Apache NiFi 3

Using DataFlow Provenance Tools

Date of Publish: 2019-03-15

https://docs.hortonworks.com/

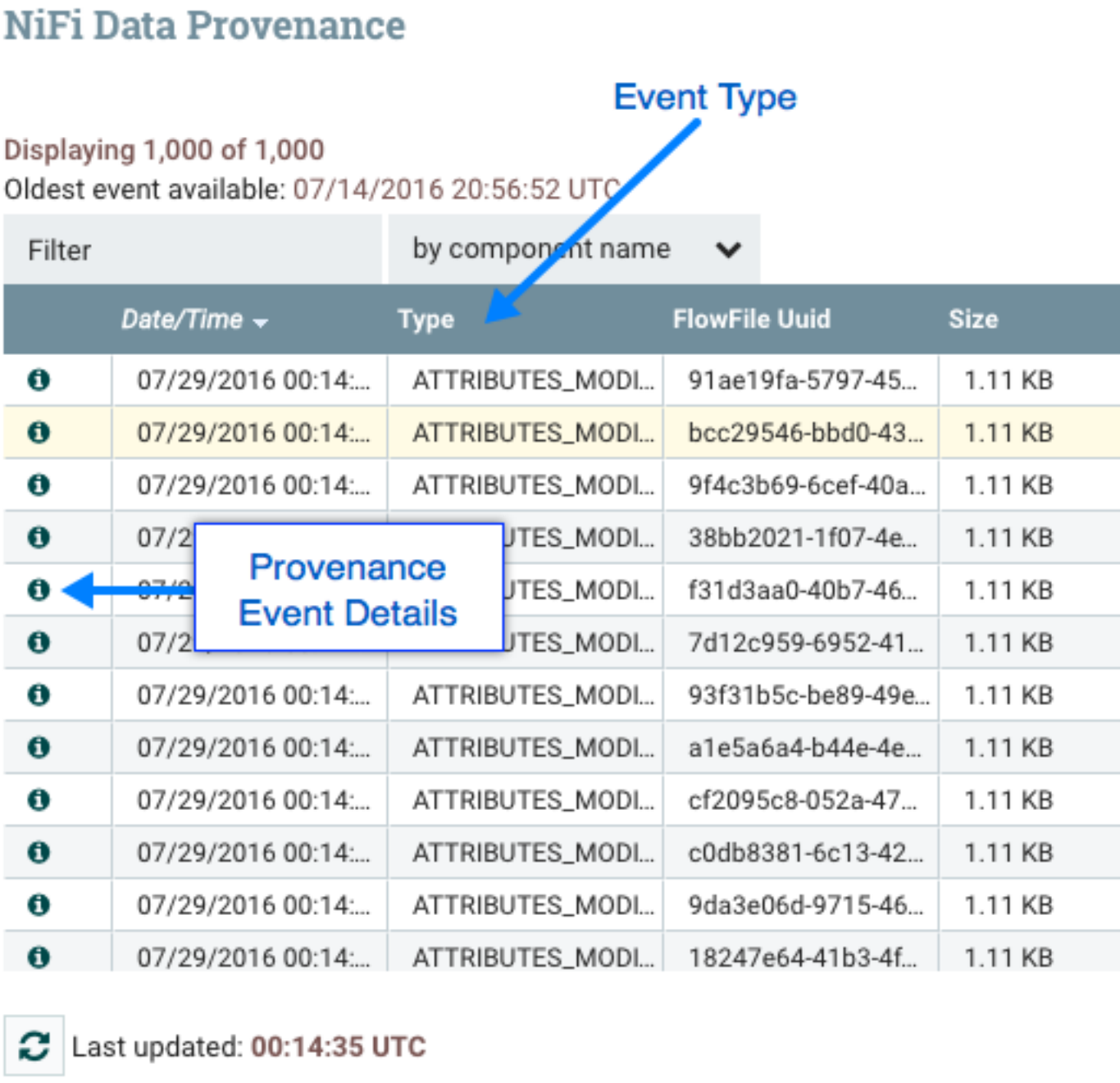
Contents

Data Provenance

While monitoring a dataflow, users often need a way to determine what happened to a particular data object (FlowFile). NiFi's Data Provenance page provides that information. Because NiFi records and indexes data provenance details as objects flow through the system, users may perform searches, conduct troubleshooting and evaluate things like dataflow compliance and optimization in real time. By default, NiFi updates this information every five minutes, but that is configurable.

To access the Data Provenance page, select "Data Provenance" from the Global Menu. This opens a dialog window that allows the user to see the most recent Data Provenance information available, search the information for specific items, and filter the search results. It is also possible to open additional dialog windows to see event details, replay data at any point within the dataflow, and see a graphical representation of the data's lineage, or path through the flow. (These features are described in depth below.)

When authorization is enabled, accessessing Data Provenance information requires the 'query provenance' Global Policy as well as the 'view provenance' Component Policy for the component which generated the event. In addition, access to event details which include FlowFile attributes and content require the 'view the data' Component Policy for the component which generated the event.



Provenance Events

Each point in a dataflow where a FlowFile is processed in some way is considered a 'provenance event'. Various types of provenance events occur, depending on the dataflow design. For example, when data is brought into the flow, a RECEIVE event occurs, and when data is sent out of the flow, a SEND event occurs. Other types of processing events may occur, such as if the data is cloned (CLONE event), routed (ROUTE event), modified (CONTENT_MODIFIED

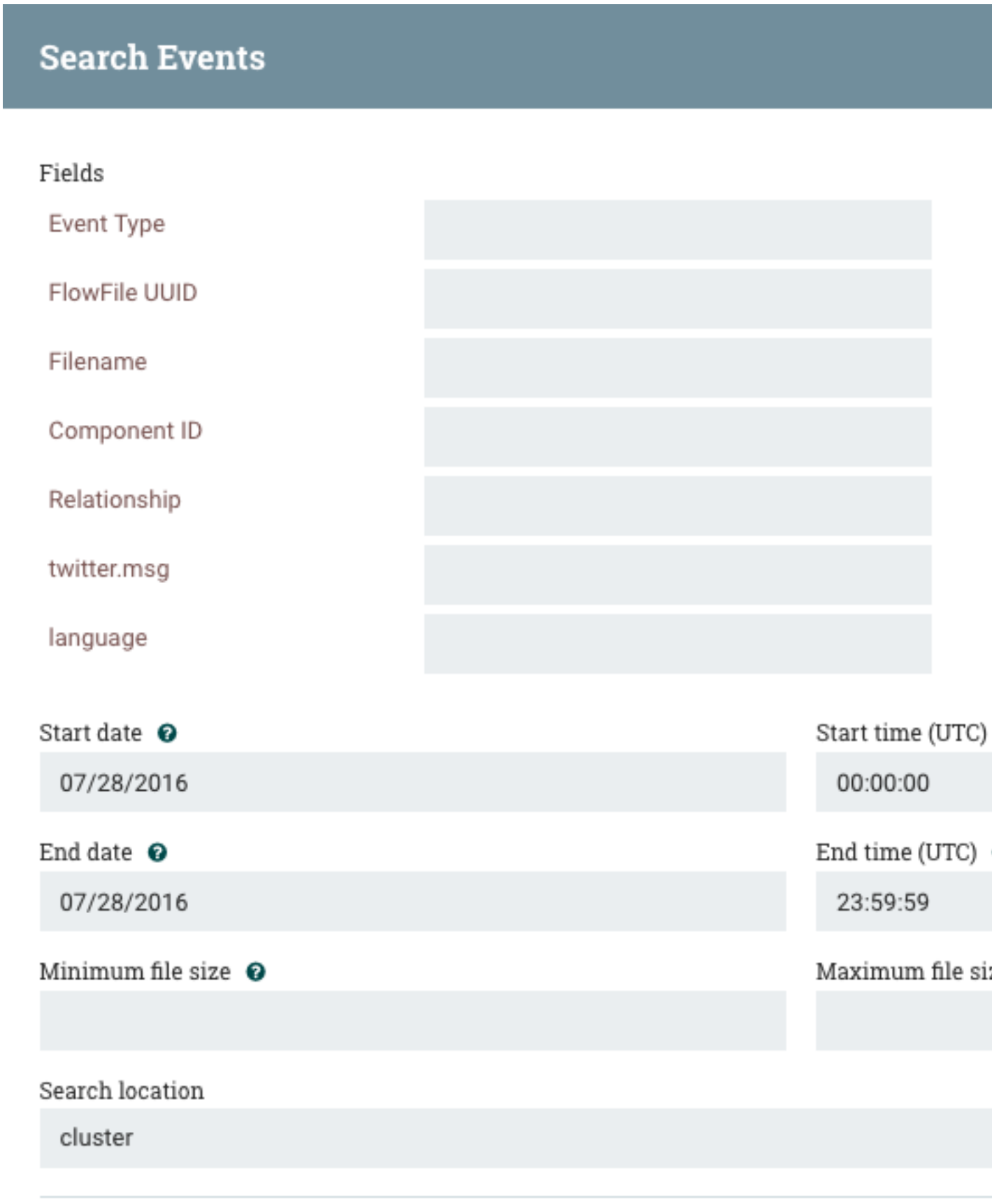
or ATTRIBUTES_MODIFIED event), split (FORK event), combined with other data objects (JOIN event), and ultimately removed from the flow (DROP event).

