Apache NiFi 3

Managing a Data Flow

Date of Publish: 2020-04-28



https://docs.cloudera.com/

Contents

Command and Control of the DataFlow	3
Starting a Component	
Stopping a Component	
Enabling/Disabling a Component	
Remote Process Group Transmission	
Individual Port Transmission	4

Encrypted Content Repository	6
What is it?	
How does it work?	7
StaticKeyProvider	7
FileBasedKeyProvider	7
Key Rotation	8
Writing and Reading Content Claims	8
Potential Issues	8

Encrypted FlowFile Repository	9
What is it?	
How does it work?	
StaticKeyProvider	
FileBasedKeyProvider	
Key Rotation	
Writing and Reading FlowFiles	
Potential Issues	
Experimental Warning	

Other	lanagement Features12

Command and Control of the DataFlow

When a component is added to the NiFi canvas, it is in the Stopped state. In order to cause the component to be triggered, the component must be started. Once started, the component can be stopped at any time. From a Stopped state, the component can be configured, started, or disabled.

Starting a Component

In order to start a component, the following conditions must be met:

- The component's configuration must be valid.
- All defined Relationships for the component must be connected to another component or auto-terminated.
- The component must be stopped.
- The component must be enabled.
- The component must have no active tasks. For more information about active tasks, see the "Anatomy of ..." sections under Monitoring of DataFlow.

Components can be started by selecting all of the components to start and then clicking the "Start" button (



) in the Operate Palette or by right-clicking a single component and choosing Start from the context menu.

If starting a Process Group, all components within that Process Group (including child Process Groups) will be started, with the exception of those components that are invalid or disabled.

Once started, the status indicator of a Processor will change to a Play symbol (

,			
,			

).

Stopping a Component

A component can be stopped any time that it is running. A component is stopped by right-clicking on the component and clicking Stop from the context menu, or by selecting the component and clicking the "Stop" button (



) in the Operate Palette.

If a Process Group is stopped, all of the components within the Process Group (including child Process Groups) will be stopped.

Once stopped, the status indicator of a component will change to the Stop symbol (

).

Stopping a component does not interrupt its currently running tasks. Rather, it stops scheduling new tasks to be performed. The number of active tasks is shown in the top-right corner of the Processor (See Anatomy of a Processor for more information).

Enabling/Disabling a Component

When a component is enabled, it is able to be started. Users may choose to disable components when they are part of a dataflow that is still being assembled, for example. Typically, if a component is not intended to be run, the component is disabled, rather than being left in the Stopped state. This helps to distinguish between components that are intentionally not running and those that may have been stopped temporarily (for instance, to change the component's configuration) and inadvertently were never restarted.

When it is desirable to re-enable a component, it can be enabled by selecting the component and clicking the "Enable" button



) in the Operate Palette. This is available only when the selected component or components are disabled. Alternatively, a component can be enabled by checking the checkbox next to the "Enabled" option in the Settings tab of the Processor configuration dialog or the configuration dialog for a Port.

Once enabled, the component's status indicator will change to either Invalid (



) or Stopped (

), depending on whether or not the component is valid.

A component is then disabled by selecting the component and clicking the "Disable" button (



) in the Operate Palette, or by clearing the checkbox next to the "Enabled" option in the Settings tab of the Processor configuration dialog or the configuration dialog for a Port.

Only Ports and Processors can be enabled and disabled.

Remote Process Group Transmission

Remote Process Groups provide a mechanism for sending data to or retrieving data from a remote instance of NiFi. When a Remote Process Group (RPG) is added to the canvas, it is added with the Transmission Disabled, as indicated by the icon (



) in the top-left corner. When Transmission is Disabled, it can be enabled by right-clicking on the RPG and clicking the "Enable transmission" menu item. This will cause all ports for which there is a Connection to begin transmitting data. This will cause the status indicator to then change to the Transmission Enabled icon (

).

If there are problems communicating with the Remote Process Group, a Warning indicator (

A

) may instead be present in the top-left corner. Hovering over this Warning indicator with the mouse will provide more information about the problem.

