Brought to you by:

CLOUDERA

Apache® NiFi™

for dummies®

A Wiley Brand

Move data easily, securely, and efficiently

Learn how to configure processors

Set up NiFi to regulate dataflows

Cloudera Special Edition

Foreword by Mark Payne

Cloudera

At Cloudera, we believe that data can make what is impossible today, possible tomorrow. We empower people to transform complex data into clear and actionable insights. Cloudera delivers an enterprise data cloud for any data, anywhere, from the Edge to AI. Powered by the relentless innovation of the open source community, Cloudera advances digital transformation for the world's largest enterprises. Learn more at cloudera.com.

Attunity

Attunity is a leading provider of data integration and big data management software solutions that enable availability, delivery, and, management of data across heterogeneous enterprise platforms, organizations, and the cloud. Software solutions include data replication and distribution, test data management, change data capture (CDC), data connectivity, enterprise file replication (EFR), managed file transfer (MFT), data warehouse automation, data usage analytics, and cloud data delivery. Attunity has supplied innovative software solutions to its enterprise-class customers for over 20 years and has successful deployments at thousands of organizations worldwide.

Apache® NiFi™

Cloudera Special Edition

Foreword by Mark Payne

for dummies®
A Wiley Brand

Apache® NiFi™ For Dummies®, Cloudera Special Edition

Published by: **John Wiley & Sons, Inc.**, 111 River Street, Hoboken, NJ 07030-5774, www.wiley.com

ISBN 978-1-119-62406-6 (pbk); ISBN 978-1-119-62407-3 (ebk)

Manufactured in the United States of America

10 9 8 7 6 5 4 3 2 1

Table of Contents

Foreword

Nearly ten years ago, I was presented with an amazing opportunity. I was fortunate enough to join a team of three incredibly talented engineers to build a new platform. This platform would be responsible for handling the ever-increasing volumes of data that would be streamed through my organization. It would have to allow users to quickly add new sources of data on the fly and route, analyze, process, and transform the data, all before delivering it to its final destination. In short, the goal was to get the right data to the right place, in the right format and schema, at the right time.

Thus began a long and engaging journey. Over the next year, I would take on more and more responsibilities, ultimately taking full responsibility for the development of the framework. The volume and variety of the data continued to increase. My organization decided that if the software were to be open sourced, it would not only benefit us but also many others who were on a similar journey. So, in November of 2014, this software was donated to the Apache Software Foundation and became known as Apache NiFi.

Since its debut, NiFi has been adopted by companies and organizations across every industry. The authors of this book have been with them through it all—training the users, assessing the strengths and weaknesses of the platform, and even getting their hands dirty to improve the code. They have seen a vast number of different use cases for NiFi across these industries and been active in the day-to-day use of the software to solve their critical dataflow problems. I have had the pleasure of working alongside some of the authors to tackle their most difficult dataflow challenges and learn from their experiences. Others have written of their experiences using NiFi and established that they have a great understanding of not just the software itself but also where it came from and where it is heading.

This book is not intended to provide an in-depth understanding of every aspect of NiFi but rather is meant to provide an understanding of what NiFi is and explain when, how, and why to use NiFi. Additionally, it will explore the features that make the software unique. Through tutorials, examples, and explanations, it

provides an excellent overview and walkthrough of NiFi that will benefit the uninitiated and experienced users alike.

While reading this book, you will gain a firm grasp on NiFi fundamentals and how to use the software. You should also be able to relate them to some of the challenges that you are facing and understand how NiFi can help to address them. Most importantly, I hope that you enjoy the read and that it encourages you to read more about NiFi and explore it on your own.

Cheers,

Mark Payne

Apache NiFi Committer and PMC Member, Sr. Member of Technical Staff, Cloudera

Introduction

Apache NiFi was built to automate and manage the flow of data between systems and address the global enterprise dataflow issues. It provides an end-to-end platform that can collect, curate, analyze and act on data in real-time, on-premises, or in the cloud with a drag-and-drop visual interface.

About This Book

This book gives you an overview of NiFi, why it's useful, and some common use cases with technical information to help you get started, debug, and manage your own dataflows.

Icons Used in This Book

REMEMBER

Remember icons mark the information that's especially important to know.

TIP

The Tip icon points out helpful suggestions and useful nuggets of information.

WARNING

The Warning icon marks important information that may save you headaches.

Beyond the Book

NiFi is an open-source software project licensed under the Apache Software Foundation. You can find further details at `https://nifi.apache.org/`.

Where to Go from Here

The book is modular so you can start with Chapter 1, but feel free to roam around to the chapters that fit best. You can also pick out a topic that interests you from the Table of Contents.

IN THIS CHAPTER

- What to use NiFi for
- Learning NiFi core concepts
- Understanding the expression and query languages for NiFi

Chapter 1
Why NiFi?

The information age caused a shift from an industry-based economy to a computer-based economy. For organizations, the information age has led to a situation in which immense amounts of data are stored in complete isolation, which makes sharing with others for collaboration and analysis difficult.

In response to this situation, several technologies have emerged, such as Hadoop data lakes, but they lack one major component—data movement. The capability to connect databases, file servers, Hadoop clusters, message queues, and devices to each other in a single pane is what Apache NiFi accomplishes. NiFi gives organizations a distributed, resilient platform to build their enterprise dataflows on.

In this chapter, we discuss how Apache NiFi can streamline the development process and the terminology and languages that you need to be successful with NiFi.

The Advantages to Apache NiFi

The ability to bring subject matter experts closer to the business logic code is a central concept when building a NiFi flow. Code is abstracted behind a drag-and-drop interface allowing for

groups to collaborate much more effectively than looking through lines of code. The programming logic follows steps, like a white-board, with design intent being apparent with labels and easy-to-understand functions.

Apache NiFi excels when information needs to be processed through a series of incremental steps. Examples of this include:

- **Files landing on an FTP server:** An hourly data dump is made available by a vendor and needs to be parsed, enriched, and put in a database
- **Rest requests from a web application:** A website needs to make complex rest API calls and middleware must make a series of database lookups
- **Secure transmission of logs:** An appliance at a remote site needs to transmit information back to the core datacenter for analysis
- **Filtering of events data:** Event data is being streamed and needs to be evaluated for specific conditions before being archived

NiFi Core Concepts

NiFi is a processing engine that was designed to manage the flow of information in an ecosystem. Everything starts with a piece of data that flows through multiple stages of logic, transformation, and enrichment.

REMEMBER

When building flows in NiFi, keep in mind where the data is coming from and where it will ultimately land. In many ways NiFi is a hybrid information controller and event processor. An event can be anything from a file landing in an FTP to an application making a `REST` request. When you consider information flow as a series of distinct events rather than a batch operation, you open a lot of possibilities.

One of the biggest paradigm shifts teams may face is going from monolithic scheduled events to sequence of individual tasks. When big data first became a term, organizations would run gigantic SQL operations on millions of rows. The problem was that this

type of operation could only be done after the data was fully loaded and staged. With NiFi, those same companies can consider their SQL databases as individual rows at time of ingest. This situation allows for data to be enriched and served to the end consumer in a much faster and reliable fashion. Due to the fact that each row is individually analyzed, a corrupt value would only cause that individual event to fail rather then the entire procedure.