The provenance event types are:

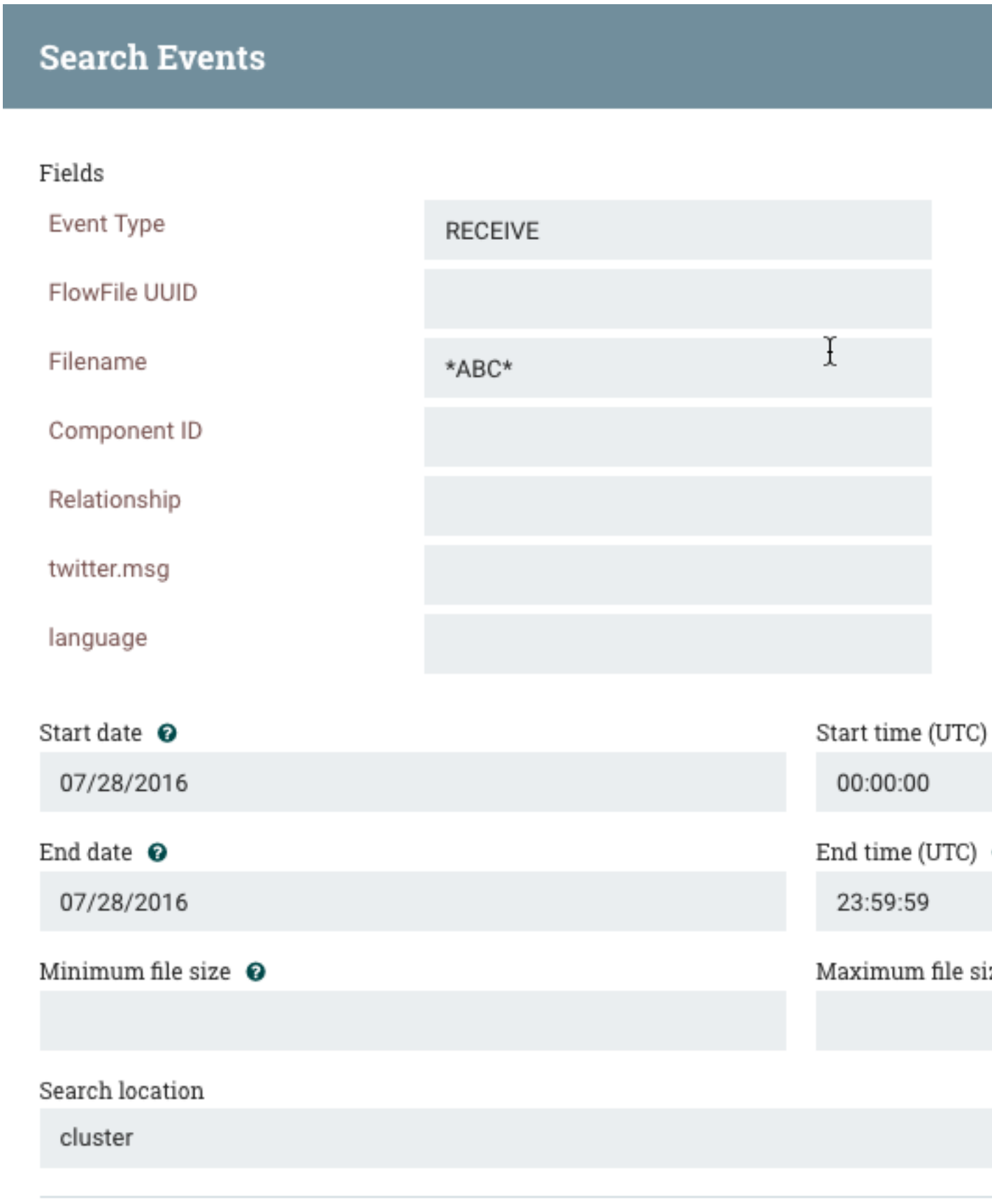
Provenance Event	Description
ADDINFO	Indicates a provenance event when additional information such as a new linkage to a new URI or UUID is added
ATTRIBUTES_MODIFIED	Indicates that a FlowFile's attributes were modified in some way
CLONE	Indicates that a FlowFile is an exact duplicate of its parent FlowFile
CONTENT_MODIFIED	Indicates that a FlowFile's content was modified in some way
CREATE	Indicates that a FlowFile was generated from data that was not received from a remote system or external process
DOWNLOAD	Indicates that the contents of a FlowFile were downloaded by a user or external entity
DROP	Indicates a provenance event for the conclusion of an object's life for some reason other than object expiration
EXPIRE	Indicates a provenance event for the conclusion of an object's life due to the object not being processed in a timely manner
FETCH	Indicates that the contents of a FlowFile were overwritten using the contents of some external resource
FORK	Indicates that one or more FlowFiles were derived from a parent FlowFile
JOIN	Indicates that a single FlowFile is derived from joining together multiple parent FlowFiles
RECEIVE	Indicates a provenance event for receiving data from an external process
REPLAY	Indicates a provenance event for replaying a FlowFile
ROUTE	Indicates that a FlowFile was routed to a specified relationship and provides information about why the FlowFile was routed to this relationship
SEND	Indicates a provenance event for sending data to an external process
UNKNOWN	Indicates that the type of provenance event is unknown because the user who is attempting to access the event is not authorized to know the type

Searching for Events

One of the most common tasks performed in the Data Provenance page is a search for a given FlowFile to determine what happened to it. To do this, click the Search button in the upper-right corner of the Data Provenance page. This opens a dialog window with parameters that the user can define for the search. The parameters include the processing event of interest, distinguishing characteristics about the FlowFile or the component that produced the event, the timeframe within which to search, and the size of the FlowFile.

