Cloudera Flow Management Operator for Kubernetes 2.11.0

Configuring NiFi CR

Date published: 2024-06-11
Date modified: 2025-09-30

https://docs.cloudera.com/

Legal Notice

© Cloudera Inc. 2026. All rights reserved.

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

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

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

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

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

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

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

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

Contents

Configuring a NiFi instance

NiFi instances are configured through the CRs used to deploy them.

A custom resource (CR) is a YAML file that describes your desired NiFi deployments. This single file contains all configuration information required for the NiFi instance, no additional configuration is required after deployment.

This documentation provides sample configuration code snippets to help you create a CR.

Resource recommendations for NiFi deployments

Learn about the recommended resource sizes for NiFi deployments. Every NiFi deployment is unique on the basis of the purpose it serves, therefore the values here are just recommendations not requirements. Actual values may substantially differ depending on your use case.

Resource Type	Amount
CPU	2+ per Pod
Memory	4Gi+ per Pod
PVC/PV	5 per Pod
Secrets	4 + #Pods
ConfigMaps	13
Services	1 + #Connections
Pods	1 min, 3+ recommended
StatefulSet	1
Deployment	0
Ingress	1 + #IngressConnection

Example NiFi cluster size

The following list of resources represents the whole of a deployed NiFi cluster managed by the Cloudera Flow Management Operator for Kubernetes. The following example is run in kind with cert-manager and ingress-nginx deployed as dependencies.

```
apiVersion: cfm.cloudera.com/v1alpha1
kind: Nifi
metadata:
  name: mynifi
spec:
  replicas: 3
  nifiVersion: 1.0.0
  image:
    repository: container.repository.cloudera.com/cloudera/cfm-nifi-k8s
    tag: 2.8.0-b94-nifi_1.25.0.2.3.13.0-36
    pullSecret: cloudera-container-repository-credentials
    pullPolicy: IfNotPresent
  tiniImage:
    repository: container.repository.cloudera.com/cloudera/cfm-tini
    tag: 2.8.0-b94
    pullSecret: cloudera-container-repository-credentials
    pullPolicy: IfNotPresent
  persistence:
    size: 1Gi
    contentRepo:
      size: 1Gi
    flowfileRepo:
      size: 1Gi
    provenanceRepo:
      size: 2Gi
    data: {}
  security:
    initialAdminIdentity: anonymous
    nodeCertGen:
      issuerRef:
        name: self-signed-ca-issuer
        kind: ClusterIssuer
    singleUserAuth:
      enabled: true
      credentialsSecretName: creds
  hostName: nifi.io
  uiConnection:
    type: Ingress
    annotations:
      nginx.ingress.kubernetes.io/affinity: cookie
      nginx.ingress.kubernetes.io/affinity-mode: persistent
      nginx.ingress.kubernetes.io/backend-protocol: HTTPS
      nginx.ingress.kubernetes.io/ssl-passthrough: "true"
      nginx.ingress.kubernetes.io/ssl-redirect: "true"
  resources:
    nifi:
      requests:
        cpu: "1"
        memory: 2Gi
      limits:
        cpu: "4"
        memory: 4Gi
    log:
      requests:
        cpu: 50m
        memory: 128Mi

StatefulSet
$ kubectl get statefulset
NAME     READY   AGE
mynifi   3/3     24h
Pods
$ kubectl get pod
NAME       READY   STATUS    RESTARTS   AGE
mynifi-0   7/7     Running   0          23h
mynifi-1   7/7     Running   0          23h
mynifi-2   7/7     Running   0          23h
ConfigMaps
$ kubectl get configmap
NAME                              DATA   AGE
mynifi-authorizers                1      24h
mynifi-authorizers-empty          1      24h
mynifi-bootstrap                  1      24h
mynifi-certificate-setup-script   1      24h
mynifi-decommission-script        1      24h
mynifi-identities-config          1      24h
mynifi-logback                    1      24h
mynifi-login-identity-providers   1      24h
mynifi-nifi-cli-properties        1      24h
mynifi-nifi-properties            1      24h
mynifi-start-script               1      24h
mynifi-state-management           1      24h
mynifi-stop-script                1      24h
Secrets
```

```
$ kubectl get secret
NAME                         TYPE                DATA   AGE
creds                        Opaque              2      27h
mynifi-0-node-cert           kubernetes.io/tls   5      24h
mynifi-1-node-cert           kubernetes.io/tls   5      23h
mynifi-2-node-cert           kubernetes.io/tls   5      23h
mynifi-keystorepassword      Opaque              1      24h
mynifi-proxy-cert            kubernetes.io/tls   3      27h
mynifi-sensitive-props-key   Opaque              1      24h
Certificates
$ kubectl get certificates
NAME                 READY   SECRET               AGE
mynifi-0-node-cert   True    mynifi-0-node-cert   24h
mynifi-1-node-cert   True    mynifi-1-node-cert   23h
mynifi-2-node-cert   True    mynifi-2-node-cert   23h
mynifi-proxy-cert    True    mynifi-proxy-cert    24h
Services
$ kubectl get service
NAME         TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)
AGE
mynifi       ClusterIP   None           <none>        6007/TCP,5000/TCP   24
h
mynifi-web   ClusterIP   10.96.28.159   <none>        8443/TCP
24h
Ingresses
$ kubectl get ingresses
NAME         CLASS    HOSTS     ADDRESS     PORTS   AGE
mynifi-web   <none>   nifi.io   localhost   80      24h

PersistentVolumeClaims
$ kubectl get persistentvolumeclaim
NAME                             STATUS   VOLUME
         CAPACITY   ACCESS MODES   STORAGECLASS   AGE
content-repository-mynifi-0      Bound    pvc-d5b00d05-d8ee-4b5c-abe4-2cae61
61fa4b   1Gi        RWO            standard       24h
content-repository-mynifi-1      Bound    pvc-3a510ebf-2f63-409b-992b-5a08
480d4b31   1Gi        RWO            standard       23h
content-repository-mynifi-2      Bound    pvc-f1bafe8e-b8b2-485c-a0c8-ddb583
ac994b   1Gi        RWO            standard       23h
data-mynifi-0                    Bound    pvc-67072ae8-b1ae-445c-b81a-0771
9417a441   1Gi        RWO            standard       24h
data-mynifi-1                    Bound    pvc-4d20e11b-0d93-4b1b-95ff-a19189
506686   1Gi        RWO            standard       23h
data-mynifi-2                    Bound    pvc-71b92ed3-3a70-4c4d-a848-44f8
79896f59   1Gi        RWO            standard       23h
flowfile-repository-mynifi-0     Bound    pvc-c9c0ea11-0f59-4eeb-b9ac-a795b5
62c059   1Gi        RWO            standard       24h
flowfile-repository-mynifi-1     Bound    pvc-58200919-b8b2-43be-8d2f-16b1
f7d39a75   1Gi        RWO            standard       23h
flowfile-repository-mynifi-2     Bound    pvc-869c0159-51c7-4857-aa52-8c775c
709692   1Gi        RWO            standard       23h
provenance-repository-mynifi-0   Bound    pvc-97dc6f41-b8ea-4682-9f60-74c8
756fb344   2Gi        RWO            standard       24h
provenance-repository-mynifi-1   Bound    pvc-bd529e22-7024-4bef-a951-0a3fb7
53277b   2Gi        RWO            standard       23h
provenance-repository-mynifi-2   Bound    pvc-cbcc2b50-3404-4fea-8c3c-6a2f
e2bfdb13   2Gi        RWO            standard       23h
state-mynifi-0                   Bound    pvc-d0e72637-2b7d-40b7-a147-724094
2599a4   1Gi        RWO            standard       24h
state-mynifi-1                   Bound    pvc-c17f3fe2-b93d-4774-b49d-5564
e794c671   1Gi        RWO            standard       23h
state-mynifi-2                   Bound    pvc-19a010f0-0397-496b-a77d-89944b
94a9b0   1Gi        RWO            standard       23h
```

