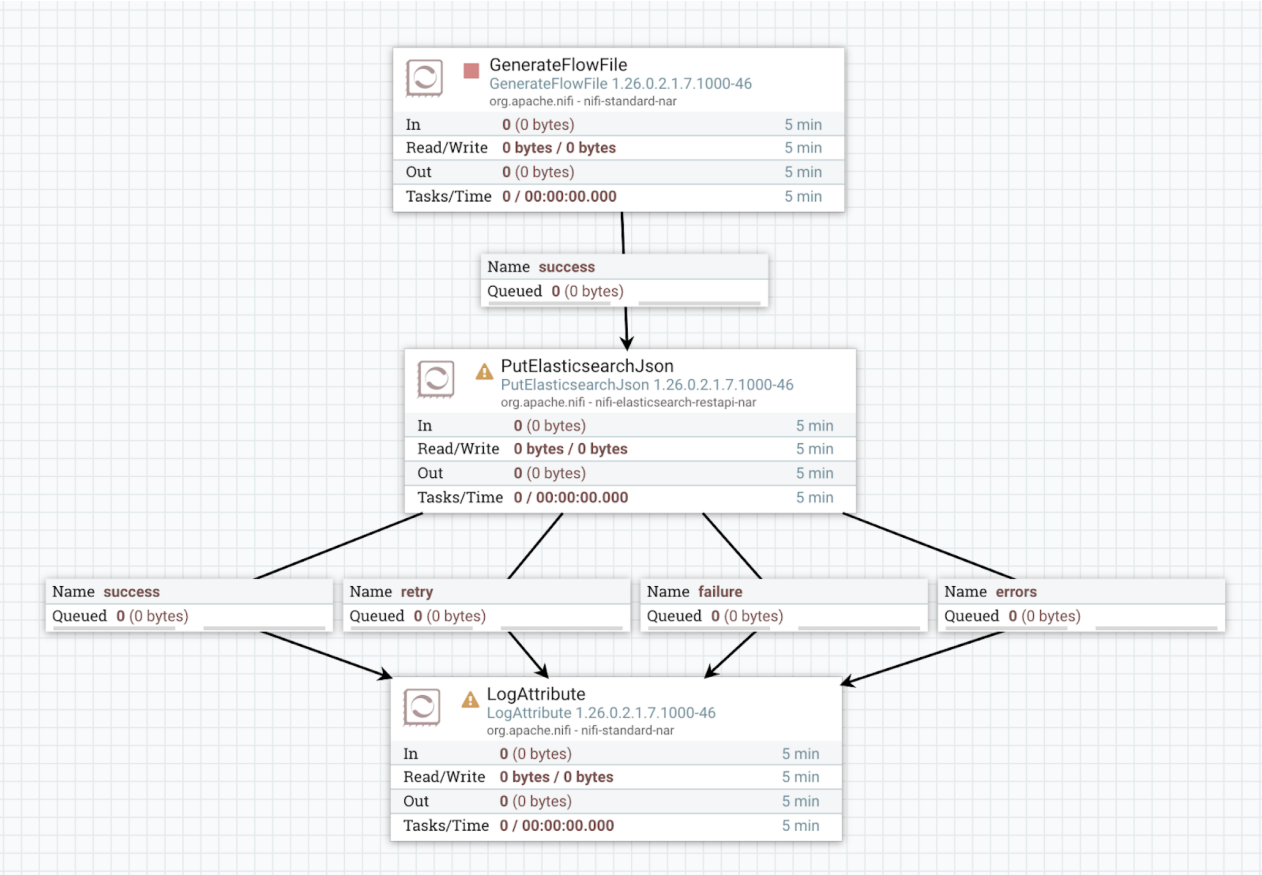


The process group defines a variable called **TCP Listener Port**, which is referenced by the ListenTCPRecord processor.

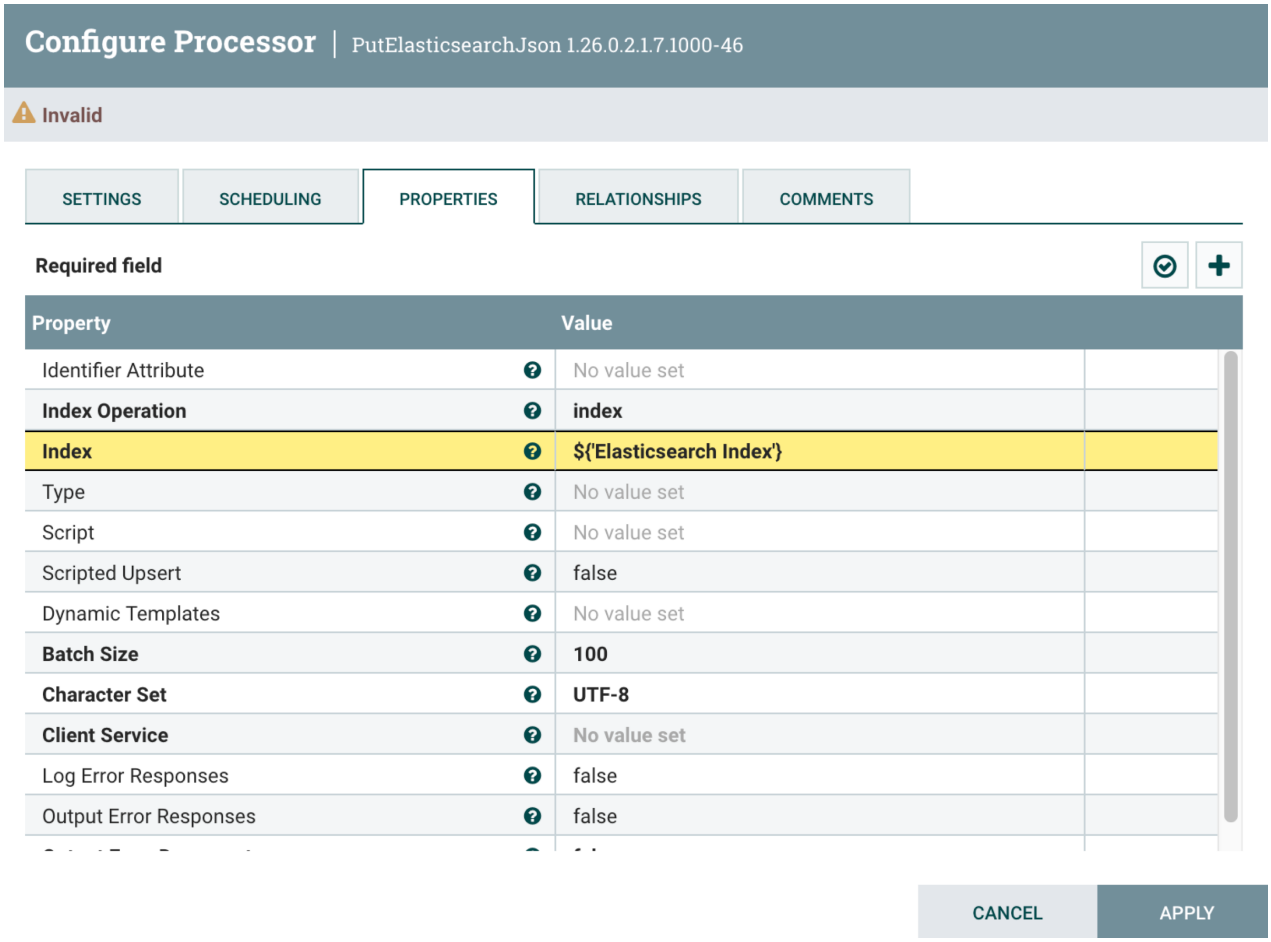


Elastic Process Group

This process group also contains a simple flow:



The process group defines a variable called **Elasticsearch Index**. This variable is referenced in the PutElasticsearchJson processor's "Index" property.



Your goal is to migrate the components used in Cloudera Flow Management 2.1.7.4000 powered by NiFi 1 to NiFi 2-compatible components used in Cloudera Flow Management 4.12.0 powered by NiFi 2.

The example guides you through component migration one process group at a time, simplifying the process and maintaining a clear activity log. While this example flow is simple, the step-by-step approach shows how this method improves clarity for more complex migrations. In real-world scenarios, you have to define a migration strategy based on your flow's structure, individual process group migration may not always be necessary. Migrating a parent process group automatically applies the changes recursively to its descendants.

Procedure

1. Move the migrated_flow.json.gz file, the output of variable migration from Step 3, into the input folder (/etc/migration-tool-input) and rename it to flow.json.gz for clarity.

   This is a NiFi-1 compatible flow that no longer contains variables.
2. Run Stage 1 component migration on the **TCP Listener** process group using the following command.

   ```
   bin/migration.sh nifi migrate-components \
   -i /etc/migration-tool-input/flow.json.gz \
   -od /etc/migration-tool-output/components \
   -pgid b41940d7-0194-1000-42fc-458834630567 \
   --sourceCompatibleOutput
   ```

   This generates a sourceVersion folder that contains the output files of the Stage 1 migration.

   ```
   components
   ### sourceVersion
       ### activity_log.json
   ```

```
    ### migrated_flow.json.gz
```

activity_log.json

- The log describes all the actions that were performed for this stage of the process group migration.
- Log of all actions performed during this stage of the process group migration.

migrated_flow.json.gz

- A modified NiFi 1 flow, which is not compatible with NiFi 2 yet.
- It contains everything the original flow did, except the **TCP Listener** process group was modified as described in the activity log.

3. Validate the Stage 1 component migration output for the **TCP Listener** process group.
   a) Load the migrated_flow.json.gz into a NiFi 1 instance and check the flow.
   b) Review the activity_log.json file and check for any manual-change-requests or manual-validation-requests. If none are present, proceed to the next step.

      In this example, you can see the following information in the activity log:

      ```
      {
       "sequence" : 2,
       "type" : "change-info",
       "subject" : "b41966ad-0194-1000-a08d-a92489457356",
       "message" : "Component [org.apache.nifi.processors.standard.ListenTCPRecord] has been deprecated (NIFI-13509)",
       "context" : {
         "rule" : "36227b60-75f0-40dc-8caf-a2ec577aa54c"
      }
      ```

      Search for the value of the “subject” element in NiFi’s search box.

      You are directed to the ListenTCPRecord processor. No manual modifications are needed at this stage. However, it is important to note that the ListenTCPRecord processor is deprecated and is not available in NiFi 2. Once the full component migration (Stage 1 and Stage 2) is complete, instructions will be provided on how to handle this deprecation.
   c) If manual changes are necessary, update the migrated_flow.json.gz on the NiFi canvas after loading it. Once the flow is validated and meets expectations, continue with the next step.
4. Run Stage 1 component migration on the **Elastic** process group.
   a) Move migrated_flow.json.gz from Step 2 into the input folder (/etc/migration-tool-input) and rename it to flow.json.gz for clarity.
   b) Make a backup of the output folder (/etc/migration-tool-output/components) before running the next migration step.

      Note:

      This step is crucial because the same output folder is used as in the previous migration, and its contents will be overwritten. To retain a record of changes made at each stage, it is better to create a backup.
   c) Run Stage 1 component migration on the **Elastic** process group using the following command.

      ```
      bin/migration.sh nifi migrate-components \
      -i /etc/migration-tool-input/flow.json.gz \
      -od /etc/migration-tool-output/components \
      -pgid b42881c7-0194-1000-3cdf-1bd453a0ed0f \
      --sourceCompatibleOutput
      ```

      This generates a sourceVersion folder that contains the two output files of the Stage 1 migration.

      activity_log.json

      - Log of all actions performed during this stage of the migration.

migrated_flow.json.gz

- A modified NiFi 1 flow, which is not compatible with NiFi 2 yet.
- It contains everything the original flow did, except the Elastic process group was modified with the actions described in the activity log.

5. Validate the Stage 1 component migration output for the **Elastic** process group.
   a) Load the migrated_flow.json.gz into a NiFi 1 instance and check the flow.
   b) Review the activity_log.json file.

      It contains a change-info entry that informs you of changes to a NiFi 2 processor, identified by the subject ID.

      ```
      {
       "sequence" : 2,
       "type" : "change-info",
       "subject" : "b428aad6-0194-1000-d73d-9f2332a59f04",
       "message" : "Property [Max JSON Field String Length] has been added (N
      IFI-12343); Property [put-es-json-not_found-is-error] has been renamed t
      o [put-es-not_found-is-error] (NIFI-12255); Property [put-es-json-error-
      documents] has been removed (NIFI-12255); Relationships [success] has be
      en renamed to [original]; Relationship [successful] has been added (NIFI
      -12255);",
       "context" : {
         "rule" : "6501a693-b524-4d1a-b11e-69747c2651e8"
      }
      ```

      To identify the processor affected by these changes, search for the value of the “subject” element in NiFi’s search box.

      You are directed to the PutElasticsearchJson processor. No manual modifications are needed at this stage. However, it is important to note that changes will be applied to the PutElasticsearchJson processor during Stage 2 of the migration. Once the full component migration (Stage 1 and Stage 2) is complete, instructions will be provided on how to handle this change.

   At this stage, you have completed Stage 1 of both variable and component migration for the process groups in your flow. After reviewing the logs, you confirmed that no manual changes were needed. You can proceed with a full component migration using the migrated_flow.json.gz from Step 4.

6. Run full component migration (Stage 1 and 2).
   a) Move migrated_flow.json.gz from Step 4 into the input folder (/etc/migration-tool-input) and rename it to flow.json.gz for clarity.
   b) Make a backup of the output folder (/etc/migration-tool-output/components) before running the next migration step.

      Note:

      This step is crucial because the same output folder is used as in the previous migration, and its contents will be overwritten. To retain a record of changes made at each stage, it is better to create a backup.

   c) Run the full component migration using the following command.

      ```
      bin/migration.sh nifi migrate-components \
      -i /etc/migration-tool-input/flow.json.gz \
      -od /etc/migration-tool-output/components
      ```

      This generates the following output:

      ```
      components
      ### sourceVersion
      #   ### activity_log.json
      #   ### migrated_flow.json.gz
      ### targetVersion
          ### activity_log.json
      ```

```
    ### migrated_flow.json.gz
```

The contents of the previously generated sourceVersion folder will be overwritten. The contents represent the NiFi 1-compatible version of the migrated flow. Since the root process group only contained the two process groups on which you already performed Stage 1 component migration, the activity log will only include the same entries as those in Steps 2 and 4.

Additionally, a targetVersion directory is created, containing the output files of the Stage 2 part of the migration:

migrated_flow.json.gz

- NiFi 2-compatible flow in terms of variables
- You can load it into NiFi 2 and validate it

activity_log.json

- List of all actions performed during this stage of the migration, and any manual steps that you need to perform.

7. Validate the Stage 2 component migration output.
   a) Load the targetVersion/migrated_flow.json.gz into a NiFi 2 instance and check the flow.
   b) Review the targetVersion/activity_log.json file.

   In this example, you can see the following manual-change-request entry.

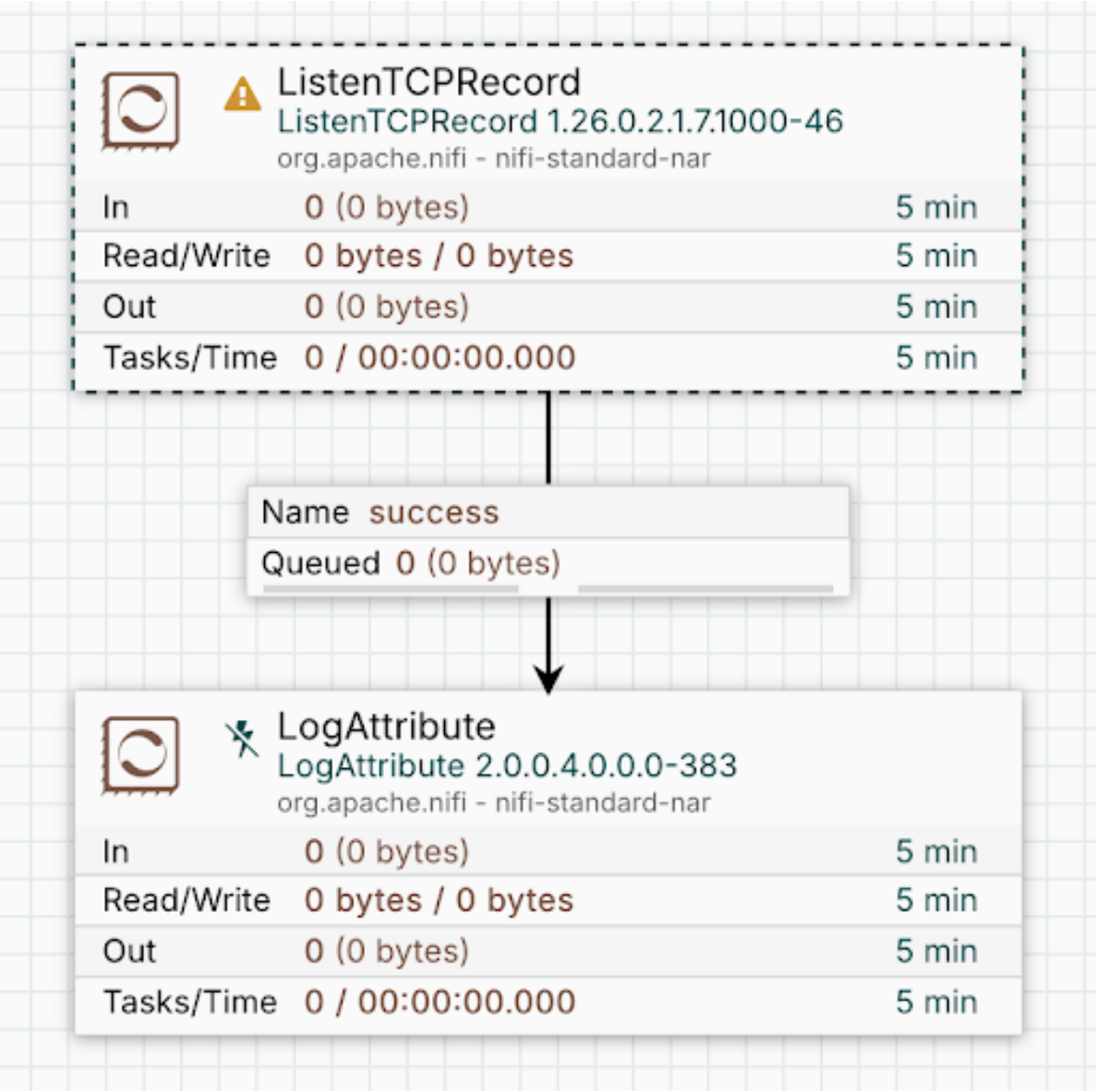
```
{
 "sequence" : 3,
 "type" : "manual-change-request",
 "subject" : "b428aad6-0194-1000-d73d-9f2332a59f04",
 "message" : "New relationship [original] has been added, it has to be
 connected to a downstream processor or terminated.",
 "context" : {
   "rule" : "6501a693-b524-4d1a-b11e-69747c2651e8"
}
```

To identify the processor affected by these changes, search for the value of the “subject” element in NiFi’s search box. It refers to the PutElasticsearchJson processor. You will find an unbound "original" relationship that needs to be connected to a downstream processor or terminated. Make the necessary modifications manually. Once completed, this change request is resolved.

You can see another manual-change-request entry in the Activity Log.

```
{
 "sequence" : 6,
 "type" : "manual-change-request",
 "subject" : "b41966ad-0194-1000-a08d-a92489457356",
 "message" : "The component has been deprecated. It is suggested to r
eplace it with a combination of [org.apache.nifi.processors.standard.Lis
tenTCP] and [org.apache.nifi.processors.standard.ConvertRecord] processo
rs",
 "context" : {
   "rule" : "36227b60-75f0-40dc-8caf-a2ec577aa54c"
}
```

To identify the processor affected by these changes, search for the value of the “subject” element in NiFi’s search box. It refers to the ListenTCPRecord processor. Open the TCP Listener process group. You can see that the processor is marked with dashed borders and its version number still shows that of the NiFi 1 instance. It means that it is a "ghost processor", not available in NiFi 2. The log message suggests replacing it with a combination of ListenTCP and ConvertRecord processors.



Instantiate and configure the ListenTCP and ConvertRecord processors, connect them appropriately, and then remove the ghost processor.

Besides these changes there are no further manual-change-requests or manual-validation-requests in the activity log.

8. Wrap up the migration.
   a) Review the completed flow to ensure it meets your use cases and that all necessary changes have been made.
   b) Enable the controller services associated with the flow in the NiFi 2 instance.
   c) Once the controller services are enabled, start the processors in your migrated NiFi 2 flow.

Using migrate-all with flow.json.gz as input

Learn how to use the migrate-all command to migrate templates, variables, and components in a single operation using a flow.json.gz file as input.

This command combines the functionality of the migrate-templates, migrate-variables, and migrate-components commands for ease and convenience. While migrate-all simplifies the migration process, for larger flows, it is usually better to run the three migration commands separately. You may also want to migrate process groups individually to ensure activity logs and manual validation remain manageable.

The migrate-all command can be useful in the following scenarios:

- The flow you want to migrate is simple and of manageable size.
- The flow is large and complex, and you need an initial high-level overview of the migration outcome before performing a step-by-step migration. Running migrate-all provides a preview of the final migrated flow. The Activity Log shows you the actions taken during migration and helps assess the required manual intervention by listing manual-change-requests and manual-validation-requests.

Migrating templates, variables, and components together using migrate-all with flow.json.gz as input

Before you begin

1. Offload all flowfiles from NiFi.
2. Stop all processors and disable all controller services in NiFi.
3. Stop NiFi.
4. Copy the flow.json.gz file from NiFi's conf directory to the Migration Tool's input folder (/etc/migration-tool-input).

Procedure

1. Run the following command to migrate the flow.

```
bin/migration.sh nifi migrate-all \
-i /etc/migration-tool-input/flow.json.gz \
-od /etc/migration-tool-output
```

   This generates the following output:

```
sourceVersion
### NiFi_Flow_b4175f57-0194-1000-8470-9251a24519b4
#   ### elastic_template.json
#   ### elastic_template.xml
### activity_log.json
### migrated_flow.json.gz
targetVersion
### NiFi_Flow_b4175f57-0194-1000-8470-9251a24519b4
#   ### elastic_template.json
### activity_log.json
### migrated_flow.json.gz
```

2. Review the migration output.
   - The sourceVersion directory contains the NiFi 1-compatible version of the flow, including the exported template and its process group counterpart, along with the Activity Log.
   - The targetVersion directory contains the NiFi 2-compatible version of the flow and its corresponding Activity Log.
3. Address issues in NiFi 1 (if needed).

   If sourceVersion/activity_log.json contains manual-change-requests, follow these steps:

   a) Load sourceVersion/migrated_flow.json.gz into your NiFi 1 instance.
   b) Apply the required manual changes.
   c) Run the migrate-all command again using the manually modified flow.
4. Validate the migrated flow.
   a) Rename targetVersion/migrated_flow.json.gz to flow.json.gz.
   b) Load flow.json.gz into your NiFi 2 instance to inspect the migrated flow.
   c) Open targetVersion/activity_log.json and review any manual-change-requests and manual-validation-requests.
   d) Apply the required manual modifications to the flow.

Migrating a process group using migrate-all with flow.json.gz as input

For complex flows, you can migrate one process group and on one stage at a time. See below for an example workflow.

Procedure

1. Migrate the TCP Listener process group using the following command.

```
bin/migration.sh nifi migrate-all \
-i /etc/migration-tool-input/flow.json.gz \
-od /etc/migration-tool-output -pgid b41940d7-0194-1000-42fc-458834630567
 \
--sourceCompatibleOutput
```

2. Use the output sourceVersion/migrated_flow.json.gz (the result of the previous command moved to the input folder) as the input for migrating the **Elastic** process group.

```
bin/migration.sh nifi migrate-all \
-i /etc/migration-tool-input/flow.json.gz \
-od /etc/migration-tool-output -pgid b42881c7-0194-1000-3cdf-1bd453a0ed0f
 \
--sourceCompatibleOutput
```

3. Use the output sourceVersion/migrated_flow.json.gz (the result of the previous command moved to the input folder) as the input for migrating the root process group and completing the Stage 2 migration.

```
bin/migration.sh nifi migrate-all \
-i /etc/migration-tool-input/flow.json.gz \
-od /etc/migration-tool-output
```

This produces a NiFi 2-compatible version of the flow.

Important:

Always ensure that the input for a migration command is a NiFi 1 flow, not a NiFi 2 flow.

Handling file formats missing sensitive property values

When migrating NiFi flows, the handling of sensitive properties depends on the file format being used.

- flow.json and flow.json.gz files contain all property values, including sensitive ones, in encrypted form.
- Other formats, such as flow definitions or templates, do not include sensitive property values.

As a result, the Migration Tool has limited capabilities when handling sensitive properties. It cannot determine whether sensitive property values were originally set, and therefore cannot guarantee a complete migration of those values.

Important: Always review the migration results carefully when working with file formats that omit sensitive property values.

Example: GetHDFS processor

For the GetHDFS processor, if a controller service is not used for Kerberos credentials, configuration in NiFi 1 requires:

- Kerberos Principal (not sensitive)
- Exactly one of Kerberos Keytab (not sensitive) or Kerberos Password (sensitive)

Kerberos Password is classified as a sensitive property. In NiFi 2, these properties are replaced by a controller service that manages Kerberos credentials.

Migration Tool behavior

- When working with flow.json:

    - The Migration Tool identifies which properties are populated and migrates them to the appropriate controller service.
    - It also validates the configuration and generates alerts through Manual Change Requests or Manual Validation Requests if inconsistencies exist.
    - For example, it can detect misconfigurations and alert you if all three properties are filled or a Kerberos Principal is set without a Kerberos Keytab or Kerberos Password is. These issues are listed as Manual Change Requests or Manual Validation Requests in the Activity Log.
- When working with flow definitions:

  Because the sensitive Kerberos Password value is missing, the Migration Tool cannot determine if Kerberos Password was originally populated or not. In this case, the original configuration is inferred based on the values of the other properties.

    - If both Kerberos Principal and Kerberos Keytab are filled, Kerberos Password is assumed empty, and the Migration Tool creates a KerberosKeytabUserService.
    - If only Kerberos Principal is populated, Kerberos Password is assumed originally filled, and the Migration Tool creates a KerberosPasswordUserService with an empty Kerberos Pas sword property.
    - If Kerberos Principal is empty, the Migration Tool assumes that Kerberos Password was not set. This typically indicates that either a controller service was already used for Kerberos credentials or Kerberos was not involved at all.

Example: GetAzureQueueStorage processor

For the GetAzureQueueStorage processor, credential configuration requires:

- Storage Account Name (sensitive)
- Exactly one of Storage Account Key or SAS Token (both sensitive)

Since all three properties are sensitive and the sensitive values are missing, the Migration Tool cannot reliably determine which property was populated in the original configuration. In such cases, final configuration decisions must be performed manually.

The Migration Tool relies on the original configuration being valid. If it is not, the tool may make incorrect assumptions, apply unintended changes, and may not be able to alert you about them. The Migration Tool provides minimal alerting when handling missing sensitive properties to prevent excessive noise in the Activity Log.

Post-migration requirements

- Verify all sensitive property configurations for file formats that do not include values.
- Ensure that all required sensitive values are populated in NiFi following import.

Sensitive parameters

Even in file types where sensitive property values are not available, parameter references are maintained, although their actual values may be empty. Similarly, in a complete flow.json,it is valid for parameter values to be empty (whether they are sensitive or not).

This means that the presence of a parameter does not guarantee that it resolves to a non-empty value. NiFi evaluates a component’s property configuration based on the value it resolves to, not merely the presence of a parameter reference.

For example, in the GetAzureQueueStorage processor, it is valid (although somewhat unusual) to configure all three properties (Storage Account Name, Storage Account Key, and SAS Token) with parameter references, even if only Storage Account Name and exactly one of Storage Account Key and SAS Token resolve to a non-empty value.

As a result, the Migration Tool generally does not distinguish between property values defined directly or through parameters when making decisions and assumptions about sensitive properties.

The exception occurs when the Migration Tool cannot make meaningful inferences from the configuration alone. In such cases, it can attempt to infer values based on the presence or absence of parameter references. For example:

- If the GetAzureQueueStorage processor contains parameter references for Storage Account Name and Storage Account Key, but SAS Token is empty, the Migration Tool can infer that the referenced parameters have values.
- If all three properties contain parameter references, or none of them do, the Migration Tool defers the resolution to you.

Migrating a flow using flow definition JSON as input

This section provides step-by-step examples of how to run different migrations with the Cloudera Flow Migration Tool using a **flow definition JSON** file as input.

The flow definition JSON file can be generated by right-clicking a process group in NiFi and selecting Download definition.

Migrating templates using flow definition JSON as input

Learn how to use the Cloudera Flow Migration Tool to extract and transform templates for compatibility with NiFi 2 using a flow.json file as input. NiFi 2 does not support templates, so this step is required to ensure compatibility with the target version.

Flow definition JSON files do not contain any templates so this step can be skipped. However, if you run the template migration command using a flow definition JSON file, it will produce the following output:

```
sourceVersion
### activity_log.json
targetVersion
### activity_log.json
```

The generated activity_log.json file will contain only minimal information, such as the start and end timestamps of the migration process. No actual templates will be listed, as they are not present in the input.

Migrating variables using flow definition JSON as input

Learn how to use the Cloudera Flow Migration Tool to convert variables to parameters and parameter contexts from flow definition JSON files.

Example flows for migrating variables

The following NiFi flow demonstrates both variable and component migration. It shows how variables are used within process groups and how they are referenced by individual components.

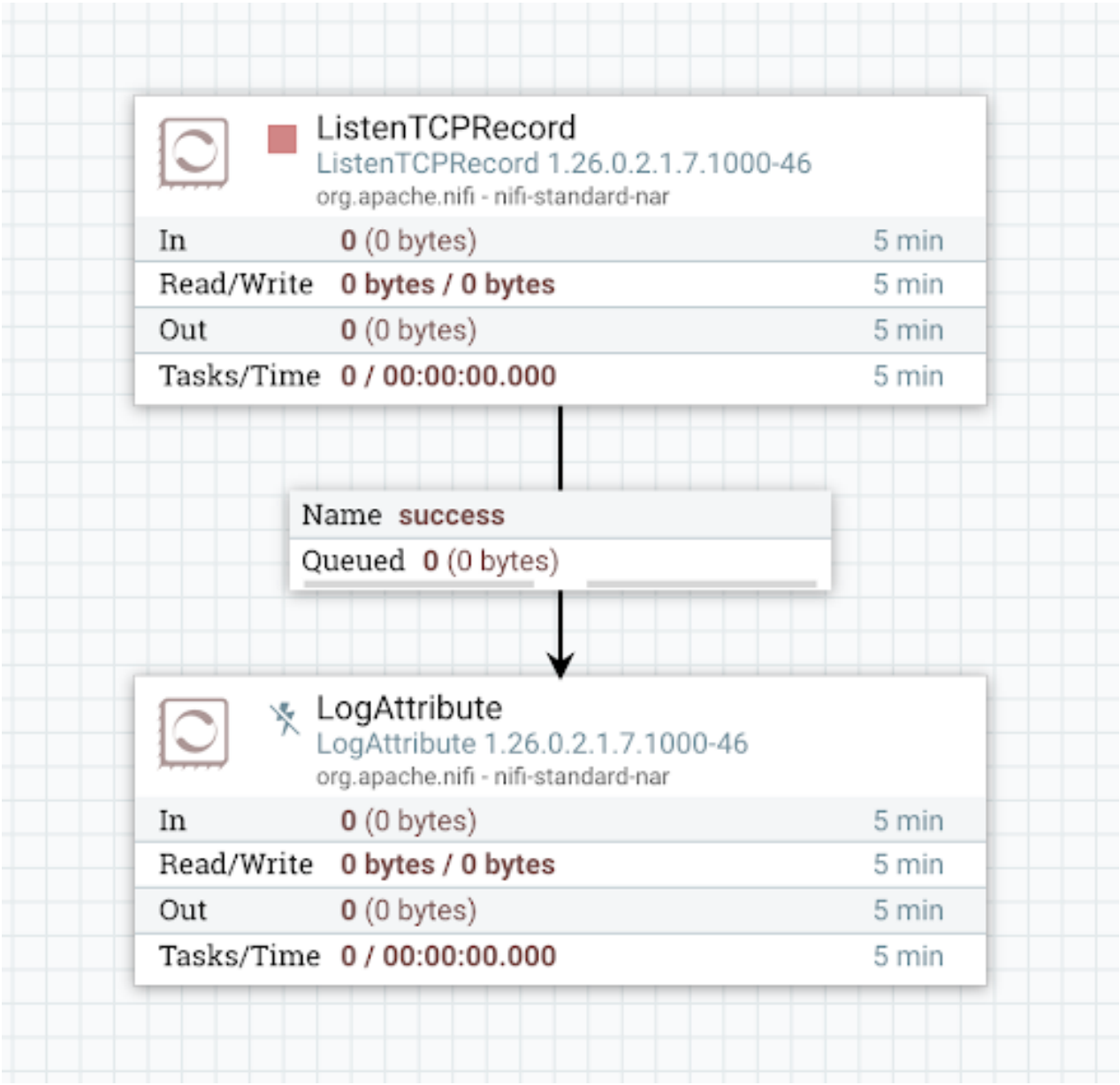
Flow definition file:

- **TCP_Listener_flow_definition.json**

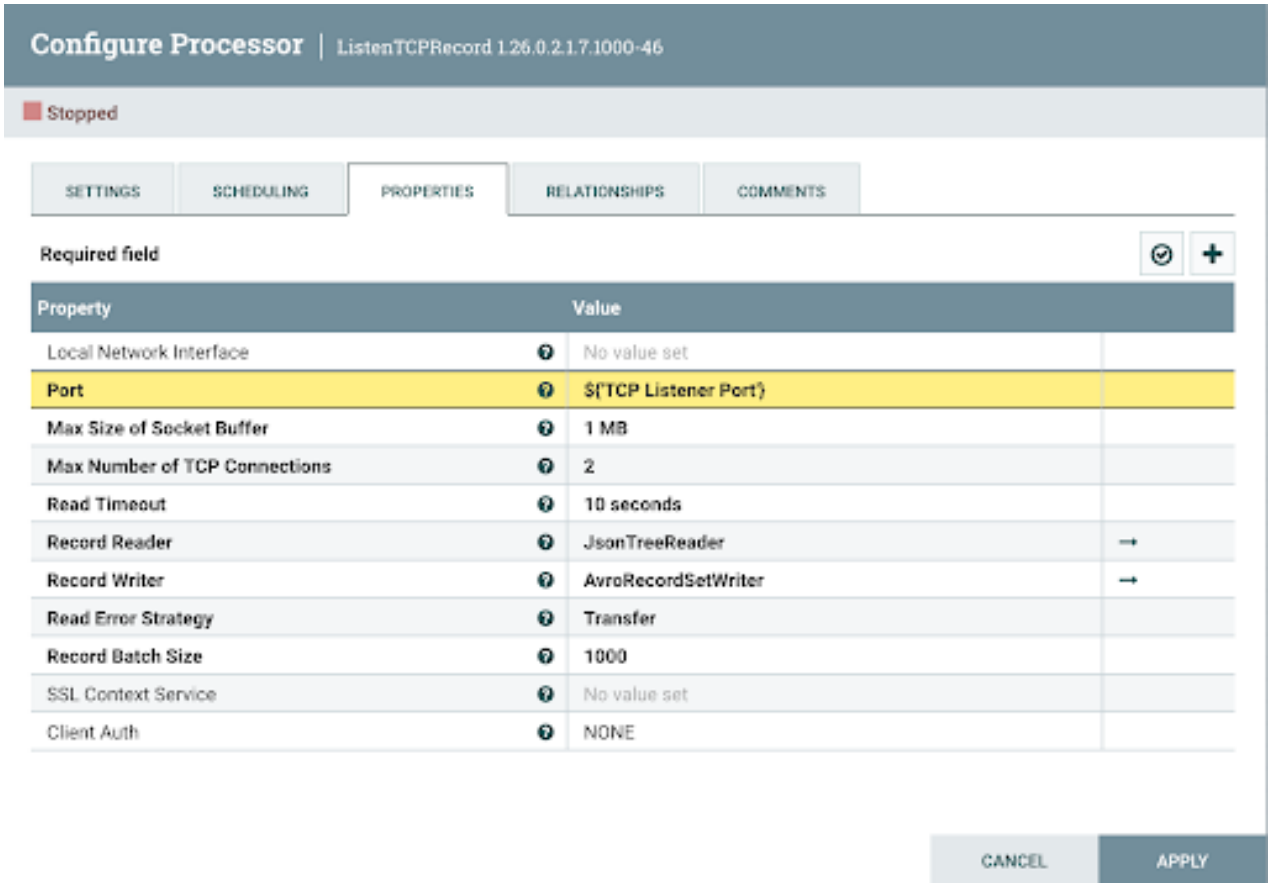
  This file contains the TCP Listener process group (ID: b41940d7-0194-1000-42fc-458834630567)

TCP Listener Process Group

This process group contains the following simple flow:



The process group defines a variable called **TCP Listener Port**, which is referenced by the ListenTCPRecord processor.