Individual Port Transmission

There are times when the DFM may want to either enable or disable transmission for only a specific port within the Remote Process Group. This can be accomplished by right-clicking on the Remote Process Group and choosing the "Manage remote ports" menu item. This provides a configuration dialog from which ports can be configured:

Rem	ote Process	Group Port	S				
Name NiFi Flo	w			URLs http://l	ocalhost:8080/r	iifi, http://localho	st:8081/nifi
Input p	oorts			Output	ports		
	Input1 No description spe	cified.			Output1 No description spec	cified.	
	Concurrent Tas 1	ks 😧	Compressed No	ø	Concurrent Tas 1	ks 😧	Compressed Yes
	Batch Settings				Batch Settings		
	Count 10	Size 10 KB	Duration 10 sec		Count No value set	Size No value set	Duration No value set
	Input2 No description spec	rified			Output2 No description spec	cified	
	Concurrent Tas		Compressed		Concurrent Tas		Compressed
	1		No		1		No
	Batch Settings	O Size	Duration		Batch Settings	O Size	Duration
	Count No value set	Size No value set	No value set		Count No value set	Size No value set	No value set
							CLOSE

The left-hand side lists all of the Input Ports that the remote instance of NiFi allows data to be sent to. The right-hand side lists all of the Output Ports from which this instance is able to pull data. If the remote instance is using secure communications (the URL of the NiFi instance begins with https://, rather than http://), any ports that the remote instance has not made available to this instance will not be shown.

Note: If a port that is expected to be shown is not shown in this dialog, ensure that the instance has proper permissions and that the Remote Process Group's flow is current. This can be checked by closing the Remote Process Group Ports dialog and looking at the bottom-left corner of the Remote Process Group. The date and time when the flow was last refreshed is displayed. If the flow appears to be outdated, it can be updated by right-clicking on the Remote Process Group and selecting "Refresh remote". (See Anatomy of a Remote Process Group for more information).

Each port is shown with its Name, its Description, configured number of Concurrent Tasks, and whether or not data sent to this port will be Compressed. Additionally, the port's configured Batch Settings (Count, Size and Duration) are displayed. To the left of this information is a toggle switch to turn the port on or off. Ports that have no connections attached to them are grayed out:

)

ame iFi Fl	low Cu	urrently Trai	nsmitting	URLs http:/	/localhost:8080/r	iifi, http://localho	ost:8081/nifi		
nput	ports			Outp	ıt ports				
	Input1 No description spe	cified.			Output1 No description spe	cified.			
	Concurrent Tas 1	sks 🕜	Compressed No	A	Concurrent Tas 1	ks 🛿	Compressed Yes		
	Batch Settings	0			Batch Settings 😧				
	Count	Size	Duration		Count	Size	Duration		
	10	10 KB	10 sec	Edit	No value set	No value set	No value set		
	Input2 No description spe	cified.			Output2 No description spe	cified.			
	Concurrent Tas	sks 🕜	Compressed		Concurrent Tas	ks 🕜	Compressed		
Т	1		No		1		No		
L	Batch Settings	0			Batch Settings	0			
L	Count	Size	Duration		Count	Size	Duration		
L	No value set	No value set	No value set		No value set	No value set	No value set		
ot (Connected								

The on/off toggle switch provides a mechanism to enable and disable transmission for each port in the Remote Process Group independently. Those ports that are connected but are not currently transmitting can be configured by clicking the pencil icon

below the on/off toggle switch. Clicking this icon will allow the DFM to change the number of Concurrent Tasks, whether or not compression should be used when transmitting data to or from this port, and Batch Settings.

For an Input Port, the batch settings control how NiFi sends data to the remote input port in a transaction. NiFi will transfer flow files, as they are queued in incoming relationships, until any of the limits (Count, Size, Duration) is met. If none of the settings are configured, a 500 milliseconds batch duration is used by default.

For an Output Port, the batch settings tells the remote NiFi how NiFi prefers to receive data from the remote output port in a transaction. The remote NiFi will use the specified settings (Count, Size, Duration) to control the transfer of flow files. If none of the settings are configured, a 5 seconds batch duration is used by default.

Encrypted Content Repository

Ø

While OS-level access control can offer some security over the flowfile content 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 content repository allows for all data to be encrypted before being persisted to the disk. For more information on the internal workings of the content repository, see NiFi In-Depth - Content Repository.

What is it?