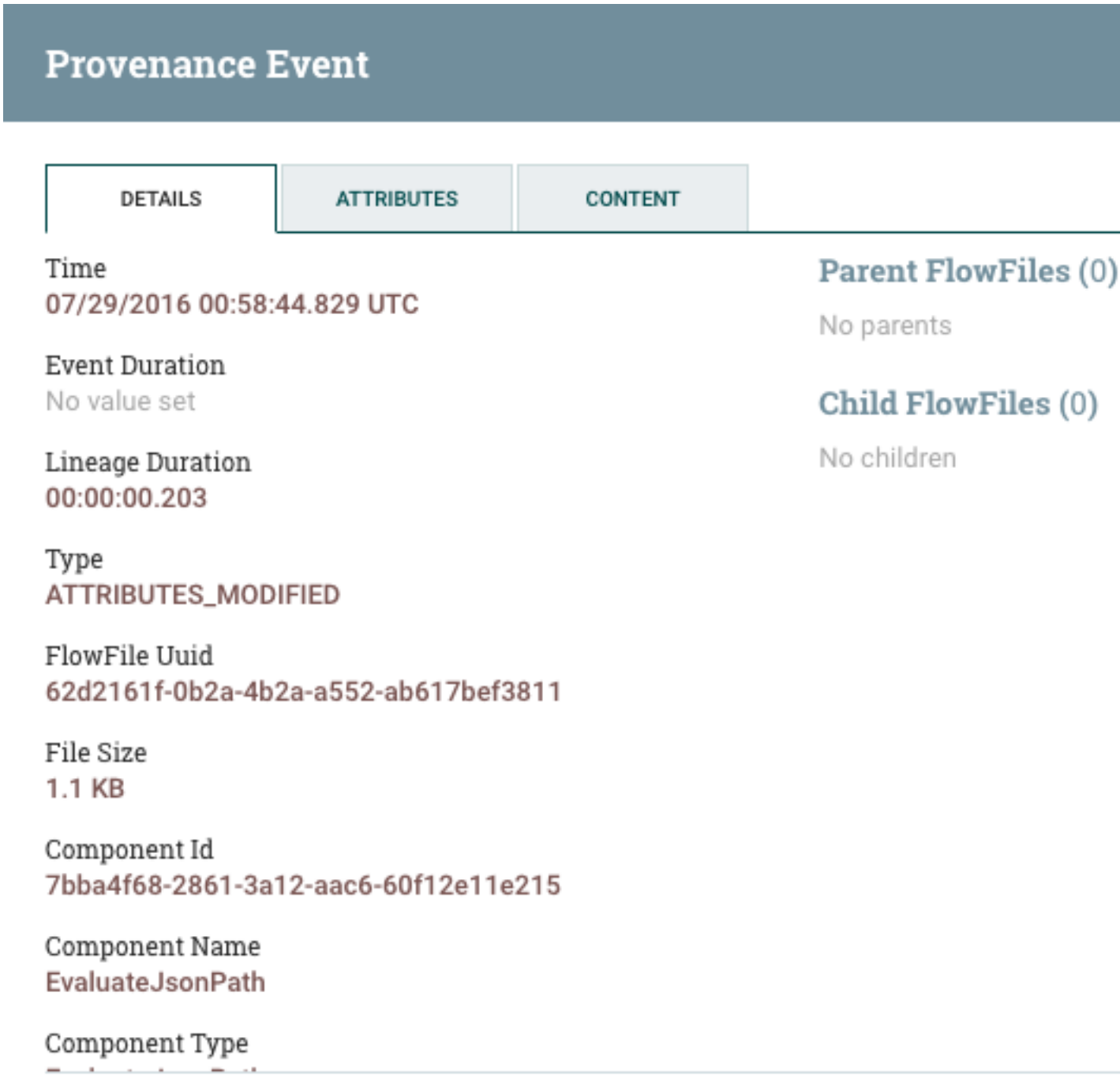

Search Events
Fields
Event Type
FlowFile UUID
Filename
Component ID
Relationship
twitter.msg
language
Start date
07/28/2016
Start time (UTC)
00:00:00
End date
07/28/2016
End time (UTC)
23:59:59
Minimum file size
Maximum file siz
Search location
cluster

For example, to determine if a particular FlowFile was received, search for an Event Type of "RECEIVE" and include an identifier for the FlowFile, such as its uuid or filename. The asterisk (*) may be used as a wildcard for any number of characters. So, to determine whether a FlowFile with "ABC" anywhere in its filename was received at any time on Jan. 6, 2015, the search shown in the following image could be performed:


Search Events
Fields
Event Type
RECEIVE
FlowFile UUID
Filename
ABC
Component ID
Relationship
twitter.msg
language
Start date
07/28/2016
Start time (UTC)
00:00:00
End date
07/28/2016
End time (UTC)
23:59:59
Minimum file size
Maximum file siz
Search location
cluster

Details of an Event

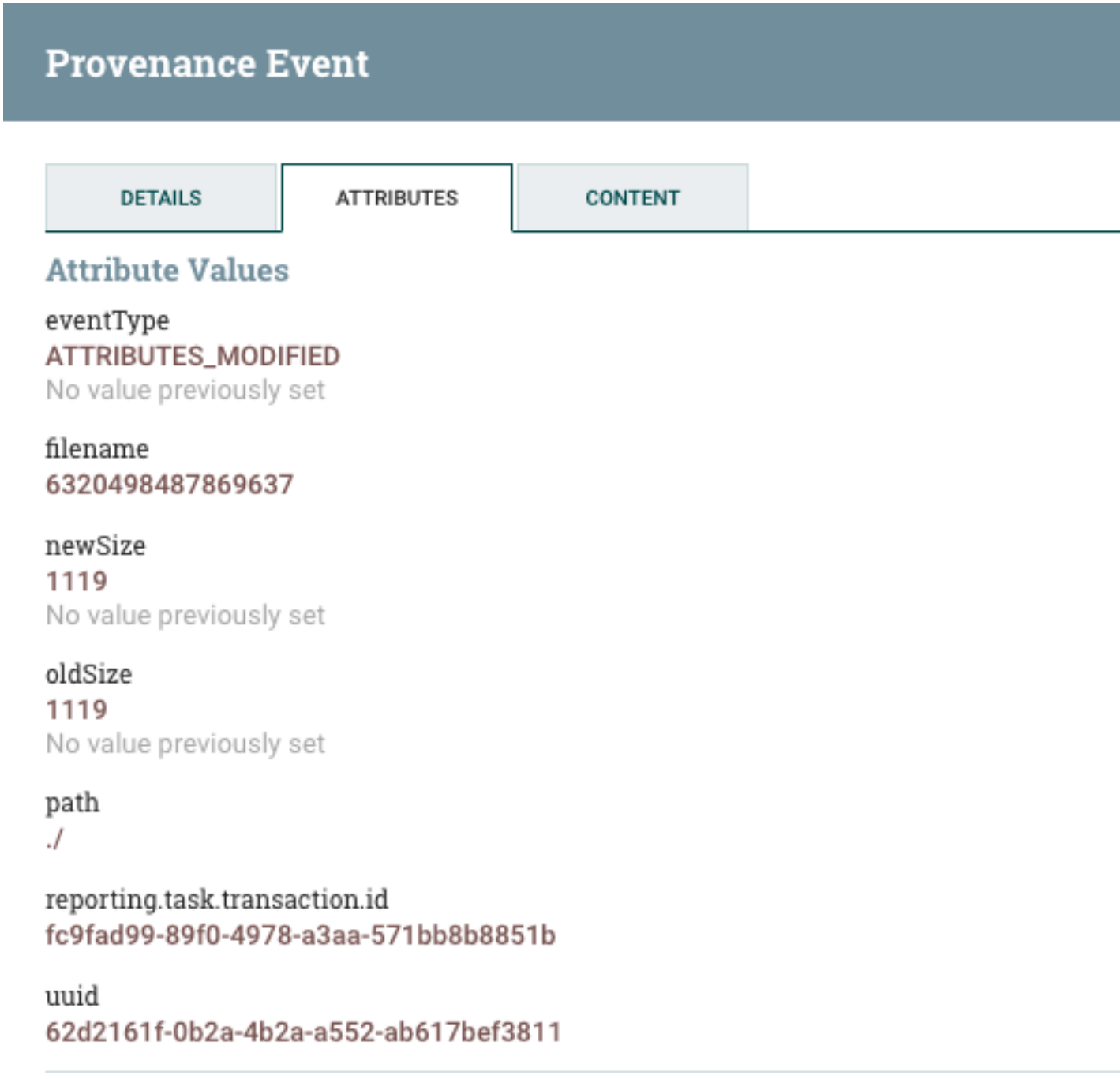
In the far-left column of the Data Provenance page, there is a View Details icon for each event ( ). Clicking this button opens a dialog window with three tabs: Details, Attributes, and Content.