Note: Apache NiFi 2 supports only parameters, not variables, so variable migration is required to ensure compatibility with the target version.

The example guides you through the variable migration process and shows how to maintain a clear activity log. Although the example uses a simple flow, the step-by-step approach can be applied to more complex scenarios. In real-world scenarios, Cloudera recommends defining a migration strategy based on the structure of each flow. In some cases, individual flow definition migration may not be necessary.

Before you begin

Copy the TCP_Listener_flow_definition.json file to the Migration Tool's input folder (/etc/migration-tool-input).

Procedure

1. Run Stage 1 variable migration on TCP_Listener_flow_definition.json, using the following command.

```
bin/migration.sh nifi migrate-variables \
-i /etc/migration-tool-input/TCP_Listener_flow_definition.json \
-od /etc/migration-tool-output/variables \
--sourceCompatibleOutput
```

   This generates a sourceVersion folder that contains the output files of the Stage 1 migration.

```
variables
### sourceVersion
    ### activity_log.json
    ### migrated_output
        ### migrated_TCP_Listener.json
```

activity_log.json

- Log of all actions performed during this stage of the migration.
- The following changes were made during the variable migration:
  - A new parameter context was created with a new parameter called TCP Listener Port, which replaces the corresponding variable.
  - The TCP Listener Port variable was removed.

migrated_TCP_Listener.json

- A modified flow definition, which is not compatible with NiFi 2 yet.
- It contains everything the original flow definition did, except the TCP Listener process group now references a parameter instead of the removed variable.
- This is still a NiFi 1 flow definition, so do not load this file directly into NiFi 2. Use an empty NiFi 1 instance to review the modification made by the tool. The flow definition will appear unchanged, but variables will be replaced with parameters of the same name, which are now referenced by the processor instead.
- Changes in the syntax: Variables were previously referenced using ${}, whereas parameters are now referenced using #{}.
  - **NiFi 1 variables**: ${variable_name}
  - **NiFi 2 parameters**: #{parameter_name}



Note:

The file name is composed of the migrated prefix and the name of the top-level process group and not the original file name.

2. Validate the Stage 1 variable migration output for the TCP_Listener_flow_definition.json.
   a) Check the new flow definition in a NiFi 1 instance to verify that the flow matches your expectations.
      1. Start your NiFi instance.
      2. Create a new process group.
      3. Load sourceVersion/migrated_output/migrated_TCP_Listener.json to NiFi 1.
      4. Remove the ‘migrated_‘ prefix from the process group name.
      5. Confirm that variables were correctly converted to parameters.
   b) Review the activity_log.json file and check for any manual-change-requests or manual-validation-requests.
   c) If there are manual-change-requests or manual-validation-requests to handle, follow these steps:
      1. Make the modifications on the NiFi canvas, indicated by the manual-change-requests or manual-validation-requests in the sourceVersion/activity_log.json.
      2. Right-click on the TCPListener process group.
      3. Select Download flow definition without external services.

      Save the file as migrated_TCP_Listener.json and overwrite the one in the sourceVersion/migrated_output folder.
   d) If no requests are present in the log, proceed to the next step.

   At this stage, the flow definition no longer contains variables and uses parameters instead. If the flow meets your expectations, you can either run a full variable migration to validate your flow in NiFi 2 or proceed with migrating the components.
3. Run full variable migration (Stage 1 and 2) on TCP_Listener_flow_definition.json.
   a) Move migrated_TCP_Listener.json from Step 2 into the input folder (/etc/migration-tool-input) and rename it to TCP_Listener_flow_definition.json for clarity.
   b) Make a backup of the output folder (/etc/migration-tool-output/variables) before running the next migration step.

Note:

This step is crucial because the same output folder is used as in the previous migration, and its contents will be overwritten. To retain a record of changes made at each stage, it is better to create a backup.

```
bin/migration.sh nifi migrate-variables \
-i /etc/migration-tool-input/TCP_Listener_flow_definition.json \
-od /etc/migration-tool-output/variables
```

The contents of the previously generated sourceVersion folder will be overwritten, but this is not a concern as the tool generates the same sourceVersion output as before.

Additionally, a targetVersion directory is created, containing the output files of the Stage 2 part of the migration.

migrated_TCP_Listener.json

- NiFi 2-compatible flow definition in terms of variables
- You can load it into NiFi 2 and validate it

activity_log.json

- List of all actions performed during this stage of the migration

4. Proceed with component migration using the input folder from step 3.

Migrating components using flow definition JSON as input

Learn how to use the Cloudera Flow Migration Tool to migrate components from flow definition JSON files.

Example flow for migrating components

The following NiFi flow demonstrates both variable and component migration.

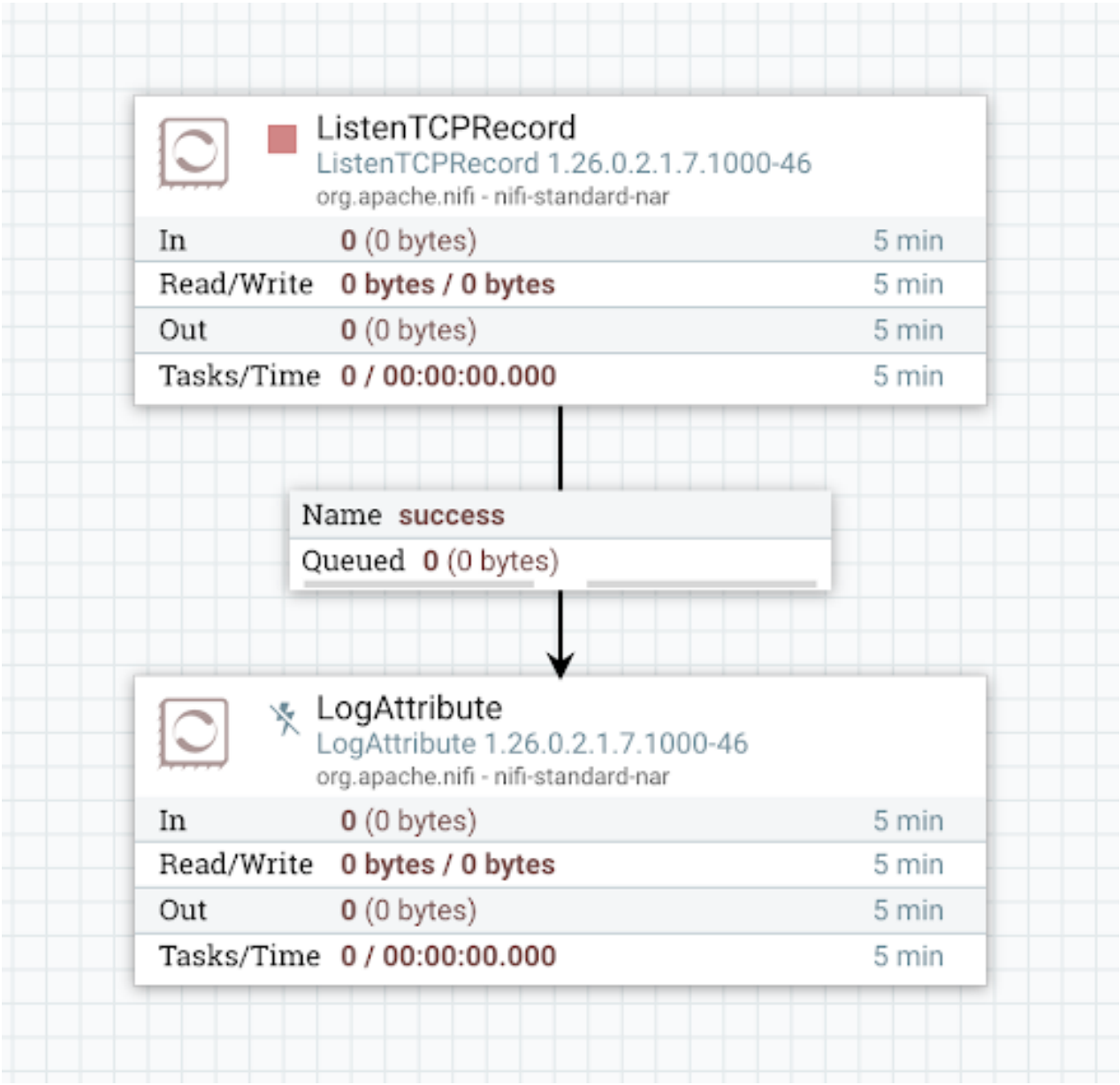
Flow definition file:

- **TCP_Listener_flow_definition.json**

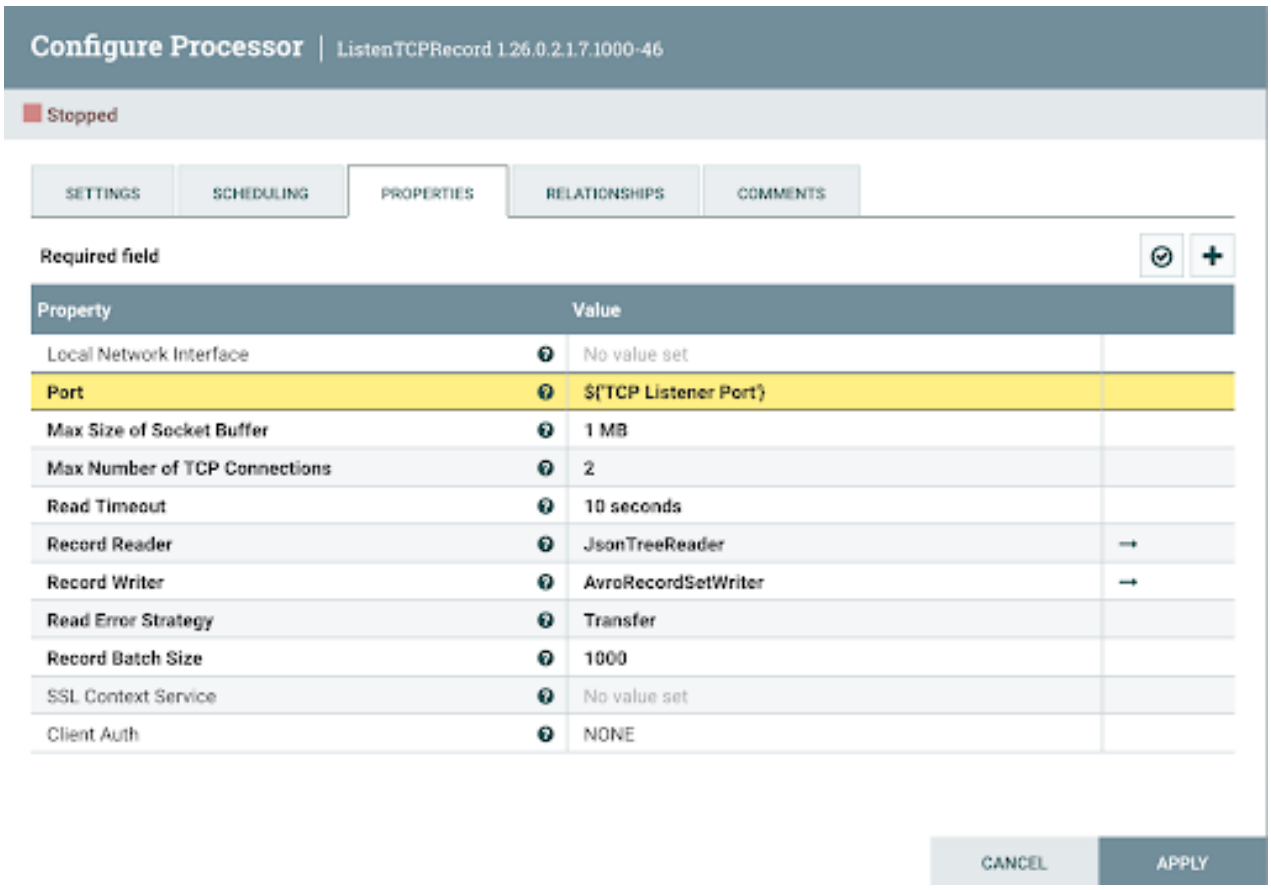
  This file contains the TCP Listener process group (ID: b41940d7-0194-1000-42fc-458834630567)

TCP Listener Process Group

This process group contains the following simple flow:



The process group defines a variable called **TCP Listener Port**, which is referenced by the ListenTCPRecord processor.



The example guides you through the variable migration process and shows how to maintain a clear activity log. Although the example uses a simple flow, the step-by-step approach can be applied to more complex scenarios. In real-world scenarios, Cloudera recommends defining a migration strategy based on the structure of each flow.

Procedure

1. Use the flow definition files in the /etc/migration-tool-input from step 3.

   This is a NiFi-1 compatible flow that no longer contains variables.
2. Run Stage 1 component migration on TCP_Listener_flow_definition.json using the following command.

   ```
   bin/migration.sh nifi migrate-components \
   -i /etc/migration-tool-input/TCP_Listener_flow_definition.json \
   -od /etc/migration-tool-output/components \
   --sourceCompatibleOutput
   ```

   This generates a sourceVersion folder that contains the output files of the Stage 1 migration.

   ```
   components
   ### sourceVersion
       ### activity_log.json
       ### migrated_output
           ### migrated_TCP_Listener.json
   ```

   activity_log.json

   - The log describes all the actions that were performed for this stage of the process group migration.
   - Log of all actions performed during this stage of the process group migration.

migrated_TCP_Listener.json

- A modified NiFi 1 flow, which is not compatible with NiFi 2 yet.
- It contains everything the original flow did, except the **TCP Listener** process group was modified as described in the activity log.

3. Validate the Stage 1 component migration output for TCP_Listener_flow_definition.json.
   a) Load the migrated_TCP_Listener.json into a NiFi 1 instance and check the flow.
   b) Review the activity_log.json file and check for any manual-change-requests or manual-validation-requests. If none are present, proceed to the next step.

      In this example, you can see the following information in the activity log:

      ```
      {
       "sequence" : 2,
       "type" : "change-info",
       "subject" : "b41966ad-0194-1000-a08d-a92489457356",
       "message" : "Component [org.apache.nifi.processors.standard.ListenTCPR
      ecord] has been deprecated (NIFI-13509)",
       "context" : {
         "rule" : "36227b60-75f0-40dc-8caf-a2ec577aa54c"
      }
      ```

      1. Open the targetVersion/migrated_TCP_Listener.json file and search for the "subject" ID, b41966ad-0194-1000-a08d-a92489457356. This ID refers to the ListenTCPRecord processor.
      2. Search for the value of the identifier as “subject” element in NiFi’s search box.

         You are directed to the ListenTCPRecord processor. No manual modifications are needed at this stage. However, it is important to note that the ListenTCPRecord processor is deprecated and is not available in NiFi 2. Once the full component migration (Stage 1 and Stage 2) is complete, instructions will be provided on how to handle this deprecation.

   c) If manual changes are necessary, update the migrated_TCP_Listener.json on the NiFi canvas after loading it. Once the flow is validated and meets expectations, continue with the next step.

   At this stage, you have completed Stage 1 of both variable and component migration for your flow definition file. After reviewing the logs, you confirmed that no manual changes were needed. You can proceed with a full component migration using the /etc/migration-tool-input from Step 3.

4. Run full component migration (Stage 1 and 2) for TCP_Listener_flow_definition.json.
   a) Make a backup of the output folder (/etc/migration-tool-output/components) before running the next migration step.

      Note:

      This step is crucial because the same output folder is used as in the previous migration, and its contents will be overwritten. To retain a record of changes made at each stage, it is better to create a backup.

   b) Run the full component migration using the following command.

      ```
      bin/migration.sh nifi migrate-components \
      -i /etc/migration-tool-input/TCP_Listener_flow_definition.json \
      -od /etc/migration-tool-output/components
      ```

      This generates the following output:

      ```
      components
      ### sourceVersion
      #   ### activity_log.json
      #   ### migrated_output
      # ### migrated_TCP_Listener.json
      ### targetVersion
          ### activity_log.json
          ### migrated_output
      ```

```
### migrated_TCP_Listener.json
```

The contents of the previously generated sourceVersion folder will be overwritten. The contents represent the NiFi 1 compatible version of the migrated flow definition. Since you already performed Stage 1 component migration, the activity log will only include the same entries as those in Steps 2 and 4.

Additionally, a targetVersion directory is created, containing the output files of the Stage 2 part of the migration:

migrated_TCP_Listener.json

- NiFi 2-compatible flow definition
- You can load it into NiFi 2 and validate it

activity_log.json

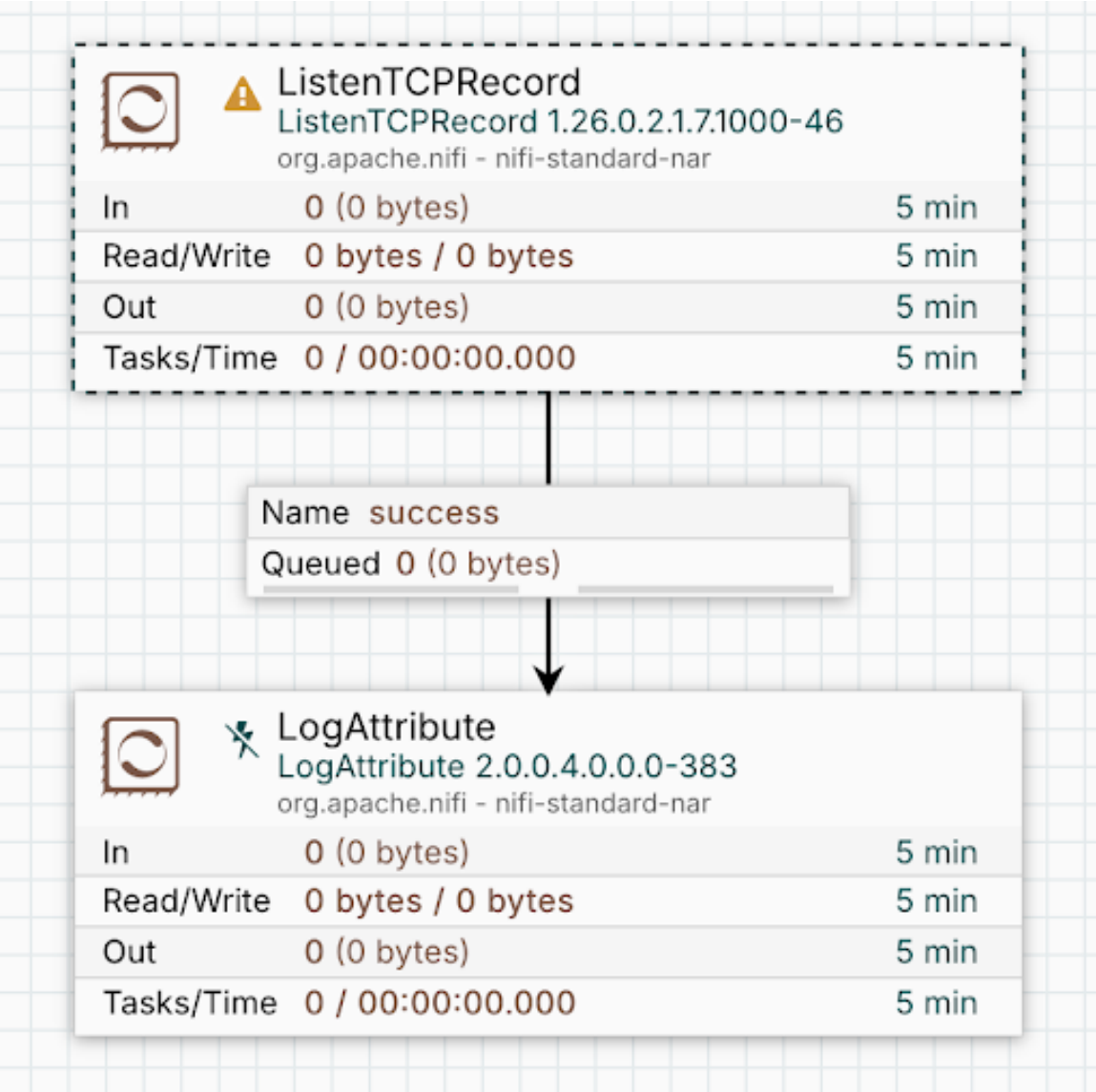
- List of all actions performed during this stage of the migration, and any manual steps that you need to perform.

5. Validate the Stage 2 component migration output for TCP_Listener_flow_definition.json.
   a) Load the targetVersion/migrated_output/migrated_TCP_Listener.json into a NiFi 2 instance and check the flow.
   b) Review the targetVersion/activity_log.json file.

   In this example, you can see the following manual-change-request entry.

```
{
 "sequence" : 6,
 "type" : "manual-change-request",
 "subject" : "b41966ad-0194-1000-a08d-a92489457356",
 "message" : "The component has been deprecated. It is suggested to r
eplace it with a combination of [org.apache.nifi.processors.standard.Lis
tenTCP] and [org.apache.nifi.processors.standard.ConvertRecord] processo
rs",
 "context" : {
   "rule" : "36227b60-75f0-40dc-8caf-a2ec577aa54c"
}
```

   1. Open the targetVersion/migrated_TCP_Listener.json file and search for the "subject" ID, b41966ad-0194-1000-a08d-a92489457356. This ID refers to the ListenTCPRecord processor.
   2. To identify the processor affected by these changes, search for the identifier value of the “subject” element in NiFi’s search box. It refers to the ListenTCPRecord processor. Open the TCP Listener process group. You can see that the processor is marked with dashed borders and its version number still shows that of the NiFi 1 instance. It means that it is a "ghost processor", not available in NiFi 2. The log message suggests replacing it with a combination of ListenTCP and ConvertRecord processors.



Instantiate and configure the ListenTCP and ConvertRecord processors, connect them appropriately, and then remove the ghost processor.

Besides these changes there are no further manual-change-requests or manual-validation-requests in the activity log.

3. Download the flow definition and you have the fully NiFi 2 compatible TCP_Listener_flow_definition.json.

Using migrate-all with flow definition JSON as input

Learn how to use the migrate-all command to migrate variables and components in a single operation using a flow definition JSON file as input.

This command combines the functionality of the migrate-variables and migrate-components commands for ease and convenience.

Note:

Template migration is not applicable for this input type.

While migrate-all simplifies the migration process, for larger flows, it is usually better to run the individual migration commands separately. You may also want to migrate process groups individually to ensure activity logs and manual validation remain manageable.

The migrate-all command can be useful in the following scenarios:

- The flow you want to migrate is simple and of manageable size.
- The flow is large and complex, and you need an initial high-level overview of the migration outcome before performing a step-by-step migration. Running migrate-all provides a preview of the final migrated flow. The Activity Log shows you the actions taken during migration and helps assess the required manual intervention by listing manual-change-requests and manual-validation-requests.

Migrating variables and components together using migrate-all with flow definition JSON as input

Before you begin

1. Offload all flowfiles from NiFi.
2. Stop all processors and disable all controller services in NiFi.
3. Stop NiFi.
4. Copy the flow.json.gz file from NiFi's conf directory to the Migration Tool's input folder (/etc/migration-tool-input).

Procedure

1. Run the following command to migrate the flow.

   ```
   bin/migration.sh nifi migrate-all \
   -i /etc/migration-tool-input/flow.json.gz \
   -od /etc/migration-tool-output
   ```

   This generates the following output:

   ```
   sourceVersion
   ### NiFi_Flow_b4175f57-0194-1000-8470-9251a24519b4
   #   ### elastic_template.json
   #   ### elastic_template.xml
   ### activity_log.json
   ### migrated_flow.json.gz
   targetVersion
   ### NiFi_Flow_b4175f57-0194-1000-8470-9251a24519b4
   #   ### elastic_template.json
   ### activity_log.json
   ### migrated_flow.json.gz
   ```

2. Review the migration output.
   - The sourceVersion directory contains the NiFi 1-compatible version of the flow, including the exported template and its process group counterpart, along with the Activity Log.
   - The targetVersion directory contains the NiFi 2-compatible version of the flow and its corresponding Activity Log.
3. Address issues in NiFi 1 (if needed).

   If sourceVersion/activity_log.json contains manual-change-requests, follow these steps:

   a) Load sourceVersion/migrated_flow.json.gz into your NiFi 1 instance.
   b) Apply the required manual changes.
   c) Run the migrate-all command again using the manually modified flow.
4. Validate the migrated flow.
   a) Rename targetVersion/migrated_flow.json.gz to flow.json.gz.

b) Load flow.json.gz into your NiFi 2 instance to inspect the migrated flow.
c) Open targetVersion/activity_log.json and review any manual-change-requests and manual-validation-requests.
d) Apply the required manual modifications to the flow.

Migrating a flow using a directory with flow definition JSON files as input

This example demonstrates how to migrate both variables and components with the Cloudera Flow Migration Tool using a **directory that contains multiple flow definition JSON files** as input. It shows how variables are used within process groups and referenced by individual components.

Migrating variables using a directory with flow definition JSON files as input

Learn how to use the Cloudera Flow Migration Tool to convert variables to parameters and parameter contexts from a directory of flow definition JSON files.

Example flows for migrating variables

The input directory contains the following flow definition files:

- **TCP_Listener_flow_definition.json**

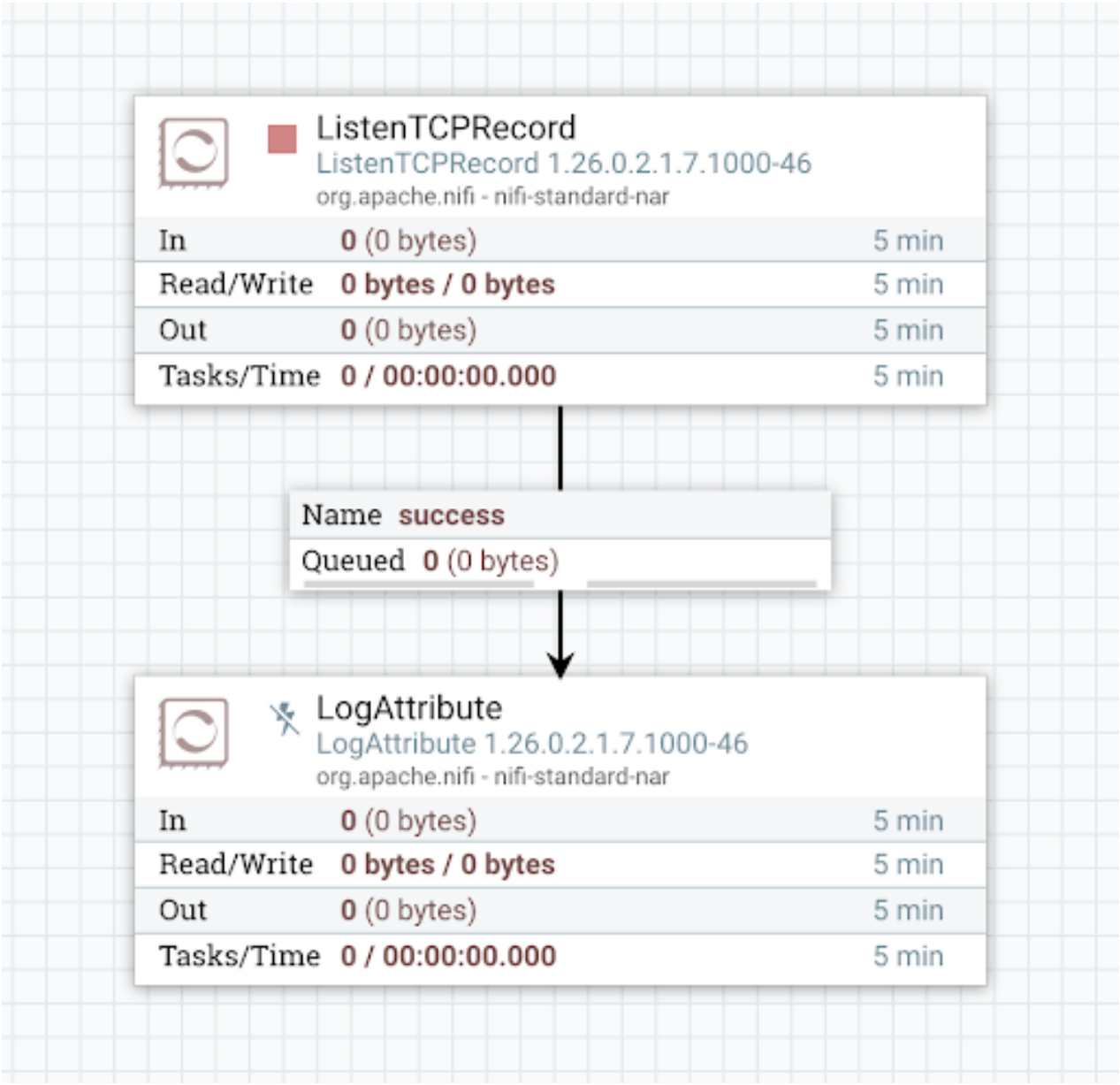
  This file contains the TCP Listener process group (ID: b41940d7-0194-1000-42fc-458834630567)
- **Elastic_flow_definition.json**

  This file contains process group Elastic (ID: b42881c7-0194-1000-3cdf-1bd453a0ed0f)

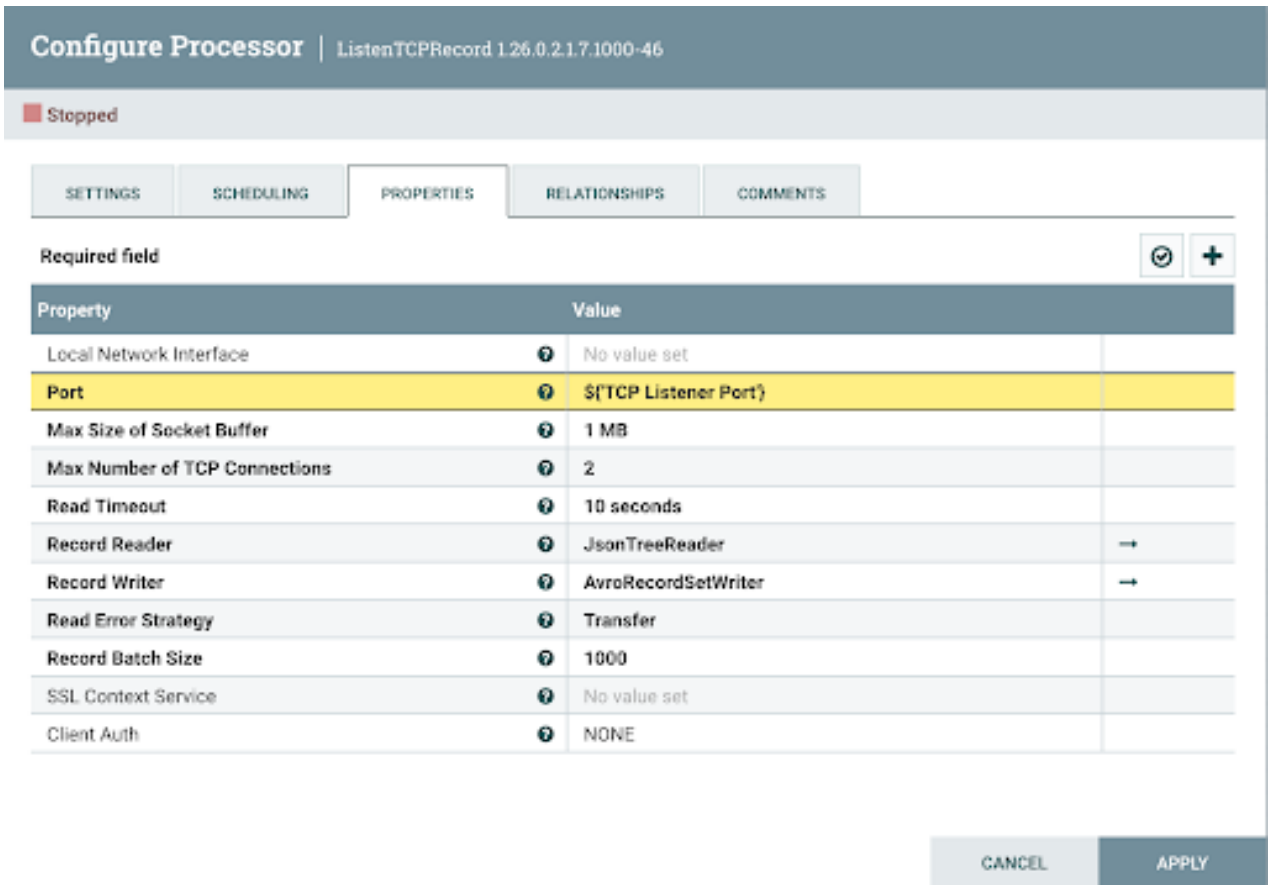


TCP Listener Process Group

This process group contains the following simple flow:

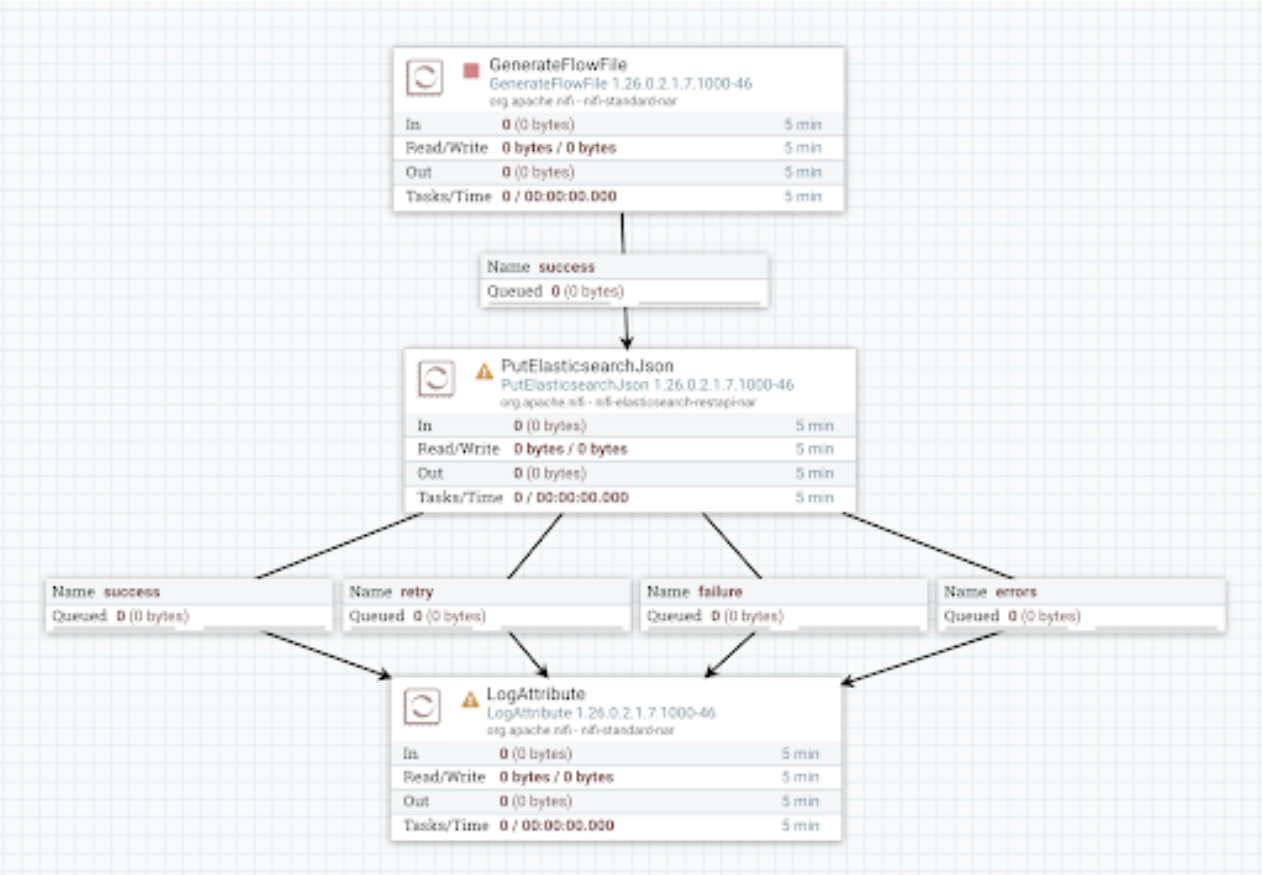


The process group defines a variable called **TCP Listener Port**, which is referenced by the ListenTCPRecord processor.