NiFi consists of three main components:

- **Flowfiles:** Information in NiFi consists of two parts: the attributes and the payload. Flowfiles typically start with a default set of attributes that are then added to by additional operations. Attributes can be referenced via the NiFi expression language, which you can find out about in the "NiFi Expression Language and Other Query Languages" section. The payload is typically the information itself and can also be referenced by specific processors.
- **Flowfile processors:** These do all the actual work in NiFi. They're self-contained segments of code that in most cases have inputs and outputs. One of the most common processors, `GetFTP`, retrieves files from an FTP server and creates a flowfile. The flowfile includes attributes about the directory it was retrieved from — such as, creation date, filename, and a payload containing the file's contents. This flowfile can then be processed by another common processor, `RouteOnAttribute`. This processor looks at an incoming flowfile and applies user-defined logic based on the attributes before passing it down the chain.
- **Connections:** These detail how flowfiles should travel between processors. Common connections are for `success` and `failure`, which are simple error handling for processors. Flowfiles that are processed without fault are sent to the `success` queue while those with problems are sent to a `failure` queue. Processors such as `RouteOnAttribute` have custom connections based on the rules created.

  Additional connection types may be `Not Found` or `Retry` and depend on the processor itself. Connections can also be `Auto–Terminated` if the user wishes to immediately discard a specific type of event. Configuring the advanced features of connections, such as backpressure, is covered in Chapter 3.

Figure 1-1 shows a basic flow incorporating these three basic concepts. A processor gets files from a local directory and creates flowfiles. These flowfiles go through the connection to another processor that puts the data into Hadoop.

REMEMBER

Processors can be turned on and off (started/stopped), which is indicated by a green triangle (running) or red square (stopped). A stopped processor doesn't evaluate flow files.

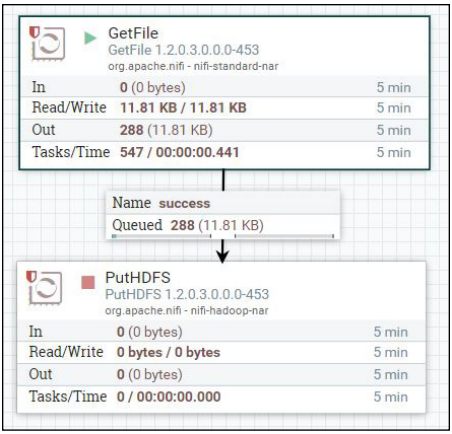

FIGURE 1-1: The first flow.

Processors are configurable by righting-click; four tabs are available:

- **Settings:** This tab allows you to rename how the processor appears and auto-terminate relationships. If a processor allows user-defined relationships to be created (such as `RouteOnAttribute`), they also appear here after created. More advanced settings, for example, penalty and yield duration, allow for how to handle re-trying flowfiles if the first attempt fails.
- **Scheduling:** NiFi provides several different scheduling options for each processor. For most cases, the Timer-Driven strategy is most appropriate. This can accommodate running on a specified interval or running as fast as NiFi can schedule it (when data is available) by setting the scheduling period to 0 seconds.

REMEMBER

Additionally, you can increase the concurrency on this tab. Doing so allocates additional threads to the processor, but to be mindful of the number of threads available to NiFi and oversubscription.

- **Comments:** Allows developers to add comments at the per processor level.
- **Properties:** This tab is where the processor's specific settings are configured. If the processor allows custom properties to be configured, click the plus sign in the top-right to add them. Some properties allow for the NiFi Expression Language.

TIP

To tell whether a property allows for the NiFi Expression Language, hover over the question mark next to the property name and see if the Supports Expression Language property is `true` or `false`.

NiFi Expression Language and Other Query Languages

The NiFi expression language is the framework in which attributes (metadata) can be interacted with. The language is built on the attribute being referenced with a preceding `${` and proceeding `}`. For example, if you want to find the path of a file retrieved by `GetFile`, it would be `${path}`. Additional terms can be added for transformation and logic expressions, such as `contains` or `append`. Multiple variables can be nested to have a multi-variable term. Examples include

- **Check whether the file has a specific name**

```
${filename:contains('Nifi')}
```

- **Add a new directory to the path attribute**

```
${path:append('/new_directory')}
```

- **Reformat a date**

```
${string_date:toDate("yyyy-MM-DD")}
```

- **Mathematical operations**

```
${amount_owed:minus(5)}
```

- **Multi-variable greater than**

```
${variable_one:gt(${variable_two})}
```

Some processors require a Boolean expression language term to filter events such as `RouteOnAttribute`, shown in Figure 1-2.

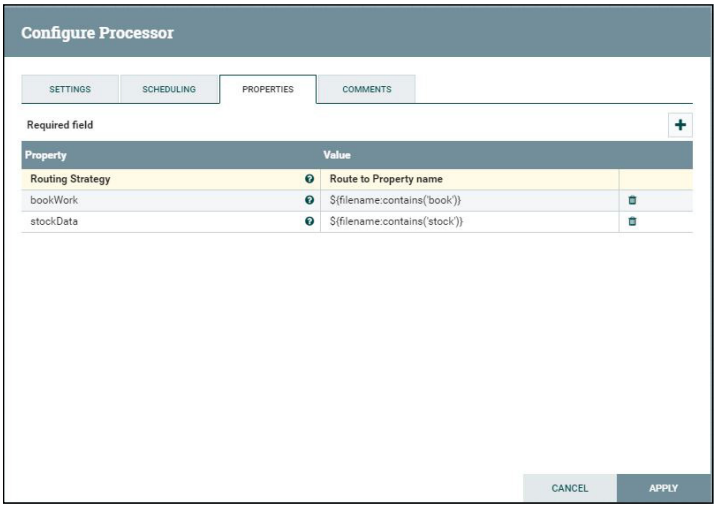

FIGURE 1-2: This processor requires a Boolean expression.

While others, such as `UpdateAttribute`, allow more freeform use of the language, as shown in Figure 1-3.

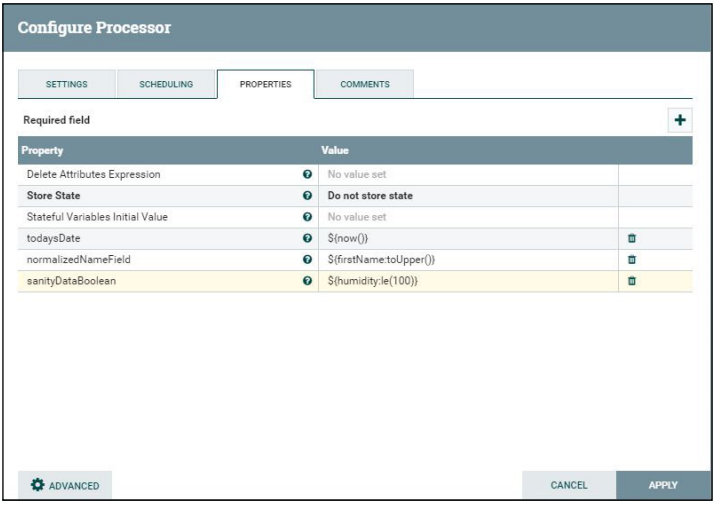

FIGURE 1-3: This processor gives you a lot of freedom in your language.

JSONPath

When referencing JSONs with processors such as `EvaluateJsonPath`, you use the JSONPath expression language. In this language, the JSON hierarchy is referenced with a `$` to represent the root and the names of the nested fields get a value, such as `$.account.user_name.first_name`. For example, you can enumerate a list of accounts, such as `$.account[0].user_name.first_name`. Additional complex operations are available:

- **Search any level of JSON for a field called `Version`**

```
$..Version
```

- **Filter only for subversions greater than 5**

```
$..Version[?(@.subVersion>5)]
```

XPath/XQuery

The XPath/Query language is available for accessing data in XMLs through processors such as `EvaluateXPath` and `EvaluateXQuery`.