Related Information

kind documentation

Group, version, kind, meta

This is the initial section of your YAML file that you need to specify in all cases.

You need to add the following section to the top of each NiFi CR you write. It defines the group “cfm.cloudera.com”, the version “v1alpha1”, the kind “Nifi”, and the name of your cluster and the NiFi nodes. It can also specify the name space in which resources will be deployed. It is expected that a single NiFi cluster is deployed in a given namespace. You can also specify namespace during deployment, if that is what you want, omit namespace from the CR

```
apiVersion: cfm.cloudera.com/v1alpha
kind: Nifi
metadata:
  name:[***NIFI CLUSTER NAME***]
  namespace: [***NIFI CLUSTER NAMESPACE***]
```

Replace *[***NIFI CLUSTER NAME***]* and *[***NIFI CLUSTER NAMESPACE***]* with the desired cluster name and cluster namespace respectively.

Configuring environment variables

Environment variables can be added to the NiFi container using the following spec:

```
spec:
  statefulset:
    env:
      - name: [***VARIABLE NAME***]
        value: [***VARIABLE VALUE***]
```

Configuring NiFi image

Specify location of the image used for deployment.

This is how you specify the NiFi image repository, image version, and NiFi version to be used for deployment. This describes the images used for running NiFi. This also provides a way of manually upgrading the NiFi version in an existing cluster or very quickly rolling out NiFi clusters with new versions.

```
spec:
  nifiVersion: [***NIFI VERSION***]
  image:
    repository: container.repository.cloudera.com/cdp-private/cfm-nifi-k8s
    tag: [***IMAGE VERSION TAG***]
```

Note:

container.repository.cloudera.com/cdp-private/cfm-nifi-k8s is the default repository for Cloudera Kubernetes images. If your Kubernetes cluster has no internet access or you want to use a self-hosted repository, replace it with the relevant path.

Note: nifiVersion is a mandatory parameter that currently defaults to 1.0.0. This may cause misconfiguration issues with NiFi 2.x clusters.

Configuring volumes and mounts

Arbitrary volumes and volume mounts can be added to the NiFi container. This can be used to provide Python scripts or other artifacts to the NiFi runtime. Combined with a Cloud Storage Interface driver and Persistent Volume Claim, you can give NiFi access to files in cloud storage such as EFS or S3 buckets.

Define your custom volumes and mounts with the following spec:

```
spec:
  statefulset:
    volumes:
      - name: foo-volume
        persistentVolumeClaim: foo-pvc
    volumeMounts:
      - name: foo-volume
        mountPath: /opt/nifi/foo
```

To learn more about Kubernetes volumes, see Persistent volumes in the Kubernetes documentation.

Configuring cluster size

Specify the number of pods in your deployment.

This section configures the number and capacity of your pods in the cluster.

```
spec:
  replicas: [***NUMBER OF REPLICAS***]
  resources:
    nifi:
      requests:
        cpu: "[***CPU IN CORES***]"
        memory: [***MEMORY IN BITES***]
      limits:
        cpu: "[***CPU IN CORES***]"
        memory: [***MEMORY IN BITES***]
    log:
      requests:
        cpu: [***CPU IN CORES***]
        memory: [***MEMORY IN BITES***]
```

Configuring out of memory recovery

You can optionally specify the step size in memory increase to prevent out of memory (OOM) crashes to your pods. You can also specify an upper bound to memory increase, to prevent infinite scaling.

The Cloudera Flow Management Operator for Kubernetes can detect an Out of Memory event in a NiFi cluster and scale up the memory footprint when configured for Out of Memory Recovery. This feature is not preventative but responsive, the NiFi cluster must first run out of memory and fail a Readiness check before the recovery attempt will be made, potentially impacting Flow performance. OOM Recovery is intended to be a safe guard and is not a replacement for good cluster sizing. If OOM Recovery has triggered, it is recommended that you reevaluate your NiFi resource sizing.