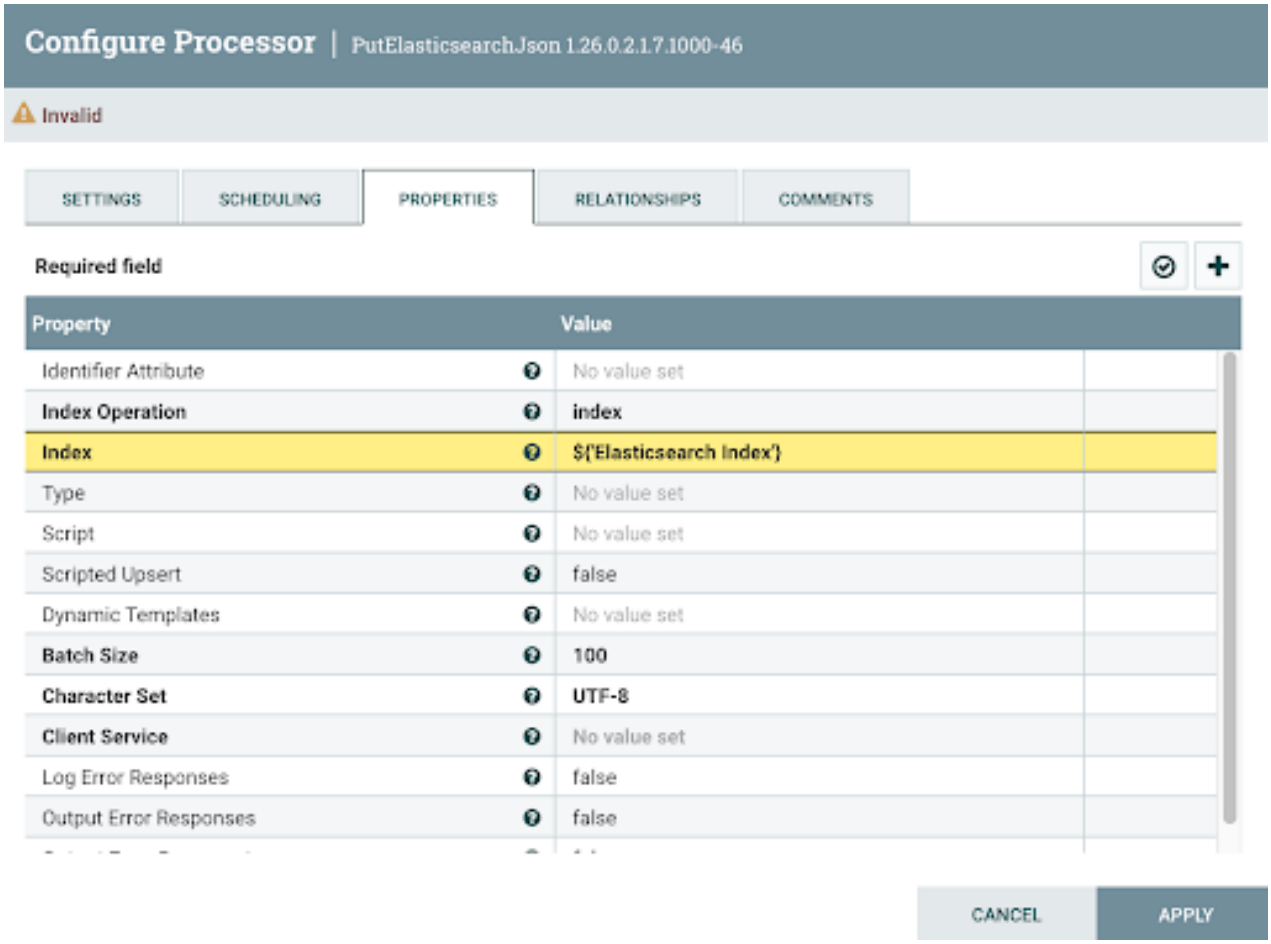


Elastic Process Group

This process group also contains a simple flow:



The process group defines a variable called **Elasticsearch Index**, which is referenced in the PutElasticsearchJson processor's Index property.



Your goal is to migrate the variables used in Cloudera Flow Management 2.1.7.4000 powered by NiFi 1 to parameters used in Cloudera Flow Management 4.12.0 powered by NiFi 2.

Note: Apache NiFi 2 supports only parameters, not variables, so variable migration is required to ensure compatibility with the target version.

About this task

The example guides you through the variable migration process and shows how to maintain a clear activity log. Although the example uses simple flow definitions, the step-by-step approach can be applied to more complex scenarios. In real-world scenarios, Cloudera recommends defining a migration strategy based on your flow's structure.

Before you begin

Copy the TCP_Listener_flow_definition.json and Elastic_flow_definition.json files to the Migration Tool's input folder (/etc/migration-tool-input).

Procedure

1. Run Stage 1 variable migration on the MigrationTool input directory (/etc/migration-tool-input) using the following command.

```
bin/migration.sh nifi migrate-variables \
-i /etc/migration-tool-input \
-od /etc/migration-tool-output/variables \
--sourceCompatibleOutput
```

This generates a sourceVersion folder that contains the output files of the Stage 1 migration.

```
variables
### sourceVersion
    ### activity_log.json
    ### migrated_output
        ### migrated_Elastic.json
        ### migrated_TCP_Listener.json
```

activity_log.json

- Log of all actions performed during this stage of the migration.
- The following changes were made during the variable migration:
  - A new parameter context was created with a new parameter called TCP Listener Port, which replaces the corresponding variable.
  - The TCP Listener Port variable was removed.
  - A new parameter context was created with a new parameter called Elasticsearch Index, which replaces the corresponding variable of the same name.
  - The new parameter is referenced from the PutElasticsearchJson processor.
  - The Elasticsearch Index variable was removed.



Note:

One activity log file is created for the entire migration, even if the input is a directory containing multiple flow definition JSON files.

migrated_TCP_Listener.json

- A modified flow definition, which is not compatible with NiFi 2 yet.
- It contains everything the original flow definition did, except the TCP Listener process group now references a parameter instead of the removed variable.
- This is still a NiFi 1 flow definition, so do not load this file directly into NiFi 2. Use an empty NiFi 1 instance to review the modification made by the tool. The flow definition will appear unchanged, but variables will be replaced with parameters of the same name, which are now referenced by the processor instead.
- Changes in the syntax: Variables were previously referenced using ${}, whereas parameters are now referenced using #{}.
  - **NiFi 1 variables**: ${variable_name}
  - **NiFi 2 parameters**: #{parameter_name}

migrated_Elastic.json

- A modified NiFi 1 flow definition, which is not compatible with NiFi 2 yet.
- It contains everything the original flow definition did, but the Elastic process group now references a parameter instead of the removed variable.



Note:

The file name is composed of the migrated prefix and the name of the top-level process group and not the original file name.

2. Validate the Stage 1 variable migration output for TCP_Listener_flow_definition.json.
   a) Check the new flow definition in a NiFi 1 instance to verify that the flow matches your expectations.
      1. Start your NiFi instance.
      2. Create a new process group.
      3. Load sourceVersion/migrated_output/migrated_TCP_Listener.json to NiFi 1.
      4. Remove the ‘migrated_‘ prefix from the process group name.
      5. Confirm that variables were correctly converted to parameters.

b) Review the activity_log.json file and check for any manual-change-requests or manual-validation-requests.
c) If there are manual-change-requests or manual-validation-requests to handle, follow these steps:

1. Make the modifications on the NiFi canvas, indicated by the manual-change-requests or manual-validation-requests in the sourceVersion/activity_log.json.
2. Right-click on the TCPListener process group.
3. Select Download flow definition without external services.
4. Save the file as migrated_TCP_Listener.json and overwrite the one in the sourceVersion/migrated_output folder.

d) If no requests are present in the log,

1. Make a backup of the original TCP_Listener_flow_definition.json file.
2. Right-click on the TCPListener process group.
3. Select Download flow definition without external services.
4. Save the file as migrated_TCP_Listener.json and overwrite the one in the sourceVersion/migrated_output folder.

e) Proceed to running Stage 1 variable migration on Elastic_flow_definition.json.

3. Validate the Stage 1 variable migration output for Elastic_flow_definition.json.
a) Check the new flow definition in a NiFi 1 instance to verify that the flow matches your expectations.

1. Start your NiFi instance.
2. Create a new process group.
3. Load sourceVersion/migrated_output/migrated_Elastic.json to NiFi 1.
4. Remove the ‘migrated_‘ prefix from the process group name.
5. Confirm that variables were correctly converted to parameters.

b) Review the activity_log.json file and check for any manual-change-requests or manual-validation-requests.
c) If there are manual-change-requests or manual-validation-requests to handle, follow these steps:

1. Make the modifications on the NiFi canvas, indicated by the manual-change-requests or manual-validation-requests in the sourceVersion/activity_log.json.
2. Right-click on the Elastic process group.
3. Select Download flow definition without external services.
4. Save the file as migrated_Elastic.json and overwrite the one in the sourceVersion/migrated_output folder.

d) If no requests are present in the log,

1. Right-click on the Elastic process group.
2. Select Download flow definition without external services.
3. Save the file as migrated_Elastic.json and overwrite the one in the sourceVersion/migrated_output folder.

At this point, both flow definitions have been updated to replace variables with parameters. If the result aligns with your expectations, you can either perform a full variable migration to validate the flow in NiFi 2 or continue with migrating the components.

4. Run full variable migration (Stage 1 and 2) using the following command.

```
bin/migration.sh nifi migrate-variables \
-i /etc/migration-tool-input \
-od /etc/migration-tool-output/variables
```

The contents of the previously generated sourceVersion folder will be overwritten, but this is not a concern as the tool generates the same sourceVersion output as before.

Additionally, a targetVersion directory is created, containing the output files of the Stage 2 part of the migration.

```
variables
### sourceVersion
#    ### activity_log.json
#    ### migrated_output
```

```
#        ### migrated_Elastic.json
#        ### migrated_TCP_Listener.json
###targetVersion
    ### activity_log.json
    ### migrated_output
        ### migrated_Elastic.json
        ### migrated_TCP_Listener.json
```

migrated_Elastic.json

- NiFi 2-compatible flow definition in terms of variables
- You can load it into NiFi 2 and validate it

migrated_TCP_Listener.json

- NiFi 2-compatible flow definition in terms of variables
- You can load it into NiFi 2 and validate it

activity_log.json

- List of all actions performed during this stage of the migration

Note:

One activity log file is created for the whole migration command.

5. Proceed with component migration using the input folder from step 3.

Migrating components using a directory with flow definition JSON files as input

Learn how to use the Cloudera Flow Migration Tool to migrate components from flow definition JSON files.

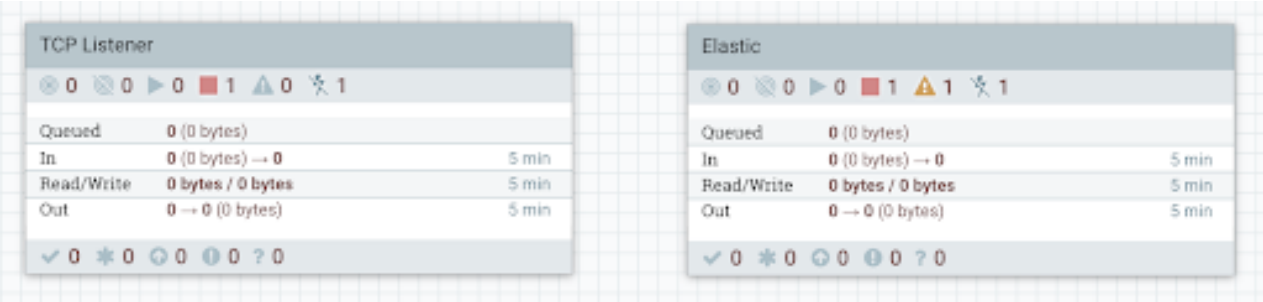
Example flows for migrating components

The input directory contains the following flow definition files:

- **TCP_Listener_flow_definition.json**

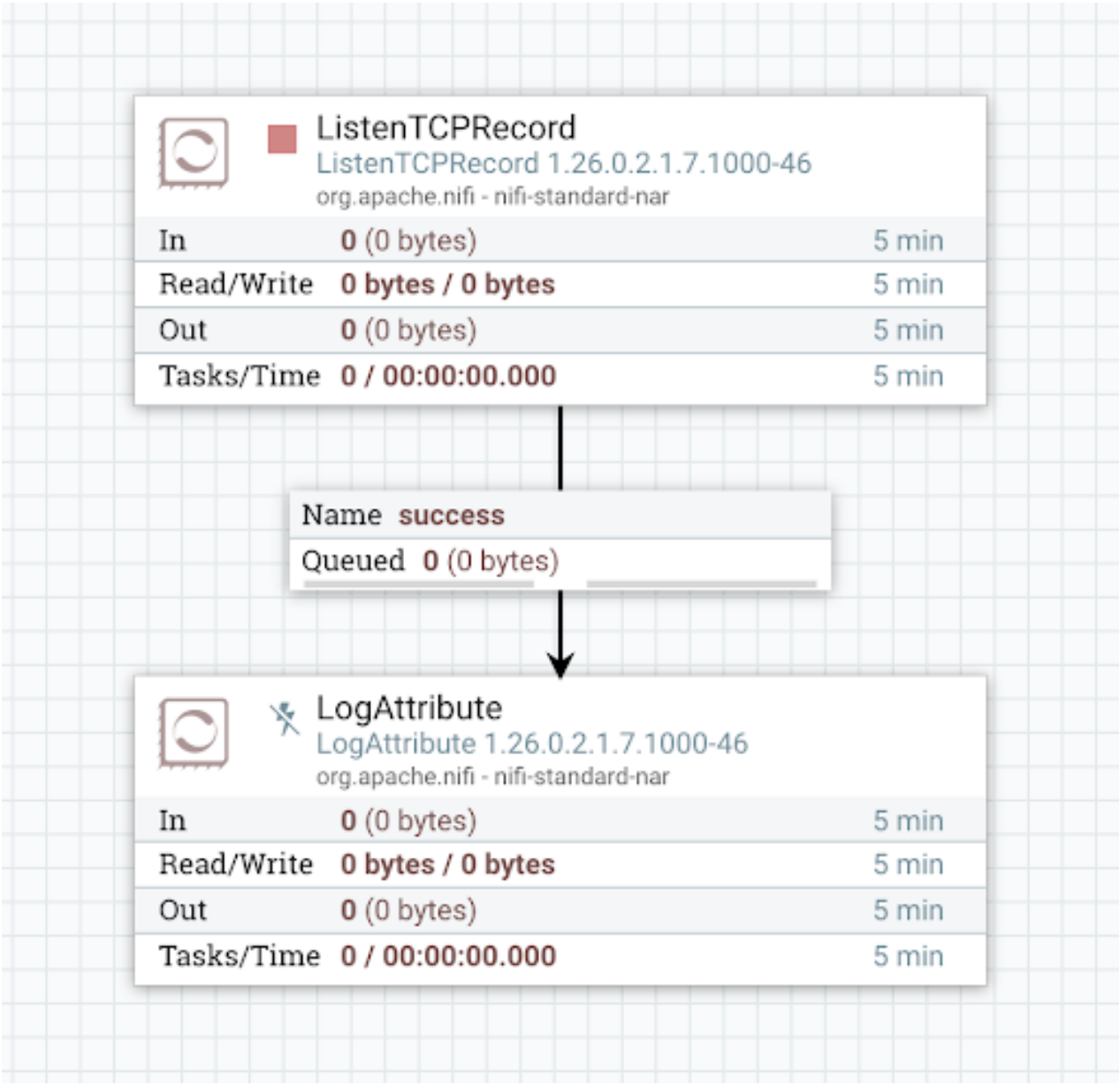
  This file contains the TCP Listener process group (ID: b41940d7-0194-1000-42fc-458834630567)
- **Elastic_flow_definition.json**

  This file contains process group Elastic (ID: b42881c7-0194-1000-3cdf-1bd453a0ed0f)

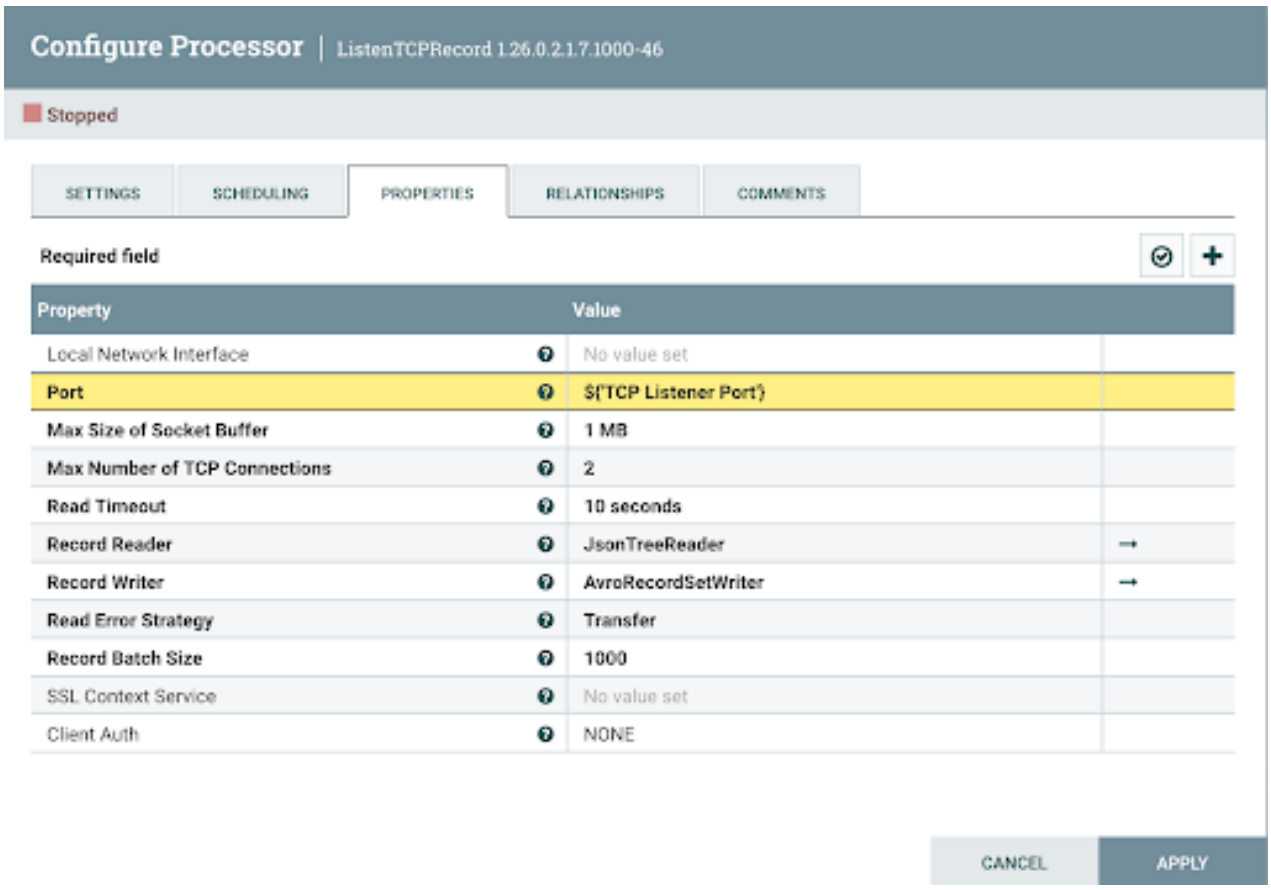


TCP Listener Process Group

This process group contains the following simple flow:

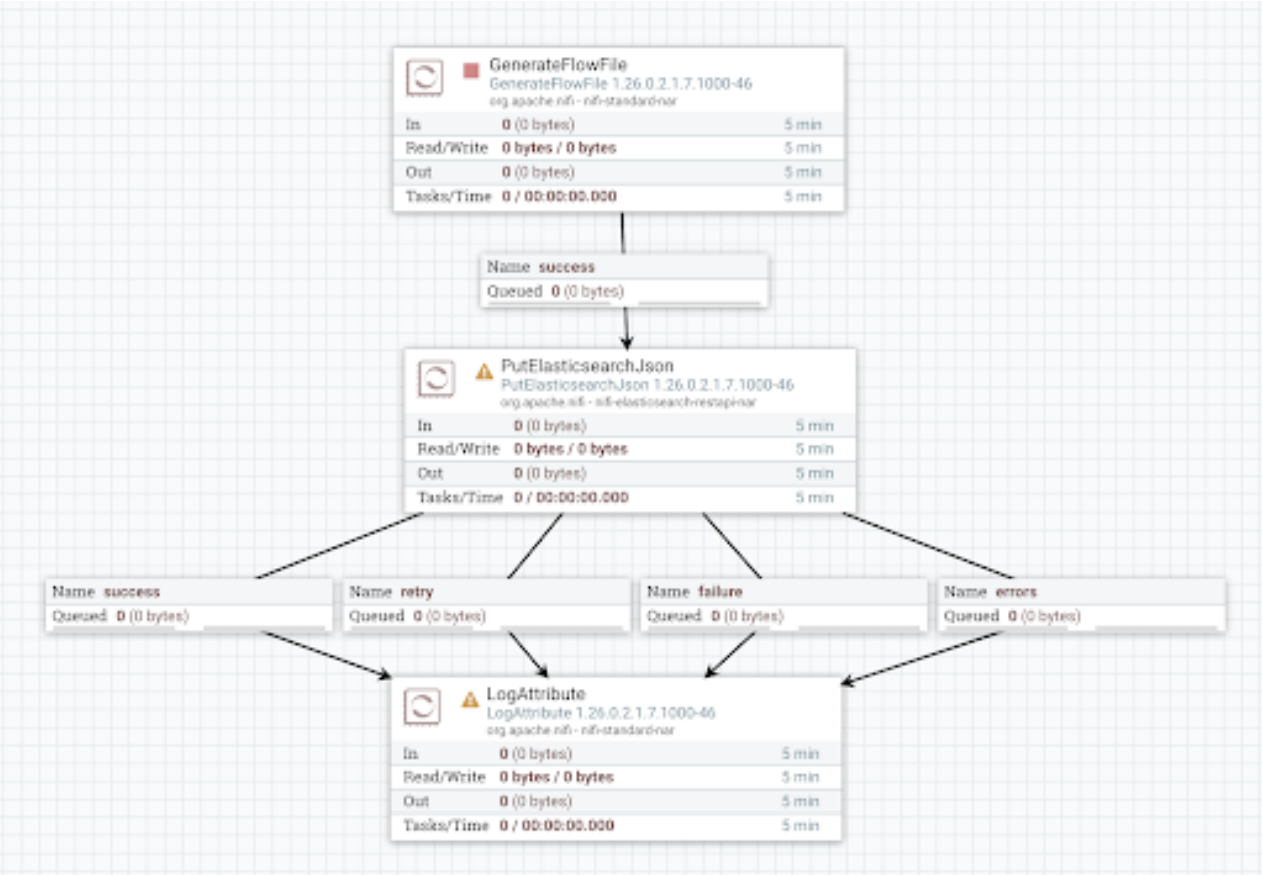


The process group defines a variable called **TCP Listener Port**, which is referenced by the ListenTCPRecord processor.

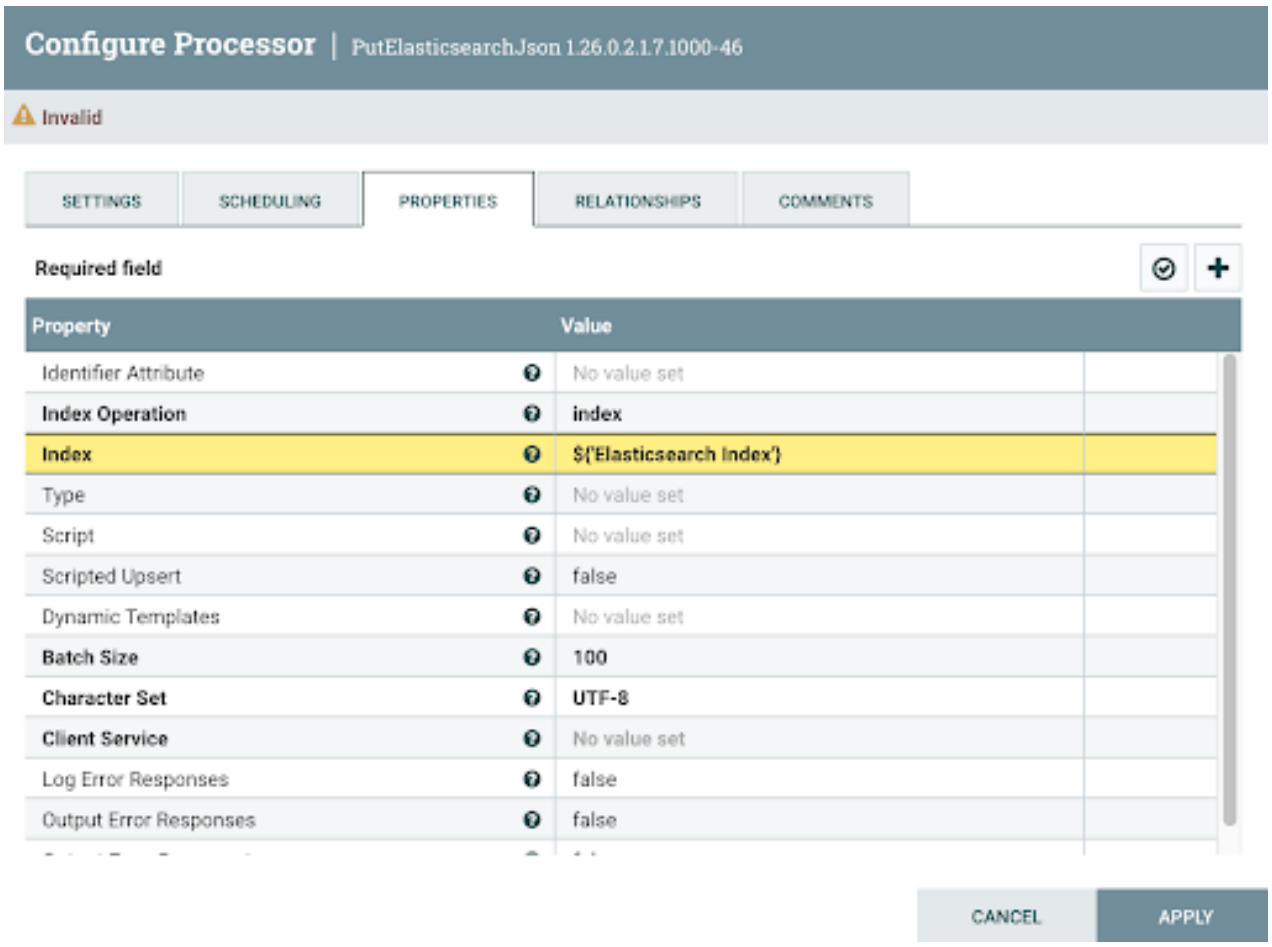


Elastic Process Group

This process group also contains a simple flow:



The process group defines a variable called **Elasticsearch Index**, which is referenced in the PutElasticsearchJson processor's Index property.



Your goal is to migrate the components used in Cloudera Flow Management 2.1.7.4000 powered by NiFi 1 to parameters used in Cloudera Flow Management 4.12.0 powered by NiFi 2.

Note: Apache NiFi 2 supports only parameters, not variables, so variable migration is required to ensure compatibility with the target version.

About this task

The example guides you through component migration one flow definition at a time, simplifying the process and maintaining a clear activity log. Although the example uses simple flow definitions, the step-by-step approach can be applied to more complex scenarios. In real-world scenarios, Cloudera recommends defining a migration strategy based on your flow's structure.

Procedure

1. Make a backup of the output folder (/etc/migration-tool-output/components) before running the next migration step.

   Note:

   This step is crucial because the same output folder is used as in the previous migration, and its contents will be overwritten. To retain a record of changes made at each stage, it is better to create a backup.
2. Copy the migrated_TCP_Listener.json and migrated_Elastic.json files to the Migration Tool's input folder (/etc/migration-tool-input).
3. Rename the file, overwriting the existing ones, to TCP_Listener_flow_definition.json and Elastic_flow_definition.json for clarity.

4. Run Stage 1 component migration using the following command.

```
bin/migration.sh nifi migrate-components \
-i /etc/migration-tool-input \
-od /etc/migration-tool-output/components \
--sourceCompatibleOutput
```

This generates a sourceVersion folder that contains the output files of the Stage 1 migration.

```
components
###sourceVersion
    ### activity_log.json
    ### migrated_output
        ### migrated_Elastic.json
        ### migrated_TCP_Listener.json
```

activity_log.json

- The log describes all the actions that were performed for this stage of the process group migration.
- Log of all actions performed during this stage of the process group migration.



Note:

One activity log file will be created for the entire command.

migrated_TCP_Listener.json

- A modified NiFi 1 flow, which is not compatible with NiFi 2 yet.
- It contains everything the original flow did, except the **TCP Listener** process group was modified as described in the Activity Log.

migrated_Elastic.json

- A modified NiFi 1 flow definition which is not compatible with NiFi 2 yet.
- It contains everything the original flow definition did, except the Elastic process group was modified with the actions described in the Activity Log.

5. Validate the Stage 1 component migration output for TCP_Listener_flow_definition.json.
   a) Load the migrated_TCP_Listener.json into a NiFi 1 instance and check the flow.
   b) Review the activity_log.json file and check for any manual-change-requests or manual-validation-requests. If none are present, proceed to the next step.

   In this example, you can see the following information in the Activity Log:

```
{
 "sequence" : 2,
 "type" : "change-info",
 "subject" : "b41966ad-0194-1000-a08d-a92489457356",
 "message" : "Component [org.apache.nifi.processors.standard.ListenTCPR
ecord] has been deprecated (NIFI-13509)",
 "context" : {
   "rule" : "36227b60-75f0-40dc-8caf-a2ec577aa54c"
}
```

   1. Open the targetVersion/migrated_TCP_Listener.json file and search for the "subject" ID, b41966ad-0194-1000-a08d-a92489457356. This ID refers to the ListenTCPRecord processor.
   2. Search for the value of the identifier as “subject” element in NiFi’s search box.

      You are directed to the ListenTCPRecord processor. No manual modifications are needed at this stage. However, it is important to note that the ListenTCPRecord processor is deprecated and is not available in NiFi 2. Once the full component migration (Stage 1 and Stage 2) is complete, instructions will be provided on how to handle this deprecation.

c) If manual changes are necessary, update the migrated_TCP_Listener.json on the NiFi canvas after loading it. Once the flow is validated and meets expectations, continue with the next step.

6. Validate the Stage 1 component migration output for Elastic_flow_definition.json.
   a) Load the migrated_Elastic.json into a NiFi 1 instance and check the flow.
   b) Review the activity_log.json file. It contains a change-info entry that informs you of changes to a NiFi 2 processor, identified by the subject ID.

```
{
 "sequence" : 2,
 "type" : "change-info",
 "subject" : "b428aad6-0194-1000-d73d-9f2332a59f04",
 "message" : "Property [Max JSON Field String Length] has been added (N
IFI-12343); Property [put-es-json-not_found-is-error] has been renamed t
o [put-es-not_found-is-error] (NIFI-12255); Property [put-es-json-error-
documents] has been removed (NIFI-12255); Relationships [success] has be
en renamed to [original]; Relationship [successful] has been added (NIFI
-12255);",
 "context" : {
   "rule" : "6501a693-b524-4d1a-b11e-69747c2651e8"
}
```

      1. Open the targetVersion/migrated_output/migrated_Elastic.json file and search for the "subject" ID, b428aad6-0194-1000-d73d-9f2332a59f04. This ID refers to the PutElasticsearchJson processor.
      2. To identify the processor affected by these changes, search for the identifier value of the “subject” element in NiFi’s search box.You are directed to the PutElasticsearchJson processor. No manual modifications are needed at this stage. However, it is important to note that changes will be applied to the PutElasticsearchJson processor during Stage 2 of the migration. Once the full component migration (Stage 1 and Stage 2) is complete, instructions will be provided on how to handle this change.