Much like JSONPath, it allows for data to either be exactly specified or searched:

- **Specify the value of the account holder's first name**

```
/account/user_name/first_name
```

- **Specify the value of the first account if multiple accounts are present in the XML**

```
/account[0]/user_name/first_name
```

- **Search any level of XML for a field called `Version`**

```
//Version
```

- **Filter only for subversions greater than 5**

```
//Version[subVersion>5]
```

IN THIS CHAPTER

- Importing a NiFi template
- Creating a NiFi dataflow
- Understanding the Hello World example

Chapter 2
Getting Started with NiFi

Apache NiFi is one of the most flexible, intuitive, feature rich dataflow management tools within the open-source community. NiFi has a simple drag-and-drop user interface (UI), which allows administrators to create visual dataflows and manipulate the flows in real time and it provides the user with information pertaining to audit, lineage, and backpressure.

For example, to really begin to understand some of the capabilities, it's best to start with a simple Hello World dataflow. The traditional Hello World example (as every technologist is used to starting with when learning any programming language) is a bit different with a dataflow management tool such as NiFi. This simple example demonstrates the flexibilities, ease of use, and intuitive nature of NiFi.

In this chapter, we explain how to import a NiFi template, create a NiFi dataflow, and how data is processed and stored.

DOWNLOADING NIFI AND CLONING A REPO

Installing and starting NiFi is outside the scope of this book. Information on how to install and start NiFi can be found at `https://nifi.apache.org/docs.html`.

You can download Apache NiFi at `https://nifi.apache.org/download.html`. The demo GitHub repository we follow throughout this book was tested with version 1.1.0 but should work with later versions of NiFi. Once NiFi is running, you need to clone or download the GitHub repository to follow along with the Hello World dataflow example in this chapter.

To clone the GitHub repository from a command line, type the following in a command line

```
git clone https://github.com/drnice/NifiHelloWorld.git
```

To download the GitHub repository, point your web browser at `https://github.com/drnice/NifiHelloWorld`, click the green Clone or Download button, and select Download ZIP. After it's downloaded, extract the zip file.

After the repository is cloned or downloaded and extracted, note the location of the files. Remember to change the location we use in this chapter to your location.

Note: Update these files in the repository to reflect your location:

- `server.sh` (the content of this file points to the location of `wsclient.html`, and should point to a location on the computer)
- `server.sh = cat /Users/drice/Documents/websocket/index.html`
- `index.html` (which is included in the repo)

Importing a NiFi Template

When you have NiFi successfully installed and running, you can import the `HelloWorld.xml` file, which is a NiFi template cloned from the GitHub repository (see the sidebar on how to clone the GitHub repository).

Follow these steps to import a NiFi template (see Figure 2-1):

1. **Launch NiFi by pointing your browser to this location `http://localhost:9090/nifi/` (or similar location based on how NiFi was installed).**
2. **Click the Upload Template button within the Operate window.**
3. **Click the magnifying glass icon.**
4. **Browse to the location where you downloaded the GitHub repository, and select the `HelloWorld.xml` file.**
5. **Click Open and then Upload.**

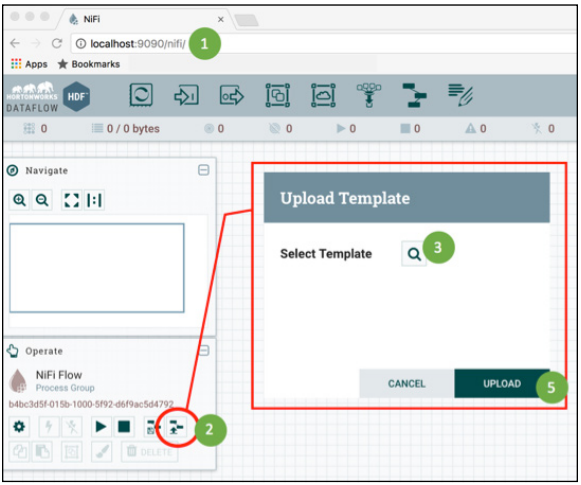

FIGURE 2-1: Importing `HelloWorld.xml` NiFi template.

Adding a NiFi Template to the NiFi Canvas

With the NiFi template uploaded, you can add the Hello World template to the NiFi canvas by following these steps (see Figure 2-2):

1. **Click the Template icon in the grey navigation bar at the top of the screen and drag and drop it anywhere on the canvas.**

2. **In the Add Template window, choose the Hello World template.**
3. **Click the Add button.**

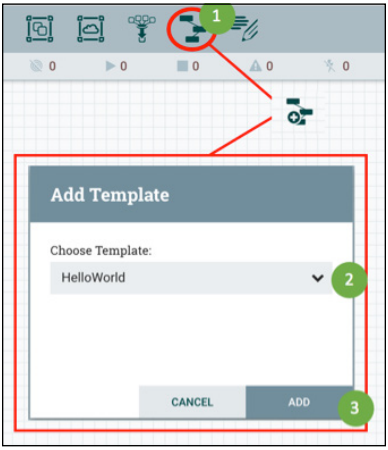

FIGURE 2-2: Adding the Hello World template to the NiFi canvas.

The Hello World dataflow opens on your canvas, as shown in Figure 2-3. The NiFi processors have a yellow exclamation icon next to them. You need to modify these processors (along with the `PutFile` processor and the `ExecuteStreamCommand` processor) for the Hello World dataflow to work.

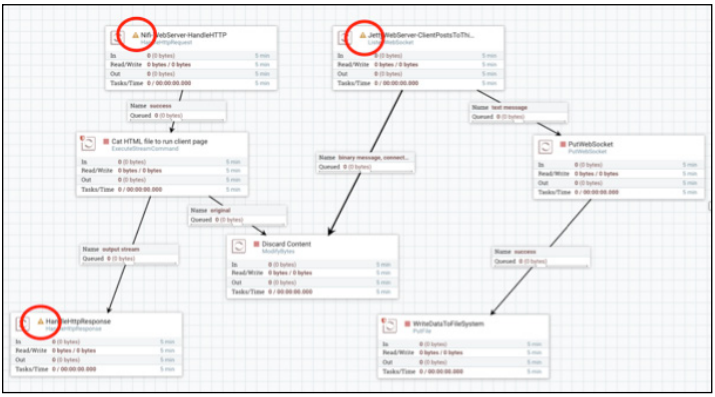

FIGURE 2-3: The Hello World example dataflow.

Setting Up and Running the Hello World Example

With the Hello World template added onto the NiFi canvas, you can clean up some properties and configurations within the processors that have a yellow exclamation icon next to them. The yellow exclamation icon indicates those processors need attention.

Configuring HandleHttpRequest NiFi processor

Start by correcting the `HandleHttpRequest` NiFi processor by following these steps:

1. **Right-click the `Nifi-WebServer-HandleHTTP` NiFi processor and select Configure.**
2. **In the Configure Processor window, click the Properties tab.**
3. **Click the right-pointing arrow in the third column next to `StandardHttpContextMap` (see Figure 2-4).**

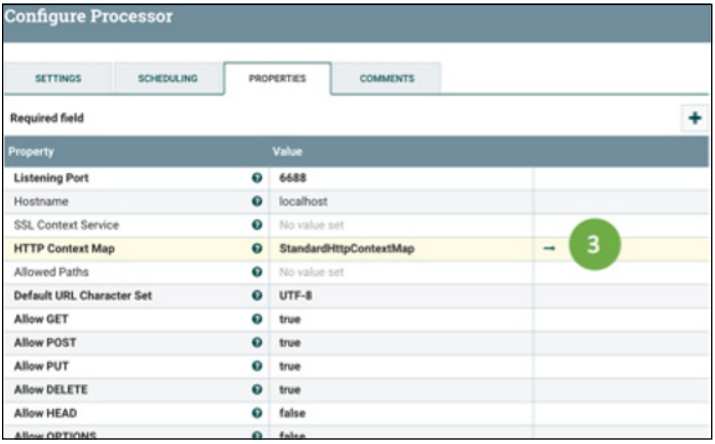

FIGURE 2-4: Configure the `HandleHttpRequest` processor.