OOM Recovery has two fields to configure: stepSize and upperBound. stepSize defines the amount of memory that should be added for each OOM event. upperBound defines the maximum amount of memory to which the OOM Recovery process is allowed to grow.

```
spec:
  outOfMemoryRecovery:
    stepSize: [***DEFINES THE MEMORY INCREASE EVERY TIME PODS ARE
 OOMKILLED***]
```

```
    upperBound: [***SPECIFIES THE UPPER LIMIT OF MEMORY INCREASE FOR MEMORY
 PROTECTION***]
```

For example:

```
spec:
  outOfMemoryRecovery:
    stepSize: 1Gi
    upperBound: 8Gi
  resources:
    nifi:
      requests:
        cpu: "1"
        memory: 4Gi
```

The above spec starts with NiFi containers at 4Gi and will grow by 1Gi for every OOM that occurs until the NiFi container memory reaches 8Gi. When only memory requests are provided, the NiFi container memory request will grow. If memory limits are provided, only the memory limit will grow.

Note: This can break Quality of Service for the Pod, in the future the requests and limits will grow proportionately.

Once the OOM Recovery has taken effect, it will never automatically scale down. Removal of the OOM Recovery growth will occur when a NiFi resource spec change is detected or when OOM Recovery is removed from the NiFi spec.

NiFi Resource Conditions

The following status field and condition have been added to track the OOM Recovery process:

```
status:
  conditions:
  - lastTransitionTime: "2025-04-15T16:16:15Z"
    message: NiFi has vertically scaled for OOM recovery
    observedGeneration: 2
    reason: OOMRecoveryScaleUp
    status: "False"
    type: VerticallyScaleUp
  outOfMemoryRecoveryGrowth: 500Mi
```

The field outOfmemoryRecoveryGrowth tracks how much the NiFi memory has already grown. The VerticallySc aleUp condition provides the last time the cluster scaled up as well as if the scaling action is complete or not. While the status of VerticallyScaleUp is “True”, the scaling is in progress. Once the scaling action is complete, the status is set to “False”.

Configuring cluster scheduling

Set a schedule for a NiFi cluster.

A field, clusterSchedule, defines the “up time” of your NiFi cluster. During the down time, the cluster is suspended which means all data and data flow configuration is still persisted. For more information, see *Cluster Suspension*.

The clusterSchedule supports two time formats for declaring a schedule: cron and time range.

Cron

The cron scheduler takes in a standard Cron expression and a run duration. The following example unsuspends the NiFi cluster once every three hours, starting at 00:00, and runs for one hour before suspending again.

```
spec:
  clusterSchedule:
    cron:
      schedule: "0 */3 * * *"
      runDuration: 1h
```

The runDuration field supports setting minutes (30m) and hours (2h) or a combination (2h30m).

TimeRange

TimeRange specifies two wall clock time instances between which the NiFi cluster will run. The following example runs daily between the hours of 12:00 and 13:00 UTC.

```
spec:
  clusterSchedule:
    timeRange:
      startTime: "12:00:00Z"
      stopTime: "13:00:00Z"
```

To configure timeRange to begin at night and finish in the morning, timeRange supports setting a stopTime that is earlier than the startTime. The following example begins running at 22:00 UTC and ends at 03:00 UTC the following day.

```
spec:
  clusterSchedule:
    timeRange:
      startTime: "22:00:00Z"
      stopTime: "03:00:00Z"
```

Both startTime and stopTime support numerical time offsets for timezones. For example, “12:00:00-04:00” is noon (12pm) in Eastern Standard Time and 16:00 in UTC.

NiFi resource conditions

The NiFi Resource Status will show a ClusterScheduled condition.

When there is no schedule, the condition status will be False and will indicate that there is no schedule.

```
  conditions:
  - lastTransitionTime: "2025-04-15T15:51:52Z"
    message: NiFi cluster has no schedule
    observedGeneration: 2
    reason: NoClusterSchedule
    status: "False"
    type: ClusterScheduled
```

When a schedule is set and the NiFi cluster is currently not running, the ClusterScheduled condition will have status True and state the time at which the cluster will be restored.

```
  conditions:
  - lastTransitionTime: "2025-04-15T20:45:12Z"
    message: NiFi cluster is scheduled to be restored at 2025-04-15T20:50:0
0Z
    observedGeneration: 1
    reason: ClusterScheduledForRestoration
    status: "True"
    type: ClusterScheduled
```

When a schedule is set and the NiFi cluster is currently running, the ClusterScheduled condition will have status True and state the time at which the cluster will be suspended.

```
  conditions:
  - lastTransitionTime: "2025-04-15T20:45:12Z"
```

```
    message: NiFi cluster is scheduled to be suspended at 2025-04-15T20:48:
00Z
    observedGeneration: 1
    reason: ClusterScheduledForSuspension
    status: "True"
    type: ClusterScheduled
```

Configuring bootstrap JVM settings

Learn about Java memory calculations, defaults and how to override them with custom values.

Cloudera Flow Management Operator for Kubernetes calculates JVM memory setting Max Direct Memory Size, Min Heap Size (xms), and Max Heap Size (xmx) based on container memory limits or requests.

In bootstrap settings, java.arg.10 is DirectMem, java.arg.2 is Min Heap, and java.arg.3 is Max Heap.

Java memory is calculated and set in the following order:

1. **Based on memory of the NiFi resource**
    - The minimum DirectMem allowed for NiFi is 512MB. DirectMem is set to the maximum value between 512MB and 10% of the memory limit. If you do not provide a memory limit, the same calculation is made on the memory request in the specifications.
    - Min Heap Size (xms) and Max Heap Size (xmx) is set to 75% of the memory limit subtracting the calculated DirectMem.
2. **Defaults**

    If you do not specify memory for the NiFi resource, the following default values are automatically set:

    - java.arg.2: -Xms2g
    - java.arg.3: -Xmx2g
    - java.arg.10: -XX:MaxDirectMemorySize=512m

Advanced configuration: Custom values to override inbuilt memory calculations

You can set each java argument for memory as part of NiFi specifications, under the configOverride key.

```
spec:
  configOverride:
    bootstrapConf:
      upsert:
        java.arg.2: -Xms2g
        java.arg.3: -Xmx2g
        java.arg.10: -XX:MaxDirectMemorySize=512m
```

Configuring persistence

Specify storage size and class globally, or for individual repositories.