   At this stage, you have completed Stage 1 of both variable and component migration for your flow definition files. After reviewing the logs, you confirmed that no manual changes were needed. You can proceed with a full component migration using the /etc/migration-tool-input from Step 5.

7. Run full component migration (Stage 1 and 2).
   a) Make a backup of the output folder (/etc/migration-tool-output/components) before running the next migration step.

      Note:

      This step is crucial because the same output folder is used as in the previous migration, and its contents will be overwritten. To retain a record of changes made at each stage, it is better to create a backup.

   b) Run the full component migration using the following command.

```
bin/migration.sh nifi migrate-components \
-i /etc/migration-tool-input \
-od /etc/migration-tool-output/components
```

      This generates the following output:

```
components
###sourceVersion
#    ### activity_log.json
#    ### migrated_output
#        ### migrated_Elastic.json
#        ### migrated_TCP_Listener.json
###sourceVersion
    ### activity_log.json
    ### migrated_output
        ### migrated_Elastic.json
        ### migrated_TCP_Listener.json
```

The contents of the previously generated sourceVersion folder will be overwritten. The contents represent the NiFi 1 compatible version of the migrated flow definition. Since you already performed Stage 1 component migration, the activity log will only include the same entries as those in Steps 2 and 4.

Additionally, a targetVersion directory is created, containing the output files of the Stage 2 part of the migration:

migrated_TCP_Listener.json

- NiFi 2-compatible flow definition
- You can load it into NiFi 2 and validate it

migrated_Elastic.json

- NiFi 2-compatible flow definition
- You can load it into NiFi 2 and validate it

activity_log.json

- List of all actions performed during this stage of the migration, and any manual steps that you need to perform.

8. Validate the Stage 2 component migration output for TCP_Listener_flow_definition.json.

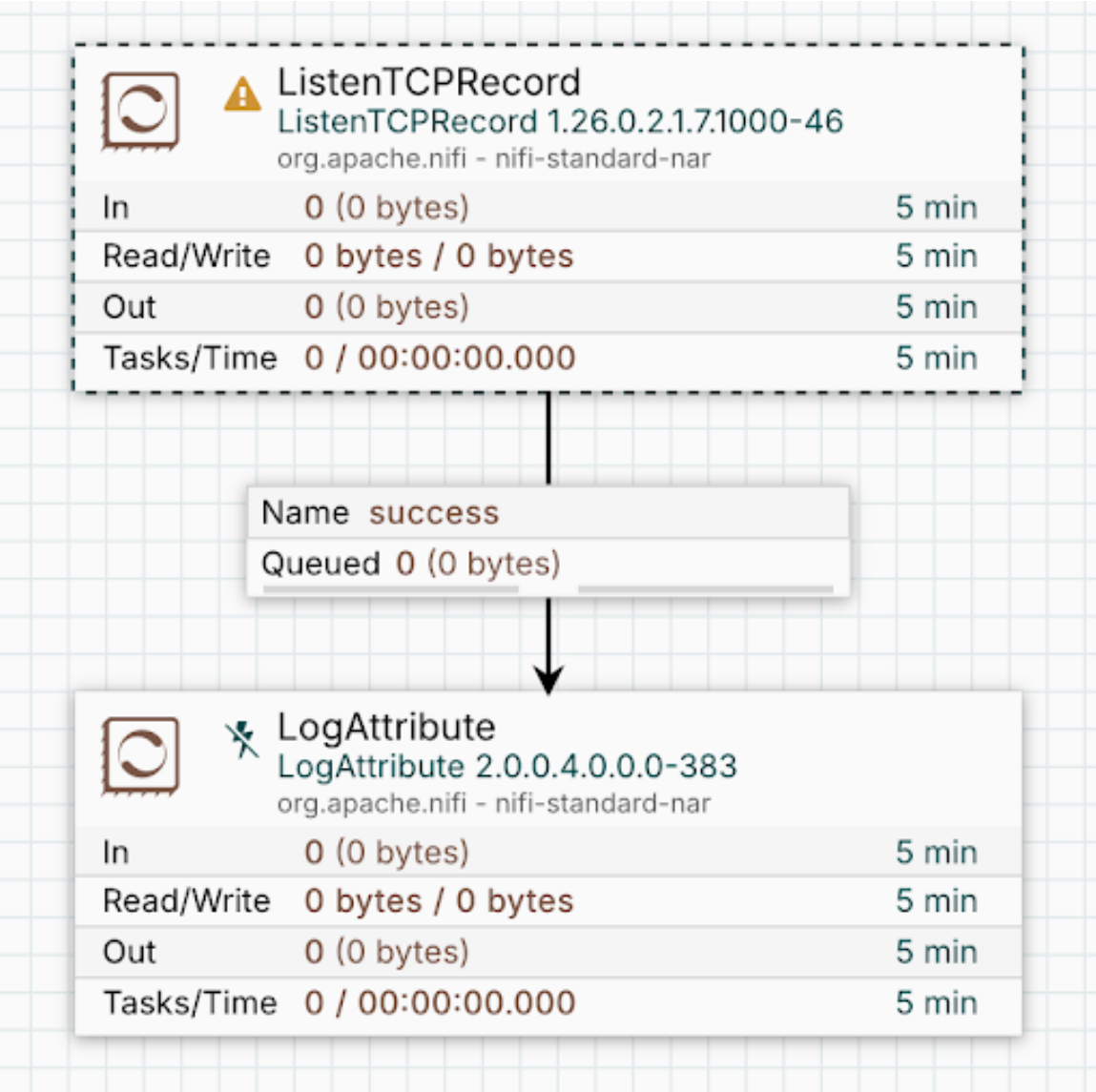
a) Load the targetVersion/migrated_output/migrated_TCP_Listener.json into a NiFi 2 instance and check the flow.

b) Review the targetVersion/activity_log.json file.

In this example, you can see the following manual-change-request entry.

```
{
 "sequence" : 6,
 "type" : "manual-change-request",
 "subject" : "b41966ad-0194-1000-a08d-a92489457356",
 "message" : "The component has been deprecated. It is suggested to r
eplace it with a combination of [org.apache.nifi.processors.standard.Lis
tenTCP] and [org.apache.nifi.processors.standard.ConvertRecord] processo
rs",
 "context" : {
   "rule" : "36227b60-75f0-40dc-8caf-a2ec577aa54c"
}
```

1. Open the targetVersion/migrated_TCP_Listener.json file and search for the "subject" ID b41966ad-0194-1000-a08d-a92489457356. This ID refers to the ListenTCPRecord processor.
2. To identify the processor affected by these changes, search for the identifier value of the “subject” element in NiFi’s search box. It refers to the ListenTCPRecord processor. Open the TCP Listener process group. You can see that the processor is marked with dashed borders and its version number still shows that of the NiFi 1 instance. It means that it is a "ghost processor", not available in NiFi 2. The log message suggests replacing it with a combination of ListenTCP and ConvertRecord processors.



Instantiate and configure the ListenTCP and ConvertRecord processors, connect them appropriately, and then remove the ghost processor.

Besides these changes there are no further manual-change-requests or manual-validation-requests in the Activity Log.

c) Download the flow definition and you have the fully NiFi 2 compatible TCP_Listener_flow_definition.json.

9. Validate the Stage 2 component migration output for Elastic_definition.json.

a) Load the targetVersion/migrated_output/migrated_Elastic.json into a NiFi 2 instance and check the flow.

b) Review the targetVersion/activity_log.json file.

In this example, you can see the following manual-change-request entry.

```
{
 "sequence" : 3,
 "type" : "manual-change-request",
 "subject" : "b428aad6-0194-1000-d73d-9f2332a59f04",
 "message" : "New relationship [original] has been added, it has to be
 connected to a downstream processor or terminated.",
 "context" : {
   "rule" : "6501a693-b524-4d1a-b11e-69747c2651e8"
```

```
}
```

1. Open the targetVersion/migrated_Elastic.json file and search for the "subject" ID b428aad6-0194-1000-d73d-9f2332a59f04. This ID refers to the PutElasticsearchJson processor.
2. To identify the processor affected by these changes, search for the identifier value of the “subject” element in NiFi’s search box. It refers to the PutElasticsearchJson processor. You will find an unbound "original" relationship that needs to be connected to a downstream processor or terminated. Make the necessary modifications manually. Once completed, this change request is resolved.

c) Download the flow definition and you have the fully NiFi 2 compatible Elastic_flow_definition.json.

Using migrate-all using a directory with flow definition JSON files as input

Learn how to use the migrate-all command to migrate templates, variables, and components in a single operation using a directory with flow definition JSON files as input.

About this task

This command combines the functionality of the migrate-variables and migrate-components commands for ease and convenience.

Note:

Template migration is not applicable for this input type.

While migrate-all simplifies the migration process, for larger flows, it is usually better to run the individual migration commands separately. You may also want to migrate process groups individually to ensure activity logs and manual validation remain manageable.

The migrate-all command can be useful in the following scenarios:

- The flow you want to migrate is simple and of manageable size.
- The flow is large and complex, and you need an initial high-level overview of the migration outcome before performing a step-by-step migration. Running migrate-all provides a preview of the final migrated flow. The Activity Log shows you the actions taken during migration and helps assess the required manual intervention by listing manual-change-requests and manual-validation-requests.

Before you begin

Copy the TCP_Listener_flow_definition.json and Elastic_flow_definition.json to the Migration Tool’s input folder (/etc/migration-tool-input).

Procedure

1. Run the following command to migrate the flow.

```
bin/migration.sh nifi migrate-all \
-i /etc/migration-tool-input/ \
-od /etc/migration-tool-output
```

This generates the following output:

```
sourceVersion
    ### activity_log.json
    ### migrated_output
        ### migrated_Elastic.json
        ### migrated_TCP_Listener.json

targetVersion
    ### activity_log.json
    ### migrated_output
        ### migrated_Elastic.json
```

```
        ### migrated_TCP_Listener.json
```

2. Review the migration output.
   - The sourceVersion directory contains the NiFi 1-compatible version of the flow definition and the Activity Log.
   - The targetVersion directory contains the NiFi 2-compatible version of the flow definition and its corresponding Activity Log.
3. Address issues in NiFi 1 (if needed).

   If sourceVersion/activity_log.json contains manual-change-requests, follow these steps:

   a. Load sourceVersion/migrated_output/<flow_definition.json> into your NiFi 1 instance.
   b. Apply the required manual changes.
   c. Run the migrate-all command again using the manually modified flow.
4. Validate the migrated flow.

   a. Load targetVersion/migrated_output/<flow_definition.json> into your NiFi 2 instance to inspect the migrated flow.
   b. Open targetVersion/activity_log.json and review any manual-change-requests and manual-validation-requests.
   c. Apply the required manual modifications to the flow.

Migrating a flow using template.xml as input

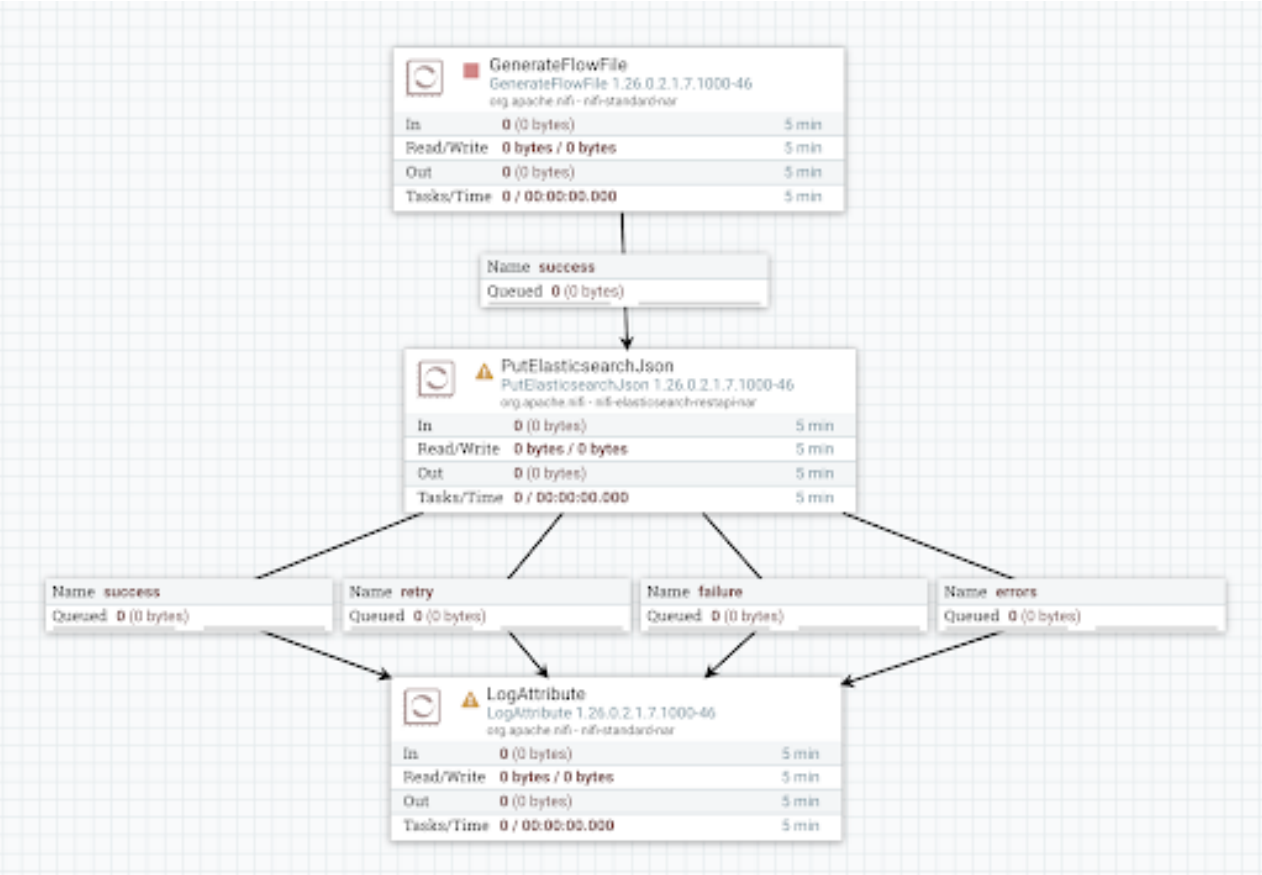
This section provides step-by-step examples of how to run different migrations with the Cloudera Flow Migration Tool using a **template.xml** file as input.

Migrating templates using template.xml as input

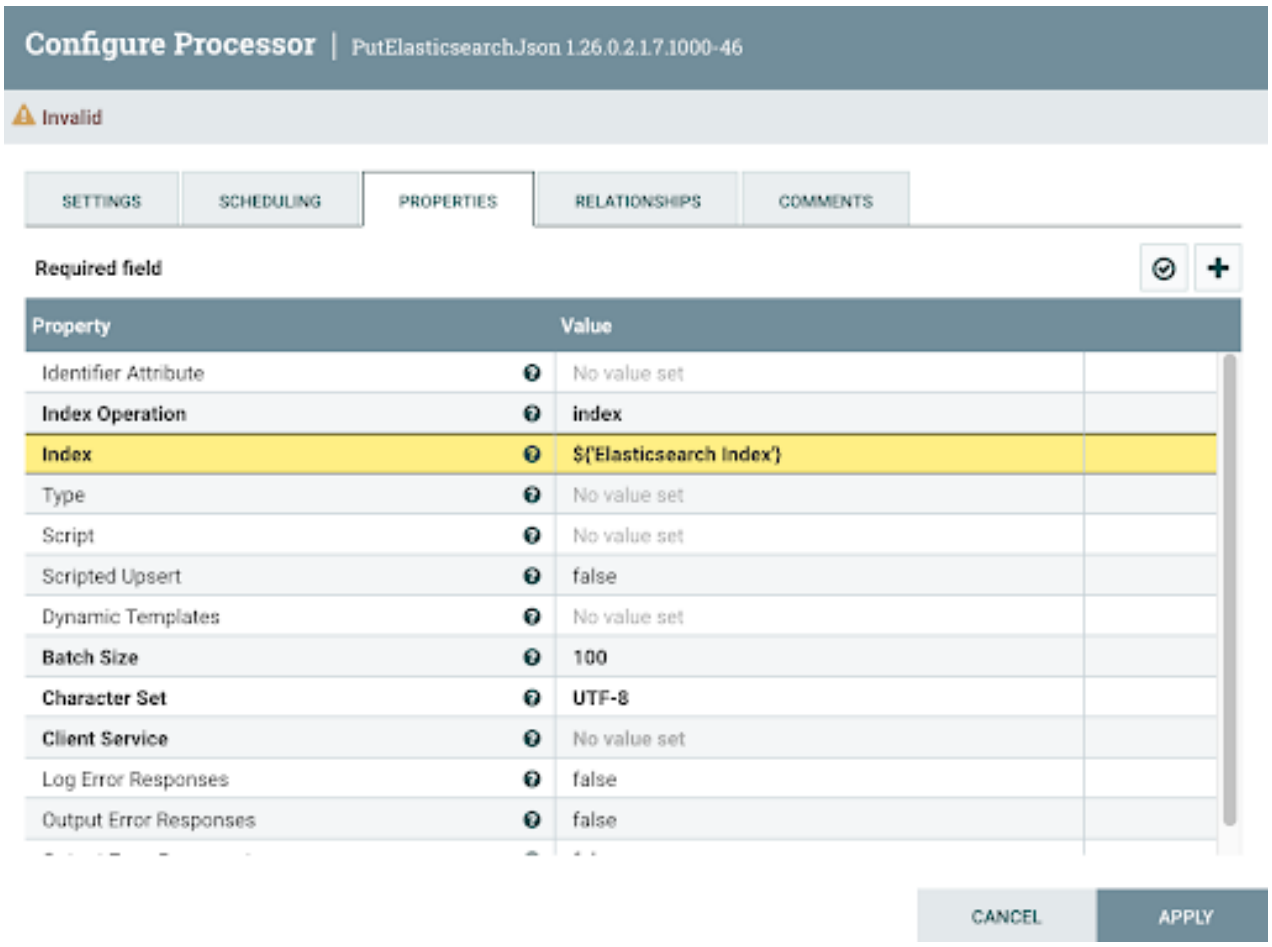
Learn how to use the Cloudera Flow Migration Tool to migrate a downloaded template.xml file and transform templates for compatibility with NiFi 2. NiFi 2 does not support templates, so this step is required to ensure compatibility with the target version. This command combines the functionality of the migrate-variables and migr ate-components commands so for templates it is equivalent with migrate-all. While the migrate-templates command simplifies the migration process, for larger flows, it is usually better to run the two migration commands separately. You may also want to migrate process groups individually to ensure activity logs and manual validation remain manageable.

Example flow for migrating a template

You have a template.xml file called Elastic_template.xml, which was created from a process group called **elastic**. This process group contains the following simple flow:



The process group has a variable, **Elasticsearch Index**, referenced in the index property of the PutElasticsearchJson processor.



Your goal is to migrate this template to NiFi 2 used in Cloudera Flow Management 4.12.0.

Before you begin

Copy the elastic_template.xml file to the Migration Tool's input folder (/etc/migration-tool-input).

Procedure

1. Run Stage 1 template migration using the following command.

```
bin/migration.sh nifi migrate-templates \
-i /etc/migration-tool-input/elastic_template.xml \
-od /etc/migration-tool-output/templates \
-sco
```

This generates a sourceVersion folder that contains the output files of the migration.

```
templates
### sourceVersion
    ### Elastic_template_b4175f57-0194-1000-8470-9251a24519b4
    #   ### Elastic_template.json
    ### activity_log.json
```

The folder name Elastic_template_b4175f57-0194-1000-8470-9251a24519b is a temporary process group which was created during the migration process and used to separate the components in the template from other components that may be present during the migration.

Elastic_template.json

- Flow definition containing the contents of the original template
- Equivalent to manually converting a template to a flow definition in NiFi 1 by:
  a. Instantiating the template on the canvas.
  b. Right-clicking the process group.
  c. Selecting Download flow definition without external services.
- Modified by the Migration Tool to ensure compatibility:
  - Variables (not supported in NiFi 2) converted into parameters
  - Parameter context created to hold the new parameters
  - Processors updated to reference parameters instead of variables
- This is still a NiFi 1 flow definition, so do not load this file directly into NiFi 2. Use an empty NiFi 1 instance to review the modification made by the tool. The flow will appear unchanged, but variables will be replaced with parameters of the same name, which are now referenced by the processor instead.
- Changes in the syntax: Variables were previously referenced using ${}, whereas parameters are now referenced using #{}.
  - **NiFi 1 variables**: ${variable_name}
  - **NiFi 2 parameters**: #{parameter_name}

activity_log.json

- Log of all actions performed during this stage of the migration.
- The following were changes made during the template migration:
  - A new parameter context was created with a new parameter in it.
  - The parameter context was assigned to the process group.
  - The PutElasticsearchJson processor was updated to reference the new parameter.
- Components are referenced by a unique ID, not by name.

2. Validate the Stage 1 template migration output.
   a) Check the new flow definition in a NiFi 1 instance to verify that the flow matches your expectations.
      1. Start your NiFi 1 instance.
      2. Create a new process group.
      3. Load elastic_template.json to NiFi 1.
      4. Confirm that variables were correctly converted to parameters.
   b) Review activity_log.json and address any manual-validation-requests or manual-validation-requests.
   c) If there are manual-change-requests or manual-validation-requests to handle, follow these steps:
      1. Make the modifications on the NiFi canvas, indicated by the manual-change-requests or manual-validation-requests in the sourceVersion/activity_log.json.
      2. Fetch the flow.json.gz file from the NiFi conf directory.
      3. Right-click on the Elastic process group.
      4. Select Download flow definition without external services.
      5. Save the file as Elastic_Listener.json and overwrite the one in the sourceVersion/<process_group>folder.
   d) If no requests are present in the log, proceed to the next step.
3. Run full template migration (Stage 1 and 2), using the following command.

```
bin/migration.sh nifi migrate-templates \
-i /etc/migration-tool-input/Elastic_template.xml \
-od /etc/migration-tool-output/templates
```

The contents of the previously generated sourceVersion folder will be overwritten, but this is not a concern as the tool generates the same sourceVersion output as before. Additionally, a targetVersion directory is created, containing the output files of the Stage 2 part of the migration.

```
targetVersion
### Elastic_template_b4175f57-0194-1000-8470-9251a24519b4
#   ### Elastic_template.json
### activity_log.json
```

Elastic_template.json

- NiFi 2-compatible flow definition
- Contains the contents of the original template converted into an exported process group.
- This version is compatible with NiFi 2, but no longer supports NiFi 1.



Note:

An elastic_template.xml is not generated as NiFi 2 does not support XML templates.

activity_log.json

- List of all actions performed during this stage of the migration.

4. Validate the Stage 2 template migration output.
   a) Check the new flow definition in a NiFi 2 instance to verify that the flow matches your expectations.
      1. Start your NiFi 2 instance.
      2. Create a new process group.
      3. Load elastic_template.json into a NiFi 2 instance.

         The new process group will be called **elastic_template** and will contain another process group named **elastic**, matching the name of the process group that was converted into a template in NiFi 1 before you started the migration.
      4. Verify parameter replacements and processor updates.
   b) Review activity_log.json and address any manual-change-requests or manual-validation-requests.

      In this case, you can see the following manual-change-request:

```
{
 "sequence" : 4,
 "type" : "manual-change-request",
 "subject" : "a1468d69-3f30-3f83-a7c1-91dad09890f7",
 "message" : "New relationship [original] has been added, it has to be
 connected to a downstream processor or terminated.",
 "context" : {
   "rule" : "6501a693-b524-4d1a-b11e-69747c2651e8"
}
```

      1. Open the targetVersion/Elastic_template.json file and search for the "subject" ID, a1468d69-3f30-3f83-a7c1-91dad09890f7. This ID refers to the PutElasticsearchJson processor.
      2. Go to the NiFi 2 canvas and check the processor. You will find that it has an unbound “original” relationship that needs to be connected to a downstream component.
      3. Make the required change manually on the canvas.
      4. Once done, export the process group. This exported process group is now a fully NiFi 2-compatible version of the original template.
      5. Save the file.

Results

You have finished the template migration process.

Migrating variables using template.xml as input

Learn how to use the Cloudera Flow Migration Tool to convert variables to parameters and parameter contexts for template files.

Example flows for migrating variables

The following NiFi flow is used to demonstrate both variable and component migration. It shows how variables are used within process groups and how they are referenced by individual components.

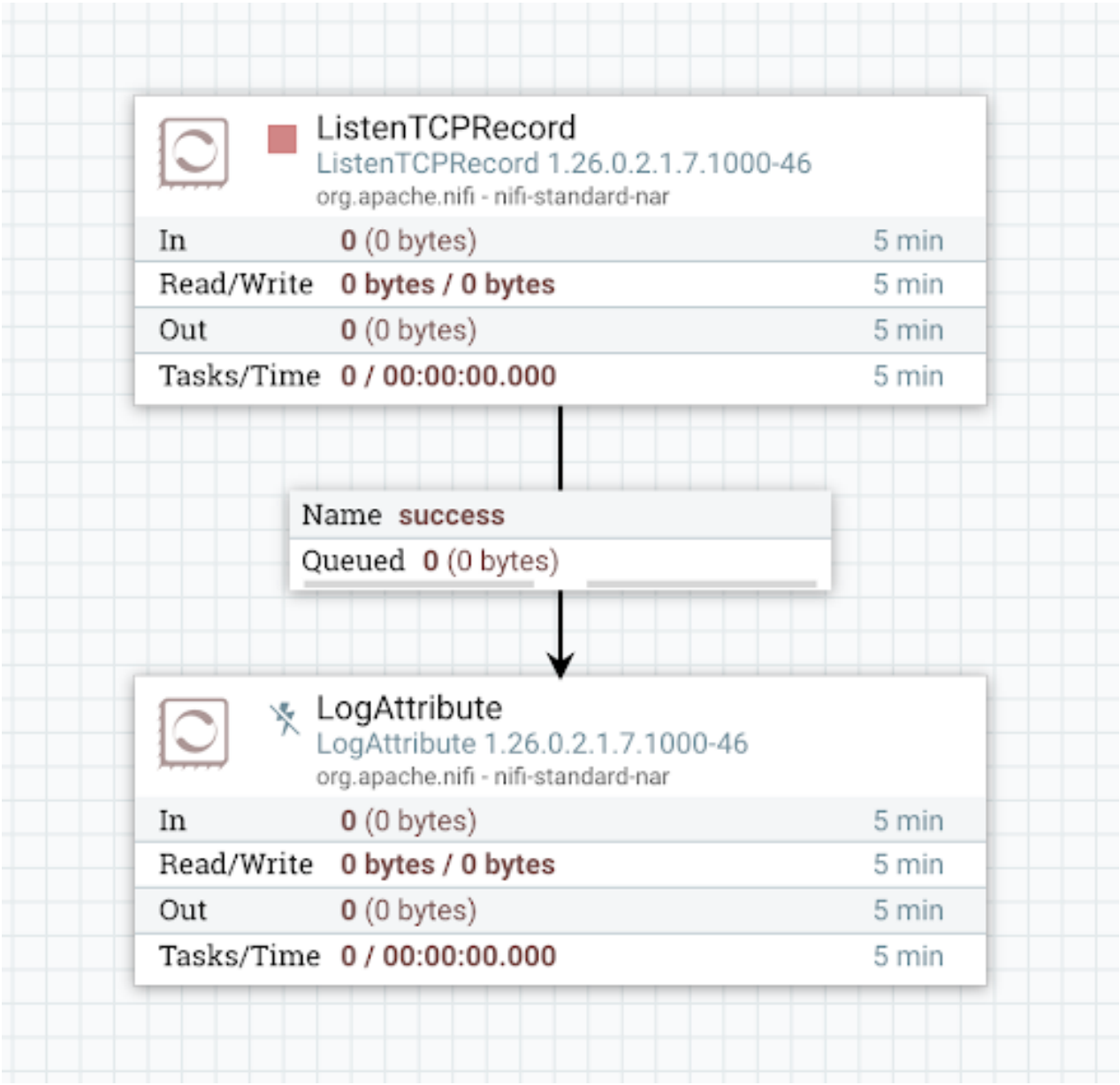
Flow definition file:

- **TCP_Listener_template.xml**

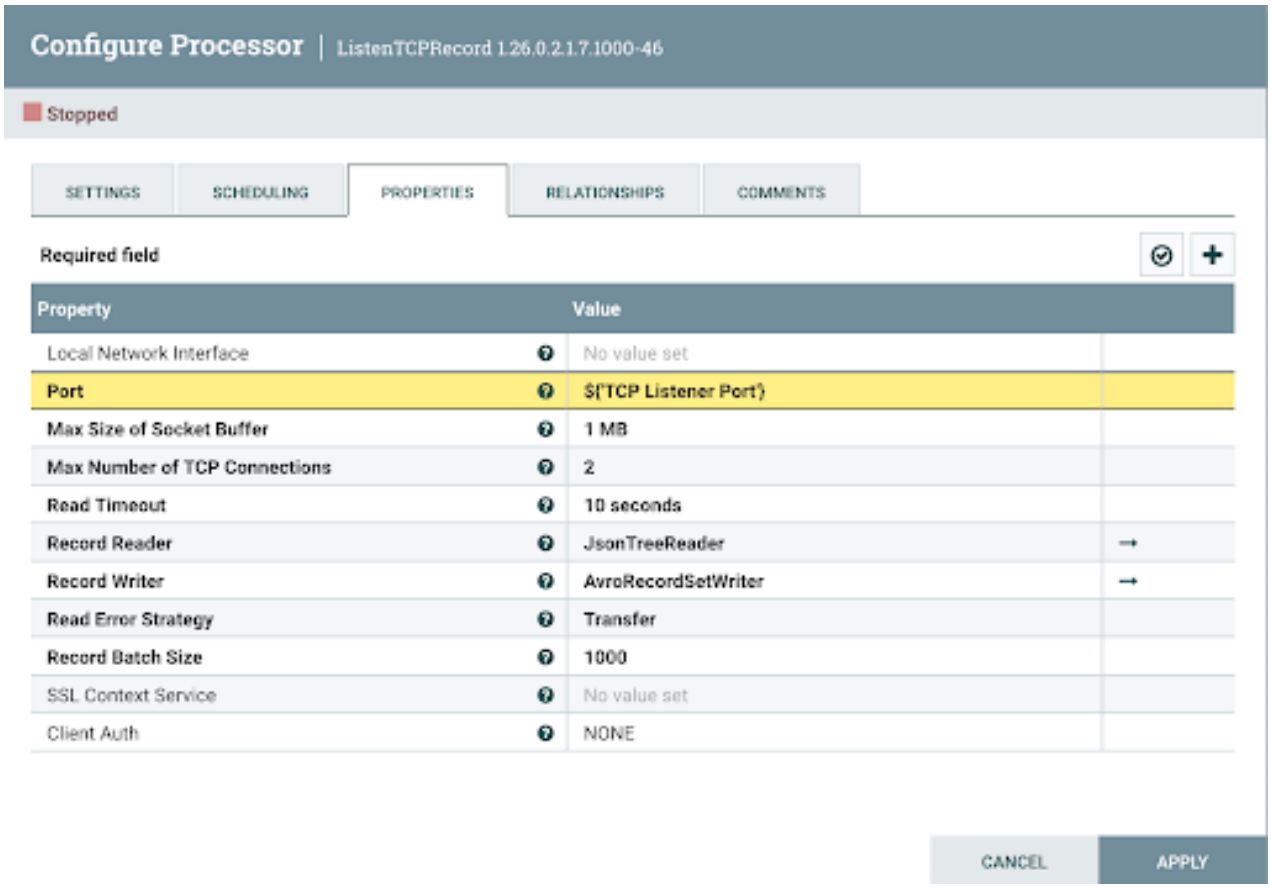
  The file consists of process group TCP Listener (ID: b41940d7-0194-1000-42fc-458834630567).

TCP Listener Process Group

This process group contains the following simple flow:



The process group defines a variable called **TCP Listener Port**, which is referenced by the ListenTCPRecord processor.



Note: NiFi 2 supports only parameters, not variables, so variable migration is required to ensure compatibility with the target version.

The example guides you through variable migration and maintaining a clear activity log. While this example template is simple, the step-by-step approach shows how this method improves clarity for more complex migrations. In real-world scenarios, you have to define a migration strategy based on your flow's structure. Individual template migration may not always be necessary.

Before you begin

Copy the TCP_Listener_template.xml file to the Migration Tool's input folder (/etc/migration-tool-input).

Procedure

1. Run Stage 1 variable migration on the TCP_Listener_template.xml file, using the following command.

```
bin/migration.sh nifi migrate-variables \
-i /etc/migration-tool-input/TCP_Listener_template.xml \
-od /etc/migration-tool-output/variables \
--sourceCompatibleOutput
```

This generates a sourceVersion folder that contains the output files of the Stage 1 migration.

```
variables
### sourceVersion
    ### activity_log.json
    ### migrated_output
```

```
### migrated_TCP_Listener_template.json
```

activity_log.json

- Log of all actions performed during this stage of the migration.
- The following were changes made during the template migration:
  - A new parameter context was created with a new parameter called **TCP Listener Port**, which replaces the corresponding variable.
  - The **TCP Listener Port** variable was removed.

migrated_TCP_Listener_template.json

- A modified NiFi 1 flow, which is not compatible with NiFi 2 yet.
- It contains everything the original flow did, except the **TCP Listener** process group now references a parameter instead of the removed variable.
- This is still a NiFi 1 flow definition, so do not load this file directly into NiFi 2. Use an empty NiFi 1 instance to review the modification made by the tool. The flow definition will appear unchanged, but variables will be replaced with parameters of the same name, which are now referenced by the processor instead.
- Changes in syntax: Variables were previously referenced using ${}, whereas parameters are now referenced using #{}.
  - **NiFi 1 variables**: ${variable_name}
  - **NiFi 2 parameters**: #{parameter_name}



Note:

The file name is composed of the migrated prefix and the name of the top-level process group and not the original file name.