4. **In the Process Group Configuration window, click the lightning bolt on the same row as the `StandardHttpContextMap` to enable this controller service (see Figure 2-5).**

5. **Leave the Scope as Service Only and click Enable and then Close.**
6. **Close the Process Group Configuration window.**

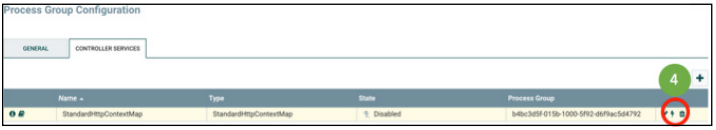

FIGURE 2-5: Enable the `StandardHttpContextMap` controller.

With the `HandleHttpRequest` NiFi Processor configured properly, there is now a red box next to it, indicating the processor isn't running.

Configuring the other processors

When you know how to configure the `HandleHttpRequest` NiFi processor, you can easily configure the other processors needed to run the Hello World dataflow: `ListenWebSocket`, `PutFile`, and `ExecuteStreamCommand`. Just as you do with `HandleHttpRequest`, right-click the processor you want to configure, and select Configure. Then on the Properties tab of the Process Group Configuration window, change these settings for each processor:

- **`ListenWebSocket`:** Click the right-pointing arrow in the third column next to `JettyWebSocketServer`. Click its lightning bolt to enable the service. Select Service Only for the Scope. Then click the Enable and Close buttons.
- **`PutFile`:** Change the Directory property to the directory where you want the data from the web application to write to and click the Apply button.
- **`ExecuteStreamCommand`:** Change the Command Path property to the directory where you downloaded the NiFi HelloWorld GitHub repository and extracted the `server.sh` file. Then click the Apply button.

When you have the processors configured correctly, each has a red box next to it, indicating the processor isn't running, instead of the yellow exclamation icon.

Running the Hello World example

To start the Hello World flow, follow these steps (see Figure 2-6):

1. **Click anywhere on the NiFi canvas so that nothing is selected (meaning no processor or connection is highlighted).**
2. **Click the Start button on the left side of the Operate window.**

   Every red box for each processor changes to a green triangle pointing to the right, indicating the processor is running.

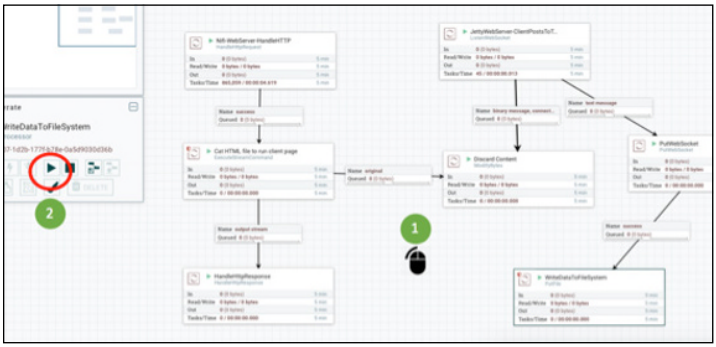

FIGURE 2-6: Running the Hello World dataflow.

Now you can launch a web browser that NiFi is hosting. Follow these steps (see Figure 2-7):

1. **Point a web browser to `http://localhost:6688/`.**

   You see a text box along with three buttons: Open, Send, and Close.
2. **Click the Open button.**
3. **Enter** Hello World **in the text box.**
4. **Click the Send button.**

   Congratulations, you just submitted Hello World through your NiFi flow.

To validate that Hello World flowed through NiFi, open the destination path defined in the `PutFile` NiFi processor and you see a file that contains the phrase "Hello World" inside it.

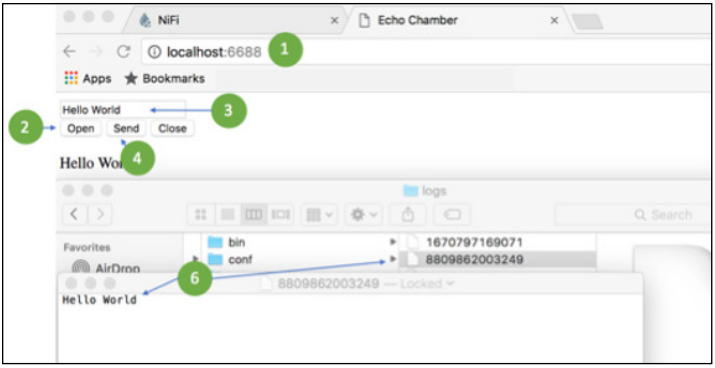

FIGURE 2-7: Executing the Hello World demo.

Understanding the Hello World Example

Now that you've successfully executed the Hello World dataflow, you can better understand how it works behind the scenes.

The `HandleHttpReqest` NiFi processor starts a HTTP server in NiFi and listens for HTTP requests on a specific port. In this case, the NiFi processor is already set up on port 6688. When you pointed your browser to `http://localhost:6688`, NiFi handled this request and passed it to the next process, which executed the `server.sh shell` script.

This shell script launches the HTML file inside of NiFi. Like any webserver, the call is then routed to see whether the response is successful and this is how the HTML file is presented in the web browser.

The content of the `index.html` file is a simple form. The Open command established a connection from the web client to the `ListenWebSocket` connection over port 9998 (the `ListenWebSocket` was pre-configured to listen to this port when you imported the NiFi template). After the connection is established, you can publish any data over the web socket connection.

Eventually, the data put on the `WebSocket` connection is routed to the `PutFile` processor within NiFi where the data is stored on a local disk in the directory you specified.

IN THIS CHAPTER

- Getting information through the NiFi's user interface
- Setting up backpressure to help processes run smoothly
- Debugging with provenance
- Getting information from the NiFi server logs

Chapter 3
General Debugging & Monitoring

The last thing anyone wants is for a processor to stop running unexpectedly, which prevents it from completing or preventing others from running altogether. Fortunately, NiFi offers several methods to monitor them.

In this chapter, we discuss how to interpret information about your processes through the user interface, set up backpressure to allow NiFi to regulate itself, use provenance to help with debugging efforts when things don't go as planned, and monitoring processes through the NiFi server logs when you want more detail.

Debugging through the User Interface

You can glean a lot of information right from the user interface, through the status bar, the Summary window, and the Status History menu.

Status bar

The status bar is located at the top of the user interface under the drag-and-drop toolbox. It provides metrics related to:

- Nodes in the cluster
- Threads running
- Flowfile count and content size
- Remote process groups in transmitting or disabled state
- Processors status (for example, which ones are running, stopped, invalid, disabled, the last time the UI was refreshed)

The amount of information reported in the status bar is minimal. When you need more in-depth information, choose the Summary found in the menu.

Summary

The Summary window contains tabs for processors, input ports, output ports, remote process groups (RPGs), connections, and process groups.

The processor groups located on the canvas contain their own status bars and general metrics; they're also available on the Summary's Process Groups tab in a tabular report. The Connections tab provides basic information as well: name, relation type being connected, the destination, queue size, % of queue threshold used, and output size in bytes. Metrics that track the input and outputs are tracked over a five-minute window.


TIP

Funnels are notoriously used as anti-patterns to store flowfiles that may not be expiring. While valid in development, such setups can back up and cause other operational issues. With the Summary's Connections tab, all the funnels on the canvas can be identified to validate that they're being used downstream and not just dead ends from other processors, as shown in Figure 3-1.