This section specifies the storage to be used for the NiFi repositories. You can define storage globally, or have overrides for specific repositories. In case of OpenShift, the storage classes have to be specified at the OpenShift level to match the IOPS expectations for your NiFi workloads.

The Cloudera Flow Management Operator for Kubernetes can configure persistent disk storage for the following directories:

- state
- data

- FlowFile Repository
- Content Repository
- Provenance Repository

In the persistence spec, you can define a default size and StorageClass which applies to each of the directories. The spec can be further configured to define specific sizes and StorageClasses for each directory, if necessary.

```
spec:
  persistence:
    size: [***SIZE IN GIGABITES***]
    storageClass: [***STORAGE CLASS***]
    contentRepo:
      size: [***SIZE IN GIGABITES***]
      storageClass: [***STORAGE CLASS***]
    flowfileRepo:
      size: [***SIZE IN GIGABITES***]
    provenanceRepo:
      size: [***SIZE IN GIGABITES***]
```

Configuring assets

Learn about configuring access to NiFi assets.

You can make NiFi assets, like configuration files available to your NiFi cluster using the assets field. This field allows you to specify a mount path within the NiFi Pods to which the provided pre-existing Persistent Volume Claim (PVC) is mounted. Cloudera Flow Management Operator for Kubernetes does not provide a method of loading assets into this volume. Using the example below, all files located in the volume associated with my-nifi-assets-volume-claim are accessible at the path /opt/nifi/nifi-assets/ for use within your flow.

Before you can start using assets with NiFi deployed through Cloudera Flow Management Operator for Kubernetes, you have to provide the following:

- A volume provisioner which supports creating volumes that are ReadWriteMany (RWX), for example nfs-provisioner.
- A pod running some kind of software or process (for example, an FTP server) attached to the RWX volume for loading files onto the volume. Alternatively, you can use the `kubectl cp` command to directly copy files into most containers, such as Ubuntu.

After meeting the above prerequisites, you need to create a Persistent Volume Claim (PVC) which creates the required RWX volume. For example:

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: nifi-assets
spec:
  storageClassName: [***STORAGE CLASS***]
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 1Gi
```

where *[***STORAGE CLASS***]* refers to the Storage Class associated with the RWX volume provisioner.

With the PVC created and the PV provisioned, attach the pod you have created, expose the pod via Ingress or Service if required, and load the asset files.

```
spec:
  assets:
    mountPath: [***ASSETS PATH***]
    persistentVolumeClaim:
      name: nifi-assets
```

where *[***ASSETS PATH***]* is the filesystem path within the NiFi container where the assets are located, for example, /opt/nifi/assets.

Related Information

ReadWriteMany (RWX)

nfs-provisioner

Persistent Volume Claim (PVC)

Configuring NAR providers

Provide custom NARs to NiFi.

NAR provider volumes

Custom NARs can be provided to NiFi via Kubernetes volumes. The volumes used for NARs must support RWX (Read Write Many) access mode, such as an NFS volume. You need a Persistent Volume Claim which references your RWX volume, such as:

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: [***YOUR VOLUME CLAIM NAME***]
spec:
  storageClassName: "nfs"
  accessModes:
  - ReadWriteMany
  resources:
    requests:
      storage: 10Mi
```

The above storage class provisions an NFS volume from the nfs-server-provisioner, if it is installed in your cluster. Cloud provider classes like EFS or S3 on AWS can be used if their CSI drivers are installed.

Finally, provide your persistent volume claim to the NiFi spec as follows. You can optionally provide a subPath for the volume if you wish to specify only a certain directory within that volume.

```
spec:
  narProvider:
    volumes:
      - volumeClaimName: [***YOUR VOLUME CLAIM NAME***]
        subPath: [***OPTIONAL SUBPATH***]
      - volumeClaimName: [***ANOTHER VOLUME CLAIM***]
```

Related Information

Persistent volumes | Kubernetes

NFS Server Provisioner | ArtifactHUB

CSI Driver for Amazon EFS | GitHub

Mountpoint CSI driver for Amazon S3 | GitHub

Configuring Kubernetes state management

Specify Kubernetes native state management provider as the state management provider of your cluster.

Cloudera's distribution of NiFi comes with a Kubernetes native state management provider. This is the recommended state management for use with Cloudera Flow Management Operator for Kubernetes. However, as it is not the default state management provider set by Cloudera Flow Management Operator for Kubernetes, you need to add this section to the configuration. Without this configuration, a ZooKeeper cluster is expected.

To configure the Kubernetes state management provider, use the below YAML.

```
spec:
  stateManagement:
   clusterProvider:
     id: kubernetes-provider
     class: org.apache.nifi.kubernetes.state.provider.KubernetesConfigMapS
tateProvider
  configOverride:
    nifiProperties:
      upsert:
        nifi.cluster.leader.election.implementation: “KubernetesLeaderElect
ionManager”
```

Configuring node certificate generation

Learn about certificate generation options.

Note:

NiFi deployed by the Cloudera Flow Management Operator for Kubernetes is secure by default. If you do not provide an authorization method, NiFi is configured in single user auth mode with username "demonstration-user" and password "demonstration-password". If you do not specify Node Certificate Generation, node certificates are generated using a self-signed Cluster Issuer nifi-node-certs that is deployed with the operator.