The Details tab shows various details about the event, such as when it occurred, what type of event it was, and the component that produced the event. The information that is displayed will vary according to the event type. This

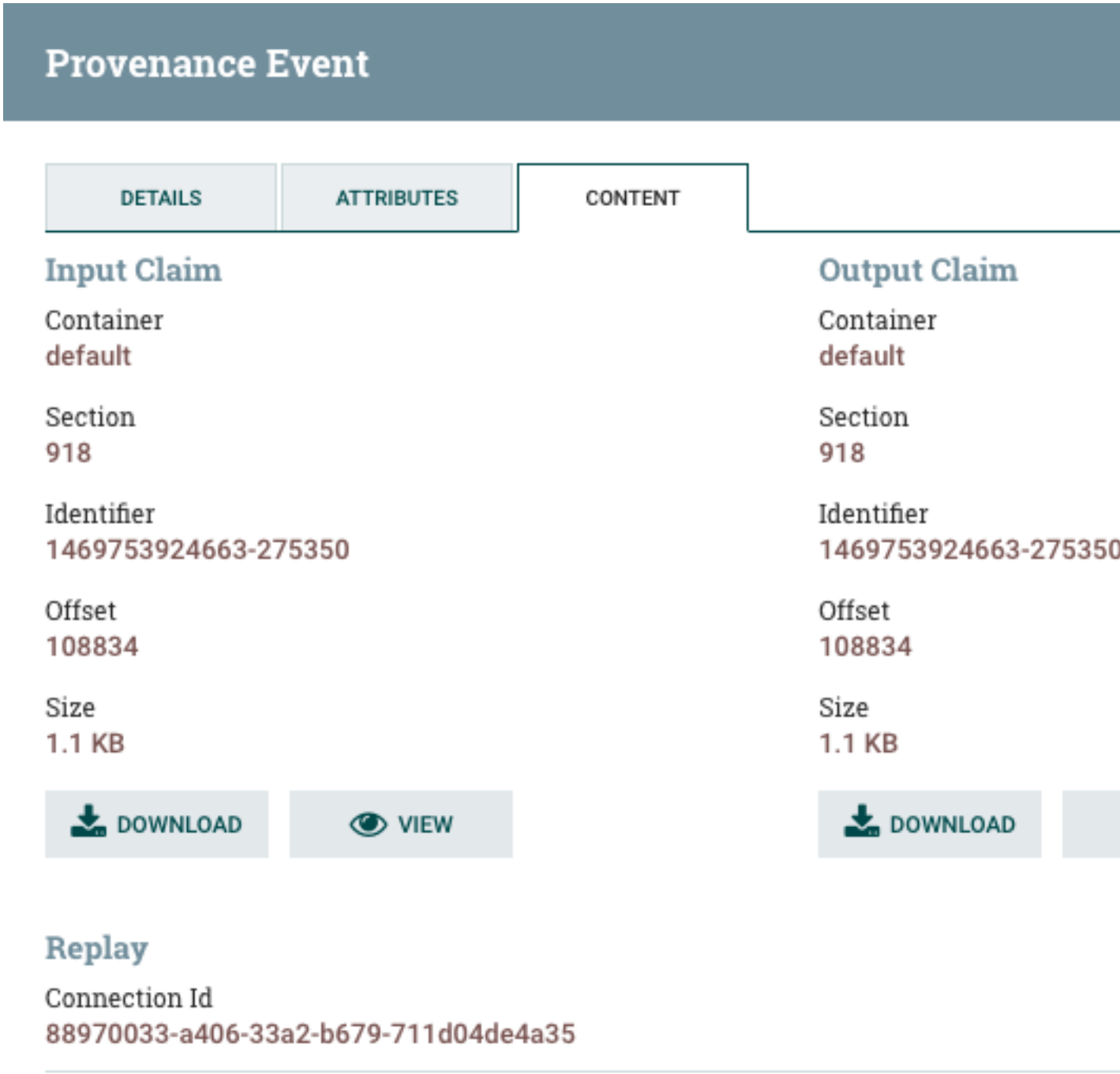
tab also shows information about the FlowFile that was processed. In addition to the FlowFile's UUID, which is displayed on the left side of the Details tab, the UUIDs of any parent or children FlowFiles that are related to that FlowFile are displayed on the right side of the Details tab.

The Attributes tab shows the attributes that exist on the FlowFile as of that point in the flow. In order to see only the attributes that were modified as a result of the processing event, the user may select the checkbox next to "Only show modified" in the upper-right corner of the Attributes tab.



Replaying a FlowFile

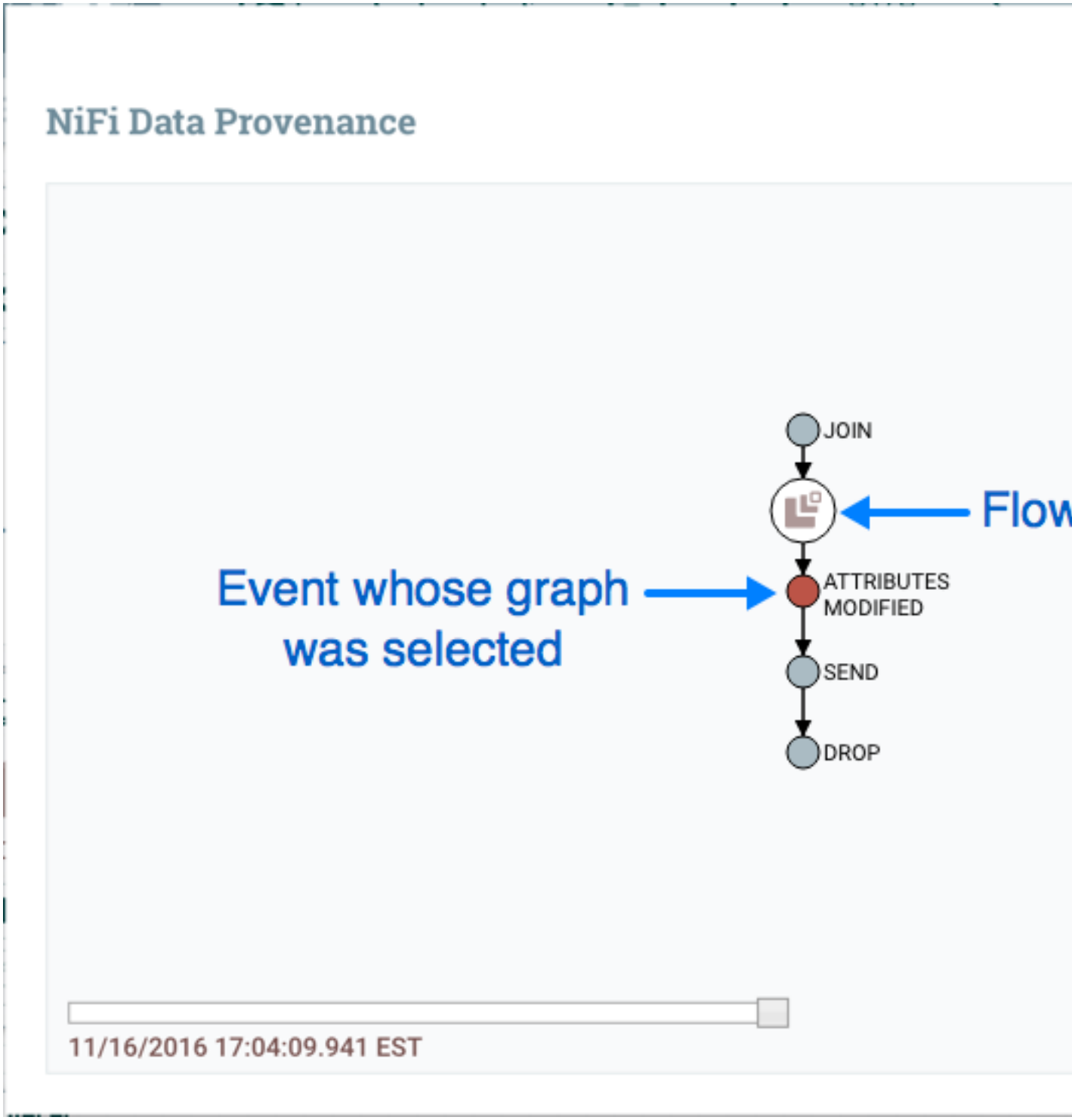
A DFM may need to inspect a FlowFile's content at some point in the dataflow to ensure that it is being processed as expected. And if it is not being processed properly, the DFM may need to make adjustments to the dataflow and replay the FlowFile again. The Content tab of the View Details dialog window is where the DFM can do these things. The Content tab shows information about the FlowFile's content, such as its location in the Content Repository and its size. In addition, it is here that the user may click the Download button to download a copy of the FlowFile's content as it existed at this point in the flow. The user may also click the Submit button to replay the FlowFile at this point in the flow. Upon clicking Submit, the FlowFile is sent to the connection feeding the component that produced this processing event.