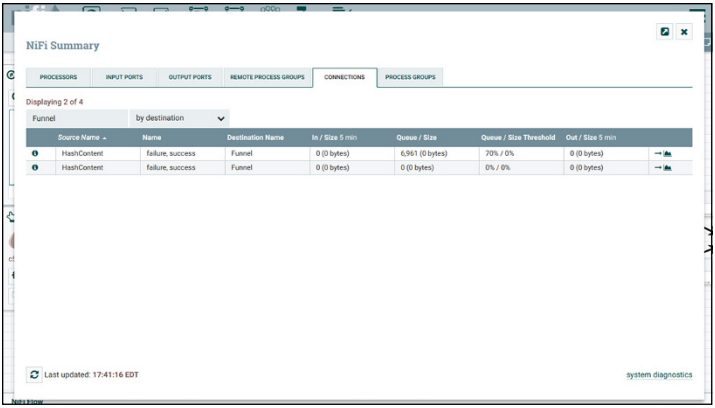

FIGURE 3-1: You can keep track of funnels through the Connections tab.

Additionally, at the bottom right of the Summary on any tab is the system diagnostics link. There are three tabs:

- The **JVM** tab shows metrics about on and off-heap utilization and garbage collection counts along with total time.
- **System,** as shown in Figure 3-2, shows the number of CPU cores and the amount of space used on the partition that the repository is stored on.
- **Version** contains detailed build numbers of NiFi, the version of Java NiFi is running with, and details on the OS.

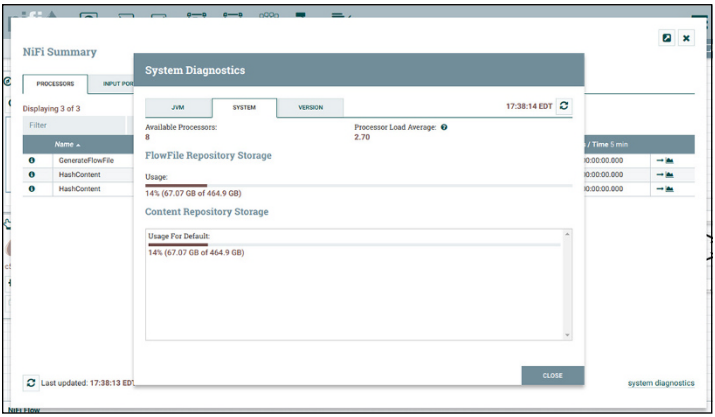

FIGURE 3-2: The system storage use for the NiFi repositories.

Status History

The Status History menu is one of the most useful features, next to Data Provenance, in debugging a slow flow. The Status History menu contains all the generic information expected such as the name and the time the status has been collected for. The graph in the menu, as shown in Figure 3-3, can visualize many metrics related to the processor or connection (over 5 minutes) and includes separate plotting for each NiFi node in the cluster.

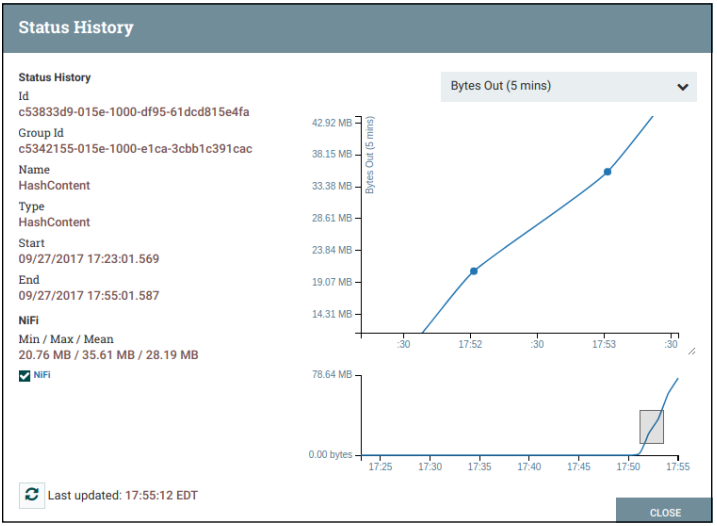

FIGURE 3-3: The graph and information that you can find via the Status History menu.

TIP

The line graph in the Status History menu can be zoomed in by dragging from one part of the graph to the next.

Configuring Backpressure

Backpressure isn't typically thought of as a first-class citizen in many data movement systems, but NiFi provides it as a first-class feature. From an operational perspective, backpressure enables you to design a system that self regulates the amount of storage it's utilizing to prevent it from crashing! In NiFi, backpressure is configured at the connection level, where you can manage the backpressure policies.

It's important to understand how the backpressure is configured in the flow to understand its behavior. If a connection is ever completely utilized by storage or flowfile count, the processor upstream of the connection stops processing and waits for room to be made in the connection. This problem is typically caused by a slow processor downstream of the connection that can't consume the flowfiles as fast as upstream processors can place them into the connection queue.

The simplest visual to monitor appears on the Connections tab of the Summary window with both flowfile and storage size icons that represent capacity: Green (most), Yellow, and Red (least). (See the earlier "Summary" section for more about the Connections tab.) It also lists the source and target processor names along with utilization and last five-minute metrics. The Connections tab can be targeted for a specific node or the entire NiFi cluster.


TIP

Use this information to identify specific ingestion methods that skew, such as a Kafka processor that has a high skew to a specific keyed partition where a specific NiFi node would be responsible for a single partition receiver.

You can configure backpressure from two perspectives: value and infrastructure storage:

- **Value:** Accepting that it's impossible to store everything is the first step to designing systems that have the capability to self regulate their contents.
- **Storage:** The infrastructure itself has physical limitations on total storage available for Flowfile, Content, and Provenance repositories to use. By defining the value of specific messages in the flow, the flowfiles of lesser importance can be dropped while holding onto more important ones.

NiFi supports three ways to configure the backpressure policies of a connection:

- **Flowfile count:** Ensure only a specific number of files are in the queue to be processed and expire others.
- **ContentSize:** Limit the amount of downstream flow storage used and also ensure that the total storage for all connections is set up in a manner that prevents the flow from filling a disk completely.

» **Time:** Expire data that remained in the queue for too long so that it no longer holds any value in being processed.

WARNING

Filling the storage mounts on a NiFi server can lead to very odd behavior of the repositories, which can result in requiring special actions to restore normal operation of the NiFi server that filled. Refer to the Summary menu's system diagnostics link for detailed storage use by the NiFi nodes (see the earlier section).

The Flowfile repository is much smaller than the Content repository. For example, a sample flow in a NiFi cluster with three nodes holding 32,500 flowfiles totaling 872.5MB results in the repositories on a single node (1/3) looking like the following table. It's important to note that in a cluster, work is distributed and some servers could have more work depending on the ingestion and transformations taking place in the flow on each server.

Node	Repository	Size
1	Content	285MB
1	Flowfile	5.4MB
2	Content	305MB
2	Flowfile	6.2MB
3	Content	265MB
3	Flowfile	5.9MB

TIP

If you're trying to empty a queue and flowfiles remain, the downstream processor may have a lease on the files. Stopping the processor allows you to clear out these files.

Checking Provenance

NiFi's data provenance provides debugging capabilities that allow for flowfiles themselves to be tracked from start to end inside the workflow. Flowfiles can have their contents inspected, downloaded, and even replayed. The combination of these features enables ease of troubleshooting to find out why a specific path was taken in the workflow. This capability allows you to make minor changes to the workflow based on what occurred and replay the message to ensure the new path is correctly taken.

Both processors and connections have data provenance available by right-clicking; alternatively, you can access the complete Provenance repository from the Provenance menu. The Provenance menu includes the Date/Time, ActionType, the Unique Flowfile ID, and other stats. On the far left is a small 'i' encircled in blue; click this icon, and you get the flowfile details. On the right, what looks like three little circles connected together is Lineage.