Cloudera Flow Management Operator for Kubernetes provides automatic certificate generation for each NiFi node in a given cluster by way of cert-manager certificates to secure intra-cluster communication between NiFis. To configure nodeCertGen, a cert-manager Issuer or ClusterIssuer is required. A self-signed Issuer setup is sufficient for development environments. In production environments use a third-party authority, or internal signing CAs.

```
spec:
  security:
    nodeCertGen:
      issuerRef:
        name: self-signed-ca-issuer
        kind: ClusterIssuer
```

Related Information

Issuers and ClusterIssuers

Configuring additional CA bundles

Add custom certificates to the NiFi truststore to allow NiFi to trust third party services.

There are two methods for adding certificates to NiFi's truststore: in-line in the custom resource or through a Secret/ConfigMap. For multiple certificates, it is recommended to provide them via Secret/ConfigMap to maintain readability of the NiFi custom resource.

In-line

```
spec:
  security:
    additionalCABundles: [***BASE64 ENCODED CERT CHAIN***]
```

Secret/ConfigMap

First create a Secret with the needed Certificates. The referenced files may have multiple certificates in them.

```
kubectl create secret generic nifi-additional-cas --from-file=cert1.cr
t=[***A CERTIFICATE FILE***] --from-file=cert2.crt=[***ANOTHER CERTIFICATE
 FILE***]
```

Then supply the Secret/ConfigMap name to the following spec:

```
spec:
  security:
    additionalCABundlesRef:
      name: nifi-additional-cas
      kind: Secret
```

Configuring NiFi properties

Learn how to override default NiFi configuration settings provided by Cloudera Flow Management Operator for Kubernetes from the CR file.

NiFI settings are available as part of the specification, under the configOverride key. They can be provided in one of the following ways:

- inline,
- as a ConfigMap
- as a Secret

```
spec:
  configOverride:
    nifiProperties:
      upsert:
        nifi.cluster.load.balance.connections.per.node: "1"
        nifi.cluster.load.balance.max.thread.count: "4"
        nifi.cluster.node.connection.timeout: "60 secs"
        nifi.cluster.node.read.timeout: "60 secs"
    bootstrapConf:
      upsert:
        java.arg.2: -Xms2g
        java.arg.3: -Xmx2g
        java.arg.13: -XX:+UseConcMarkSweepGC
```

Overriding NiFi settings using ConfigMaps and Secrets

Learn about overriding default NiFi settings using ConfigMaps and Secrets.

The ConfigMap or Secret values are available to inject into the environment for the following files:

- authorizers.xml
- bootstrap.conf
- logback.xml
- login-identity-providers.xml
- nifi.properties
- state-management.xml

Each of these config overrides must be in an individual ConfigMap with the key being the filename to be replaced. Using this ConfigMap or Secret reference method entirely overrides the defaults provided by the Cloudera Flow Management Operator for Kubernetes, which may impact cluster operation.

```
NiFiSpec
```

```
spec:
  configOverride:
    authorizersObjectReference:
      kind: "ConfigMap"
      name: "custom-authorizers"

ConfigMapSpec
data:
  authorizers.xml: |
      <authorizers>
        <authorizer>
          <identifier>single-user-authorizer</identifier>
          <class>org.apache.nifi.authorization.single.user.SingleUserAuth
orizer</class>
        </authorizer>
      </authorizers>
```

Configuring scaling

Learn about scaling NiFi clusters either manually or automatically, using HPA.

It is possible to manually scale up and down the NiFi cluster size by editing the replicas value in the deployment file and applying the changes. It is also possible to specify an HPA to automatically scale the NiFi cluster (replica count) based on the Kubernetes resources (CPU/memory).

To manually scale the cluster, simply edit the replicas field to your desired replica count.

For autoscaling, apply a Horizontal Pod Autoscaling (HPA) resource targeting the NiFi CR, as follows:

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
 name: nifi-hpa
spec:
 maxReplicas: 3
 minReplicas: 1
 metrics:
   - type: Resource
     resource:
       name: cpu
       target:
         type: Utilization
         averageUtilization: 75
  scaleTargetRef:
   apiVersion: cfm.cloudera.com/v1alpha1
   kind: Nifi
   name: [***NIFI CLUSTER NAME***]
```

Configuring pod affinity

Pod affinity controls where pods are deployed based on node configuration and placement of other pods.

To learn more about Pod Affinity, read *Assigning Pods to Nodes* in the Kubernetes documentation.

You can configure the affinity settings of the NiFi pod in the NiFi Custom Resource under spec.statefulset. The following example represents the default configuration which will be added to the Custom Resource in the defaulting webhook.

Note:

If any affinity is provided in spec.statefulset, the default in the example will not be applied.

```
spec:
  statefulset:
    affinity:
      podAntiAffinity:
        preferredDuringSchedulingIgnoredDuringExecution:
        - podAffinityTerm:
            labelSelector:
              matchExpressions:
              - key: app.kubernetes.io/instance
                operator: In
                values:
                - mynifi
            topologyKey: kubernetes.io/hostname
          weight: 1
```

This default configuration attempts to spread NiFi cluster pods to different nodes of the Kubernetes cluster. If there are more NiFi pods than available Kubernetes nodes, then some pods will coexist on the same node.

Related Information

Assigning Pods to Nodes | Kubernetes documentation

Configuring connections to NiFi

Learn about configuring connections for your NiFi cluster.

Cloudera Flow Management Operator for Kubernetes provides a flexible method of configuring connections to NiFi called Connections. Using Connections, a Service, Ingress, or Route can be configured to route to a specific port on NiFi. For defining Connections targeting an arbitrary port on NiFi, use the spec.connections array. For configuring connection to the NiFi Web UI, use the spec.uiConnection field. This documentation provides a full reference for Connections.

Configuring session affinity

Learn about configuring session affinity. It makes possible to keep connection to the web UI alive in clusters with several nodes.

Regardless of your connection type, a NiFi cluster with more than one node requires session affinity of some type for the Web UI to operate. This is because each NiFi node can supply its own web UI and if a LoadBalancer shifts you to another instance, your authentication tokens become invalid. The best method of applying session affinity varies greatly depending on the Kubernetes cluster provider. In the simplest case, defining session affinity on the web Service resource itself is sufficient:

```
spec:
  uiConnection:
    serviceConfig:
      sessionAffinity: ClientIP
```

In certain clouds, for example AWS, the backing LoadBalancer resources do not support session affinity, and cause provisioning to break.

Configuring arbitrary connections

Learn about configuring a connections array.

You can use the connections array to flexibly define routing to ports on NiFi. The below example configures an Ingress resource with some annotations and labels provided. The Ingress will expose a URL https://nifi.io/listenTCP which routes to port 9432 on NiFi. Additionally, the backing Service is configured to have two extra ports, 8496 and 8495.