Viewing FlowFile Lineage

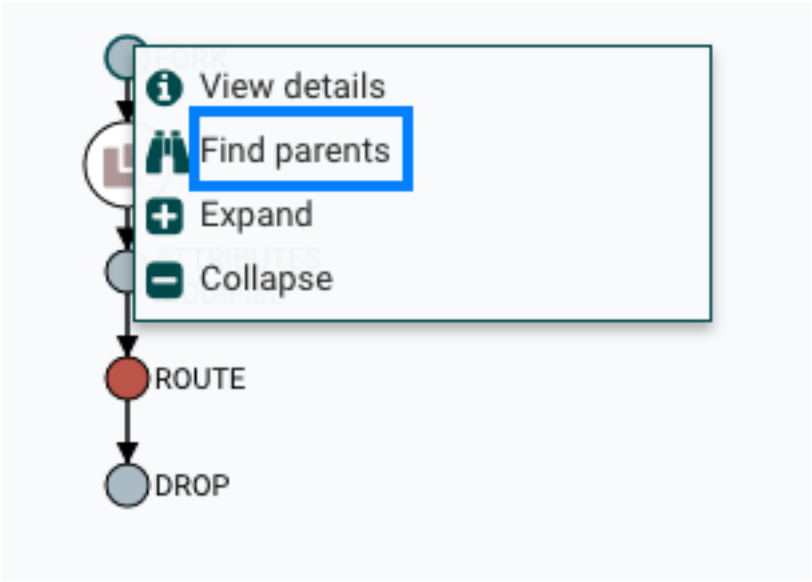
It is often useful to see a graphical representation of the lineage or path a FlowFile took within the dataflow. To see a FlowFile's lineage, click on the "Show Lineage" icon ( ) in the far-right column of the Data Provenance table.

This opens a graph displaying the FlowFile () and the various processing events that have occurred. The selected event will be highlighted in red. It is possible to right-click or double-click on any event to see that event's details. To see how the lineage evolved over time, click the slider at the bottom-left of the window and move it to the left to see the state of the lineage at earlier stages in the dataflow.

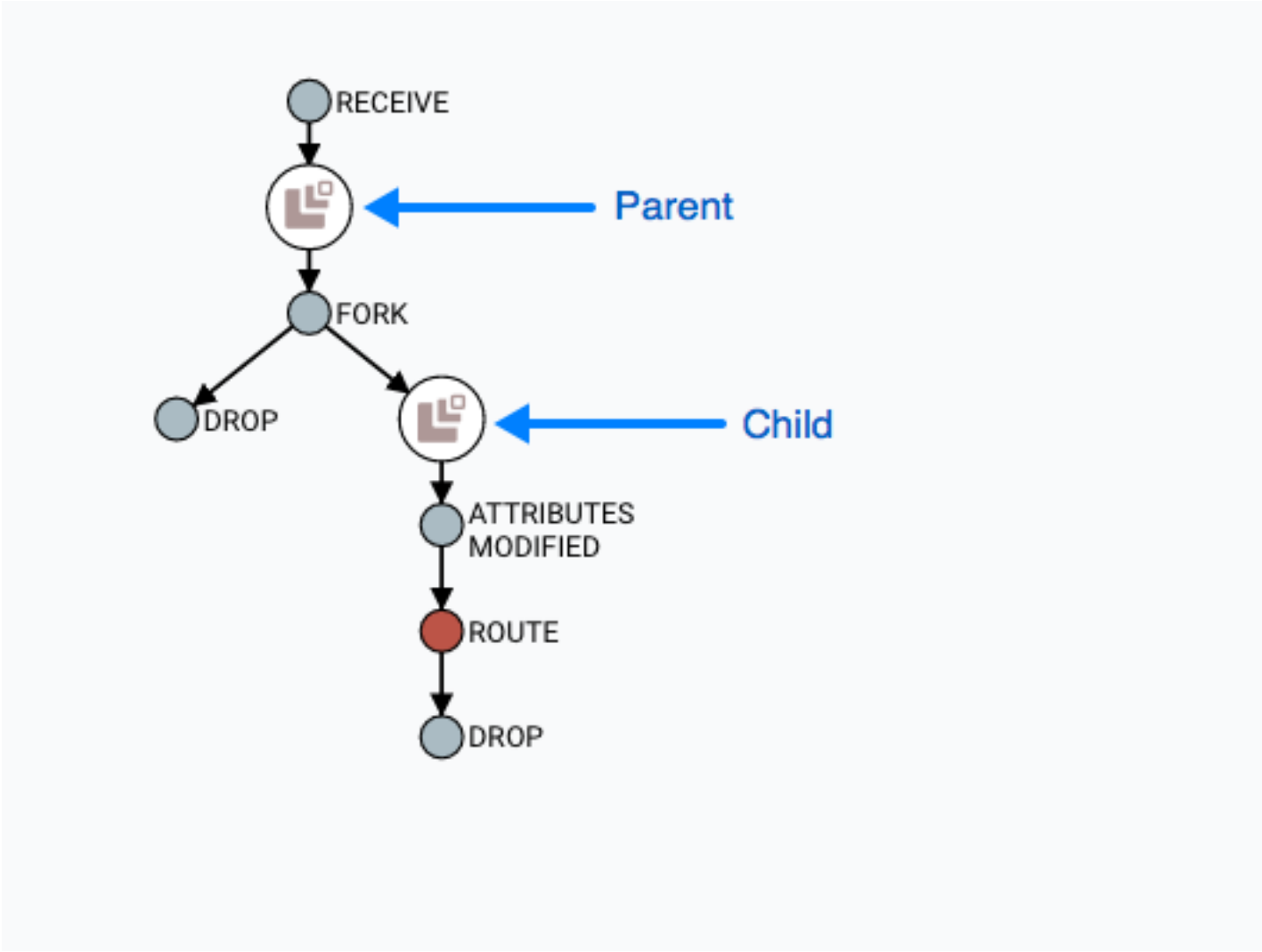


Find Parents

Sometimes, a user may need to track down the original FlowFile that another FlowFile was spawned from. For example, when a FORK or CLONE event occurs, NiFi keeps track of the parent FlowFile that produced other FlowFiles, and it is possible to find that parent FlowFile in the Lineage. Right-click on the event in the lineage graph and select "Find parents" from the context menu.