The EncryptedFileSystemRepository is a new implementation of the content repository which encrypts all content data 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 FileSystemRepository was introduced in NiFi 0.2.1 and provided the only persistent content repository implementation. The encrypted version wraps that implementation with functionality to return to the Session (usually StandardProcessSession) a special OutputStream/InputStream which encrypt and decrypt the serialized bytes respectively. This allows all components to continue interacting with the content repository interface in the same way as before and continue operating on content data in a streaming manner, without requiring any changes to handle the data protection.

The fully qualified class org.apache.nifi.content.EncryptedFileSystemRepository is specified as the content repository implementation in nifi.properties as the value of nifi.content.repository.implementation. In addition, new 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 ./ encrypt-config.sh tool in the NiFi Toolkit.

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

FileBasedKeyProvider

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

```
key1=NGCpDpxBZNN0DBodz0p1SDbTjC2FG5kp1pCmdUKJ1xxtcMSo6GC4fMlTyy1mPeKOxzLut3DRX
+51j6PC05SznA==
key2=GYxPbMMDbnraXs09eGJudAM5jTvVYp05XtImkAg4JY4rIbmHOiVUUI6OeOf7ZW
+hH42jtPgNW9pSkkQ9HWY/vQ==
key3=SFe11xuz7J89Y/IQ7YbJPOL0/YKZRFL/
VUxJgEHxx1Xpd/8ELA7wwN59K1KTr3BURCcFP5YGmwrSKfr4OE4Vlg==
key4=kZprfcTSTH69UuOU3jMkZfrtiVR/eqWmmbdku3bQcUJ/
+UToecNB51zOVEMBChyEXppyXXC35Wa6GEXFK6PMKw==
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.

Data Protection vs. Key Protection

Even though the flowfile content is encrypted using AES/CTR to handle streaming data, if using the Config Encrypt Tool or FileBasedKeyProvider, those keys will be protected using AES/GCM to provide authenticated encryption over the key material.

Key Rotation

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

Writing and Reading Content Claims

Once the repository is initialized, all content claim write operations are serialized using RepositoryObjectStreamEncryptor (the only currently existing implementation is RepositoryObjectAESCTREncryptor) to an OutputStream. The actual implementation is EncryptedContentRepositoryOutputStream, which encrypts the data written by the component via StandardProcessSession inline and the encryption metadata (keyId, algorithm, version, IV) is serialized and prepended. The complete OutputStream is then written to the repository on disk as normal.

• • •						15	712586893	85-93		
35476	9B0B6FFB	02362410	383EC20D	A3F7D291	39162128	39505A8C	Ø3611B32	30134F27	BFCE3366	. o. 6\$ 8>9 !(9PZ. a 20 0'3
235512	45A676BE	88092BE9	97CFA022	258EA2E7	45306174	4D184807	8685CA61	34D1F55E	E0D784A7	E.v +"%E0atM Ha4^
235548	ØB4FA4AD	29F16A3E	4D869375	CF782A76	219C8591	E9891CD6	535DE430	6937D6CD	BC80510B	0).j>Mu.x*v!S].0i7Q
235584	B75B1E87	431CB0C3	6CD1525A	71BBFE1B	53175F46	47EC48DB	A6180448	EEC9736D	21FECF7E	.[.Cl.RZq S _FG.H Hsm!
235620	811957A5	C3AF5BA3	EACØA5A7	87AD86D0	2D0C5998	2883FED7	ØA375BB6	C462E571	016E4734	. W[Y.(7[b.q nG
235656	043BF9D0	533819D8	998275BE	F1B3F5A8	428E59A3	624154D3	F60FDE9C	FBC3AFA6	821C6BCF	;S8uB.Y.bAT k
235692	069C6B4E	725AA965	0D1793BC	937922F7	4B5359B8	F4AC05F0	B7634170	A514A6DB	27A5DBDD	.kNrZ@ey".KSYcAp'
235728	38318592	EDF212B5	001ADA13	E2BB2238	36277CB9	E64D1B3E	24FE8FA0	5C4E0E31	AF533CEA	81 N 1.S<
235764	4D9B8F96	79B4C7C3	79AD58A8	4400DA18	8DC1E015	184B7315	D3E14B28	7C358478	551143E8	Myy.X.D KsK(5.xU C
235800	F85B9A9A	1311809B	43BECB3A	C2051E63	E2893607	83891419	FD155819	132CEE4E	350EC48F	.[C:. c6 X ,.N5 .
235836	8CD3ED6A	65C6C5B3	98DA6292	A2E1E3B8	9FCB2B7A	1F5F0AF7	D94870B7	7FB28AC1	5E5A74FD	jeb+zHp^Zt
235872	A47102CE	5766B6A2	B6CD26B6	4A0D3D44	333A1CC2	95A6CEA7	1672B302	4941A0A9	B7967D9C	.q .Wf&.J =D3: r. IA.�}
235908	BF4BF557	5382C6BF	AC81D069	B9639D7A	F8B1B398	F0D02842	0A909F49	D7AAF742	56A5BA75	.K.WSi.c.z(BIBV
235944	BD06EAEC	C1B4BDF8	CDA290D5	E43C67E7	8ABC59BB	4696EAF4	73A9D73E	6D9E2DCD	46C97AC6	
235980	8480E439	2C8D5F1B	0C08592A	CA44C50F	EDBF9BB2	AF946BE5	6926A7C6	F718AE89	BE9A7107	9, Y*.Dk.i&q
236016	ABØF92B6	CAFF7AAD	C6671E4D	9F21CF49	9ADB64FB	D20036DC	42A351D7	F43EB274	BBE166EF	zg M.!.Id 6.B.Q>.tf
236052	6D7DD879	5B1F0BE6	DAC65404	FA54CD18	6401898E	7A4839D9	1104A7AD	CF0000C3	306579A7	m}.y[T .T. dzH90ey
236088	13EE834E	9AE9AAEF	3892B434	7F93D1BC	51B751B2	BC3AB1BD	FF5C0066	2E765297	37A0D5B2	N84Q.Q:\ f.vR.7
236124	674EC77B	BB780B2D	7F95AD2E	CEAB9375	289B7405	6B783E9D	D5C02187	7D4A562C	646E8679	gN.{.xu(.t kx>!.}JV,dn.
236160	54A495D6	D08533FD	309049EF	41ADEB3A	8CD8A1BA	3E1A35B8	0D20D823	306F3C62	58FA382D	T3.0.I.A:> 5#0o <bx.8< td=""></bx.8<>
236196	8A865F91	AC768E30	717FC82C	A336BC9A	ØAD29620	99B82D5E	140ABA2E	268B9916	D3988EØA	v.0q .,.6^&
236232	199534D3	A26DE78A	70A3C766	072E7F72	2AB31CCA	380000AC	ED000573	72003F6F	72672E61	.4mpf . r*8 sr ?org.
236268	70616368	652E6E69	66692E73	65637572	6974792E	7265706F	7369746F	72792E53	74726561	pache.nifi.security.repository.Stre