Lineage is visualized as a large directed acyclic graph (DAG) that shows the steps in the flow where modifications or routing took place on the flowfile. Right-click a step in the Lineage to view details about the flowfile at that step or expand the flow to understand where it was potentially cloned from. At the very bottom left of the Lineage UI is a slider with a play button to play the processing flow (with scaled time) and understand where the flowfile spent the most time or at which point it got routed.

Inside the flowfile details, you can find a detailed analysis of both the content and its metadata attributes. More interesting metrics are potentially the Queue Positions and Durations along with the worker the flowfile is located on. The Content tab allows you to investigate before and after versions of a flowfile after it's been processed (see Figure 3-4); just read the data in the browser or download it for later. To correct a problem, use the Replay capability to make a connection to the flow and replay the flowfile again. (And then inspect it again to be sure it runs the way you want.)

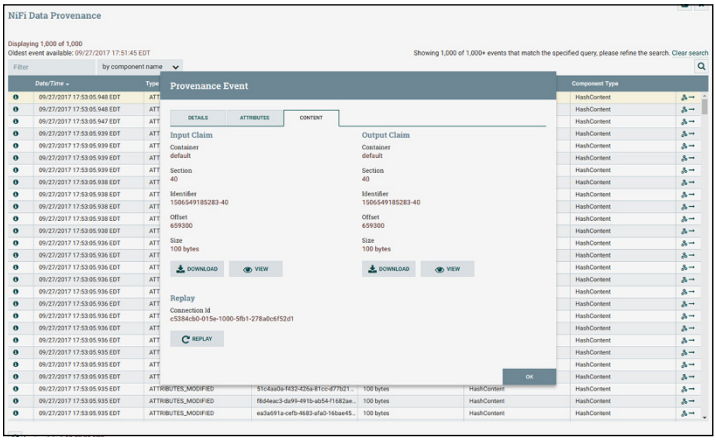

FIGURE 3-4: A major advantage to provenance is the capability to view the before and after content of a flowfile.

Checking the NiFi Server Logs

Each NiFi server has a set of application and bootstrapping logs. NiFi uses SLF4J to provide a robust and configurable logging framework that you can configure to provide as much detail as you want. The logs contain detailed information about processes occurring on the server.

By default, the NiFi server logs are located in the `logs/` directory found in the NiFi folder. This folder is also called `$NIFI_HOME`. If you downloaded and untarred/unzipped NiFi, the directory is `NIFI_HOME`.

The `nifi-app.log` application log contains more details about processors, remote process groups (for site to site), Write Ahead Log functions, and other system processes.

The `nifi-bootstrap.log` bootstrap contains entries on whether the NiFi server is started, stopped, or dead. It also contains the complete command with classpath entries used to start the NiFi service.


TIP

The `logback.xml` located in `$NIFI_HOME/conf` can be edited on the fly without having to restart NiFi. It takes approximately 30 seconds before the new logging configuration takes effect. The log level can be configured per node and isn't a clusterwide configuration. For example, to only change the Processor log level, edit the `logback.xml` and change the logger line for `org.apache.nifi.processors` to `INFO` from `WARN`.

IN THIS CHAPTER

- Importing data into a database
- Pushing the data over a HTTP connection
- Polling and saving data for later use

Chapter 4
NiFi Use Cases

Planning the first NiFi data integration project requires attention to:

- **Data volume and velocity:** Pulling flat files from a monitored directory often leads to large files made available infrequently, which can require lots of memory to process. In contrast, when data is pushed from an external source to a NiFi listener over a TCP/IP port, each data row tends to be small in size but is received frequently.
- **Data types to ingest:** NiFi has the ability to support both binary and text data sets.
- **Capacity of connected systems:** When considering system capacity needed, pay close attention to each of the connected systems' capabilities to accept the data as it becomes available, support temporary content data storage, and store data long term.

While NiFi can support many different ingest use cases, in this chapter we examine three sample use case scenarios.

Importing Datasets into a Database

In this scenario, the requirement is to monitor a directory on disk and on a scheduled basis read all the files in the directory, validate the data in those files, and finally write the valid records into a database.

The flow consists of the following steps:

1. Periodically scans the directory and fetches the contents whenever a new file arrives.

   The `ListFile` and `FetchFile` processors accomplish this step.

2. Validates the contents of the data.

   The `ValidateRecord` processor, which accomplishes this step, is configured with a schema that describes how the data should look. In this case, the flow routs invalid records to a processor and writes them to an `errors` directory. The `ValidateRecord` processor is configured with a Record Reader so that it's capable of processing any kind of record-oriented data, such as CSV, JSON, Avro, or even unstructured log data.

3. Publishes the data to the database using the `PutDatabaseRecord` processor.

   Again, this processor uses a Record Reader so that it can process any record-oriented data.

Failures during data integration can happen, so you need to plan for errors. NiFi handles this nicely through the ability to route exceptions into a failure queue to either hold for reprocessing or to write to an output file for offline editing.

The sample scenario validates that the data file contents match a schema and any invalid records are written to an `error` directory via the `PutFile` processor. This processor can also be stopped to hold the data in the queue within NiFi. From here, you can review the input content, make any necessary corrections to the flow, and reprocess the data. See Figure 4-1.

At the same time, any records that did match the schema are written to the database by the `PutDatabaseRecord` processor.

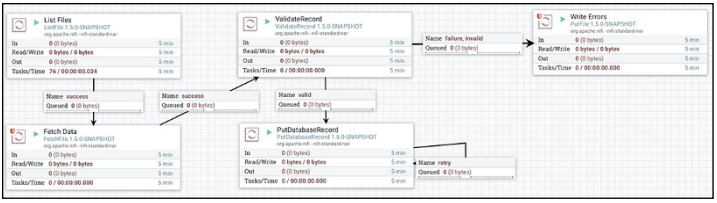

FIGURE 4-1: The steps to this dataflow.

This processor allows you to configure where the data should be published and how, including mapping of field names in the data to database column names.

TIP

The concepts used for this NiFi dataflow can be extended to a variety of different import scenarios — for example, pulling from a relational database or a RESTful query.

Listening for HTTP Posts

In this scenario, an external system pushes the data over a HTTP connection into NiFi using a `POST` option. The approach to set up a listener operation is as simple as inserting the `ListenHTTP` processor, as shown in Figure 4-2.

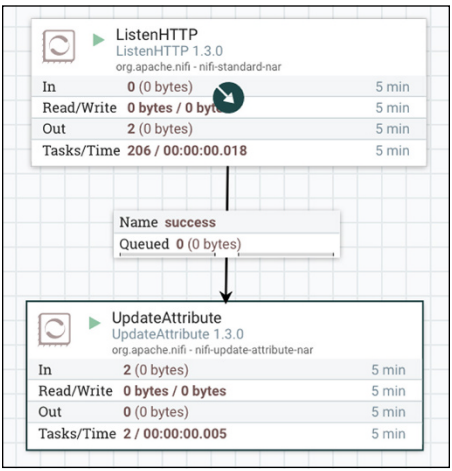

FIGURE 4-2: This flow inserts a ListenHTTP processor.

This flow is listening on the same server the NiFi process is running on and listening to a port configured in the `ListenHTTP` processor, as shown in Figure 4-3.