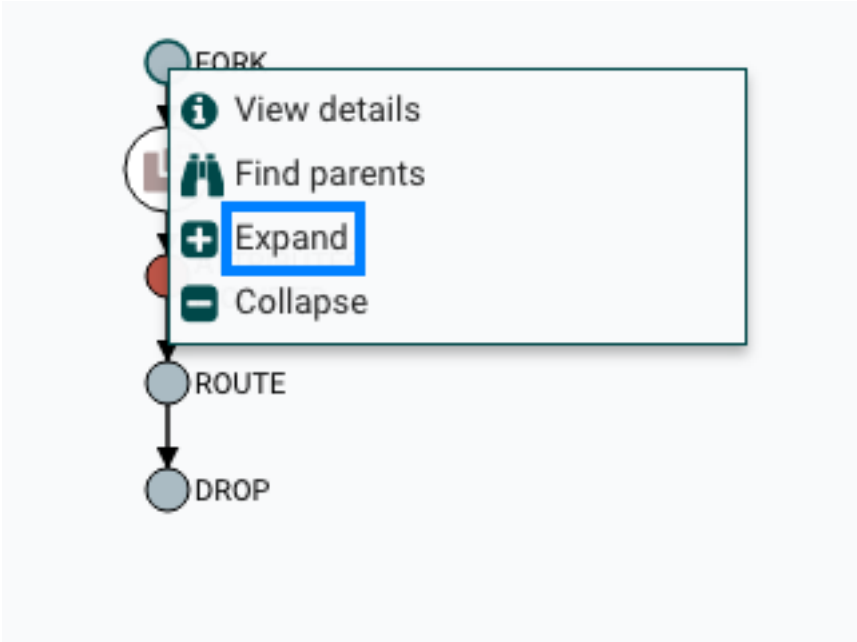


Once "Find parents" is selected, the graph is re-drawn to show the parent FlowFile and its lineage as well as the child and its lineage.

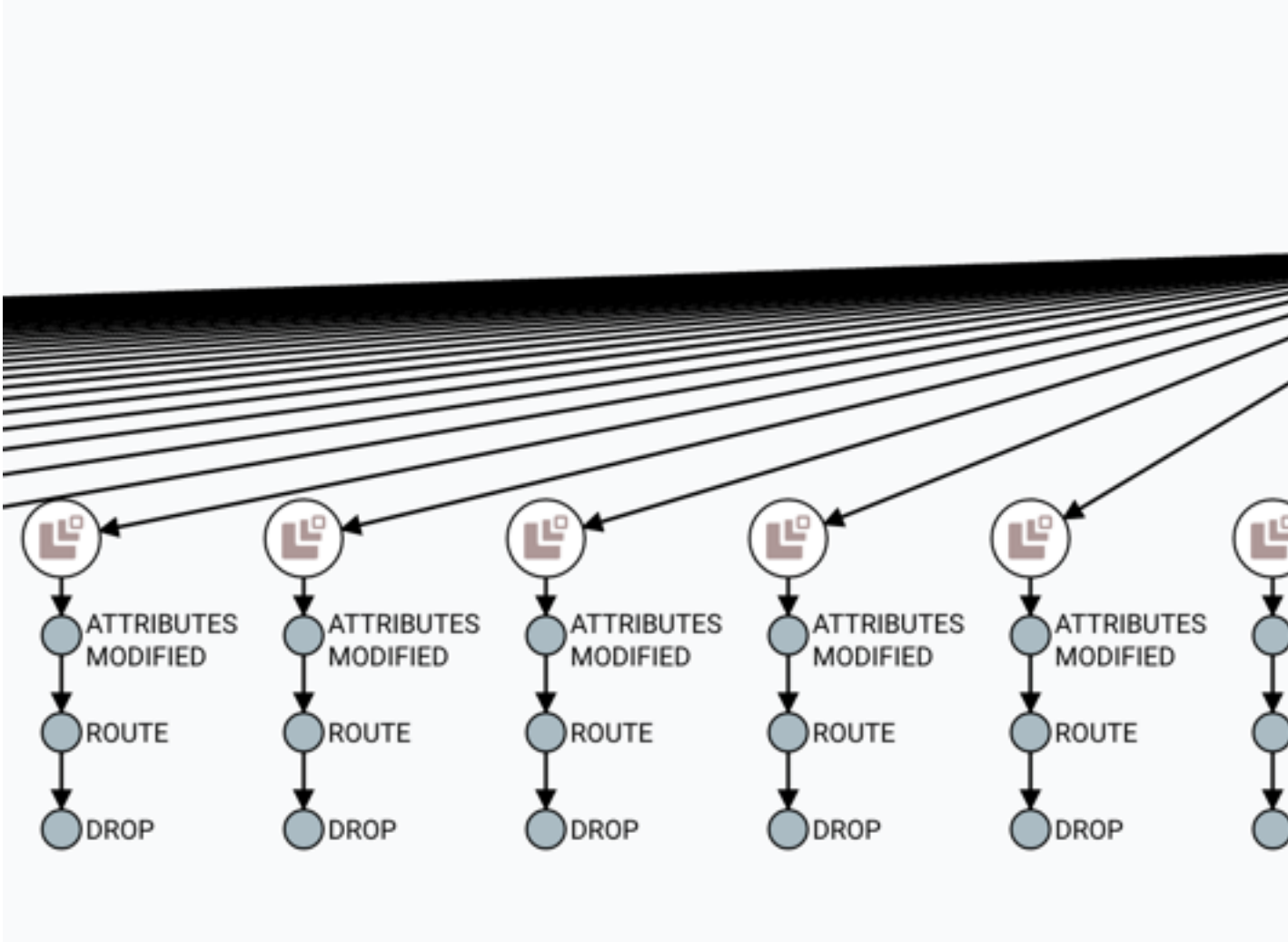


Expanding an Event

In the same way that it is useful to find a parent FlowFile, the user may also want to determine what children were spawned from a given FlowFile. To do this, right-click on the event in the lineage graph and select "Expand" from the context menu.



Once "Expand" is selected, the graph is re-drawn to show the children and their lineage.



Write Ahead Provenance Repository

By default, the Provenance Repository is implemented in a Persistent Provenance configuration. In Apache NiFi 1.2.0, the Write Ahead configuration was introduced to provide the same capabilities as Persistent Provenance, but with far better performance. Migrating to the Write Ahead configuration is easy to accomplish. Simply change the setting for the nifi.provenance.repository.implementation system property in the nifi.properties file from the default value of org.apache.nifi.provenance.PersistentProvenanceRepository to org.apache.nifi.provenance.WriteAheadProvenanceRepository and restart NiFi.

However, to increase the chances of a successful migration consider the following factors and recommended actions.

Backwards Compatibility

The WriteAheadProvenanceRepository can use the Provenance data stored by the PersistentProvenanceRepository. However, the PersistentProvenanceRepository may not be able to read the data written by the WriteAheadProvenanceRepository. Therefore, once the Provenance Repository is changed to use the WriteAheadProvenanceRepository, it cannot be changed back to the PersistentProvenanceRepository without first deleting the data in the Provenance Repository. It is therefore recommended that before changing the implementation to Write Ahead, ensure your version of NiFi is stable, in case an issue arises that requires the need to roll back to a previous version of NiFi that did not support the WriteAheadProvenanceRepository.