236304	6D696E67	456E6372	79707469	6F6E4D65	74616461	74611846	6982894D	442C0200	00787200	mingEncryptionMetadata FiMD, xr
236340	466F7267	2E617061	6368652E	6E696669	2E736563	75726974	792E7265	706F7369	746F7279	Forg.apache.nifi.security.repositor
236376	2E526570	6F736974	6F72794F	626A6563	74456E63	72797074	696F6E4D	65746164	6174619F	.RepositoryObjectEncryptionMetadata
236412	4328584E	DFDF0802	00054900	10636970	68657242	7974654C	656E6774	684C0009	616C676F	C(XN I cipherByteLengthL alg
236448	72697468	6D740012	4C6A6176	612F6C61	6E672F53	7472696E	673B5B00	07697642	79746573	rithmt Ljava/lang/String;[ivByte
236484	7400025B	424C0005	6B657949	6471007E	00024C00	07766572	73696F6E	71007E00	027870FF	t [BL keyIdq ~ L versionq ~ xp
236520	FFFFFF74	00114145	532F4354	522F4E6F	50616464	696E6775	7200025B	42ACF317	F8060854	t AES/CTR/NoPaddingur [B
236556	E0020000	78700000	00104207	611FD7A6	81A8E8EA	FFB8FFA4	9C117400	024B3174	00027631	. xp B a t K1t v
236592										
UTF-8	٥	(select s	ome data)							E
						0x18E8	F out of 0x390	30 bytes		

On content claim read, the process is reversed. The encryption metadata (RepositoryObjectEncryptionMetadata) is parsed and used to decrypt the serialized bytes, which are then deserialized into a CipherInputStream object. The delegation to the normal repository file system interaction allows for "random-access" (i.e. immediate seek without decryption of unnecessary content claims).

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

Potential Issues

• Switching between unencrypted and encrypted repositories

- If a user has an existing repository (FileSystemRepository) 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 content claims are not accessible through the provenance query interface and new content claims will overwrite the existing claims. 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-6783) but NiFi is not intended for long-term storage of content claims so the impact should be minimal. There are two scenarios for roll-over:
 - Encrypted # unencrypted if the previous repository implementation was encrypted, these claims should be handled seamlessly as long as the key provider available still has the keys used to encrypt the claims (see *Key Rotation*.)
 - Unencrypted # encrypted if the previous repository implementation was unencrypted, these claims should be handled seamlessly as the previously written claims simply need to be read with a plaintext InputStream and then be written back with the EncryptedContentRepositoryOutputStream
- There is also a future effort to provide a standalone tool in NiFi Toolkit to encrypt/decrypt an existing content 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-6783).
- 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 claims (i.e. an entire repository file won't be irrecoverable due to the encryption). Some testing has been performed on scenarios where disk space is exhausted. While the flow can no longer write additional content claims to the repository in that case, the NiFi application continues to function properly, and successfully written content claims are still available via the Provenance Query operations. Stopping NiFi and removing the content repository (or moving it to a larger disk) resolves the issue.

Encrypted FlowFile Repository

While OS-level access control can offer some security over the flowfile attribute and content claim 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 flowfile repository allows for all data to be encrypted before being persisted to the disk. For more information on the internal workings of the flowfile repository, see NiFi In-Depth - FlowFile Repository.

What is it?

The EncryptedSequentialAccessWriteAheadLog is a new implementation of the flowfile write-ahead log which encrypts all flowfile attribute data 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 SequentialAccessWriteAheadLog was introduced in NiFi 1.6.0 and provided a faster flowfile repository implementation. The encrypted version wraps that implementation with functionality to transparently encrypt and decrypt the serialized RepositoryRecord objects during file system interaction. During all writes to disk (swapping, snapshotting, journaling, and checkpointing), the flowfile containers are serialized to bytes based on a schema, and this serialized form is encrypted before writing. This allows the snapshot handler to continue interacting with the

flowfile repository interface in the same way as before and continue operating on flowfile data in a random access manner, without requiring any changes to handle the data protection.

The fully qualified class org.apache.nifi.wali.EncryptedSequentialAccessWriteAheadLog is specified as the flowfile repository write-ahead log implementation in nifi.properties as the value of nifi.flowfile.repository.wal.implementation. In addition, new 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 ./ encrypt-config.sh tool in the NiFi Toolkit.

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

FileBasedKeyProvider

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