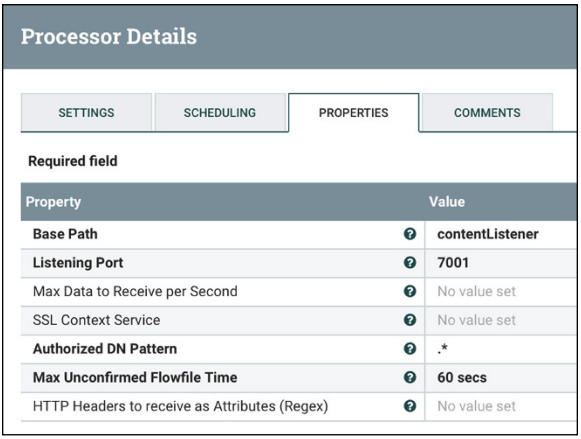

FIGURE 4-3: The ListenHTTP processor includes a port for listening.

With this configuration, any HTTP `POST` operation to the NiFi server using the URL `http://{nifi server address}:7001/contentListener` gets picked up by the listener and then gets passed on to the next step in the dataflow.

REMEMBER

The capability to specify a base path (the example in Figure 4-3 specifies `contentListener`) means that it's possible to have multiple readable HTTP endpoints for all the different workflows supported by your NiFi instance.

Polling a RESTFul API to Extract a JSON Attribute

In this scenario, the flow executes RESTful queries every 30 seconds and then saves the result as a NiFi attribute for use in other downstream functions, as shown in Figure 4-4.

The first step in this scenario is to poll the `RESTful` API to query the Yahoo Weather service every 30 seconds and return a flow file in JSON format. To control the polling interval, specify 30 seconds for the `Run Schedule` property on the Scheduling tab.

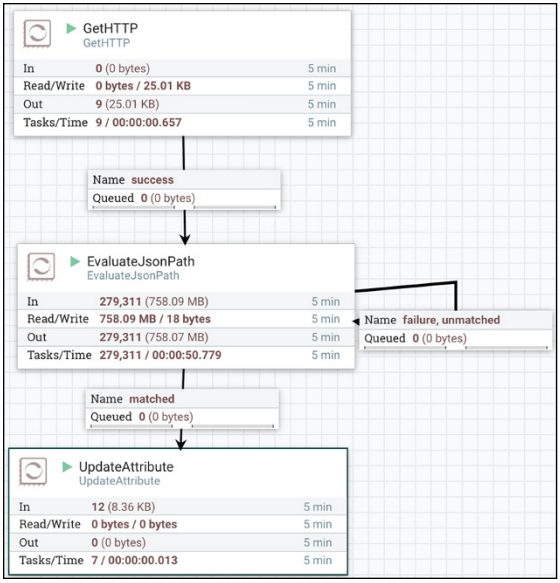

FIGURE 4-4: This flow shows a RESTful query runs every 30 seconds and saves the result.

Without any programming, you can specify a HTTP URL on the Properties tab to make the Yahoo Weather query.

To make this use case work, you need to extract the wind speed from the full JSON object. The data returned by the `GetHTTP` processor is actually a fairly complex JSON object. This is a problem because the objective is to only pull out the wind speed attribute from within this large JSON object.

To extract just the `Wind Speed` JSON attribute, you need to add a NiFi attribute using the `EvaluateJsonPath` processor, as shown in Figure 4-5.

To add it, click the + sign in the top-right corner of the `Evaluate JsonPath` processor Properties tab and add the `wind_speed` attribute. Looking at the output from the `EvaluateJsonPath`, you find a new attribute with an actual `wind_speed` value.

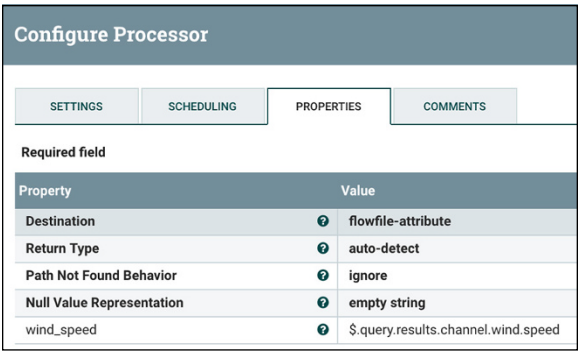

FIGURE 4-5: Use `EvaluateJsonPath` processor to extract the wind speed.

IN THIS CHAPTER

- Ingesting data with NiFi
- Using log-based CDC with NiFi
- Integrating NiFi with source systems
- Looking at performance for each source type

Chapter 5
Generating NiFi Streams from Data Sources

Organizations often struggle to ingest data from many sources in an efficient, cost-effective, and scalable fashion. They must manage hundreds and even thousands of data feeds and can't depend on custom development for every source system.

In this chapter, we explore the options on how to generate data streams for ingestion into Apache NiFi.

Options for Data Ingest with NiFi

NiFi has a variety of options for ingesting data from simple bulk data transfer to more sophisticated real-time data movement that automatically responds to source metadata changes. Figure 5-1 shows a maturity model that highlights the data ingest approaches organizations typically take on their journeys with NiFi.

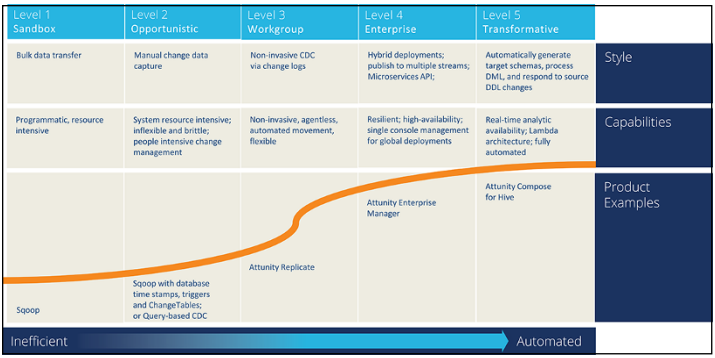

FIGURE 5-1: The NiFi data integration maturity model.

Pairing Log-Based Change Data Capture with NiFi

Change data capture (CDC) replicates just the most recent production data and metadata changes that the source has registered during a given time period, typically seconds or minutes. CDC can be the primary replication method, or it can accompany batch loads. In the latter case, CDC captures updates during the batch replication process, applies them to the target, and then maintains fully updated target data once the batch job is complete. Like full load, CDC can copy data from one source to one target, or one source to multiple targets.

CDC has two primary benefits:

- It enables faster and more accurate decisions based on the most current data.
- It reduces the need for, or even replaces, batch loading.

REMEMBER

This second benefit is critical because it makes replication more feasible for a variety of use cases. For example, killing the batch window with CDC speeds up production operations. Your analytics team might be willing to wait for the next nightly batch load to run their queries (although that's increasingly less common). But even then, companies can't stop their 24/7 production databases for that long bulk/batch load. CDC also makes cloud migrations and cloud analytics more feasible. Sending a continuous stream of small replicas over the Wide Area Network (WAN) consumes less bandwidth and time than a full-load data transfer.

Given these benefits, an increasing number of organizations are using CDC, both as a feature of replication platforms such as Attunity Replicate and as a feature of broader extract, transform, and load (ETL) offerings such as Microsoft SQL Server Integration Services (SSIS).

CDC engines can identify and copy changes in several ways:

- **Timestamps** in a dedicated table column can record the time of the last update, thereby flagging any row containing data more recent than the last CDC replication task. For example, IBM InfoSphere CDC uses this method. While timestamps are a straightforward method of tracking updates, they can slow operations by frequently querying production tables. They also can create complexity as administrators need to ensure time zones are accurately represented.
- **Triggers** log transaction events in an additional "shadow" table that can be "played back" to copy those events to the target on a regular basis. For example, Oracle GoldenGate CDC uses this method when replicating data from SQL Server 2016. While triggers enable the necessary updates from source to target, the creation of these additional tables can increase processing overhead and slows operations.
- **Log readers**, as the name implies, identify new transactions by scanning change logs from a remote server. This method is often the fastest and least disruptive of the CDC options because it requires no new table and doesn't query production operations. Attunity Replicate uses this approach. One consideration here is that some source databases and data warehouses, such as Teradata, don't have change logs, and therefore require CDC engines to regularly query the production database for changes.