```
spec:
  connections:
  - type: Ingress
    name: someConnection
    annotations:
      someanno: myanno
    labels:
      somelabel: mylabel
    ingressConfig:
      hostname: nifi.io
      paths:
      - port: 9432
        path: /listenTCP
        name: listentcp
    serviceConfig:
      ports:
      - port: 8496
        protocol: TCP
        name: porta
      - port: 8495
        protocol: UDP
        name: portb
```

Configuring NiFi Web UI connection

Learn about configuring a connection to the NiFi web UI.

You can configure a connection to the NiFi Web UI using the spec.uiConnection field. It is a standard connection field with special validation and handling. The name of this connection is always ignored and set to *[***CR NAME***]*-web. For Ingress type connections, a maximum of one path may be specified. When you configure a uiConnection, the spec.hostname field is required.

The uiConnection can support hostname routing with and without an additional context path. It is not recommended to use a context path for routing as NiFi does not support it well, but it is possible. For more information, see NiFi documentation on proxy configuration. An example using ingress-nginx is included in this section.

Related Information

NiFi proxy configuration

Hostname-only ingress example

Learn about configuring an Ingress resource using TLS files generated by Cloudera Flow Management Operator for Kubernetes.

This YAML snippet configures an Ingress resource for accessing the NiFi Web UI. It uses the TLS files generated by a Cloudera Flow Management Operator for Kubernetes created Certificate as defined in spec.security.ingressCertGen. The supplied annotations are for the ingress-nginx Ingress controller. The affinity settings enable a persistent session so that UI interactions go to the same NiFi node in the cluster. The backend-protocol setting is needed for when NiFi is configured to be secure, as it will reject any non-HTTPS connection attempts.

```
spec:
  uiConnection:
    type: Ingress
    ingressConfig:
      ingressClassName: myIngressClass
      ingressTLS:
      - hosts:
        - nifi.localhost
        secretName: mynifi-ingress-cert
    annotations:
      nginx.ingress.kubernetes.io/affinity: cookie
      nginx.ingress.kubernetes.io/affinity-mode: persistent
      nginx.ingress.kubernetes.io/backend-protocol: HTTPS
```

Hostname-only route example

Learn about configuring a Route resource to acces the NiFi web UI.

This YAML snippet configures a Route resource for accessing the NiFi web UI.

```
spec:
  uiConnection:
    type: Route
    routeConfig:
      tls:
        termination: passthrough
```

Ingress with context path example

Learn about configuring an Ingress resource that rewrites the connection path in incoming requests and does a reverse-rewrite on UI calls going to the backend.

This YAML code snippet configures an ingress UI Connection with a path. The annotations here are for the ingress-nginx ingress controller and all are required for NiFi to correctly understand the incoming requests.

In the example the path includes some regex at the end: (/|$)(.*). This regex informs the rewrite directives in the configuration-snippet and rewrite-target annotations. NiFi does not handle proxy paths well, it does not understand that https://nifi.localhost/some/path/to/nifi coming through the defined Ingress is intended to call the /nifi API to load the UI. The rewrite-target annotation addresses this by capturing the /nifi and anything that comes after and sends that as the path to the NiFi pod. It translates /some/path/to/nifi/ to /nifi/. Similarly, the NiFi web UI does not correctly form API calls going to the backend, attempting to call /nifi/ instead of /some/path/to/nifi/. This is addressed by the configuration-snippet rewrite instruction. It does the reverse of the rewrite-target, reapplying the removed context path /some/path/to. The remaining configuration-snippet lines are headers required by a NiFi behind a proxy. For more information, see the *NiFi System Administrator's Guide*.

```
spec:
  uiConnection:
    type: Ingress
    ingressConfig:
      ingressClassName: myIngressClass
      ingressTLS:
      - hosts:
        - nifi.localhost
        secretName: mynifi-ingress-cert
      paths:
      - port: 8443
        path: "/some/path/to(/|$)(.*)"
    annotations:
      nginx.ingress.kubernetes.io/affinity: cookie
      nginx.ingress.kubernetes.io/affinity-mode: persistent
      nginx.ingress.kubernetes.io/backend-protocol: HTTPS
      nginx.ingress.kubernetes.io/configuration-snippet: |-
        proxy_set_header X-ProxyScheme $scheme;
        proxy_set_header X-ProxyHost $host;
        proxy_set_header X-ProxyPort $server_port;
        proxy_set_header X-ProxyContextPath /some/path/to;
        rewrite (.*\/nifi)$ $1/ redirect;
        proxy_ssl_name mynifi.default.svc.cluster.local;
      nginx.ingress.kubernetes.io/rewrite-target: /$2
```

Configuring additional proxy hosts

Learn about adding a list of expected proxy hosts. NiFi will reject API requests sent through proxies if it is not aware of those proxy hosts.

Provide a list of expected proxy hosts to NiFi beyond the hostname provided in hostName. To add additional proxy hosts, add the following to your NiFi YAML:

```
spec:
  additionalProxyHosts:
    - [***YOUR PROXY***]
    - [***ANOTHER PROXY***]
```

Configuring authentication and authorization for NiFi

Learn about configuring the type of authentication appropriate for your use case.

Note:

NiFi requires all web and API traffic be over HTTPS to support user authentication and authorization. For information on adding an auto-generated certificate to each node, see Node certificate generation.

Note:

NiFi deployed by the Cloudera Flow Management Operator for Kubernetes is secure by default. If you do not provide an authorization method, NiFi is configured in single user auth mode with username "demonstration-user" and password "demonstration-password". If you do not specify Node Certificate Generation, node certificates are generated using a self-signed Cluster Issuer nifi-node-certs that is deployed with the operator.

Configuring the initial admin user

When you set up a secured NiFi instance for the first time, you must manually designate an "Initial Admin Identity". This initial admin user is granted access to the UI and given the ability to create additional users, groups, and policies.

NiFi requires an initial admin user which will be given sufficient privileges to configure other users and policies. When configuring an authentication method other than single user authentication, an initial admin user is required.

Specify the initial admin user with the following YAML snippet:

```
spec:
  security:
    initialAdminIdentity: [***INITIAL ADMIN IDENTITY***]
```

Replace *[***INITIAL ADMIN IDENTITY***]* with a username, LDAP distinguished name (DN), or a Kerberos principal.

Related Concepts

Configuring single user authentication

Configuring a user identity with access policies

Configuring LDAP authentication

Configuring OIDC authentication

Configuring single user authentication

Single user authentication is NiFi’s most basic authentication option, sufficient for individual development clusters and also production clusters where flows are deployed in a controlled manner, such as continuous integration (CI) or site reliability engineering (SRE). A single user is granted all permissions on the NiFi cluster, no other users can be configured.

- Configuration snippet for letting NiFi generate the password.

  ```
  spec:
    security:
      singleUserAuth:
        enabled: true
  ```

  You find the generated username and password in the app-log container logs.
- Configuration snippet for setting NiFi username and password using a Secret:

```
spec:
  security:
    singleUserAuth:
      enabled: true
      credentialsSecretName: [***YOUR CREDENTIALS SECRET***]
```

Replace:

[YOUR CREDENTIALS SECRET***]***

Create your credentials secret with the following command:

```
kubectl create secret generic [***YOUR CREDENTIALS SECRET***] --
from-literal=username=[***YOUR USER NAME***] --from-literal=pass
word=[***YOUR PASSWORD***]
```

Replace:

[YOUR CREDENTIALS SECRET***]***

with the desired credentials secret name

[YOUR USER NAME***]***

with the generated username in the app-log container logs

[YOUR PASSWORD***]***

with the generated password in the app-log container logs

Related Concepts

Configuring the initial admin user

Configuring a user identity with access policies

Configuring LDAP authentication

Configuring OIDC authentication

Configuring a user identity with access policies

The User CRD represents a NiFi user and allows you to define a user identity and access policies.

```
apiVersion: cfm.cloudera.com/v1alpha1
kind: User
metadata:
  name: nifi-user
spec:
  instanceTarget:
    kind: Nifi
    name: [***NIFI CLUSTER NAME***]
    namespace: [***TARGET NAMESPACE***]
  identity: [****NAME OF THE USER IDENTITY**]
  accessPolicies:
  - resources:
     [***
      - ACCESSIBLE
```

```
      - RESOURCES***]
    actions:
      [***
      - PERMITTED
      - ACTIONS***]
```

For example, to create a user on the NiFi cluster defined by the resource *MYNIFI* in the *DEFAULT* namespace named *USER05* with *READ* and *WRITE* access on *FLOW*, *CONTROLLER*, and *ROOT* process group components, you would use:

```
apiVersion: cfm.cloudera.com/v1alpha1
kind: User
metadata:
  name: nifi-user
spec:
  instanceTarget:
    kind: Nifi
    name: MYNIFI
    namespace: DEFAULT
  identity: USER05
  accessPolicies:
  - resources:
      - /FLOW
      - /CONTROLLER
      - /PROCESS-GROUPS/ROOT
    actions:
      - READ
      - WRITE
```

Related Concepts

Configuring the initial admin user

Configuring single user authentication

Configuring LDAP authentication

Configuring OIDC authentication

Configuring LDAP authentication

Learn how to configure an LDAP server for user authentication in your NiFi or NiFi Registry cluster.

Cloudera Flow Management Operator for Kubernetes can configure NiFi to connect to an LDAP server for user authentication.

Prerequisites:

- Full LDAP URL, i.e. ldap://*[***LDAP SERVER URL***]*:*[***LDAP PORT***]*
- Desired authentication strategy
- Authentication credentials and key/trust stores if using LDAPS.
- User search filters

For LDAP servers protected with any authentication, a Secret must be created containing the correct authentication credentials and TLS resources (if applicable). The Secret must contain the following data fields:

- managerPassword
- keystore (if TLS is configured)
- keystorePassword (if TLS is configured)
- truststore (if TLS is configured)
- truststorePassword (if TLS is configured)

Create the secret usiing the cubectl CLI utility:

```
kubectl create secret generic my-ldap-creds \
      --from-literal=managerPassword=myMan@gerPassw0rd \
   --from-file=keystore=/path/to/keystore \
   --from-literal=keystorePassword=myKeystorePassword \
   --from-file=truststore=/path/to/truststore \
   --from-literal=truststorePassword=myTruststorePassword
```

The following example shows a connection to an LDAP server protected with basic authentication with TLS.