2. Validate the Stage 1 variable migration output for the migrated_TCP_Listener_template.json file.
   a) Check the new flow definition in a NiFi 1 instance to verify that the flow matches your expectations.
      1. Start your NiFi 1 instance.
      2. Create a new process group.
      3. Load sourceVersion/migrated_output/migrated_TCP_Listener_template.json to NiFi 1.
      4. Remove the ‘migrated_‘ prefix from the process group name.
      5. Confirm that variables were correctly converted to parameters.
   b) Review the activity_log.json file and check for any manual-change-requests or manual-validation-requests.
   c) If there are manual-change-requests or manual-validation-requests to handle, follow these steps:
      1. Make the modifications on the NiFi canvas, indicated by the manual-change-requests or manual-validation-requests in the sourceVersion/activity_log.json.
      2. Right-click on the TCPListener process group
      3. Select Download flow definition without external services.
      4. Save the file as migrated_TCP_Listener_template.json and overwrite the one in the sourceVersion/migrated_output folder.
   d) If no requests are present in the log, proceed to the next step.

   At this stage, the flow definition no longer contains variables and uses parameters instead. If the flow meets your expectations, you can either run a full variable migration to validate your flow in NiFi 2 or proceed with migrating the components.
3. Run full variable migration (Stage 1 and 2) on the TCP_Listener_template.
   a) Copy the migrated_TCP_Listener_template.json file from Step 2 into the input folder (/etc/migration-tool-input) and rename it to TCP_Listener_template.json for clarity.
   b) Make a backup of the output folder (/etc/migration-tool-output/variables) before running the next migration step.

Note: This step is crucial because the same output folder is used as in the previous migration, and its contents will be overwritten. To retain a record of changes made at each stage, it is better to create a backup.

c) Run the following command.

```
bin/migration.sh nifi migrate-variables \
-i /etc/migration-tool-input/TCP_Listener_template.json \
-od /etc/migration-tool-output/variables
```

The contents of the previously generated sourceVersion folder will be overwritten, but this is not a concern as the tool generates the same sourceVersion output as before.

Additionally, a targetVersion directory is created, containing the output files of the Stage 2 part of the migration.

migrated_TCP_Listener_template.json

- NiFi 2-compatible flow in terms of variables
- You can load it into NiFi 2 and validate it

activity_log.json

- List of all actions performed during this stage of the migration.

4. Proceed with component migration using the input folder from step 3.

Migrating components using a template with variables

Learn how to use the Cloudera Flow Migration Tool to migrate components from flow definition JSON files. As templates may not contain some parameter contexts and parameters defined at the parent process group level, you have to use the flow definitions generated in the previous steps.

Example flow for migrating components

The following NiFi flow is used to demonstrate both variable and component migration.

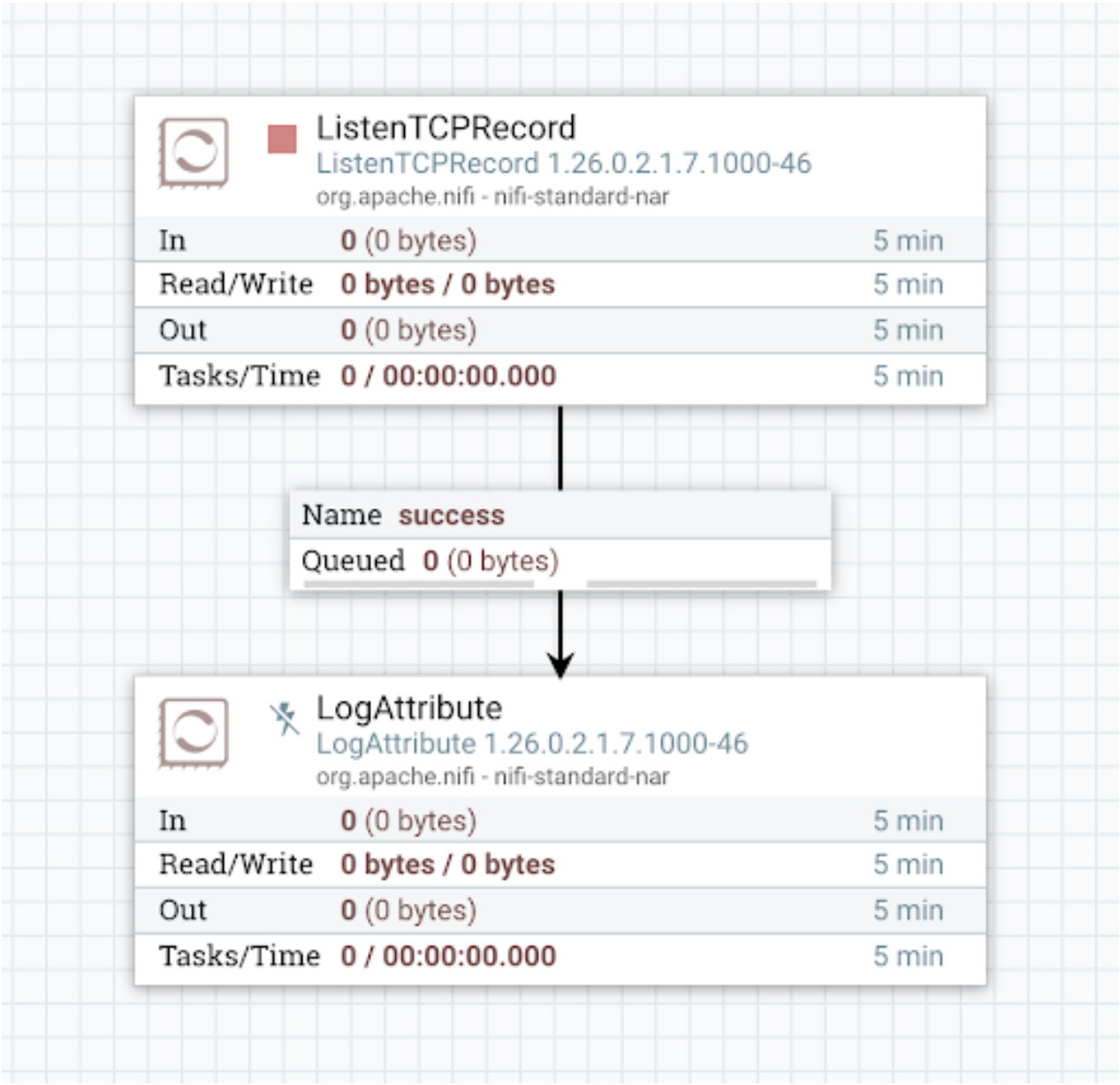
There is one flow definition file from the variable migration step:

- **TCP_Listener_template.json**

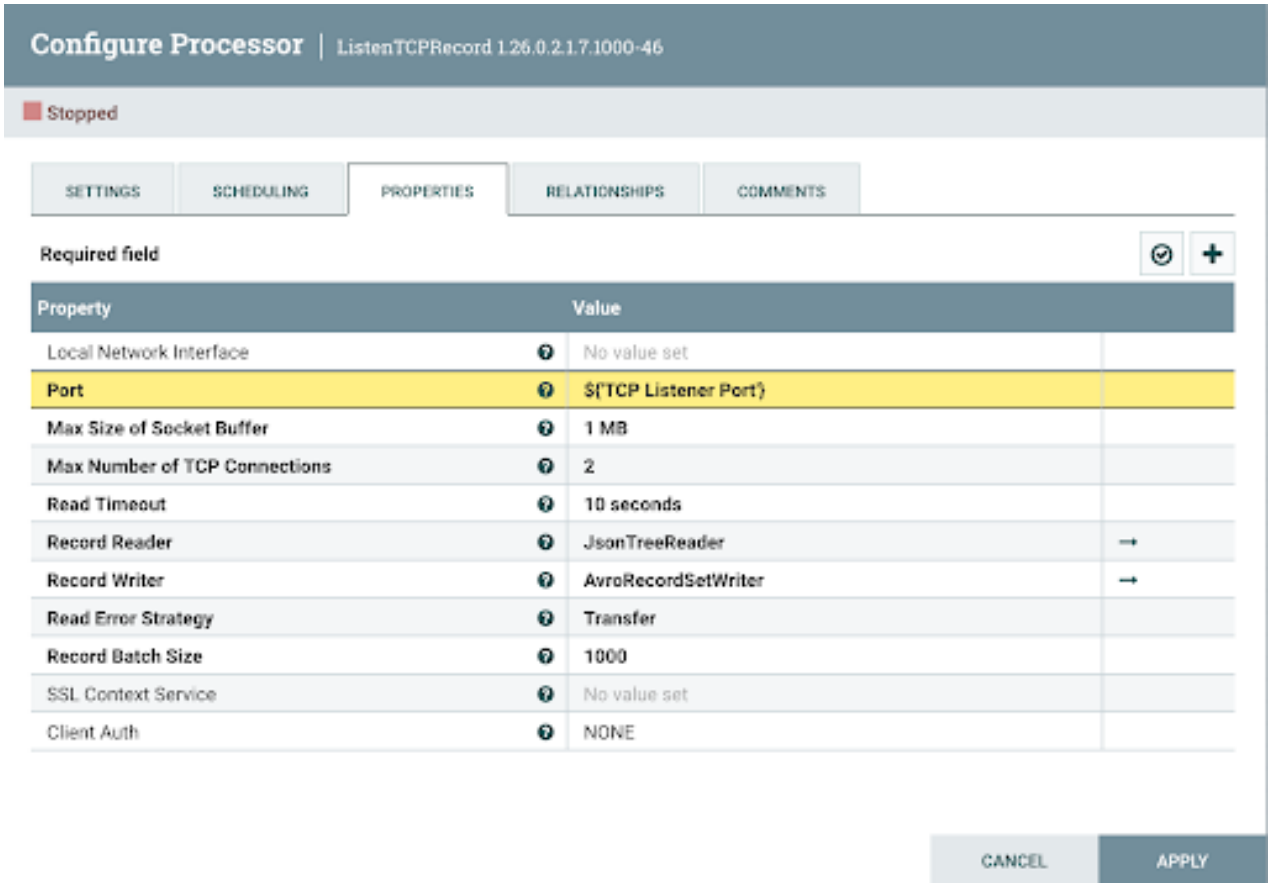
  The file consists of the process group TCP Listener (ID: b41940d7-0194-1000-42fc-458834630567).

TCP Listener Process Group

This process group contains the following simple flow:



The process group defines a variable called **TCP Listener Port**, which is referenced by the ListenTCPRecord processor.



The example guides you through component migration and maintaining a clear activity log. While this example flow definition is simple, the step-by-step approach shows how this method improves clarity for more complex migrations. In real-world scenarios, you have to define a migration strategy based on your flow's structure.

Procedure

1. Use the flow definition file in the /etc/migration-tool-input from step 3.

   This is a NiFi-1 compatible flow definition that no longer contains variables.
2. Run Stage 1 component migration on the TCP_Listener_template.json process group using the following command.

   ```
   bin/migration.sh nifi migrate-components \
   -i /etc/migration-tool-input/TCP_Listener_template.json \
   -od /etc/migration-tool-output/components \
   --sourceCompatibleOutput
   ```

   This generates a sourceVersion folder that contains the output files of the Stage 1 migration.

   ```
   components
   ### sourceVersion
       ### activity_log.json
       ### migrated_output
           ### migrated_TCP_Listener_template.json
   ```

   activity_log.json

   - The log describes all the actions that were performed for this stage of the process group migration.
   - Log of all actions performed during this stage of the process group migration.

migrated_TCP_Listener_template.json

- A modified NiFi 1 flow, which is not compatible with NiFi 2 yet.
- It contains everything the original flow did, except the **TCP Listener** process group was modified as described in the Activity Log.

3. Validate the Stage 1 component migration output for migrated_TCP_Listener_template.json.
   a) Load the migrated_TCP_Listener_template.json into a NiFi 1 instance and check the flow.
   b) Review the activity_log.json file and check for any manual-change-requests or manual-validation-requests. If none are present, proceed to the next step.

      In this example, you can see the following information in the activity log:

      ```
      {
       "sequence" : 2,
       "type" : "change-info",
       "subject" : "b41966ad-0194-1000-a08d-a92489457356",
       "message" : "Component [org.apache.nifi.processors.standard.ListenTCPRecord] has been deprecated (NIFI-13509)",
       "context" : {
         "rule" : "36227b60-75f0-40dc-8caf-a2ec577aa54c"
      }
      ```

      1. **a.** Open the targetVersion/migrated_TCP_Listener_template.json file and search for the "subject" ID, b41966ad-0194-1000-a08d-a92489457356. This ID refers to the ListenTCPRecord processor.
         b. Search for the value of the identifier as “subject” element in NiFi’s search box.

         You are directed to the ListenTCPRecord processor. No manual modifications are needed at this stage. However, it is important to note that the ListenTCPRecord processor is deprecated and is not available in NiFi 2. Once the full component migration (Stage 1 and Stage 2) is complete, instructions will be provided on how to handle this deprecation.
   c) If manual changes are necessary, update the migrated_TCP_Listener_template.json on the NiFi canvas after loading it. Once the flow is validated and meets expectations, continue with the next step.

   At this stage, you have completed Stage 1 of both variable and component migration for your flow definition file. After reviewing the logs, you confirmed that no manual changes were needed. You can proceed with a full component migration using the /etc/migration-tool-input from Step 3.
4. Run full component migration (Stage 1 and 2) for TCP_Listener_template.json.
   a) Make a backup of the output folder (/etc/migration-tool-output/components) before running the next migration step.

      Note:

      This step is crucial because the same output folder is used as in the previous migration, and its contents will be overwritten. To retain a record of changes made at each stage, it is better to create a backup.
   b) Run the full component migration using the following command.

      ```
      bin/migration.sh nifi migrate-components \
      -i /etc/migration-tool-input/TCP_Listener_template.json \
      -od /etc/migration-tool-output/components
      ```

      This generates the following output:

      ```
      components
      ### sourceVersion
      #   ### activity_log.json
      #   ### migrated_output
      # ### migrated_TCP_Listener_template.json
      ### targetVersion
          ### activity_log.json
          ### migrated_output
      ```

```
### migrated_TCP_Listener_template.json
```

The contents of the previously generated sourceVersion folder will be overwritten. The contents represent the NiFi 1-compatible version of the migrated flow. Since the root process group only contained the two process groups on which you already performed Stage 1 component migration, the activity log will only include the same entries as those in Steps 2 and 4.

Additionally, a targetVersion directory is created, containing the output files of the Stage 2 part of the migration:

migrated_TCP_Listener_template.json

- NiFi 2-compatible flow in terms of variables
- You can load it into NiFi 2 and validate it

activity_log.json

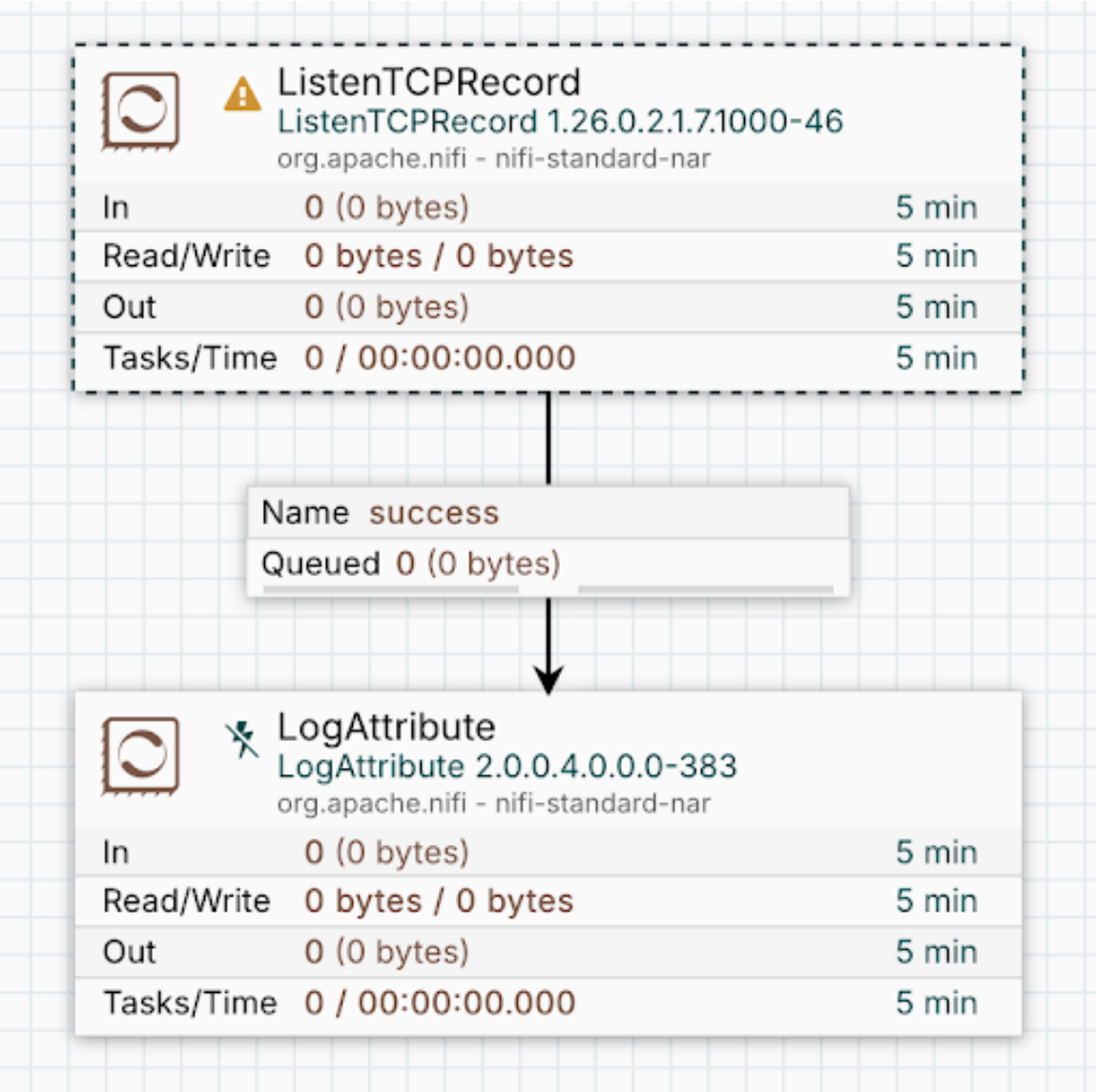
- List of all actions performed during this stage of the migration, and any manual steps that you need to perform.

5. Validate the Stage 2 component migration output for migrated_TCP_Listener_template.json.
   a) Load the targetVersion/migrated_output/migrated_TCP_Listener_template.json into a NiFi 2 instance and check the flow.
   b) Review the targetVersion/activity_log.json file.

   In this example, you can see the following manual-change-request entry.

```
{
 "sequence" : 6,
 "type" : "manual-change-request",
 "subject" : "b41966ad-0194-1000-a08d-a92489457356",
 "message" : "The component has been deprecated. It is suggested to r
eplace it with a combination of [org.apache.nifi.processors.standard.Lis
tenTCP] and [org.apache.nifi.processors.standard.ConvertRecord] processo
rs",
 "context" : {
   "rule" : "36227b60-75f0-40dc-8caf-a2ec577aa54c"
}
```

   1. Open the targetVersion/migrated_TCP_Listener_template.json file and search for the "subject" ID, b41966ad-0194-1000-a08d-a92489457356. This ID refers to the ListenTCPRecord processor.
   2. To identify the processor affected by these changes, search for the identifier value of the “subject” element in NiFi’s search box. It refers to the ListenTCPRecord processor.
   3. Open the TCP Listener process group. You can see that the processor is marked with dashed borders and its version number still shows that of the NiFi 1 instance. It means that it is a "ghost processor", not available in NiFi 2. The log message suggests replacing it with a combination of ListenTCP and ConvertRecord processors.



4. Instantiate and configure the ListenTCP and ConvertRecord processors, connect them appropriately, and then remove the ghost processor.

   Besides these changes there are no further manual-change-requests or manual-validation-requests in the activity log.

c) Download the flow definition and you have the fully NiFi 2 compatible TCP_Listener_template.json.

Migrating components using a template without variables

Learn how to use the Cloudera Flow Migration Tool to migrate components from template.xml files that do not contain variables. Component migration with template.xml input can only be used if the flow does not contain any variables. If the template contains variables, follow the *Migrate components using template with variables* section.

Example flow for migrating components

The following NiFi flow is used to demonstrate both variable and component migration.

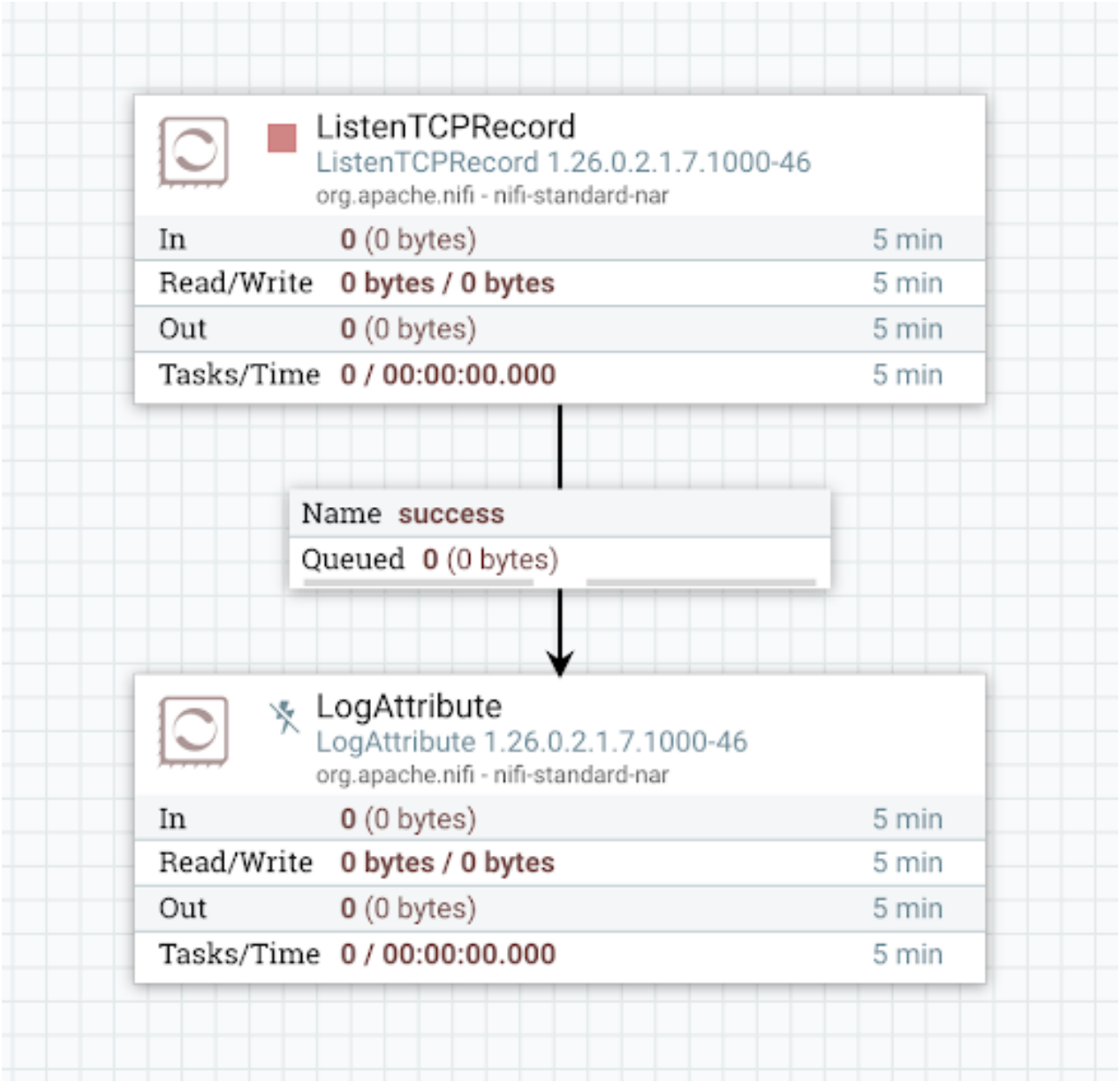
There is one template.xml file:

- **TCP_Listener_template.xml**

  The file consists of the process group TCP Listener (ID: b41940d7-0194-1000-42fc-458834630567).

TCP Listener Process Group

This process group contains the following simple flow:



The ListenTCPRecord processor does not reference any variables.

The example guides you through component migration and maintaining a clear activity log. While this example template is simple, the step-by-step approach shows how this method improves clarity for more complex migrations. In real-world scenarios, you have to define a migration strategy based on your flow's structure.

Procedure

1. Copy the TCP_Listener_template.xml file to the /etc/migration-tool-input folder.

   This is a template without any variables.
2. Run Stage 1 component migration on the TCP_Listener_template.xml file using the following command.

```
bin/migration.sh nifi migrate-components \
-i /etc/migration-tool-input/TCP_Listener_template.xml \
```

```
-od /etc/migration-tool-output/components \
--sourceCompatibleOutput
```

This generates a sourceVersion folder that contains the output files of the Stage 1 migration.

```
components
### sourceVersion
    ### activity_log.json
    ### migrated_output
        ### migrated_TCP_Listener_template.json
```

activity_log.json

- The log describes all the actions that were performed for this stage of the process group migration.
- Log of all actions performed during this stage of the process group migration.

migrated_TCP_Listener_template.json

- A modified NiFi 1 flow, which is not compatible with NiFi 2 yet.
- It contains everything the original flow did, except the **TCP Listener** process group was modified as described in the Activity Log.

3. Validate the Stage 1 component migration output for TCP_Listener_flow_definition.json.
   a) Load the migrated_TCP_Listener_template.json into a NiFi 1 instance and check the flow.
   b) Review the activity_log.json file and check for any manual-change-requests or manual-validation-requests. If none are present, proceed to the next step.

      In this example, you can see the following information in the Activity Log:

      ```
      {
       "sequence" : 2,
       "type" : "change-info",
       "subject" : "b41966ad-0194-1000-a08d-a92489457356",
       "message" : "Component [org.apache.nifi.processors.standard.ListenTCPR
      ecord] has been deprecated (NIFI-13509)",
       "context" : {
         "rule" : "36227b60-75f0-40dc-8caf-a2ec577aa54c"
      }
      ```

      Search for the value of the “subject” element using the NiFi search box.

      This will locate the ListenTCPRecord processor. No manual modifications are needed at this point. However, note that the ListenTCPRecord processor is deprecated and not available in NiFi 2. After completing both stages of the component migration process, follow-up instructions outline how to address this deprecation.
   c) If manual changes are necessary, update the migrated_TCP_Listener_template.json on the NiFi canvas after loading it. Once the flow is validated and meets expectations, continue with the next step.
4. Run full component migration (Stage 1 and 2) for TCP_Listener_template.json.
   a) Copy the migrated_TCP_Listener_template.json file and rename migrated_TCP_Listener_template.json to TCP_Listener_template.json for clarity.
   b) Make a backup of the output folder (/etc/migration-tool-output/components) before running the next migration step.

      Note: step is crucial because the same output folder is used as in the previous migration, and its contents will be overwritten. To retain a record of changes made at each stage, it is better to create a backup.
   c) Run the full component migration using the following command.

      ```
      bin/migration.sh nifi migrate-components \
      -i /etc/migration-tool-input/TCP_Listener_template.json \
      ```

```
-od /etc/migration-tool-output/components
```

This generates the following output:

```
components
### sourceVersion
#   ### activity_log.json
#   ### migrated_output
# ### migrated_TCP_Listener_template.json
### targetVersion
    ### activity_log.json
    ### migrated_output
 ### migrated_TCP_Listener_template.json
```

The contents of the previously generated sourceVersion folder will be overwritten. The contents represent the NiFi 1-compatible version of the migrated flow. Since the root process group only contained the two process groups on which you already performed Stage 1 component migration, the activity log will only include the same entries as those in Steps 2 and 4.

Additionally, a targetVersion directory is created, containing the output files of the Stage 2 part of the migration:

migrated_TCP_Listener_template.json

- NiFi 2-compatible flow in terms of variables
- You can load it into NiFi 2 and validate it

activity_log.json

- List of all actions performed during this stage of the migration, and any manual steps that you need to perform.

5. Validate the Stage 2 component migration output for TCP_Listener_flow_definition.json.
   a) Load the targetVersion/migrated_output/migrated_TCP_Listener_template.json into a NiFi 2 instance and check the flow.
   b) Review the targetVersion/activity_log.json file.

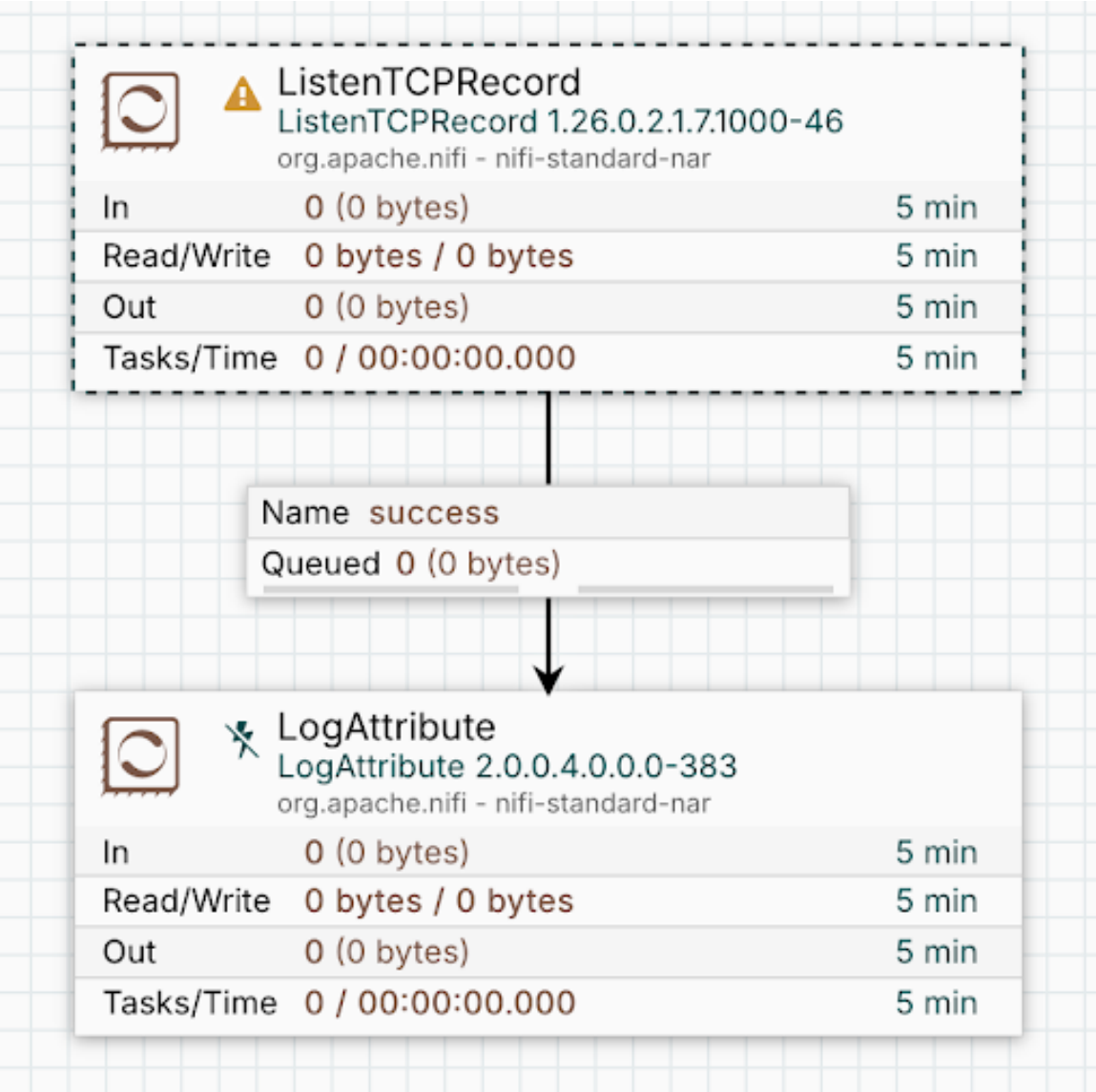
   In this example, you can see the following manual-change-request entry.

```
{
 "sequence" : 6,
 "type" : "manual-change-request",
 "subject" : "b41966ad-0194-1000-a08d-a92489457356",
 "message" : "The component has been deprecated. It is suggested to r
eplace it with a combination of [org.apache.nifi.processors.standard.Lis
tenTCP] and [org.apache.nifi.processors.standard.ConvertRecord] processo
rs",
 "context" : {
   "rule" : "36227b60-75f0-40dc-8caf-a2ec577aa54c"
}
```

   1. To identify the processor affected by these changes, search for the value of the “subject” element in NiFi’s search box.

      It refers to the ListenTCPRecord processor.
   2. Open the TCP Listener process group.

      You can see that the processor is marked with dashed borders and its version number still shows that of the NiFi 1 instance. It means that it is a "ghost processor", not available in NiFi 2. The log message suggests replacing it with a combination of ListenTCP and ConvertRecord processors.



3. Instantiate and configure the ListenTCP and ConvertRecord processors, connect them appropriately, and then remove the ghost processor.

   Besides these changes there are no further manual-change-requests or manual-validation-requests in the Activity Log.

c) Download the flow definition and you have the fully NiFi 2 compatible TCP_Listener_template.json.

Using migrate-all with template.xml as input

Template migration is not applicable for this input type, use the migrate-templates command instead.

Migrating a flow using a directory with template.xml files as input

This example demonstrates how to migrate both variables and components with the Cloudera Flow Management Migration Tool using a **directory that contains multiple template XML files** as input.

Migrating templates using a directory with template.xml files as input

Learn how to use the migrate-templates command to transform downloaded template.xml files for compatibility with NiFi 2, which no longer supports templates. This step is required to ensure successful migration to the target version.

This command combines the functionality of migrate-variables and migrate-components, making it equivalent to migrate-all for templates. While the migrate-templates command simplifies the migration process, for larger flows it is often better to run the variable and component migration steps separately. You may also choose to migrate individual process groups to keep activity logs and validation efforts manageable.