Older Existing NiFi Version

If you are upgrading from an older version of NiFi to 1.2.0 or later, it is recommended that you do not change the provenance configuration to Write Ahead until you confirm your flows and environment are stable in 1.2.0 first. This reduces the number of variables in your upgrade and can simplify the debugging process if any issues arise.

Bootstrap.conf

While better performance is achieved with the G1 garbage collector, Java 8 bugs may surface more frequently in the Write Ahead configuration. It is recommended that the following line is commented out in the bootstrap.conf file in the conf directory:

```
java.arg.13=-XX:+UseG1GC
```

System Properties

Many of the same system properties are supported by both the Persistent and Write Ahead configurations, however the default values have been chosen for a Persistent Provenance configuration. The following exceptions and recommendations should be noted when changing to a Write Ahead configuration:

- nifi.provenance.repository.journal.count is not relevant to a Write Ahead configuration
- nifi.provenance.repository.concurrent.merge.threads and nifi.provenance.repository.warm.cache.frequency are new properties. The default values of 2 for threads and blank for frequency (i.e. disabled) should remain for most installations.
- Change the settings for nifi.provenance.repository.max.storage.time (default value of 24 hours) and nifi.provenance.repository.max.storage.size (default value of 1 GB) to values more suitable for your production environment
- Change nifi.provenance.repository.index.shard.size from the default value of 500 MB to 4 GB
- Change nifi.provenance.repository.index.threads from the default value of 2 to either 4 or 8 as the Write Ahead repository enables this to scale better
- If processing a high volume of events, change nifi.provenance.repository.rollover.time from a default of 30 secs to 1 min and nifi.provenance.repository.rollover.size from the default of 100 MB to 1 GB

Once these property changes have been made, restart NiFi.

Encrypted Provenance Considerations

The above migration recommendations for WriteAheadProvenanceRepository also apply to the encrypted version of the configuration, EncryptedWriteAheadProvenanceRepository.

The next section has more information about implementing an Encrypted Provenance Repository.

Encrypted Provenance Repository

While OS-level access control can offer some security over the provenance data written to the disk in a repository, there are scenarios where the data may be sensitive, compliance and regulatory requirements exist, or NiFi is running on hardware not under the direct control of the organization (cloud, etc.). In this case, the provenance repository allows for all data to be encrypted before being persisted to the disk.

The current implementation of the encrypted provenance repository intercepts the record writer and reader of WriteAheadProvenanceRepository, which offers significant performance improvements over the legacy PersistentProvenanceRepository and uses the AES/GCM algorithm, which is fairly performant on commodity hardware. In most scenarios, the added cost will not be significant (unnoticable on a flow with hundreds of provenance events per second, moderately noticable on a flow with thousands - tens of thousands of events per second). However, administrators should perform their own risk assessment and performance analysis and decide how to move forward. Switching back and forth between encrypted/unencrypted implementations is not recommended at this time.

What is it?

The EncryptedWriteAheadProvenanceRepository is a new implementation of the provenance repository which encrypts all event record information before it is written to the repository. This allows for storage on systems where OS-level access controls are not sufficient to protect the data while still allowing querying and access to the data through the NiFi UI/API.

How does it work?

The WriteAheadProvenanceRepository was introduced in NiFi 1.2.0 and provided a refactored and much faster provenance repository implementation than the previous PersistentProvenanceRepository. The encrypted version wraps that implementation with a record writer and reader which encrypt and decrypt the serialized bytes respectively.

The fully qualified class org.apache.nifi.provenance.EncryptedWriteAheadProvenanceRepository is specified as the provenance repository implementation in nifi.properties as the value of nifi.provenance.repository.implementation. In addition, encrypted write ahead provenance repository properties must be populated to allow successful initialization.

StaticKeyProvider

The StaticKeyProvider implementation defines keys directly in nifi.properties. Individual keys are provided in hexadecimal encoding. The keys can also be encrypted like any other sensitive property in nifi.properties using the encrypted-config tool in the NiFi Toolkit.

The following configuration section would result in a key provider with two available keys, "Key1" (active) and "AnotherKey".

```
nifi.provenance.repository.encryption.key.provider.implementation=org.apache.nifi.securi
nifi.provenance.repository.encryption.key.id=Key1
nifi.provenance.repository.encryption.key=0123456789ABCDEFFEDCBA98765432100123456789ABCD
nifi.provenance.repository.encryption.key.id.AnotherKey=01010101010101010101010101010101
```

FileBasedKeyProvider

The FileBasedKeyProvider implementation reads from an encrypted definition file of the format:

```
key1=NGCpDpxBZNN0DBodz0p1SDbTjC2FG5kp1pCmdUKJlxxtcMSo6GC4fMlTyy1mPeKOxzLut3DRX
+51j6PCO5SznA==
key2=GYxPbMMDbnraXs09eGJudAM5jTvVYp05XtImkAg4JY4rIbmHOiVUUI6OeOf7ZW
+hH42jtPgNW9pSkkQ9HWY/vQ==
key3=SFe11xuz7J89Y/IQ7YbJPOL0/YKZRFL/
VUxJgEHxxlXpd/8ELA7wwN59K1KTr3BURCcFP5YGmwrSKfr4OE4Vlg==
key4=kZprfcTSTH69UuOU3jMkZfrtiVR/eqWmmbdku3bQcUJ/
+UToecNB5lzOVEMBChyEXppyXXC35Wa6GEXFK6PMKw==
key5=c6FzfnKm7UR7xqI2NFpZ+fEKBfSU7+1NvRw
+XWQ9U39MONWqk5gvoyOCdFR1kUgeg46jrN5dGXk13sRqE0GETQ==
```

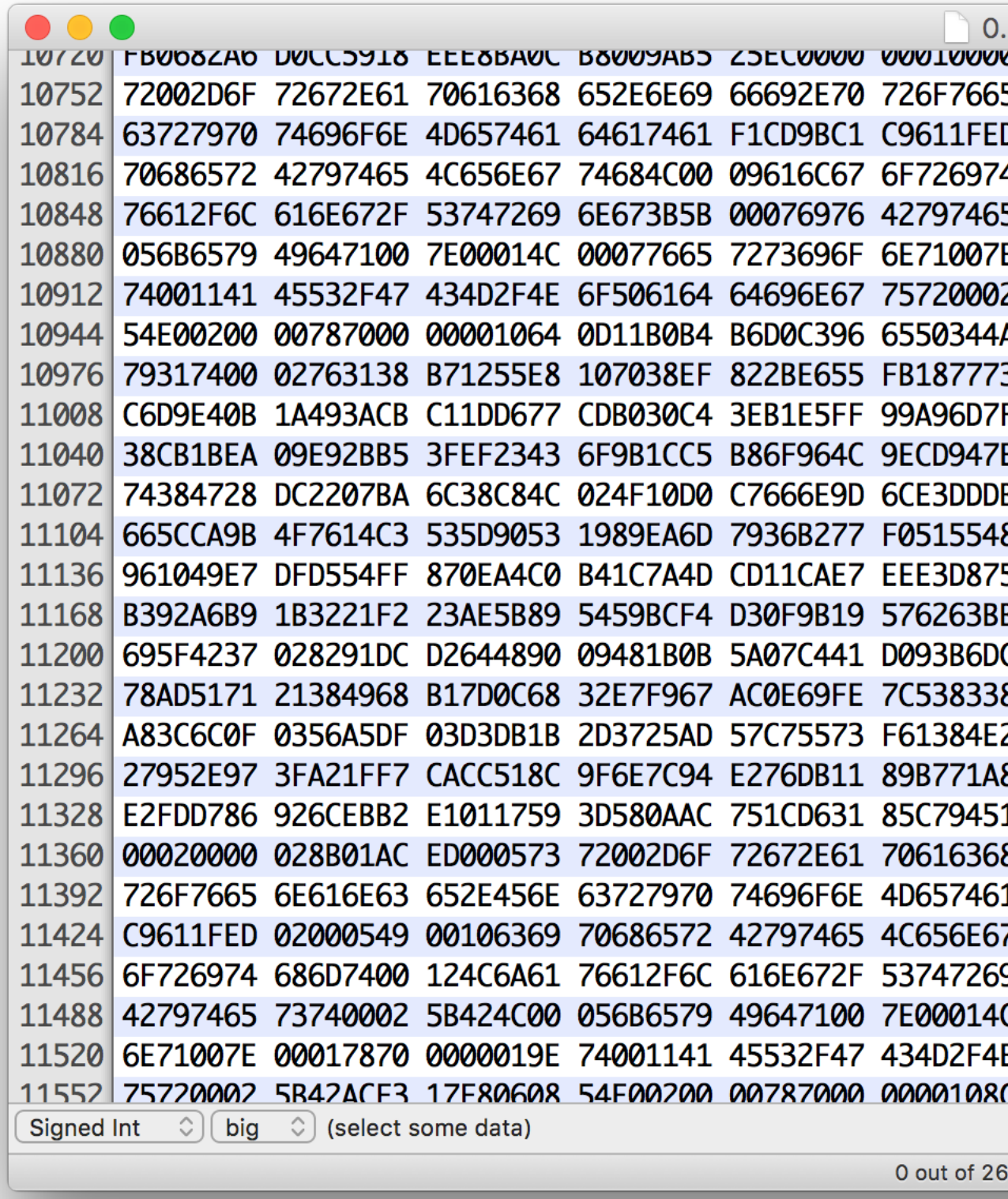
Each line defines a key ID and then the Base64-encoded cipher text of a 16 byte IV and wrapped AES-128, AES-192, or AES-256 key depending on the JCE policies available. The individual keys are wrapped by AES/GCM encryption using the master key defined by nifi.bootstrap.sensitive.key in conf/bootstrap.conf.