Attunity Replicate empowers organizations to accelerate data replication, and ingest and stream across a wide range of heterogeneous databases, data warehouses, and big data platforms. It moves data easily, securely, and efficiently with minimal operational impact.

Using Attunity Replicate, organizations can accelerate data movement to NiFi without the need for manual coding. The software can

- Simplify massive ingestion into big data platforms from thousands of sources
- Automatically generate target schemas based on source metadata
- Efficiently process big data loads with parallel threading
- Use change data capture process (CDC) to maintain true real-time analytics with less overhead

Integrating NiFi with New Sources

NiFi and Cloudera Data Flow (CDF) can help organizations harness the power of Industrial IoT. Traditional technologies, such as the extract, transform, load (ETL) tools, simply aren't suited to handle the complexity and volumes inherent in big data. Thus, enterprise CDC tools come into play to manage that complexity.

Whether you want to aggregate information from remote sensors into an on-premises data warehouse, on-premise data lake, or cloud-based versions of either, you may want to reconsider whether ETL is truly needed. In most situations, when heavy transformations are not required, an ELT model is better suited for big data.

When leveraging ELT, a best practice recommended and supported by Hadoop applications, organizations can move data rapidly, easily, and securely over WAN networks and low-bandwidth connections that include satellites and the cloud. Real-time replication of data to analytics platforms means access to faster, better business insights and more sustainable sources of competitive advantage.

TIP

Here are a few best practices to consider when integrating data from different sources:

- **Review the operational impact to source performance.** When designing streaming applications to include historical processing, being able to support known database queries, especially common patterns from source systems, typically means better performance and focus on the customer need. For example, creating aggregates and summaries of data within the data warehouse structure itself is a common practice.

- **Assess the impact on business operations.** Isolate business intelligence (BI) and analytics workloads from production systems to ensure operations and transactions run affected.
- **Reduce performance overhead to the collection of historical data.** Transaction systems typically only have current data. By storing snapshots of this data collectively over a period of time, organizations can gradually build historical datasets for analytics.

Here are selection criteria for real-time replication tools that integrate with these data sources:

- Minimal production impact
- Automation of manual tasks
- Broad platform support — data warehouses (DWs), databases (DBs), legacy data, NoSQL, Hadoop, cloud
- Scalability — add sources, targets, platform types with no/minimal process change
- Security — data access governance, secure WAN transfer
- Performance, including WAN transfer
- Filtering of replicated data
- Fault resilience and recoverability
- Source schema change propagation
- API integration with third-party management frameworks and dashboards

CDC technology has become a strategic component of integration between operational databases and BI architectures. Given today's ever-increasing struggle for more immediate access to real-time data and information, constant pressure on efficiency costs, and the exponential growth of underlying data volumes, CDC tools used for extraction have become a must have for a modern data lake project. Furthermore, the advantages of an architecture that combines powerful CDC and ETL with streaming tools such as NiFi are now hard to ignore.

Here are some key considerations when integrating with NiFi:

- **Data capture architecture options** include agents-based extracts, intermediate servers to cache and carry across corporate networks, and extracting from the transactional/journal logs versus query/trigger-based CDC processing.
- **CDC replication options** are sequential transactions, optimized batches, integration with native DW loaders, and/or message streaming.

Considering Performance by Source Type

Each type of source that you connect with NiFi performs differently, which you need to take into consideration.

Relational databases and data warehouses

A CDC tool is required for timeliness to extract, design, build, and process the information and integrate from the operational data stores. Business users need up-to-date information, with low latency ranging from hours to minutes or even seconds. This requirement is fueled by competitive pressures, new business models, and customer satisfaction requirements. The data in the warehouse is a few days old, typically updated daily or weekly. Information in a data warehouse is normally in a summary format and doesn't effectively address operational requirements. Data volumes are doubling every one to two years, making the option of moving the entire source data impractical or even infeasible. Traditionally, the entire source database is extracted, transformed, and then loaded to the target data warehouse or data mart. These batch windows represent periods of time when the operational system isn't working, but rather processing data for ETL purposes. Continuous uptime and the need to use every bit of resources for transaction processing means that batch windows in many cases are no longer permissible by the business. While NiFi can accommodate batch windows with JDBC extracts, increasingly IT organizations are simply copying production data updates real time into separate data lakes. CDC technologies are ideal for these scenarios and strategies.

Mainframe

Information residing in mainframe systems, such as DB2 iSeries or AS/400, VSAM, IMS or Adabas, is usually mission critical and carries historical business logic. Using a CDC solution eliminates the need to move data in periodic batches, thereby keeping that data in sync and providing real-time access to the data in other platforms. Log-based CDC for DB2 (leveraging the DB2 journals) as well as for VSAM and IMS eliminates the need for repeated brute-force data queries that incur a hefty MIPS price tag.


TIP

Look for CDC solutions that can unlock mainframe data without incurring the complexity and expense that come with sending ongoing queries into the mainframe database.

Messaging systems

A solution that can capture non-relational data should be able to deliver the information in a way that can be easily processed by other tools such as ETL or Enterprise Application Integration (EAI). NiFi can natively extract from JMS and messaging platforms.


TIP

If changes are processed by an ETL tool that uses SQL, look for a solution that can normalize the non-relational data and provide a relational metadata model. If changes are processed by a NiFi tool, typically in XML, JSON, or any other operational updates, look for a solution that can map the legacy data source into an XML document with a corresponding XML schema that represents the original record hierarchy.

File system

Often as applications store log files and historical transactional stories are slowly FTPed into mount points full of flat files, NiFi was built to listen on end points and process them. Furthermore, NiFi applications and processors have been built to incorporate greater processing with these scenarios. As these solutions are integrated, the native connectors have optimizations to make this possible.

Chapter 6
Six NiFi Resources

Here we present six resources that provide more information to help you successfully use Apache NiFi:

- HelloWorld sample code from Chapter 2 `https://github.com/drnice/NifiHelloWorld`
- Apache Technical Wiki `https://cwiki.apache.org/confluence/display/NIFI/FAQs`
- Cloudera DataFlow `http://bit.ly/nifi-for-dummies-cdf-page`
- 451 report: Cloudera updates DataFlow for IoT data processing at the edge `http://bit.ly/nifi-for-dummies-cdf-update`
- Download Cloudera DataFlow `http://bit.ly/nifi-for-dummies-download-cdf`
- Cloudera Flow Management Datasheet `http://bit.ly/nifi-for-dummies-cfm-datasheet`

Manage your data-in-motion with Apache NiFi

Apache NiFi is an easy to use, powerful, and reliable system to process and distribute data. It provides an end-to-end platform that can collect, curate, analyze, and act on data in real-time, on-premises, or in the cloud with a drag-and-drop visual interface. This book offers you an overview of NiFi along with common use cases to help you get started, debug, and manage your own dataflows.

Inside...

- Discover the advantages of Apache NiFi
- Create a NiFi dataflow
- Troubleshoot processors
- Learn about three NiFi use cases
- Accelerate data replication

CLOUDERA

Go to Dummies.com®
for videos, step-by-step photos, how-to articles, or to shop!

ISBN: 978-1-119-62406-6
Not for resale

for dummies® A Wiley Brand

Also available as an e-book

WILEY END USER LICENSE AGREEMENT

Go to www.wiley.com/go/eula to access Wiley’s ebook EULA.