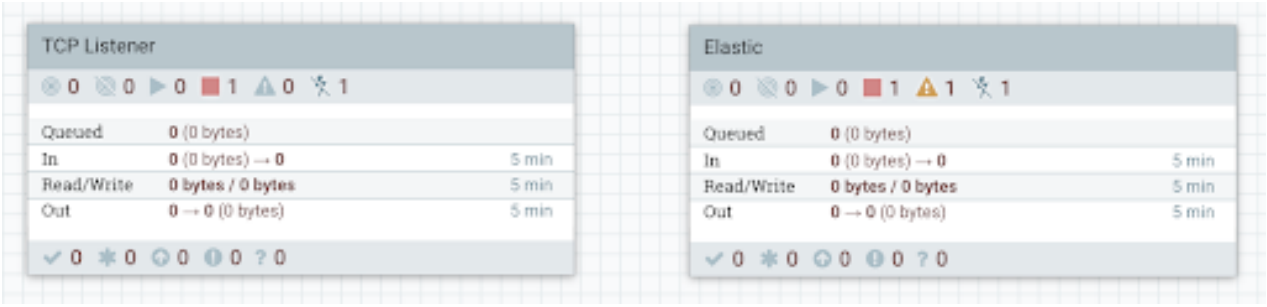
Example flow for migrating a template

This example uses two template files:

- **TCP_Listener_template.xml**

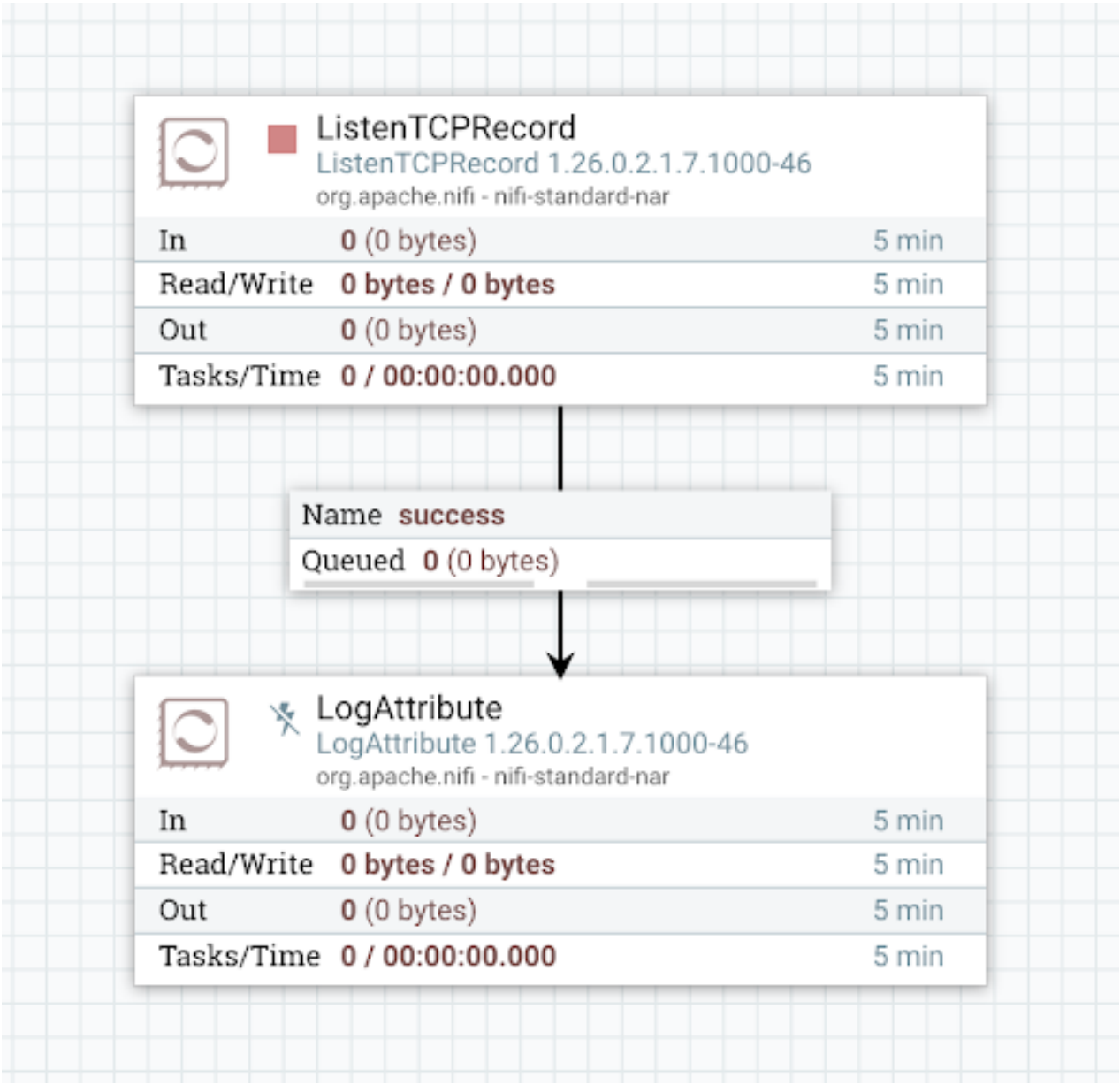
  This file contains process group TCP Listener (ID: b41940d7-0194-1000-42fc-458834630567)
- **Elastic_template.xml**

  This file contains process group Elastic (ID: b42881c7-0194-1000-3cdf-1bd453a0ed0f)

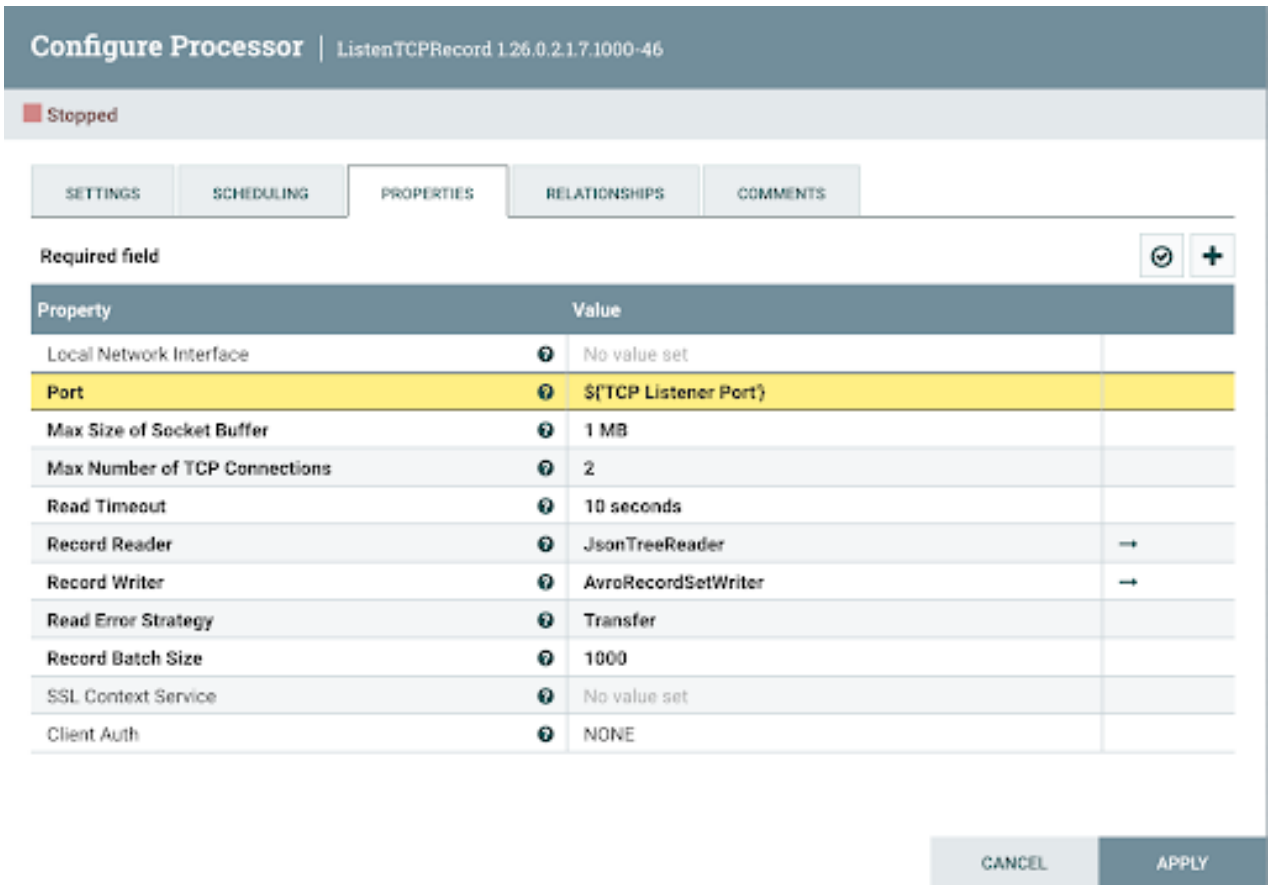


TCP Listener Process Group

This process group contains the following simple flow:

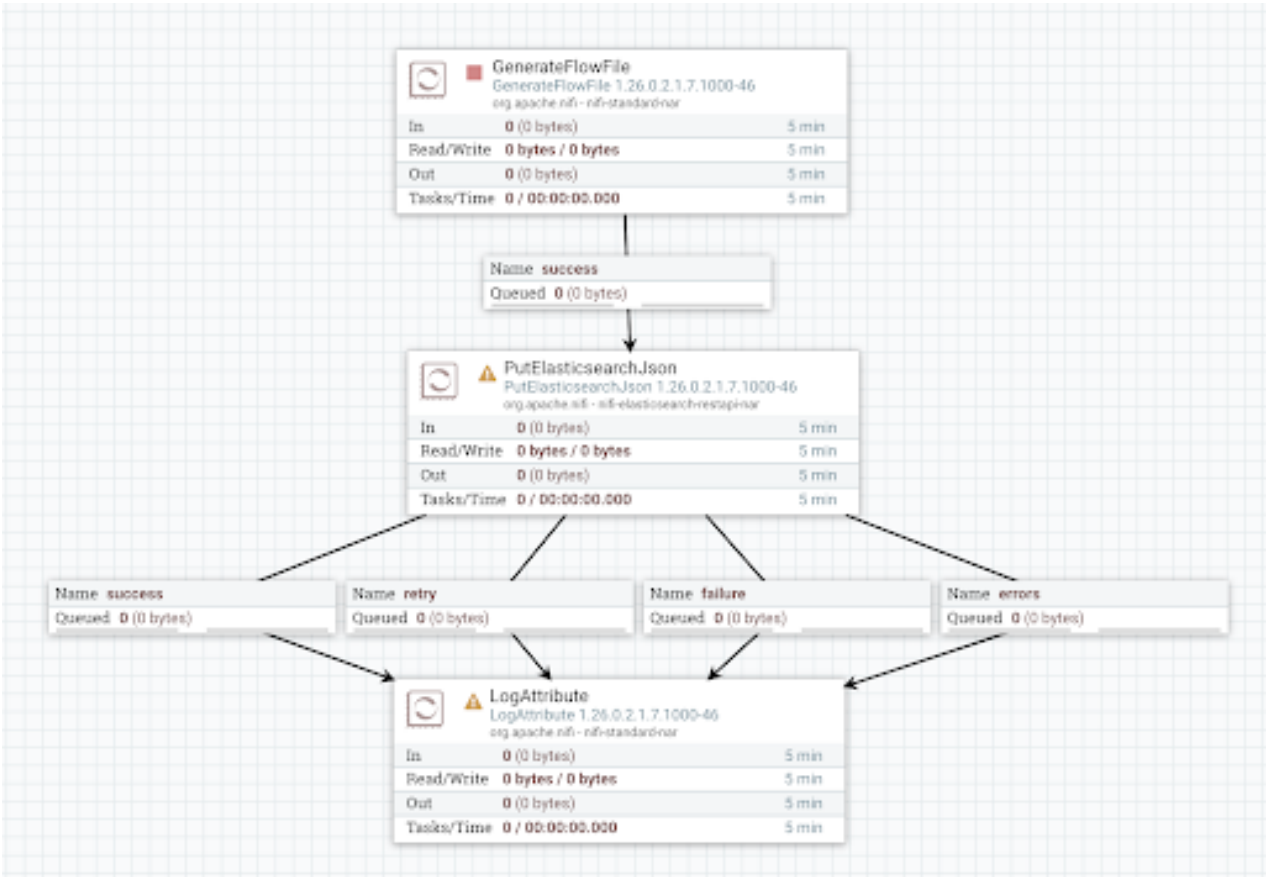


The process group defines a variable called **TCP Listener Port**, which is referenced by the ListenTCPRecord processor.

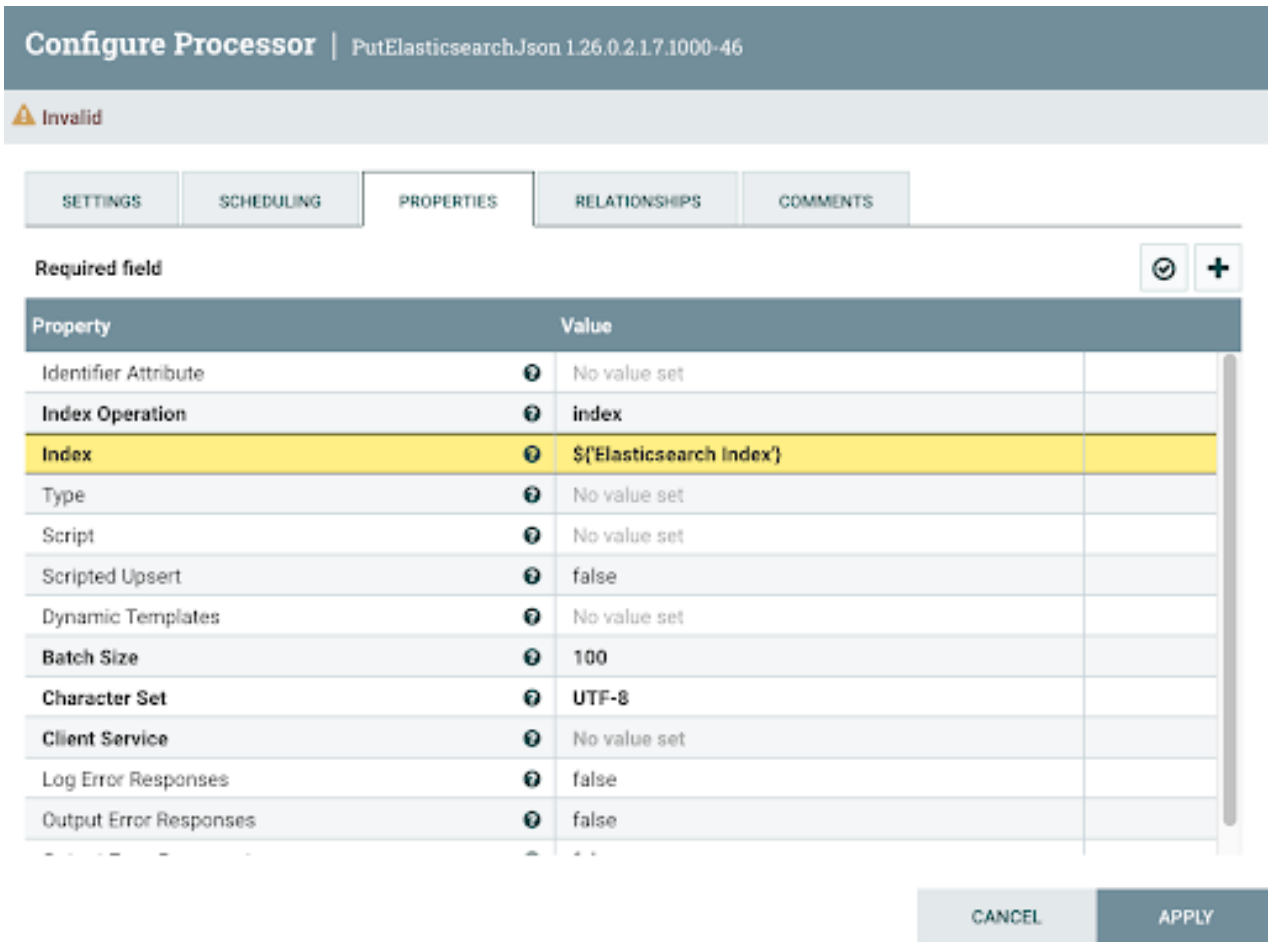


Elastic Process Group

This process group also contains a simple flow:



The process group has a variable, **Elasticsearch Index**, referenced in the Index property of the PutElasticsearchJson processor.



Your goal is to migrate the templates used in Cloudera Flow Management 2.1.7.4000 powered by NiFi 1 to parameters used in Cloudera Flow Management 4.12.0powered by NiFi 2.

Note: Apache NiFi 2 supports only parameters, not variables, so variable migration is required to ensure compatibility with the target version.

The example guides you through variable migration one process group at a time, simplifying the process and maintaining a clear activity log. While these example templates are simple, the step-by-step approach shows how this method improves clarity for more complex migrations. In real-world scenarios, you have to define a migration strategy based on your flow's structure.

Before you begin

Copy the TCP_Listener_template.xml and Elastic_template.xml files to the Migration Tool's input folder (/etc/migration-tool-input).

Procedure

1. Run Stage 1 template migration using the following command.

```
bin/migration.sh nifi migrate-templates \
-i /etc/migration-tool-input \
-od /etc/migration-tool-output/templates \
-sco
```

This generates a sourceVersion folder that contains the output files of the migration.

```
templates
### sourceVersion
    ### Elastic_template_b4175f57-0194-1000-8470-9251a24519b4
    #   ### Elastic_template.json
    ### TCP_Listener_template_b41940d7-0194-1000-42fc-458834630567
    #   ### TCP_Listener_template.json
    ### activity_log.json
```

The folder name Elastic_template_b4175f57-0194-1000-8470-9251a24519b4 and TCP_Listener_template_b41940d7-0194-1000-42fc-45883463056 are temporary process groups which were created during the migration process and used to separate the components in the template from other components that may be present during the migration.

Elastic_template.json

- Flow definition containing the contents of the original template
- Equivalent to manually converting a template to a flow definition in NiFi 1 by:
  a. Instantiating the template on the canvas.
  b. Right-clicking the process group.
  c. Selecting Download flow definition without external services.
- Modified by the Migration Tool to ensure compatibility:
  - Variables (not supported in NiFi 2) converted into parameters
  - Parameter context created to hold the new parameters
  - Processors updated to reference parameters instead of variables
- This is still a NiFi 1 flow definition, so do not load this file directly into NiFi 2. Use an empty NiFi 1 instance to review the modification made by the tool. The flow will appear unchanged, but variables will be replaced with parameters of the same name, which are now referenced by the processor instead.
- Changes in the syntax: Variables were previously referenced using ${}, whereas parameters are now referenced using #{}.
  - **NiFi 1 variables**: ${variable_name}
  - **NiFi 2 parameters**: #{parameter_name}

TCP_Listener_template.json

- A modified flow definition, which is not compatible with NiFi 2 yet.
- It contains everything the original flow definition did, except the TCP Listener process group now references a parameter instead of the removed variable.
- This is still a NiFi 1 flow definition, so do not load this file directly into NiFi 2. Use an empty NiFi 1 instance to review the modification made by the tool. The flow definition will appear unchanged, but variables will be replaced with parameters of the same name, which are now referenced by the processor instead.
- Changes in syntax: Variables were previously referenced using ${}, whereas parameters are now referenced using #{}.
  - **NiFi 1 variables**: ${variable_name}
  - **NiFi 2 parameters**: #{parameter_name}

activity_log.json

- Log of all actions performed during this stage of the migration.
- The following were changes made during the template migration:
  - A new parameter context was created with a new parameter called TCP Listener Port, which replaces the corresponding variable.

- The TCP Listener Port variable was removed.
- A new parameter context was created with a new parameter called Elasticsearch Index, which replaces the corresponding variable of the same name.
- The new parameter is referenced from the PutElasticsearchJson processor.
- The Elasticsearch Index variable was removed.

2. Validate the Stage 1 template migration output for Elastic_template.json.
   a) Check the new flow definition in a NiFi 1 instance to verify that the flow matches your expectations.
      1. Start your NiFi 1 instance.
      2. Create a new process group.
      3. Load elastic_template.json to NiFi 1.
      4. Confirm that variables were correctly converted to parameters.
   b) Review activity_log.json and address any manual-validation-requests or manual-validation-requests.
   c) If there are manual-change-requests or manual-validation-requests to handle, follow these steps:
      1. Make the modifications on the NiFi canvas, indicated by the manual-change-requests or manual-validation-requests in the sourceVersion/activity_log.json.
      2. Right-click on the Elastic process group.
      3. Select Download flow definition without external services.
      4. Save the file as Elastic_template.json and overwrite the one in the sourceVersion/<process_group> folder.
      5. Proceed to running Stage 1 variable migration on the Elastic_flow_definition.json.
   d) Proceed to running Stage 1 variable migration on the TCP_Listener_template.json.
3. Validate the Stage 1 template migration output for TCP_Listener_template.json.
   a) Check the new flow definition in a NiFi 1 instance to verify that the flow matches your expectations.
      1. Create a new process group.
      2. Load TCP_Listener_template.json to NiFi 1.
      3. Confirm that variables were correctly converted to parameters.
   b) Review activity_log.json and address any manual-change-requests or manual-validation-requests.
   c) If there are manual-change-requests or manual-inspection-requests to handle, follow these steps:
      1. Make the modifications on the NiFi canvas, indicated by the manual-change-requests or manual-validation-requests in the sourceVersion/activity_log.json.
      2. Right-click on the TCP Listener process group.
      3. Select Download flow definition without external services.
      4. Save file as TCP_Listener_template.json and overwrite the one in the sourceVersion/<process_group> folder.
   d) If none are present, proceed to the next step.
4. Run full template migration (Stage 1 and 2), using the following command.

```
bin/migration.sh nifi migrate-templates \
-i /etc/migration-tool-input \
-od /etc/migration-tool-output/templates
```

The contents of the previously generated sourceVersion folder will be overwritten, but this is not a concern as the tool generates the same sourceVersion output as before. Additionally, a targetVersion directory is created, containing the output files of the Stage 2 part of the migration.

```
targetVersion
### Elastic_template_b4175f57-0194-1000-8470-9251a24519b4
#   ### Elastic_template.json
### TCP_Listener_template_b41940d7-0194-1000-42fc-458834630567
#   ### TCP_Listener_template.json
### activity_log.json
```

Elastic_template.json

- NiFi 2-compatible flow definition
- Contains the contents of the original template converted into an exported process group.
- This version is compatible with NiFi 2, but no longer supports NiFi 1.

activity_log.json

- List of all actions performed during this stage of the migration.

Note: One activity log file will be created for the entire command.

5. Validate the Stage 2 template migration output.
   a) Check the new flow definition in a NiFi 2 instance to verify that the flow matches your expectations.
      1. Start your NiFi 2 instance.
      2. Create a new process group.
      3. Load elastic_template.json into a NiFi 2 instance.

         The new process group will be called **elastic_template** and will contain another process group named **elastic**, matching the name of the process group that was converted into a template in NiFi 1 before you started the migration.
      4. Verify parameter replacements and processor updates.
   b) Review activity_log.json and address any manual-change-requests or manual-validation-requests.

      In this case, you can see the following manual-change-request:

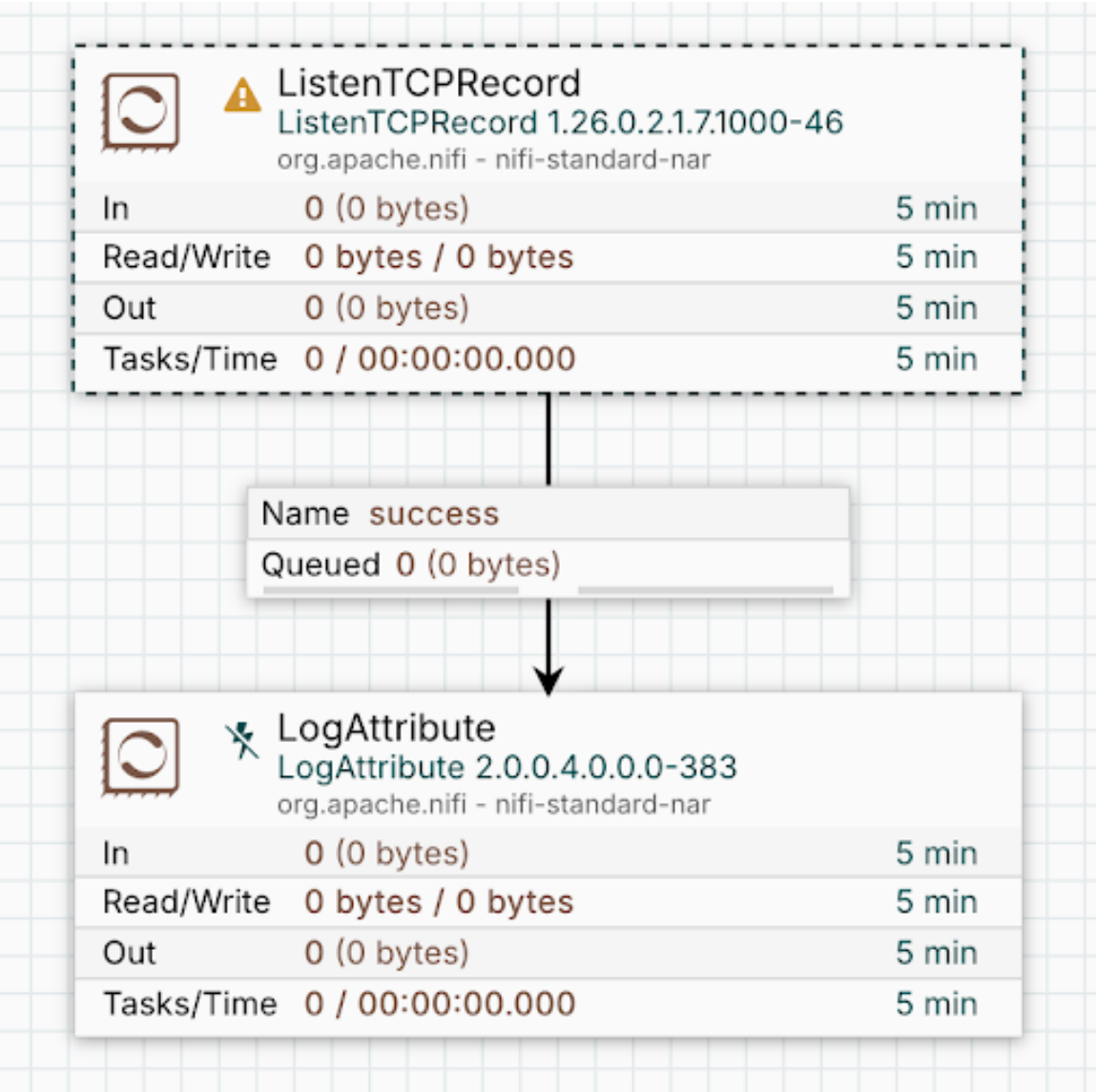
```
{
 "sequence" : 4,
 "type" : "manual-change-request",
 "subject" : "a1468d69-3f30-3f83-a7c1-91dad09890f7",
 "message" : "New relationship [original] has been added, it has to be
 connected to a downstream processor or terminated.",
 "context" : {
   "rule" : "6501a693-b524-4d1a-b11e-69747c2651e8"
}
```

      1. Open the targetVersion/Elastic_template.json file and search for the "subject" ID, a1468d69-3f30-3f83-a7c1-91dad09890f7. This ID refers to the PutElasticsearchJson processor.
      2. Go to the NiFi 2 canvas and check the processor. You will find that it has an unbound “original” relationship that needs to be connected to a downstream component.
      3. Make the required change manually on the canvas.
      4. Once done, export the process group. This exported process group is now a fully NiFi 2-compatible version of the original template.
      5. Save the file.
   c) Validate the Stage 2 component migration output for TCP_Listener_template.json.
   d) Load the targetVersion/TCP_Listener_template.json into a NiFi 2 instance and check the flow.
   e) Review the targetVersion/activity_log.json file. In this example, you can see the following manual-change-re quest entry.

```
{
 "sequence" : 6,
 "type" : "manual-change-request",
 "subject" : "b41966ad-0194-1000-a08d-a92489457356",
 "message" : "The component has been deprecated. It is suggested to r
eplace it with a combination of [org.apache.nifi.processors.standard.Lis
tenTCP] and [org.apache.nifi.processors.standard.ConvertRecord] processo
rs",
 "context" : {
```

```
    "rule" : "36227b60-75f0-40dc-8caf-a2ec577aa54c"
}
```

1. Open the targetVersion/TCP_Listener_template.json file and search for the "subject" ID, b41966ad-0194-1000-a08d-a92489457356. This ID refers to the ListenTCPRecord processor.
2. To identify the processor affected by these changes, search for the identifier value of the “subject” element in NiFi’s search box. It refers to the ListenTCPRecord processor. Open the TCP Listener process group. You can see that the processor is marked with dashed borders and its version number still shows that of the NiFi 1 instance. It means that it is a "ghost processor", not available in NiFi 2. The log message suggests replacing it with a combination of ListenTCP and ConvertRecord processors.



Instantiate and configure the ListenTCP and ConvertRecord processors, connect them appropriately, and then remove the ghost processor.

Besides these changes there are no further manual-change-requests or manual-validation-requests in the Activity Log.

f) Download the flow definition and you have the fully NiFi 2 compatible TCP_Listener_flow_definition.json.

Results

You have finished the template migration process.

Migrating variables using a directory with template.xml files as input

Learn how to use the Cloudera Flow Migration Tool to convert variables to parameters and parameter contexts for template files.

Example flows for migrating variables

The following NiFi flows are used to demonstrate both variable and component migration.

This example uses two template files:

- **TCP_Listener_template.xml**

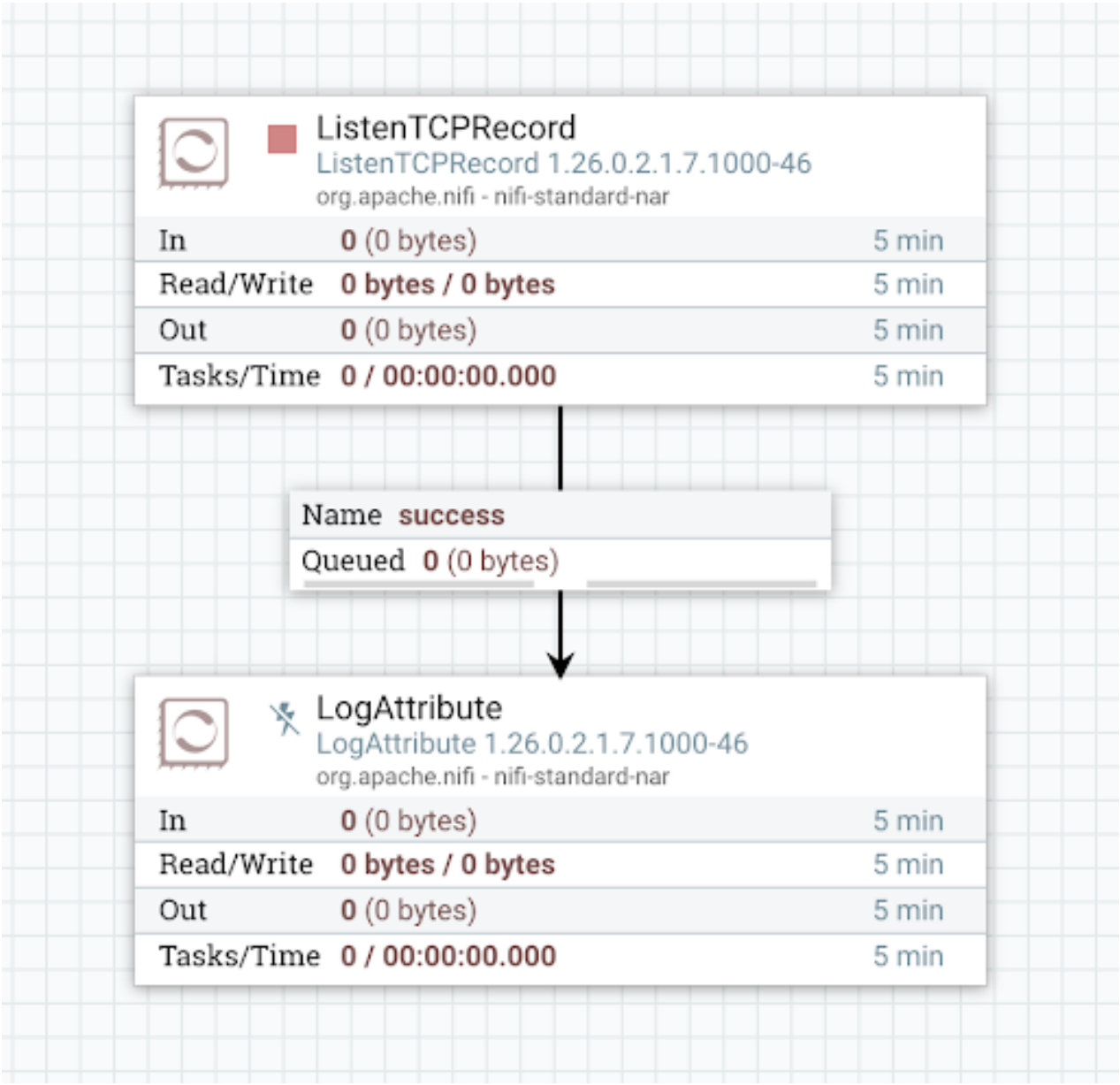
  The file consists of process group TCP Listener (ID: b41940d7-0194-1000-42fc-458834630567).
- **Elastic_flow_template.xml**

  This file contains process group Elastic (ID: b42881c7-0194-1000-3cdf-1bd453a0ed0f).