```
key1=NGCpDpxBZNN0DBodz0p1SDbTjC2FG5kp1pCmdUKJ1xxtcMSo6GC4fMlTyy1mPeKOxzLut3DRX
+51j6PCO5SznA==
key2=GYxPbMMDbnraXs09eGJudAM5jTvVYp05XtImkAg4JY4rIbmHOiVUUI6OeOf7ZW
+hH42jtPgNW9pSkkQ9HWY/vQ==
key3=SFe11xuz7J89Y/IQ7YbJPOL0/YKZRFL/
VUxJgEHxx1Xpd/8ELA7wwN59K1KTr3BURCcFP5YGmwrSKfr4OE4V1g==
key4=kZprfcTSTH69UuOU3jMkZfrtiVR/eqWmmbdku3bQcUJ/
+UToecNB51zOVEMBChyEXppyXXC35Wa6GEXFK6PMKw==
key5=c6FzfnKm7UR7xq12NFpZ+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.flowfile.repository.encryption.key.id. Previouslyencrypted flowfile records can still be decrypted as long as that key is still available in the key definition file or nifi.flowfile.repository.encryption.key.id.<OldKeyID> as the key ID is serialized alongside the encrypted record.

Writing and Reading FlowFiles

Once the repository is initialized, all flowfile record write operations are serialized using RepositoryObjectBlockEncryptor (the only currently existing implementation is RepositoryObjectAESGCMEncryptor) to the provided DataOutputStream. The original stream is swapped with a temporary wrapped stream, which encrypts the data written by the wrapped serializer/deserializer via EncryptedSchemaRepositoryRecordSerde inline and the encryption metadata (keyId, algorithm, version, IV, cipherByteLength) is serialized and prepended. The complete length and encrypted bytes are then written to the original DataOutputStream on disk as normal.