Key Rotation

Simply update nifi.properties to reference a new key ID in nifi.provenance.repository.encryption.key.id. Previously-encrypted events can still be decrypted as long as that key is still available in the key definition file or nifi.provenance.repository.encryption.key.id.<OldKeyID> as the key ID is serialized alongside the encrypted record.

Writing and Reading Event Records

Once the repository is initialized, all provenance event record write operations are serialized according to the configured schema writer (EventIdFirstSchemaRecordWriter by default for WriteAheadProvenanceRepository) to a byte[]. Those bytes are then encrypted using an implementation of ProvenanceEventEncryptor (the only current implementation is AES/GCM/NoPadding) and the encryption metadata (keyId, algorithm, version, IV) is serialized and prepended. The complete byte[] is then written to the repository on disk as normal.


10720 FB0682A6 D0CC5918 EEE8BA0C B8009AB5 25EC0000 00010000
10752 72002D6F 72672E61 70616368 652E6E69 66692E70 726F7665
10784 63727970 74696F6E 4D657461 64617461 F1CD9BC1 C9611FED
10816 70686572 42797465 4C656E67 74684C00 09616C67 6F726974
10848 76612F6C 616E672F 53747269 6E673B5B 00076976 42797465
10880 056B6579 49647100 7E00014C 00077665 7273696F 6E71007E
10912 74001141 45532F47 434D2F4E 6F506164 64696E67 75720002
10944 54E00200 00787000 00001064 0D11B0B4 B6D0C396 6550344A
10976 79317400 02763138 B71255E8 107038EF 822BE655 FB187773
11008 C6D9E40B 1A493ACB C11DD677 CDB030C4 3EB1E5FF 99A96D7F
11040 38CB1BEA 09E92BB5 3FEF2343 6F9B1CC5 B86F964C 9ECD947E
11072 74384728 DC2207BA 6C38C84C 024F10D0 C7666E9D 6CE3DDDE
11104 665CCA9B 4F7614C3 535D9053 1989EA6D 7936B277 F0515548
11136 961049E7 DFD554FF 870EA4C0 B41C7A4D CD11CAE7 EEE3D875
11168 B392A6B9 1B3221F2 23AE5B89 5459BCF4 D30F9B19 576263BE
11200 695F4237 028291DC D2644890 09481B0B 5A07C441 D093B6DC
11232 78AD5171 21384968 B17D0C68 32E7F967 AC0E69FE 7C538338
11264 A83C6C0F 0356A5DF 03D3DB1B 2D3725AD 57C75573 F61384E2
11296 27952E97 3FA21FF7 CACC518C 9F6E7C94 E276DB11 89B771A8
11328 E2FDD786 926CEBB2 E1011759 3D580AAC 751CD631 85C79451
11360 00020000 028B01AC ED000573 72002D6F 72672E61 70616368
11392 726F7665 6E616E63 652E456E 63727970 74696F6E 4D657461
11424 C9611FED 02000549 00106369 70686572 42797465 4C656E67
11456 6F726974 686D7400 124C6A61 76612F6C 616E672F 53747269
11488 42797465 73740002 5B424C00 056B6579 49647100 7E00014C
11520 6E71007E 00017870 0000019E 74001141 45532F47 434D2F4E
11552 75720002 5B42ACF3 17F80608 54E00200 00787000 00001080
Signed Int
big
(select some data)
0 out of 26

On record read, the process is reversed. The encryption metadata is parsed and used to decrypt the serialized bytes, which are then deserialized into a ProvenanceEventRecord object. The delegation to the normal schema record writer/reader allows for "random-access" (i.e. immediate seek without decryption of unnecessary records).

Within the NiFi UI/API, there is no detectable difference between an encrypted and unencrypted provenance repository. The Provenance Query operations work as expected with no change to the process.

Potential Issues

When switching between implementation "families" (i.e. VolatileProvenanceRepository or PersistentProvenanceRepository to EncryptedWriteAheadProvenanceRepository), the existing repository must be cleared from the file system before starting NiFi. A terminal command like localhost:$NIFI_HOME $ rm -rf provenance_repository/ is sufficient.

- Switching between unencrypted and encrypted repositories
  - If a user has an existing repository (WriteAheadProvenanceRepository only - not PersistentProvenanceRepository) that is not encrypted and switches their configuration to use an encrypted repository, the application writes an error to the log but starts up. However, previous events are not accessible through the provenance query interface and new events will overwrite the existing events. The same behavior occurs if a user switches from an encrypted repository to an unencrypted repository. Automatic roll-over is a future effort (https://issues.apache.org/jira/browse/NIFI-3722) but NiFi is not intended for long-term storage of provenance events so the impact should be minimal. There are two scenarios for roll-over:
    - Encrypted # unencrypted - if the previous repository implementation was encrypted, these events should be handled seamlessly as long as the key provider available still has the keys used to encrypt the events (see Key Rotation)
    - Unencrypted # encrypted - if the previous repository implementation was unencrypted, these events should be handled seamlessly as the previously recorded events simply need to be read with a plaintext schema record reader and then written back with the encrypted record writer
  - There is also a future effort to provide a standalone tool in NiFi Toolkit to encrypt/decrypt an existing provenance repository to make the transition easier. The translation process could take a long time depending on the size of the existing repository, and being able to perform this task outside of application startup would be valuable (https://issues.apache.org/jira/browse/NIFI-3723).
- Multiple repositories - No additional effort or testing has been applied to multiple repositories at this time. It is possible/likely issues will occur with repositories on different physical devices. There is no option to provide a heterogenous environment (i.e. one encrypted, one plaintext repository).
- Corruption - when a disk is filled or corrupted, there have been reported issues with the repository becoming corrupted and recovery steps are necessary. This is likely to continue to be an issue with the encrypted repository, although still limited in scope to individual records (i.e. an entire repository file won't be irrecoverable due to the encryption).