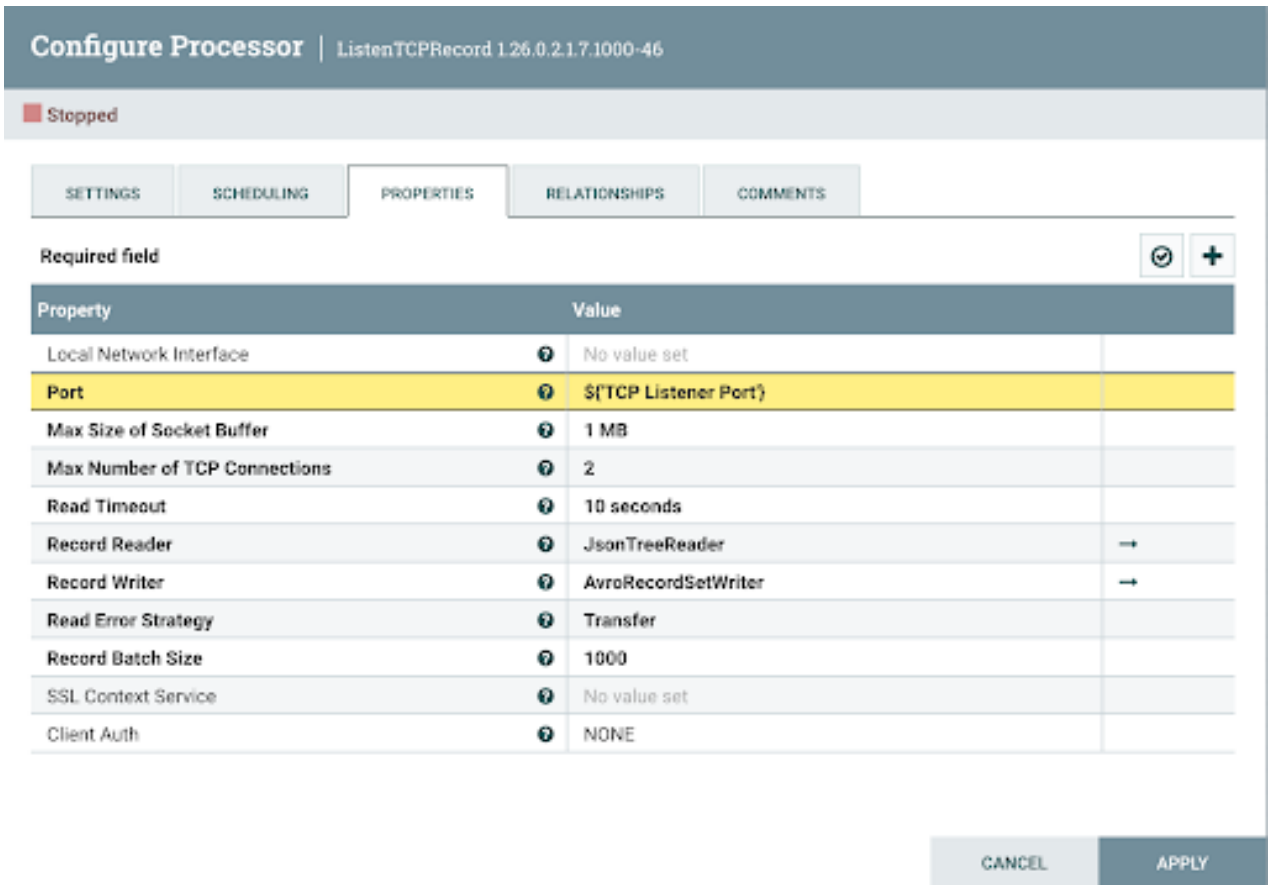


TCP Listener Process Group

This process group contains the following simple flow:

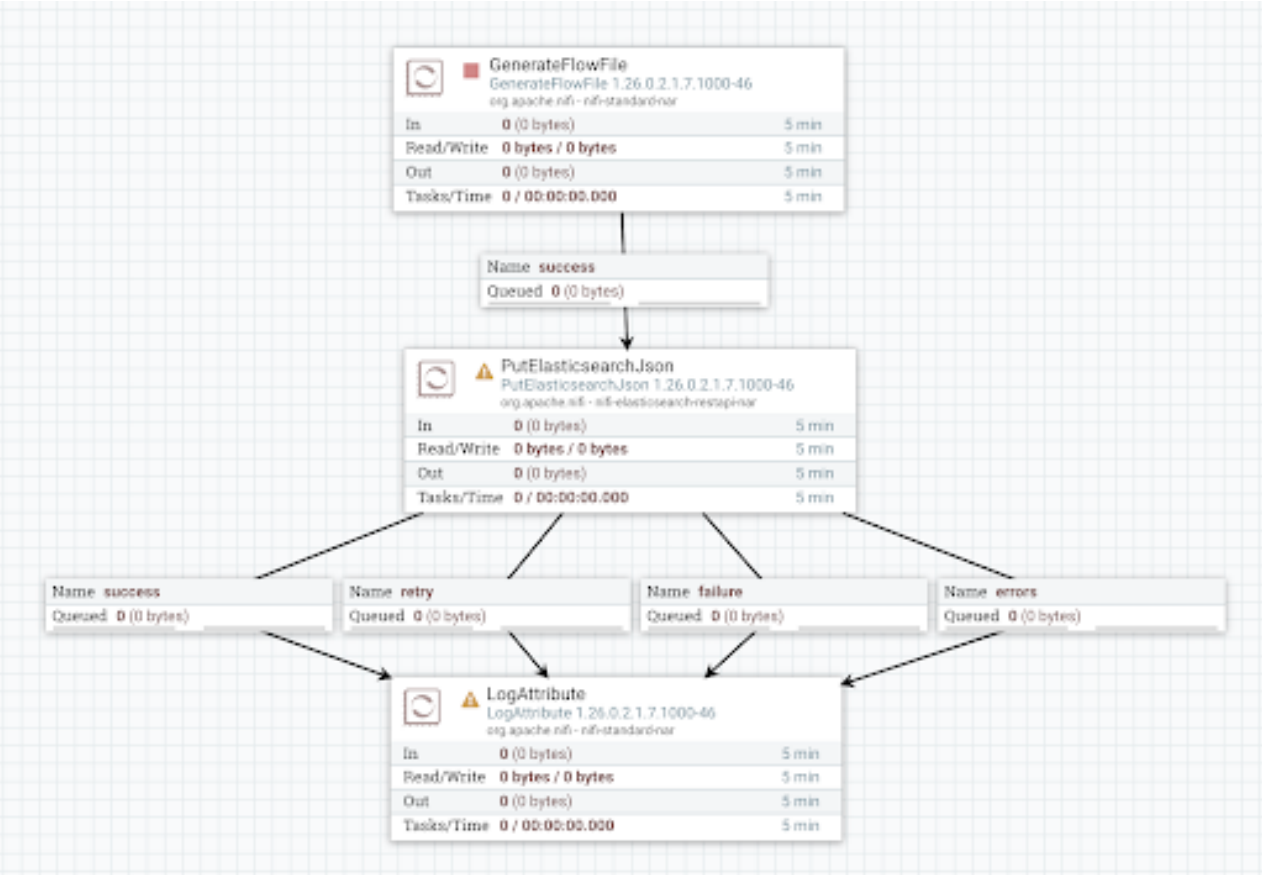


The process group defines a variable called **TCP Listener Port**, which is referenced by the ListenTCPRecord processor.

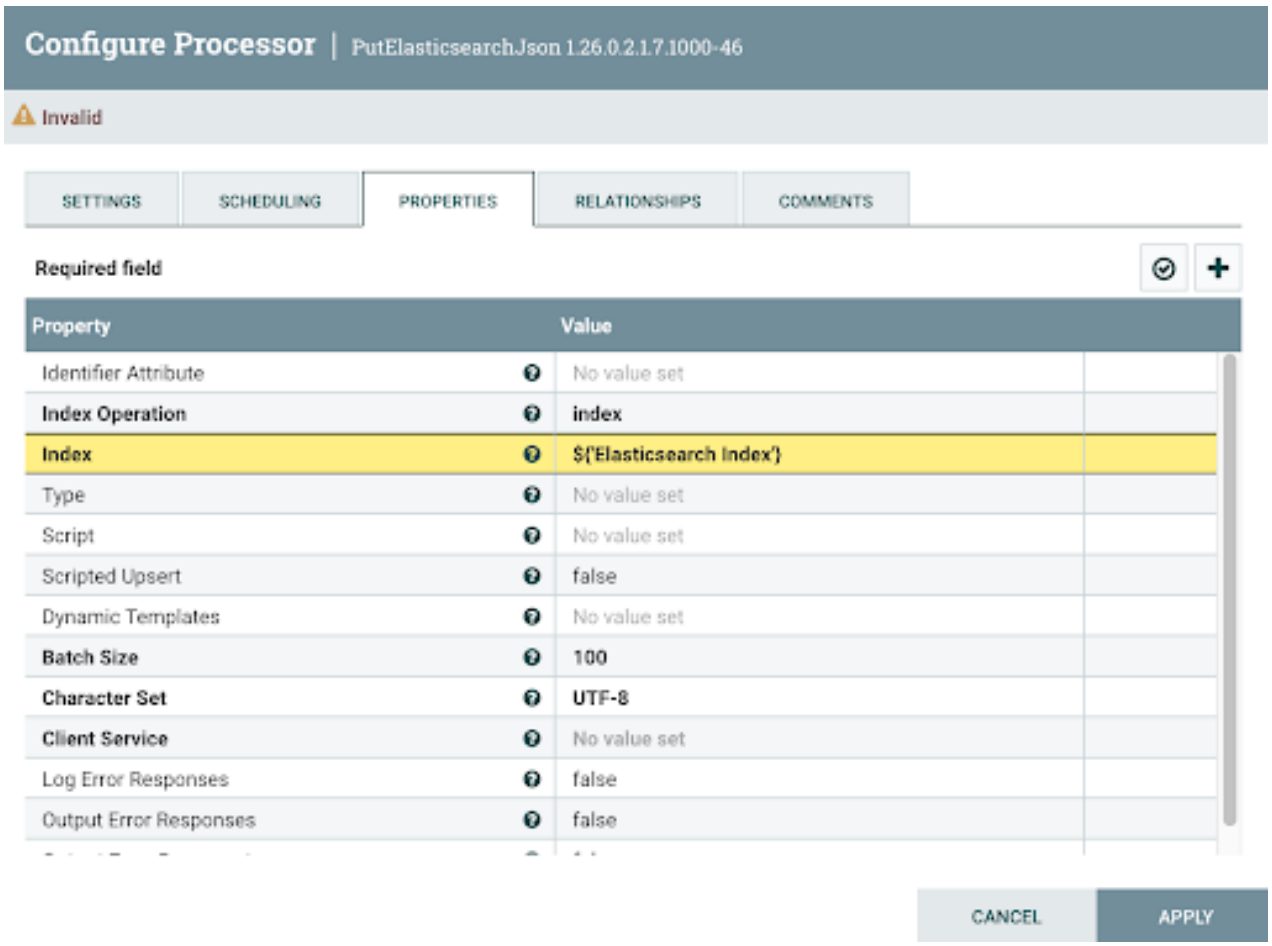


Elastic Process Group

This process group also contains a simple flow:



The process group has a variable, **Elasticsearch Index**, referenced in the Index property of the PutElasticsearchJson processor.



Your goal is to migrate the templates used in Cloudera Flow Management 2.1.7.4000 powered by NiFi 1 to parameters used in Cloudera Flow Management 4.12.0 powered by NiFi 2.

The example guides you through variable migration and maintaining a clear activity log. While this example template is simple, the step-by-step approach shows how this method improves clarity for more complex migrations. In real-world scenarios, you have to define a migration strategy based on your flow's structure. Individual template migration may not always be necessary.

Before you begin

Copy the TCP_Listener_template.xml and Elastic_template.xml files to the Migration Tool's input folder (/etc/migration-tool-input).

Procedure

1. Run Stage 1 variable migration using the following command.

```
bin/migration.sh nifi migrate-variables \
-i /etc/migration-tool-input \
-od /etc/migration-tool-output/variables \
--sourceCompatibleOutput
```

This generates a sourceVersion folder that contains the output files of the Stage 1 migration.

```
variables
### sourceVersion
    ### activity_log.json
    ### migrated_output
```

```
        ### migrated_Elastic_template.json
        ### migrated_TCP_Listener_template.json
```

activity_log.json

- A new parameter context was created with a new parameter called **TCP Listener Port**, which replaces the corresponding variable.
- The **TCP Listener Port** variable was removed.
- A new parameter context was created with a new parameter called **Elasticsearch Index**, which replaces the corresponding variable of the same name.
- The new parameter is referenced from the PutElasticsearchJson processor.
- The **Elasticsearch Index** variable was removed.

migrated_TCP_Listener_template.json

- A modified NiFi 1 flow, which is not compatible with NiFi 2 yet.
- It contains everything the original flow did, except the **TCP Listener** process group now references a parameter instead of the removed variable.
- This is still a NiFi 1 flow definition, so do not load this file directly into NiFi 2. Use an empty NiFi 1 instance to review the modification made by the tool. The flow definition will appear unchanged, but variables will be replaced with parameters of the same name, which are now referenced by the processor instead.
- Changes in the syntax: Variables were previously referenced using ${}, whereas parameters are now referenced using #{}.
  - **NiFi 1 variables**: ${variable_name}
  - **NiFi 2 parameters**: #{parameter_name}

migrated_Elastic_template.json

- A modified NiFi 1 flow definition, which is not compatible with NiFi 2 yet.
- It contains everything the original flow definition did, but the Elastic process group now references a parameter instead of the removed variable.



Note:

The file name is composed of the migrated prefix and the name of the top-level process group and not the original file name.

2. Validate the Stage 1 variable migration output for the migrated_TCP_Listener_template.json file.
   a) Check the new flow definition in a NiFi 1 instance to verify that the flow matches your expectations.
      1. Start your NiFi 1 instance.
      2. Create a new process group.
      3. Load sourceVersion/migrated_output/migrated_TCP_Listener_template.json to NiFi 1.
      4. Remove the ‘migrated_‘ prefix from the process group name.
      5. Confirm that variables were correctly converted to parameters.
   b) Review the activity_log.json file and check for any manual-change-requests or manual-validation-requests.
   c) If there are manual-change-requests or manual-validation-requests to handle, follow these steps:
      1. Make the modifications on the NiFi canvas, indicated by the manual-change-requests or manual-validation-requests in the sourceVersion/activity_log.json.
      2. Right-click on the TCPListener process group
      3. Select Download flow definition without external services.
      4. Save the file as migrated_TCP_Listener_template.json and overwrite the one in the sourceVersion/migrated_output folder.
   d) If no requests are present in the log:
      1. Make a backup of the original TCP_Listener_flow_definition.json.
      2. Right-click on the TCPListener process group

3. Select Download flow definition without external services.
4. Save file as migrated_TCP_Listener_template.json and overwrite the one in the sourceVersion/migrated_output folder.

e) Proceed to running Stage 1 variable migration on the Elastic_template.json.

At this stage, the flow definition no longer contains variables and uses parameters instead. If the flow meets your expectations, you can either run a full variable migration to validate your flow in NiFi 2 or proceed with migrating the components.

3. Validate the Stage 1 variable migration output for the Elastic_template.json.

a) Check the new flow definition in a NiFi 1 instance to verify that the flow matches your expectations.

1. Start your NiFi 1 instance.
2. Create a new process group.
3. Load sourceVersion/migrated_output/migrated_Elastic_template.json to NiFi 1.
4. Remove the ‘migrated_‘ prefix from the process group name.
5. Confirm that variables were correctly converted to parameters.

b) Review the activity_log.json file and check for any manual-change-requests or manual-validation-requests.

c) If there are manual-change-requests or manual-validation-requests to handle, follow these steps:

1. Make the modifications on the NiFi canvas, indicated by the manual-change-requests or manual-validation-requests in the sourceVersion/activity_log.json.
2. Right-click on the Elastic process group
3. Select Download flow definition without external services.
4. Save the file as migrated_Elastic_template.json and overwrite the one in the sourceVersion/migrated_output folder.

d) If no requests are present in the log:

1. Right-click on the Elastic process group
2. Select Download flow definition without external services.
3. Save file as migrated_Elastic_template.json and overwrite the one in the sourceVersion/migrated_output folder.

At this stage, both flow definitions no longer contain variables and use parameters instead. If the flow meets your expectations, you can either run a full variable migration to validate your flow in NiFi 2 or proceed with migrating the components.

4. Run full variable migration (Stage 1 and 2) using the following command.

```
bin/migration.sh nifi migrate-variables \
-i /etc/migration-tool-input \
-od /etc/migration-tool-output/variables
```

This generates a sourceVersion and targetVersion folders that contain the output files of the migration.

```
variables
### sourceVersion
#    ### activity_log.json
#    ### migrated_output
#        ### migrated_Elastic_template.json
#        ### migrated_TCP_Listener_template.json
###targetVersion
    ### activity_log.json
    ### migrated_output
        ### migrated_Elastic_template.json
        ### migrated_TCP_Listener_template.json
```

The contents of the previously generated sourceVersion folder will be overwritten, but this is not a concern as the tool generates the same sourceVersion output as before.

Additionally, a targetVersion directory is created, containing the output files of the Stage 2 part of the migration.

migrated_Elastic_template.json

- NiFi 2-compatible flow in terms of variables
- You can load it into NiFi 2 and validate it

migrated_TCP_Listener_template.json

- NiFi 2-compatible flow in terms of variables
- You can load it into NiFi 2 and validate it

activity_log.json

- List of all actions performed during this stage of the migration.

Note:

One activity log file will be created for the entire command.

5. Proceed with component migration using the input folder from step 3.

Migrating components using a directory with template files as input

Learn how to use the Cloudera Flow Migration Tool to migrate components from flow definition JSON files. As templates may not contain some parameter contexts and parameters defined at the parent process group level, you have to use the flow definitions generated in the previous steps.

Example flow for migrating components

The following NiFi flows are used to demonstrate both variable and component migration.

This example uses two template files:

- **migrated_TCP_Listener_template.json**

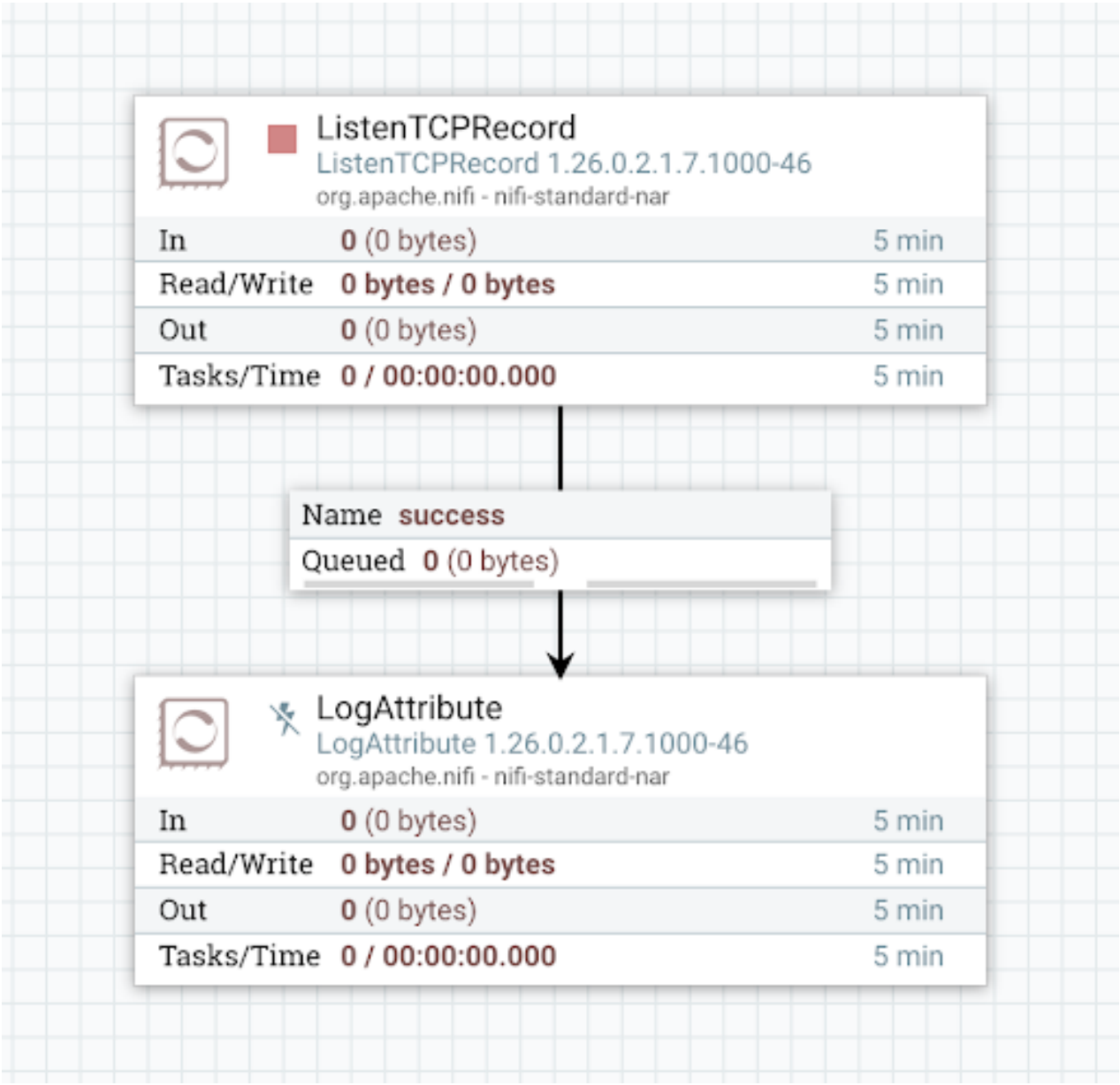
  This file contains process group TCP Listener (ID: b41940d7-0194-1000-42fc-458834630567)
- **migrated_Elastic_template.json**

  This file contains process group Elastic (ID: b42881c7-0194-1000-3cdf-1bd453a0ed0f)

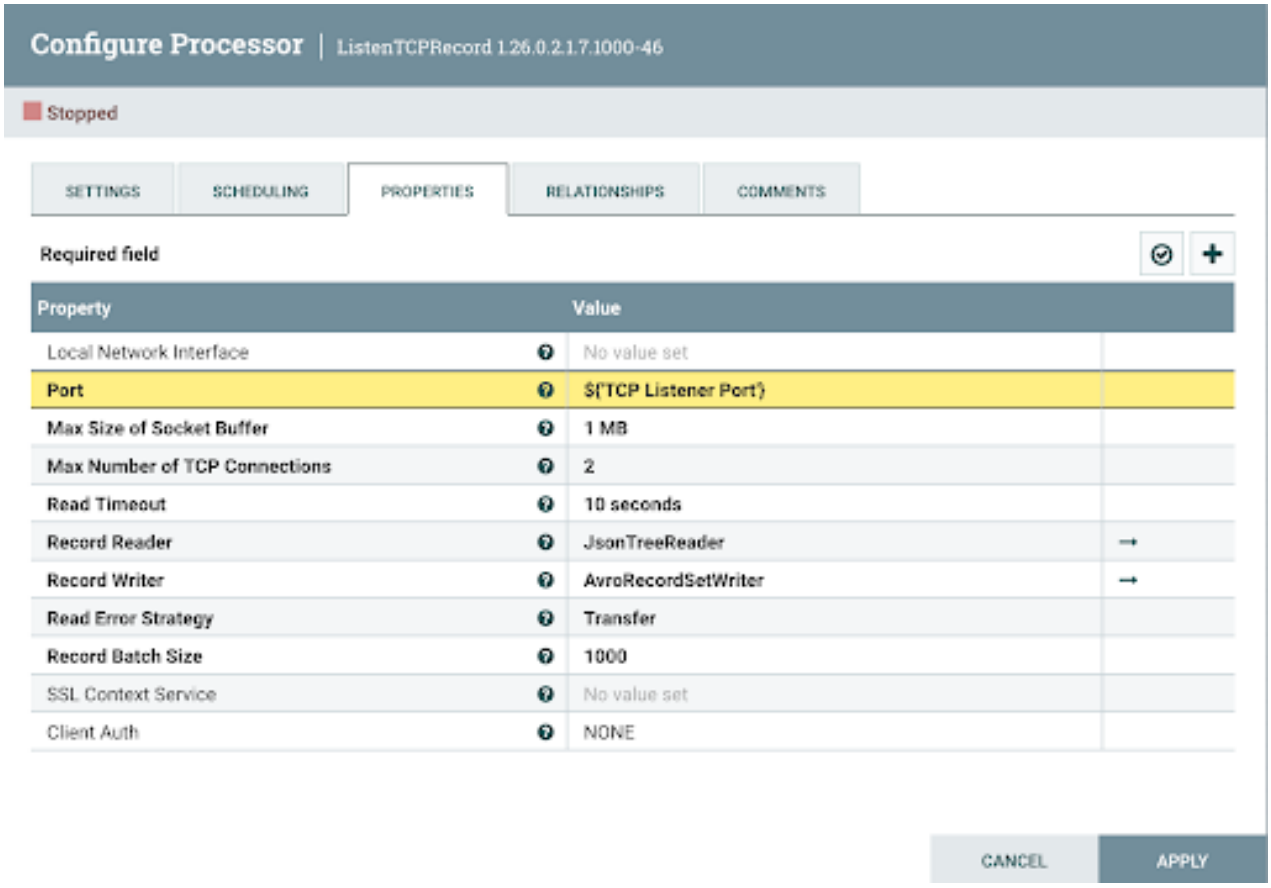


TCP Listener Process Group

This process group contains the following simple flow:

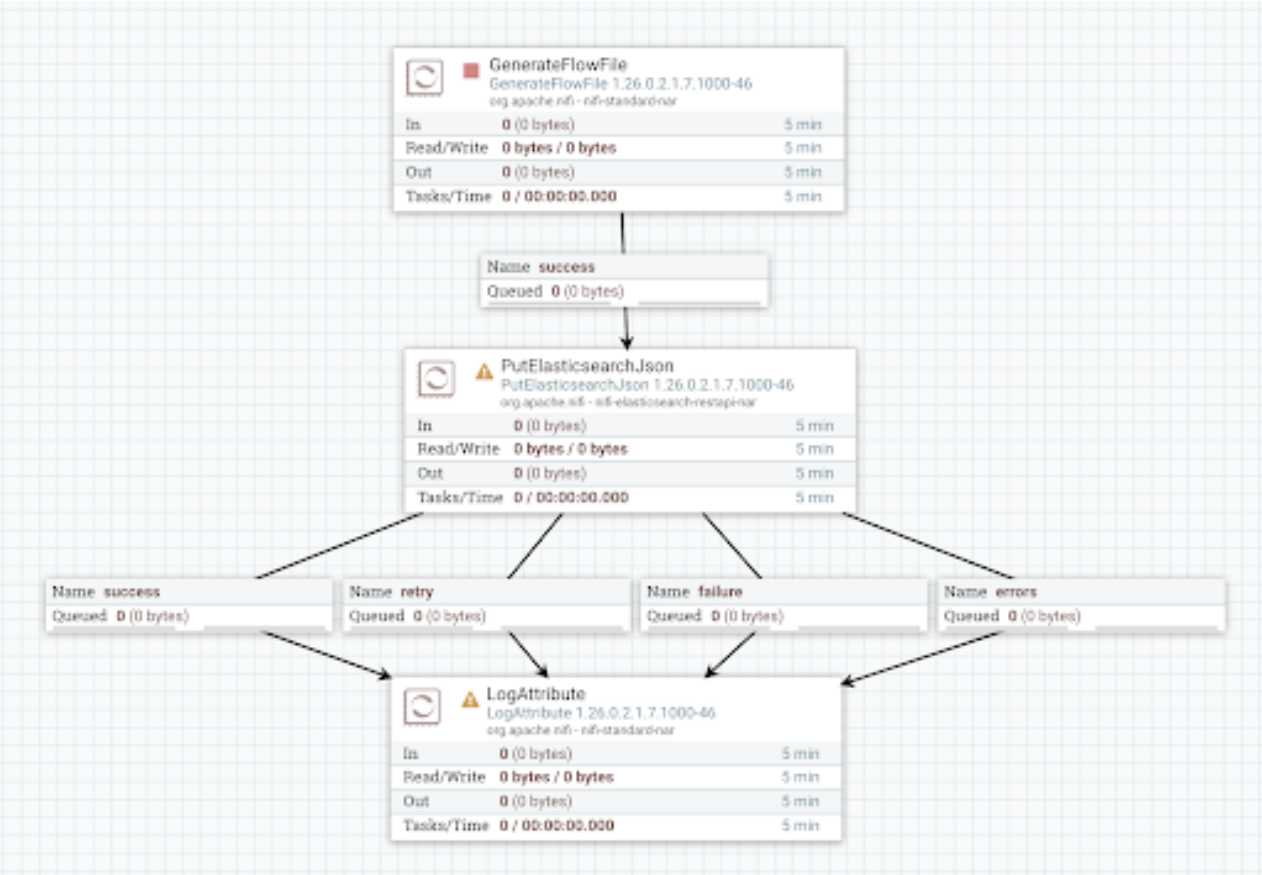


The process group defines a variable called **TCP Listener Port**, which is referenced by the ListenTCPRecord processor.

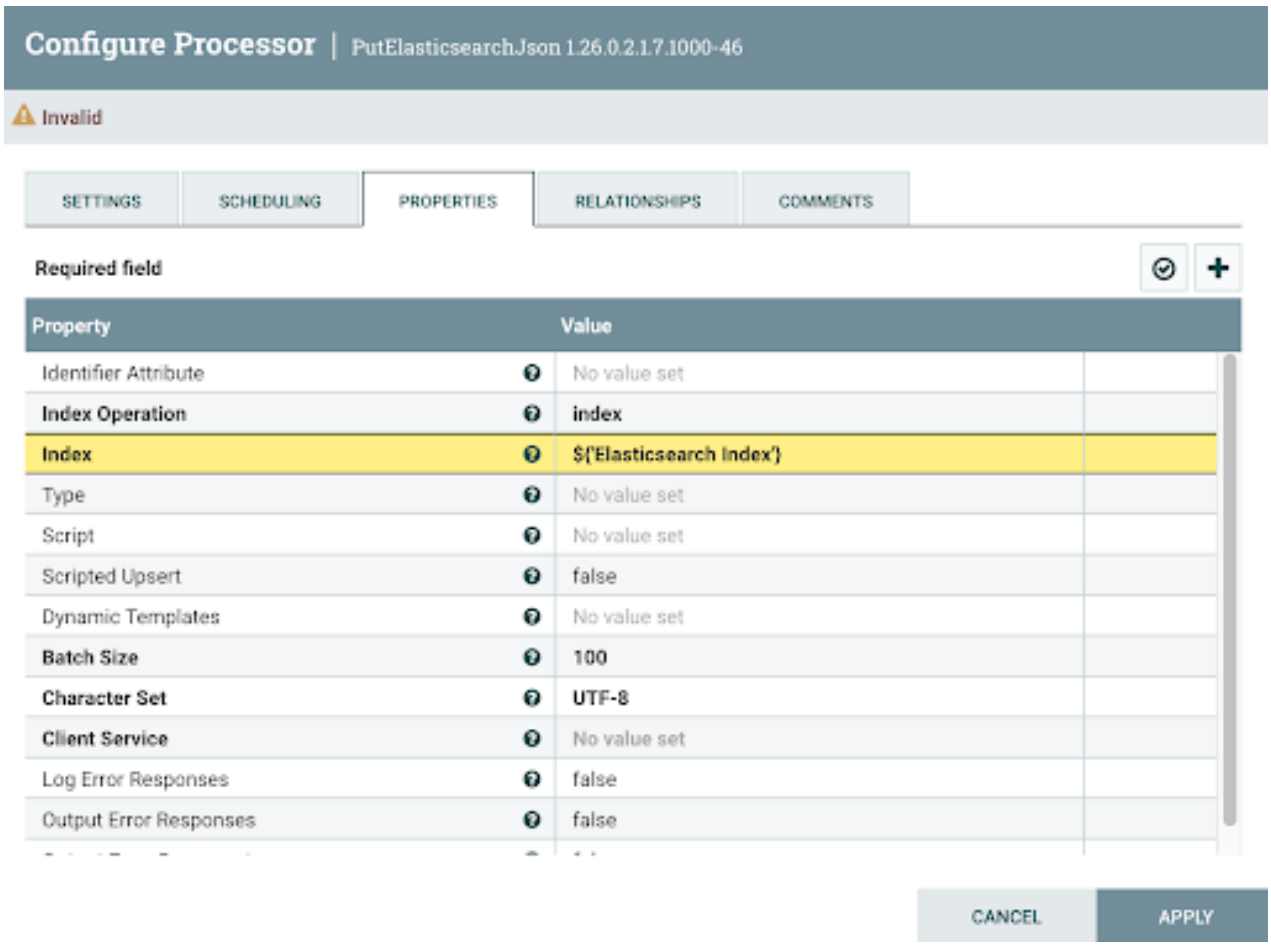


Elastic Process Group

This process group also contains a simple flow:



The process group defines a variable, **Elasticsearch Index**, which is referenced in the Index property of the PutElasticsearchJson processor.



Your goal is to migrate the components used in Cloudera Flow Management 2.1.7.4000 powered by NiFi 1 to NiFi-2 compatible components used in Cloudera Flow Management 4.12.0

The example guides you through component migration one flow definition at a time, simplifying the process and maintaining a clear activity log. While these example flow definitions are simple, the step-by-step approach shows how this method improves clarity for more complex migrations. In real-world scenarios, you have to define a migration strategy based on your flow's structure.

Procedure

1. Make a backup of the output folder (/etc/migration-tool-output/components) before running the next migration step.

   Note:

   This step is crucial because the same output folder is used as in the previous migration, and its contents will be overwritten. To retain a record of changes made at each stage, it is better to create a backup.
2. Copy the migrated_TCP_Listener_template.json and migrated_Elastic_template.json files to the Migration Tool's input folder (/etc/migration-tool-input).
3. Rename them, overwriting the existing ones, to TCP_Listener_template.json and Elastic_template.json for clarity.
4. Remove all other files from the Migration Tool's input folder (/etc/migration-tool-input).
5. Run Stage 1 component migration using the following command.

```
bin/migration.sh nifi migrate-components \
-i /etc/migration-tool-input \
-od /etc/migration-tool-output/components \
--sourceCompatibleOutput
```

This generates a sourceVersion folder that contains the output files of the Stage 1 migration.

```
components
###sourceVersion
    ### activity_log.json
    ### migrated_output
        ### migrated_Elastic_template.json
        ### migrated_TCP_Listener_template.json
```

activity_log.json

- The log describes all the actions that were performed for this stage of the process group migration.
- Log of all actions performed during this stage of the process group migration.



Note:

One activity log file will be created for the entire command.

migrated_TCP_Listener_template.json

- A modified NiFi 1 flow, which is not compatible with NiFi 2 yet.
- It contains everything the original flow did, except the **TCP Listener** process group was modified as described in the Activity Log.

migrated_Elastic_template.json

- A modified NiFi 1 flow definition which is not compatible with NiFi 2 yet.
- It contains everything the original flow definition did, except the Elastic process group was modified with the actions described in the activity log.

6. Validate the Stage 1 component migration output for TCP_Listener_template.json.
   a) Load the migrated_TCP_Listener_template.json into a NiFi 1 instance and check the flow.
   b) Review the activity_log.json file and check for any manual-change-requests or manual-validation-requests. If none are present, proceed to the next step.

      In this example, you can see the following information in the Activity Log:

      ```
      {
       "sequence" : 2,
       "type" : "change-info",
       "subject" : "b41966ad-0194-1000-a08d-a92489457356",
       "message" : "Component [org.apache.nifi.processors.standard.ListenTCPRecord] has been deprecated (NIFI-13509)",
       "context" : {
         "rule" : "36227b60-75f0-40dc-8caf-a2ec577aa54c"
      }
      ```

      Search for the value of the “subject” element in NiFi’s search box.You are directed to the ListenTCPRecord processor. No manual modifications are needed at this stage. However, it is important to note that the ListenTCPRecord processor is deprecated and is not available in NiFi 2. Once the full component migration (Stage 1 and Stage 2) is complete, instructions will be provided on how to handle this deprecation.
   c) If manual changes are necessary, update the migrated_TCP_Listener_template.json on the NiFi canvas after loading it. Once the flow is validated and meets expectations, continue with the next step.
7. Validate the Stage 1 component migration output for Elastic_template.json.
   a) Load the migrated_Elastic_template.json into a NiFi 1 instance and check the flow.
   b) Review the activity_log.json file. It contains a change-info entry that informs you of changes to a NiFi 2 processor, identified by the subject ID.