• •	•							0	.journal			
1748	ØA695616	EDACCB9A	BB1242E5	258EBAA0	207B1D86	24620000	0224ACED	00057372	00416F72	672F6170	61636865	iV B.% { .*b \$ sr Aorg.apache
1792						69746F72						.nifi.security.repository.block.BlockEncrypt
1836	696F6E4D											ionMetadata6I xr Forg.apache.nifi.se
1880	63757269											curity.repository.RepositoryObjectEncryption
1924	4D657461											Metadata.C(XN I cipherByteLengthL ald
1968	6F726974	686D7400	124C6A61	76612F6C	616E672F	53747269	6E673B5B	00076976	42797465	73740002	5B424C00	orithmt Ljava/lang/String;[ivBytest [BL
2012	056B6579	49647100	7E00024C	00077665	7273696F	6E71007E	00027870	000000D1	74001141	45532F47	434D2F4E	keyIdg ~ L versiong ~ xp .t AES/GCM/N
2056	6F506164	64696E67	75720002	5B42ACF3	17F80608	54E00200	00787000	00001062	8C410D08	C4027564	DAE6A666	oPaddingur [B T. xp b.A . udf
2100	992A7F74	00024B31	74000276	3141E990	4557ECC2	58154479	B1A0755B	D123610E	BD39023E	A5F212F7	DAB42ED8	.* t K1t v1AEWX Dyu[.#a .9 >
2144	6BA62EAB	7D1179F5	A39DEF5A	B9BCF174	4379AE7D	32E7F183	ØA846B56	46F74AF1	F9C441DA	592B9539	7A991CE7	k} yZtCy.}2kVF.JA.Y+.9z.
2188	CF84BF19	ØEAFA3C5	14B6BF83	CF067577	885C34C2	02090697	CD43EFCF	AB00F859	F4463464	8BF6E29A	92BD863C	uw.\4C Y.F4d
2232	9766E76C	A4A0D0A7	0D2808A1	B6A11DE9	D426D8C3	001112A3	8D8BF9B3	6AFAC064	FFDAD759	1A223954	B701719A	.f.l (&jdY "9T. q.
2276	1999C031	D13125B2	57A16737	24318CA6	A547FA98	12CB79E2	F603811A	826E0943	F15EAD57	938A70FE	1D05967C	1.1%.W.g7\$1GynC.^.Wp.
2320	A4DD0000	0224ACED	00057372	00416F72	672E6170	61636865	2E6E6966	692E7365	63757269	74792E72	65706F73	\$ sr Aorg.apache.nifi.security.repos
2364	69746F72	792E626C	6F636B2E	426C6F63	6B456E63	72797074	696F6E4D	65746164	61746136	C69C49D5	97A81F02	itory.block.BlockEncryptionMetadata6I
2408	00007872	00466F72	672E6170	61636865	2E6E6966	692E7365	63757269	74792E72	65706F73	69746F72	792E5265	xr Forg.apache.nifi.security.repository.Re
2452	706F7369	746F7279	4F626A65	6374456E	63727970	74696F6E	4D657461	64617461	9F432858	4EDFDF08	02000549	positoryObjectEncryptionMetadata.C(XN
2496	00106369	70686572	42797465	4C656E67	74684C00	Ø9616C67	6F726974	686D7400	124C6A61	76612F6C	616E672F	cipherByteLengthL algorithmt Ljava/lang/
2540	53747269	6E673B5B	00076976	42797465	73740002	5B424C00	Ø56B6579	49647100	7E00024C	00077665	7273696F	String;[ivBytest [BL keyIdq ~ L version
2584	6E71007E	00027870	000000D1	74001141	45532F47	434D2F4E	6F506164	64696E67	75720002	5B42ACF3	17F80608	nq ~ xp .t AES/GCM/NoPaddingur [B
2628	54E00200	00787000	000010C6	8ADF06C7	E564EF65	1E451B2E	AB38DE74	00024B31	74000276	3161CB12	5454606F	T. xpd.e E8.t K1t v1a. TT`c
2672	24A497A1	FB67CF78	36DA4CF7	9CC2E4EB	2A7438C2	50F415AE	42FB8649	F5066D26	616B226D	CB288C5E	31370E2B	\$g.x6.L*t8.PBI. m&ak"m.(.^17 +
2716	B120656D											. em8o&.7 ?.T(>. As
2760						B86DD755						601.b.]]y .b"".q.m.U.CI 4\pyk2((.R=
2804	049D7B3B							3F5E1914	DC591E24	FE75979C	E7F02705	.{;!Jp;@"[FeQ ?^ .Y \$.u'
2848	4B0EE409	9B926647	1D8723F0	3744A280	6824FE32	05AA2630	498A7F					KfG .#.7Dh\$.2 .&0I.
Signed	Int 🔍 le, h	nex 0xEDA0	;									=
							0x2 bytes s	elected at offs	et 0x1CBE out o	f 0x324B byte:	3	

On flowfile record read, the process is reversed. The encryption metadata (RepositoryObjectEncryptionMetadata) is parsed and used to decrypt the serialized bytes, which are then deserialized into a DataInputStream object.

During swaps and recoveries, the flowfile records are descrialized and reserialized, so if the active key has been changed, the flowfile records will be re-encrypted with the new active key.

Within the NiFi UI/API, there is no detectable difference between an encrypted and unencrypted flowfile repository. All framework interactions with flowfiles work as expected with no change to the process.

Potential Issues

- Switching between unencrypted and encrypted repositories
 - If a user has an existing write-ahead repository (WriteAheadFlowFileRepository) that is not encrypted (uses the SequentialAccessWriteAheadLog) and switches their configuration to use an encrypted repository, the application handles this and all flowfile records will be recovered on startup. Future writes (including reserialization of these same flowfiles) will be encrypted. If a user switches from an encrypted repository to an unencrypted repository, the flowfiles cannot be recovered, and it is recommended to delete the existing flowfile repository before switching in this direction. Automatic roll-over is a future effort (https://issues.apache.org/jira/browse/NIFI-6994) but NiFi is not intended for long-term storage of flowfile records so the impact should be minimal. There are two scenarios for roll-over:
 - Encrypted # unencrypted if the previous repository implementation was encrypted, these records should be handled seamlessly as long as the key provider available still has the keys used to encrypt the claims (see *Key Rotation*.)
 - Unencrypted # encrypted currently handled seamlesssly for SequentialAccessWriteAheadLog but there are other initial implementations which could be handled
 - There is also a future effort to provide a standalone tool in NiFi Toolkit to encrypt/decrypt an existing flowfile 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-6994).
- Multiple repositories No additional effort or testing has been applied to multiple repositories at this time. Current implementations of the flowfile repository allow only for one repository, though it can reside across multiple volumes and partitions. 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 partition/directory).

• 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). It is important for the continued operation of NiFi to ensure that the disk storing the flowfile repository does not run out of available space.

Experimental Warning

While all Apache licensed code is provided "on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied" (see https://www.apache.org/licenses/LICENSE-2.0), some features of Apache NiFi may be marked experimental. Experimental features may:

- · have undergone less extensive testing than is normal for standard NiFi features
- interact with unstable external dependencies
- be subject to change (any exposed APIs should not be considered covered under the minor release backward compatibility guidelines of https://semver.org)
- potentially cause data loss
- not be directly supported by the community in the event issues arise

Every attempt is made to provide more detailed and specific information around the nature of the experimental warning on a per-feature basis. Questions around specific experimental features should be directed to the dev@nifi.apache.org.

Other Management Features

In addition to the Summary Page, Data Provenance Page, Template Management Page, and Bulletin Board Page, there are other tools in the Global Menu (see NiFi User Interface) that are useful to the DFM. Select Flow Configuration History to view all the changes that have been made to the dataflow. The history can aid in troubleshooting, such as if a recent change to the dataflow has caused a problem and needs to be fixed. The DFM can see what changes have been made and adjust the flow as needed to fix the problem. While NiFi does not have an "undo" feature, the DFM can make new changes to the dataflow that will fix the problem.

Two other tools in the Global Menu are Controller Settings and Users. The Controller Settings page provides the ability to change the name of the NiFi instance, add comments describing the NiFi instance, and set the maximum number of threads that are available to the application. It also provides tabs where DFMs may add and configure Controller Services and Reporting Tasks. The Users page is used to manage user access, which is described in the System Administrator's Guide.