      ```
      {
       "sequence" : 2,
      ```

```
 "type" : "change-info",
 "subject" : "b428aad6-0194-1000-d73d-9f2332a59f04",
 "message" : "Property [Max JSON Field String Length] has been added (N
IFI-12343); Property [put-es-json-not_found-is-error] has been renamed t
o [put-es-not_found-is-error] (NIFI-12255); Property [put-es-json-error-
documents] has been removed (NIFI-12255); Relationships [success] has be
en renamed to [original]; Relationship [successful] has been added (NIFI
-12255);",
 "context" : {
   "rule" : "6501a693-b524-4d1a-b11e-69747c2651e8"
}
```

To identify the processor affected by these changes, search for the value of the “subject” element in NiFi’s search box.

You are directed to the PutElasticsearchJson processor. No manual modifications are needed at this stage. However, it is important to note that changes will be applied to the PutElasticsearchJson processor during Stage 2 of the migration. Once the full component migration (Stage 1 and Stage 2) is complete, instructions will be provided on how to handle this change.

At this stage, you have completed Stage 1 of both variable and component migration for your flow definition files. After reviewing the logs, you confirmed that no manual changes were needed. You can proceed with a full component migration using the /etc/migration-tool-input from Step 5.

8. Run full component migration (Stage 1 and 2).
   a) Make a backup of the output folder (/etc/migration-tool-output/components) before running the next migration step.

      Note:

      This step is crucial because the same output folder is used as in the previous migration, and its contents will be overwritten. To retain a record of changes made at each stage, it is better to create a backup.

   b) Run the full component migration using the following command.

```
bin/migration.sh nifi migrate-components \
-i /etc/migration-tool-input \
-od /etc/migration-tool-output/components
```

This generates the following output:

```
components
###sourceVersion
#    ### activity_log.json
#    ### migrated_output
#        ### migrated_Elastic_template.json
#        ### migrated_TCP_Listener_template.json
###sourceVersion
    ### activity_log.json
    ### migrated_output
        ### migrated_Elastic_template.json
        ### migrated_TCP_Listener_template.json
```

The contents of the previously generated sourceVersion folder will be overwritten. The contents represent the NiFi 1-compatible version of the migrated flow. Since the root process group only contained the two process groups on which you already performed Stage 1 component migration, the activity log will only include the same entries as those in Steps 2 and 4.

Additionally, a targetVersion directory is created, containing the output files of the Stage 2 part of the migration.

migrated_TCP_Listener_template.json

- NiFi 2-compatible flow in terms of variables

- You can load it into NiFi 2 and validate it

migrated_Elastic_template.json

- NiFi 2-compatible flow in terms of variables
- You can load it into NiFi 2 and validate it

activity_log.json

- List of all actions performed during this stage of the migration, and any manual steps that you need to perform.

9. Validate the Stage 2 component migration output for TCP_Listener_template.json.

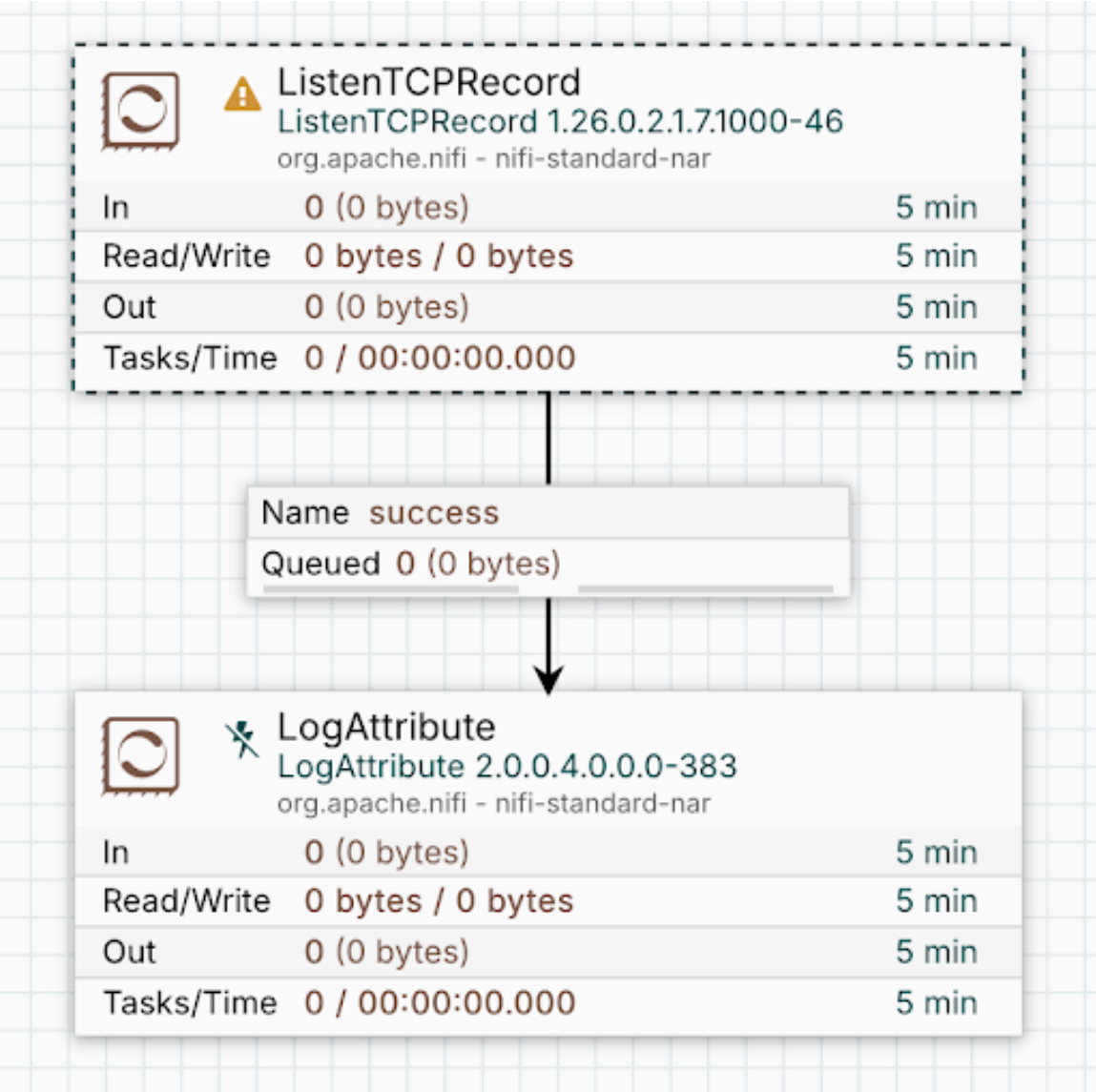
a) Load the targetVersion/migrated_output/migrated_TCP_Listener_template.json into a NiFi 2 instance and check the flow.

b) Review the targetVersion/activity_log.json file.

In this example, you can see the following manual-change-request entry.

```
{
 "sequence" : 6,
 "type" : "manual-change-request",
 "subject" : "b41966ad-0194-1000-a08d-a92489457356",
 "message" : "The component has been deprecated. It is suggested to r
eplace it with a combination of [org.apache.nifi.processors.standard.Lis
tenTCP] and [org.apache.nifi.processors.standard.ConvertRecord] processo
rs",
 "context" : {
   "rule" : "36227b60-75f0-40dc-8caf-a2ec577aa54c"
}
```

1. To identify the processor affected by these changes, search for the identifier value of the “subject” element in NiFi’s search box. It refers to the ListenTCPRecord processor.
2. Open the TCP Listener process group. You can see that the processor is marked with dashed borders and its version number still shows that of the NiFi 1 instance. It means that it is a "ghost processor", not available in NiFi 2. The log message suggests replacing it with a combination of ListenTCP and ConvertRecord processors.



3. Instantiate and configure the ListenTCP and ConvertRecord processors, connect them appropriately, and then remove the ghost processor.

   Besides these changes there are no further manual-change-requests or manual-validation-requests in the Activity Log.

c) Download the flow definition and you have the fully NiFi 2 compatible TCP_Listener_template.json.

10. Validate the Stage 2 component migration output for Elastic_template.json.

a) Load the targetVersion/migrated_output/migrated_Elastic_template.json into a NiFi 2 instance and check the flow.

b) Review the targetVersion/activity_log.json file. In this example, you can see the following manual-change-request entry.

```
{
 "sequence" : 3,
 "type" : "manual-change-request",
 "subject" : "b428aad6-0194-1000-d73d-9f2332a59f04",
 "message" : "New relationship [original] has been added, it has to be
 connected to a downstream processor or terminated.",
 "context" : {
   "rule" : "6501a693-b524-4d1a-b11e-69747c2651e8"
```

```
}
```

1. To identify the processor affected by these changes, search for the value of the “subject” element in NiFi’s search box. It refers to the PutElasticsearchJson processor. You will find an unbound "original" relationship that needs to be connected to a downstream processor or terminated.
2. Make the necessary modifications manually. Once completed, this change request is resolved.

c) Download the flow definition and you have the fully NiFi 2 compatible Elastic_template.json.

Using migrate-all using a directory with template.xml files as input

Template migration is not applicable for this input type, use the migrate-templates command instead.

Cloudera Flow Migration Tool Reference

The reference documentation provides a comprehensive guide to the Migration Tool commands, which support partial or full migration of incoming data flows. Each command addresses a specific part of the migration process, offering flexibility and control to meet a variety of migration needs. Use this reference to understand the function and configuration of each command so that you can effectively tailor the migration process to your requirements.

Additionally, the reference documentation covers the Activity Log, which provides a detailed record of migration-related events, changes, and manual interventions. The log helps you track modifications, review necessary actions, and ensure a smooth transition during the migration process. Use this reference to gain insights into how your flows were migrated, identify any manual steps required, and ensure that all necessary changes were applied correctly.

Commands

This section contains information about the commands you can use with the Cloudera Flow Migration Tool when migration your data flows from NiFi 1 to NiFi 2.

The command list includes:

- **Command descriptions**: A detailed explanation of each command and its role within the migration workflow.
- **Input arguments**: A list of common arguments shared by all commands, along with those specific to individual commands or use cases.
- **Syntax and usage examples**: Practical examples showcasing the correct syntax and usage for each command, helping you implement them in your environment.
- **Output artifacts**: Information on the output generated by each command, ensuring traceability and transparency throughout the migration process.

migrate-templates

Migrates NiFi templates using the Cloudera Flow Migration Tool.

Syntax

```
migrate-templates --input {filePath} --outputDirectory {directoryPath} [--so
urceCompatibleOutput] [--processGroupId {pgId}]
```

Required arguments

--input

- Short format: -i {filePath}
- Specifies the path to a local file containing the flow in JSON format for processing.
- The file must be accessible to the Migration Tool, and reading privileges are required.
- The original file remains unmodified.

--outputDirectory

- Short format: -od {directoryPath}
- Specifies the directory where migration results will be saved.
- The directory must be accessible to the Migration Tool, and writing privileges are required.
- Existing content in the specified directory may be overwritten by subsequent runs.

Supported arguments

--sourceCompatibleOutput

- Short format: -sco
- If specified, onlyStage 1 migration will be run.
- If omitted, both Stage 1 and Stage 2 are processed sequentially, using Stage 1's output as input for Stage 2.

--processGroupId

- Short format: -pgid {pgId}
- Defines the starting point for migration, applying transformations to the specified group and its child groups.
- If omitted, the root group is processed.

--inputType

- Short format: -it {inputType}
- The type of input provided, case sensitive. Supported types: FLOW, FLOW_GZ, FLOW_DEFINITION, TEMPLATE_XML, REGISTRY_SNAPSHOT
- If omitted, the tool tries to automatically identify the input type. Mixed inputs are not allowed.

--registryAware

- Short format: -ra
- Skips versioned process groups during migration so their content is handled by migrate-registry and other Registry-related commands instead.



Note:

The input flow.json file must represent the entire flow, regardless of whether this argument is used.

Output files of Stage 1 and Stage 2

Extracted templates

The resulting templates from the migration process.

activity_log.json

It is a log file recording details of the migration process. For more information, see *Activity Log*.



Note:

One activity log file will be created for the entire command.

flow_definitions.json

It is a flow definition representation of their respective template.

Examples

1 **Migrate templates processing Stage 1 and export templates**

```
./migration.sh nifi migrate-templates --input /etc/exported-flows/flow.json
\
--outputDirectory /etc/output/ --sourceCompatibleOutput --processGroupId b
b81df68-cd6a-461a-b724-384265875b53
```

Processes the process group specified by the ID bb81df68-cd6a-461a-b724-384265875b53 only in Stage 1. Reads the input from /etc/exported-flows/flow.json, and saves the results in /etc/output/.

2 **Migrate templates processing all stages**

```
./migration.sh nifi migrate-templates --input /etc/exported-flows/flow.json
\
--outputDirectory /etc/output/
```

Processes both Stage 1 and Stage 2 sequentially, using the output of Stage 1 as input for Stage 2.

migrate-variables

Migrates variable configurations from NiFi 1 to NiFi 2 using the Cloudera Flow Migration Tool.

Syntax

```
migrate-templates --input {filePath} --outputDirectory {directoryPath} [--so
urceCompatibleOutput] [--processGroupId {pgId}]
```

Required arguments

--input

- Short format: -i {filePath}
- Specifies the path to a local file containing the flow in JSON format for processing.
- The file must be accessible to the Migration Tool.
- The original file remains unmodified.

--outputDirectory

- Short format: -od {directoryPath}
- Specifies the directory where migration results will be saved.
- The directory must be accessible to the Migration Tool.
- Existing content in the specified directory may be overwritten by subsequent runs.

Supported arguments

--sourceCompatibleOutput

- Short format: -sco {stageName}
- If specified, only Stage 1 migration will be executed. Specifies the stage(s) to process.
- To process only Stage 1 migration, use STAGE_1.
- If omitted, both Stage 1 and Stage 2 are processed sequentially, using Stage 1's output as input for Stage 2.
- Providing unsupported or custom stage names may result in invalid stage error.

--processGroupId

- Short format: -pgid {pgId}
- Defines the starting point for migration, applying transformations to the specified group and its child groups.
- If omitted, the root group is processed.

--inputType

- Short format: -it {inputType}
- The type of input provided, case sensitive. Supported types: FLOW, FLOW_GZ, FLOW_DEFINITION, TEMPLATE_XML, REGISTRY_SNAPSHOT
- If omitted, the tool tries to automatically identify the input type. Mixed inputs are not allowed.

--registryAware

- Short format: -ra
- Skips versioned process groups during migration so their content is handled by migrate-registry and other Registry-related commands instead.

Note:

The input flow.json file must represent the entire flow, regardless of whether this argument is used.

Output files of Stage 1 and Stage 2

NiFi flow

For example: migrated_flow.json.

activity_log.json

A log file recording the details of the migration process. For more information, see *Activity Log*.

Note:

One activity log file will be created for the entire command.

Examples

1 **Migrate variables processing Stage 1**

```
./bin/migration.sh nifi migrate-variables --input /etc/exported-flows/flow.j
son --outputDirectory /etc/output/ --sourceCompatibleOutput
```

Processes the flow.json file only in Stage 1, located in /etc/exported-flows/. Saves the migrated variables and activity log in /etc/output/.

2 **Migrate variables processing all stages**

```
./bin/migration.sh nifi migrate-variables --input /etc/exported-flows/flow.j
son --outputDirectory /etc/output/
```

Processes the flow.json file in all stages and saves the results in /etc/output/.

3 **Migrate variables using full customization with all arguments**

```
./bin/migration.sh nifi migrate-variables --input /etc/exported-flows/flow.j
son --outputDirectory /etc/output/ --sourceCompatibleOutput --processGroupId
 bb81df68-cd6a-461a-b724-384265875b53 -it FLOW
```

Runs Stage 1 of the migration, starting the processing from the process group ID bb81df68-cd6a-461a-b724-384265875b53. Saves the migrated flow and the Activity Log in /etc/output/.

migrate-components

Migrates components from NiFi 1 to NiFi 2 using the Cloudera Flow Migration Tool.

Syntax

```
migrate-components --input {filePath} --outputDirectory {directoryPath} [--s
ourceCompatibleOutput] [--processGroupId {pgId}]
```

Required arguments

--input

- Short format: -i {filePath}
- Specifies the path to a local file containing the flow in JSON format for processing.

- The file must be accessible to the Migration Tool.
- The original file remains unmodified.

--outputDirectory

- Short format: -od {directoryPath}
- If specified, only Stage 1 migration will run. Specifies the stage(s) to process
- To process only Stage 1 migration, use STAGE_1.
- If omitted, both Stage 1 and Stage 2 are processed sequentially, using Stage 1's output as input for Stage 2.
- Providing unsupported or custom stage names may result in invalid stage error.

Supported arguments

--sourceCompatibleOutput

- Short format: -sco {stageName}
- To process Stage 1 migration, use STAGE_1
- If omitted, both Stage 1 and Stage 2 are processed sequentially, using Stage 1's output as input for Stage 2.
- Providing unsupported or custom stage names may result in invalid stage error.

--processGroupId

- Short format: -pgid {pgId}
- Defines the starting point for migration, applying transformations to the specified group and its child groups.
- If omitted, the root group is processed.
- Management-level components, such as Controller Services, Parameter Providers, and Reporting Tasks, are only included when this argument is not used.

--inputType

- Short format: -it {inputType}
- The type of input provided, case sensitive. Supported types: FLOW, FLOW_GZ, FLOW_DEFINITION, TEMPLATE_XML, REGISTRY_SNAPSHOT
- If omitted, the tool tries to automatically identify the input type. Mixed inputs are not allowed.

--registryAware

- Short format: -ra
- Skips versioned process groups during migration so their content is handled by migrate-registry and other Registry-related commands instead.



Note:

The input flow.json file must represent the entire flow, regardless of whether this argument is used.

Output files of Stage 1 and Stage 2

Flow file

activity_log.json

A log file recording the details of the migration process. For more information, see *Activity Log*.



Note:

One activity log file will be created for the entire command.

Examples

1 **Migrate components processing Stage 1**

```
./bin/migration.sh nifi migrate-components --input /etc/flow_original.json --outputDirectory /etc/output/ --sourceCompatibleOutput
```

Processes the flow_original.json file located in /etc/. Applies transformation rules from Stage 1 on all components. Saves the migrated flow file and the Activity Log in /etc/output/.

2 **Migrate components in a specific Process Group**

```
./bin/migration.sh nifi migrate-components --input /etc/flow_original.json --outputDirectory /etc/output/ --processGroupId 3f8d2cba-4d3b-4901-bd0b-4781ff5b5c9f
```

Starts the migration at the process group identified by 3f8d2cba-4d3b-4901-bd0b-4781ff5b5c9f. Transforms components within this group and its child groups. Outputs are saved in /etc/output/.

3 **Migrate components using full customization with all arguments**

```
./bin/migration.sh nifi migrate-components --input /etc/flow_original.json --outputDirectory /etc/output/ --sourceCompatibleOutput --processGroupId 3f8d2cba-4d3b-4901-bd0b-4781ff5b5c9f -it FLOW
```

Runs Stage 1 only and the migration begins at the specified process group. Management level components are not migrated. Saves the migrated flow and the Activity Log in /etc/output/.

migrate-all

Aggregates the commands of all migration steps when using the Cloudera Flow Migration Tool.

Usage

- Combines all migration steps into a single command.
- Outputs of each step are used as inputs for the next step.
- The order of the included steps is the following:
  - Stage 1: Migrate Templates # Migrate Variables # Migrate Components
  - Stage 2: Migrate Templates # Migrate Variables # Migrate Components
- When you combine **migrate-all** with a NiFi Registry migration workflow, Stage 2 can include a flow-reference refresh consistent with upgrade-flow. If you run individual migration commands instead of migrate-all, run the registry flow-reference step from your checklist when required.

Supported arguments

--registryAware

- Short format: -ra
- Skips versioned process groups during template, variable, and component steps so their content is handled by migrate-registry and other Registry-related commands instead.

migrate-registry

Migrates NiFi Registry snapshots used by a flow through a working NiFi Registry, using the Cloudera Flow Migration Tool.

Description

This is the primary tool for migrating registries. While there are some auxiliary commands, this command is responsible for most of the work done during registry snapshot migration. Its main purpose is to apply changes to the

snapshots provided to the command. Its behavior is very similar to the migration commands for flows or individual snapshot files (supported by other commands like migrate-components), but this command is optimized to support the migration of a high number of snapshots with the least amount of administrative overhead.

To enable this, execution of the command requires a Working Registry to be set up and properly configured in the properties. Please see the relevant section of the Prerequisites. While the command's primary purpose is always to migrate and manage migrated snapshots, the Migrate Registry Snapshot command supports different modes of execution for different steps during the migration process.

Execution modes:

start

The initial step for registry migration. This step requires a Working Registry where no snapshots from an ongoing migration are stored. If the Migration Tool detects artifacts of an ongoing registry migration within the Working Registry, the command will fail. Otherwise, the command migrates the content (snapshots) of the provided registry exports and writes the results into the Working Registry as "initial migration" results. These migrated snapshots may then be subject to manual changes and further migrations.

Additionally, this step generates a modified flow.json, which is not migrated at this point, but all registered flow references are updated to point to the Working Registry. This modified flow serves as input for later steps. The start step can be executed only once within a sequence of Migrate Registry Snapshot commands, and only as the first step. Furthermore, it can be executed only during Stage 1. Running it with both stages will result in an error.

See *Properties* for details on the naming conventions that the Migration Tool applies to flows and buckets within the Working Registry.

restart

The restart step behaves the same as start in most respects. The difference lies in how it handles the state of the Working Registry: at the beginning of execution, instead of checking for an ongoing migration, restart cleans up the Working Registry by removing all flows and buckets considered part of a migration (based on the current properties). Be aware that if the Working Registry contains other flows, this step may unintentionally remove them. All subsequent steps match the behavior of the start step. The restart step is intended to simplify restarting the migration process.

See *Properties* for details on the naming conventions that the Migration Tool applies to flows and buckets within the Working Registry.

continue

This step is responsible for further migrations of the snapshots stored in the Working Registry. During Stage 1, this step can be repeated as many times as necessary, along with applying manual changes. In Stage 2, as in flow migration, it can be executed only once. This is the only step that can be executed during Stage 2. Unlike the other options, continue reads snapshots from and writes migrated snapshots back to the Working Registry.

Syntax

```
migrate-registry {stepArgument} [--input {filePath}] [--exportDirectory {dir
ectoryPath}] --outputDirectory {directoryPath} [--sourceCompatibleOutput] [-
-registry {registryName}] [--registryBucket {bucketName}] [--registryFlow {f
lowName}]
```

Required arguments

Step argument

- The following step arguments might be provided:
  - --start (short form: -s)

    - --restart (short form: -res)
    - --continue (short form: -c)
  - A given time only exactly one step argument is expected.
  - The step argument defines which step is being executed (see above)

--outputDirectory

- Short format: -od {directoryPath}
- Specifies the directory where migration results will be saved.
- The directory must be accessible to the Migration Tool.
- Existing content in the specified directory may be overwritten by subsequent runs.
- Required argument for start and restart steps.

Supported arguments

--input

- Short format: -i {filePath}
- Specifies the path to a local file containing the flow in JSON format for processing.
- The file must be accessible to the Migration Tool.
- The original file remains unmodified.
- Required argument for start and restart steps.

--exportDirectory

- Short form: -ed {directoryPath}
- Specifies the directory where the registry exports (need to be migrated) are stored.
- The exports themselves are represented by sub-directories.
- The name of the subdirectories must be the same as the name of the matching Registry Client.
- The directory must be accessible to the Migration Tool.
- The original directory and its content remains unmodified.
- The original file remains unmodified.

--sourceCompatibleOutput

- Short format: -sco
- If specified, only Stage 1 migration will be executed. Specifies the stage(s) to process.
- To process only Stage 1 migration, use this argument.
- If omitted, both Stage 1 and Stage 2 are processed sequentially, using Stage 1's output as input for Stage 2.
- Steps start and restart are not to be run within Stage 2 and requires this argument.

--registry

- Short format: -r {registryName}
- Helps narrowing down the scope of the migration to the snapshots within the registry with the given registryName.
- The registry name equals to the directory name of the original registry export (and the name of the Registry Client matches with that).
- Can be used only with the continue step and within Stage 1.

--registryBucket

- Short format: -rb {bucketName}
- This argument requires the argument registry.
- Helps narrowing down the scope of the migration to the snapshots within the bucket with the given bucketName.
- The bucketName is the origin bucket name, not the one stored within the Working Registry (prefixed with the registry name).

- Can be used only with the continue step and within Stage 1.

--registryFlow

- Short format: -rf {flowName}
- This argument requires the argument registryBucket.
- Helps narrowing down the scope of the migration to the snapshots within the flow with the given flowName.
- The flowName must match with the name as the flow stored within the Working Registry (including line part).
- Can be used only with the continue step and within Stage 1.

Output files of Stage 1 and Stage 2

NiFi flow

When executing the start or restart step, the NiFi flow is part of the output. Not migrated but with replaced versioned flows.

Activity logs

Above the activity_log.json which contains the generic events of the execution, every migrated snapshot has its own specific log, for example: R1_Bucket_1_Flow_2-activity_log.json.

Examples

1. Starting a registry migration (Stage 1)

```
./bin/migration.sh nifi migrate-registry -s -i /my_dir/flow.json -ed /my_dir
/flow_exports -od /my_dir/out -sco
```

2. Partial migration (Stage 1)

```
./bin/migration.sh nifi migrate-registry -c -od /my_dir/out --registry R1 --
registryBucket B1 -sco
```

3. Registry Stage 1 and Stage 2 together

```
./bin/migration.sh nifi migrate-registry -c -od /my_dir/out
```

upgrade-flow

Refreshes versioned flow references in the flow JSON from the working NiFi Registry using the Cloudera Flow Migration Tool.

Description

This command updates the flow.json by refreshing the versioned flows. For every managed versioned flow, the Migration Tool upgrades it to the newest version available in the Working Registry. The flow must have already been processed by the start step.

For Stage 1, its use is optional and mainly useful for testing purposes. In Stage 2, this command is mandatory and can be executed either explicitly or implicitly as part of the migrate-all command.

Syntax

```
upgrade-flow --input {filePath} --outputDirectory {directoryPath} [--sourceC
ompatibleOutput]
```

Required arguments

--input

- Short format: -i {filePath}
- Specifies the path to a local file containing the flow in JSON format for processing.
- The file must be accessible to the Migration Tool.
- The original file remains unmodified.

--outputDirectory

- Short format: -od {directoryPath}
- Specifies the directory where migration results will be saved.
- The directory must be accessible to the Migration Tool.
- Existing content in the specified directory may be overwritten by subsequent runs.

Supported arguments

--sourceCompatibleOutput

- Short format: -sco
- If specified, only Stage 1 migration will be executed. Specifies the stage(s) to process.
- To process only Stage 1 migration, use this argument.
- When running without this argument, this command has no affect during Stage 1.

Output files of Stage 1 and Stage 2

NiFi flow

With replaced versioned flow.

Examples

1. Upgrade references during Stage 1

```
./bin/migration.sh nifi upgrade-flow -i /my_dir/flow.json -od /my_dir/out/ou
t -sco
```

2. Upgrade references during Stage 2

```
./bin/migration.sh nifi upgrade-flow -i /my_dir/flow.json -od /my_dir/out/
out
```

export-registry-result

Exports NiFi Registry migration results using the Cloudera Flow Migration Tool.

Description

During the migration process, the flow and the snapshots are in an “under migration” format. This means that snapshots are stored in the Working Registry, and all versioned flow references point to the location of the Working Registry. This is not the expected final state, but it cannot be changed during migration, as the absence of these references would make the manual work involved in the migration difficult to manage.

To address this, as the final step of the migration process, run the Export Registry Migration Result command. This command exports the snapshots in a format identical to a NiFi Toolkit registry export while updating the reference locations during the process. This command cannot be executed for Stage 1 only and applies only to Stage 2.

Syntax

```
export-registry-result --input {filePath} --outputDirectory {directoryPath}
[--exportDirectory {directoryPath}] [--keepHistory]
```

Required arguments

--input

- Short format: -i {filePath}
- Specifies the path to a local file containing the flow in JSON format for processing.
- The file must be accessible to the Migration Tool.
- The original file is not modified.

--outputDirectory

- Short format: -od {directoryPath}
- Specifies the directory where migration results will be saved.
- The directory must be accessible to the Migration Tool.
- Existing content in the specified directory may be overwritten by subsequent runs.

Supported arguments

--exportDirectory

- Short form: -ed {directoryPath}
- Specifies the directory where the registry exports (need to be migrated) are stored.
- The exports themselves are represented by sub-directories.
- The directory must be accessible to the Migration Tool.
- The original directory and its content remains unmodified.
- The original file remains unmodified.
- Necessary if keepHistory is added to the command.

--keepHistory

- Short form: -kh
- Might be added if the original snapshots should be kept in the exported result as "historical versions". As a consequence, the original history can be stored in the target registry, but as NiFi 1 flows, not being usable in NiFi 1. The original files export will not be modified regardless of the usage of this argument.
- If provided, exportDirectory must be specified as well, pointing to the original registry exports.

Output for Stage 2

NiFi flow

With final versioned flow references

Registry exports

One folder per registries involved, in the same format as with NiFi Toolkit registry export

Activity log

Execution details

Examples

1. Export without historical versions

```
./bin/migration.sh nifi export-registry-result -i /my_dir/flow.json -od /my_
dir/out
```

2. Export with history

```
./bin/migration.sh nifi export-registry-result -i /my_dir/flow.json -ed /my_
dir/flow_exports -od /my_dir/out --keepHistory
```

cleanup-working-registry

Removes migration-related buckets and flows from the working NiFi Registry using the Cloudera Flow Migration Tool.

Description

This command cleans up migration-related content from the Working Registry.

Note: Be aware that the command uses the naming conventions defined by the current state of the properties file, and if it is not properly configured, it may delete additional buckets and flows.

For more information on the naming conventions that the Migration Tool applies to flows and buckets within the Working Registry, see *Properties*.

Syntax

```
cleanup-working-registry --outputDirectory {directoryPath}
```

Required arguments

--outputDirectory

- Short format: -od {directoryPath}
- Specifies the directory where migration results will be saved.
- The directory must be accessible to the Migration Tool.
- Existing content in the specified directory may be overwritten by subsequent runs.

Output for Stage 1 and Stage 2

Activity log

Only the activity log serves as output, containing details about the removed resources.

Examples

Clean up working registry

```
./bin/migration.sh nifi cleanup-working-registry -od /my_dir/out/out
```

help

Displays a list of available commands when using the Cloudera Flow Migration Tool.

Usage

- If you use it with a specific command ($[***COMMAND_NAME***] help), it displays the available arguments for it.
- Specify the full command, including its group: ./migration.sh nifi migrate-all help

Activity Log

This section contains information about activity_log.json. This log file provides a comprehensive record of the migration process, capturing detailed information about changes, events, and required interventions.

Summary

- File name: activity_log.json
- Serves a business-level record of changes and the context of the migration process.
- Provides a detailed log of the activities during migration in JSON format that follows a structured schema: it begins with metadata and consists of a list of entries.
- Designed for human readability and automated filtering, but not for fully automatic processing.



Note: One activity log file will be created for the entire command.

Metadata

- Provides context for understanding the log entries.
- Metadata fields:
  - recordingStarts: (number) UNIX epoch timestamp indicating when logging began, the approximate start of the stage run (not suitable for performance measurements)
  - recordingEnds: (number) UNIX epoch timestamp indicating when logging concluded, the approximate end of the stage run (not suitable for performance measurements)
  - stage: (string) Identifies the stage that is run. Possible values are "STAGE_1" or "STAGE_2"

Entry

- The Activity Log contains a list of entries, each representing a distinct event using different entry types.
- Each entry serves to provide insight into the actions performed during the migration process.

Entry properties

- sequence: A unique number within the log file, that represents the entry's sequence. It can be used to order and reference entries.
- type: A string denoting the purpose of the entry. For more information, see Entry types below.
- subject: A string representing the entity involved in the entry, such as a component, group, or other relevant entity.

  Note: In the case of component migration, the entry’s subject refers to the triggering component. This is important when a component is replaced, as the log will reference the original component’s ID (which may have been deleted at this point) rather than the new component's ID.
- message: A string containing the main content or description of the entry.
- context: A JSON object with additional key-value pairs, providing further details about the entry (for example: applied rules, affected properties).

Entry types

- manual-change-request
  - Subject: Usually the ID of a component requiring manual intervention
  - Message: Describes changes that cannot be applied automatically and require human intervention. For example, deprecated components without replacements or decisions requiring domain expertise.
  - Context: Not commonly used

- Example:

```
{
      "sequence": 5,
      "type": "manual-change-request",
      "subject": "69c4f8f3-549a-4103-b8f6-f8627b689e12",
      "message": "The value [null] for property [assume-r
ole-sts-region] is not supported any more.",
      "context": {  }
    }
```

- manual-validation-request
  - Subject: Typically the ID of a component under review.
  - Message: Highlights changes that should be explicitly reviewed to ensure correctness. These changes may or may not require adjustments.
  - Context: May include a rule key with a UUID for the applied migration rule, which helps identify errors.
- change-info
  - Subject: The component being migrated.
  - Message: Describes version differences and provides context for changes, without indicating any actual transformation.
  - Context: Includes a rule key with a UUID for the applied migration rule.
  - Example:

```
{
      "sequence": 4,
      "type": "change-info",
      "subject": "69c4f8f3-549a-4103-b8f6-f8627b689e12",
      "message": "Property [assume-role-sts-region] has bee
n changed as follows: [Allowed values have been changed. Val
ues removed: [eu-isoe-west-1]]",
      "context": {
        "rule": "6ca65082-09bd-4713-9fde-f5beb0722d1c"
      }
    }
```

- change
  - Subject: The entity (for example: a component or template) that has been modified during migration.
  - Message: Records actual changes applied to the flow during migration. These entries provide specifics on transformations in response to version differences.
  - Context: Includes a rule key with a UUID and may contain additional transformation details. While change-info describes general version differences in NiFi (for example: Processor ConsumeKafka_1_0 has been removed from NiFi), change entry details the specific transformations applied to your flow to address those changes (For example: Deprecated processor ea9178e3-ad81-4c6f-b23c-88a01e08ddac has been replaced with processor d9730bae-4f88-430e-801d-8805f6069988).
  - Example:

```
{
      "sequence": 39,
      "type": "change",
      "subject": "0a8d5eb6-6791-1faa-2303-b0adcf300df1",
      "message": "Property [Kerberos Principal] has been re
moved",
      "context": {
        "rule": "65d775f9-dc70-45c9-b4d4-e18a21f8ccba"
```

```
        }
    }
```

- control
  - Subject: Unspecified, often a Process Group
  - Message: Marks the progression of migration activities, such as the start or completion of a Process Group's migration.
  - Context: No widespread usage
  - Example:

```
{
      "sequence": 4,
      "type": "control",
      "subject": "0a8d5a99-6791-1faa-78ea-9582a48a4113",
      "message": "Group migration has started",
      "context": {
      }
    }
```

Troubleshooting flow migration issues

Some operation systems may encounter issues such as “too many open files”. To resolve this, adjustments must be made at the operating system level.