```
spec:
  security:
    initialAdminIdentity: mynifiadmin
    ldap:
      authenticationStrategy: SIMPLE
      managerDN: "cn=admin,dc=example,dc=org"
      secretName: my-ldap-creds
      referralStrategy: FOLLOW
      connectTimeout: 3 secs
      readTimeout: 10 secs
      url: ldap://my-ldap-url:389
      userSearchBase: "dc=example,dc=org"
      userSearchFilter: "(uid={0})"
      identityStrategy: USE_USERNAME
      authenticationExpiration: 12 hours
 tls:
   keystoreType: jks
   truststoreType: jks
   clientAuth: NONE
   protocol: TLSv1.2
```

By default, Cloudera Flow Management Operator for Kubernetes does not deploy a UserGroupProvider using the LDAP target. This means NiFi does not pull down any users, only queries the LDAP server for authentication. This impedes configuring user access, requiring the NiFi administrator to create each user manually.

The following example shows configuring user synchronization with the LDAP server:

```
spec:
  security:
    ldap:
      sync:
        interval: 30 min
        userObjectClass: inetOrgPerson
        userSearchScope: SUBTREE
        userIdentityAttribute: cn
        userGroupNameAttribute: ou
        userGroupNameReferencedGroupAttribute: ou
        groupSearchBase: "dc=example,dc=org"
        groupObjectClass: organizationalUnit
        groupSearchScope: OBJECT
        groupNameAttribute: ou
```

Related Concepts

Configuring the initial admin user

Configuring single user authentication

Configuring a user identity with access policies

Configuring OIDC authentication

Configuring OIDC authentication

NiFi supports user authentication with Open ID Connect (OIDC) providers such as Keycloak.

To configure authentication with an Open ID Connect (OIDC) provider, you need to know the Discovery URL, clientId, and clientSecret of the authenticating server.

An example of a Discovery URL from Keycloak is:

```
https://keycloak.cfmoperator.net/realms/master/.well-known/openid-configurat
ion
```

The clientID and clientSecret fields are provided to NiFi in a Kubernetes secret. Create that secret with the following command:

```
kubectl create secret generic oidc-client-secret --from-literal=clie
ntID=[***YOUR CLIENT ID***] --from-literal=clientSecret=[***YOUR CLIENT
 SECRET***]
```

The Discovery URL and client credentials secret are provided to NiFi with the below spec:

```
spec:
  security:
    openIDAuth:
      discoveryURL: [***YOUR DISCOVERY URL***]
      clientSecretName: [***OIDC CLIENT SECRET***]
```

OpenIDAuth also provides additional options:

connectTimeout

Specify the connection timeout when communicating with the OIDC provider.

readTimeout

Specify the read timeout when communicating with the OIDC provider.

JWSAlgorithm

JWSAlgorithm is the preferred algorithm for validating identity tokens. If this value is blank, it defaults to RS256 which is required to be supported by the OIDC provider according to the specification. If this value is HS256, HS384, or HS512, NiFi attempts to validate HMAC protected tokens using the specified client secret. If this value is none, NiFi attempts to validate unsecured/ plain tokens. Other values for this algorithm attempt to parse as an RSA or EC algorithm to be used in conjunction with the JSON Web Key (JWK) provided through the jwks_uri in the metadata found at the discovery URL.

Note:

For NiFi to trust the certificate presented by the OIDC server, you must add a valid CA for your OIDC server to NiFi. For informatioon on adding a CA to NiFi, see Additional CA Bundles.

Related Concepts

Configuring the initial admin user

Configuring single user authentication

Configuring a user identity with access policies

Configuring LDAP authentication

Related Information

OpenID Connect | Apache NiFi System Administrator's Guide

Configuring JVM security providers (FIPS)

NiFi and NiFi Registry are not FIPS compliant out of the box. When booting cfm-nifi-k8s for NiFi version 1 on a FIPS enabled cluster, the Pod will enter a CrashLoop attempting to load JKS keystores. NiFi version 2 will boot but not necessarily be compliant. Follow the instructions here to add additional security providers to the NiFi JVM to enable FIPS compliance.

Prerequisites

FIPS compliance requires special security providers to be given to the NiFi and NiFi Registry containers. To fully configure these new providers, the operator requires a few pieces of information:

1. Security provider jars.
2. Keystore provider class.
3. Preferred keystore format.
4. Security providers definition.
5. Java policy for providers. (optional)

Security provider jars

These are Java jar files containing FIPS compliant security providers that you have obtained from Cloudera (CCJ and BCTLS) or another vendor, such as Safelogic. The jars should be referred to by the environment variable PROVIDER_JAR_PATH.

The rest of this document will show examples using ccj and bctls from Cloudera's archive mirror.

Keystore provider class

The provider class that should be used for constructing keystores and truststores. Using ccj, this would be com.safelogic.cryptocomply.jcajce.provider.CryptoComplyFipsProvider. This will be provided to NiFi by environment variable KEYSTORE_PROVIDER_CLASS.

Preferred keystore format

The default keystore format JKS is a weak format and generally not FIPS compliant. Your security provider may provide a different format, such as Bouncy Castle FIPS KeyStore (BCFKS). This will be supplied to NiFi by environment variable KEYSTORE_TYPE.

Security providers definition

The security providers to add to the JVM must be provided in a file with one provider per line.

CCJ example:

```
$ cat additional-security-providers.txt
com.safelogic.cryptocomply.jcajce.provider.CryptoComplyFipsProv
ider
org.bouncycastle.jsse.provider.BouncyCastleJsseProvider fips:CCJ
```

A path reference to this file must be provided with an environment variable SECURITY_PROVIDERS_PATH.

Java policy for providers

For some providers, additional permissions may need to be given via Java policy. A standard Java policy file can be provided, see this CCJ example:

```
$ cat additional-java-policy.txt
grant {
    //CCJ Java Permissions
    permission java.lang.RuntimePermission "getProtectionDomain";
    permission java.lang.RuntimePermission "accessDeclaredMembers
";
    permission java.util.PropertyPermission "java.runtime.name",
"read";
    permission java.security.SecurityPermission "putProviderProp
erty.CCJ";
    //CCJ Key Export and Translation
    permission com.safelogic.cryptocomply.crypto.CryptoServicesP
ermission "exportKeys";
    //CCJ SSL
    permission com.safelogic.cryptocomply.crypto.CryptoService
sPermission "tlsAlgorithmsEnabled";
    //CCJ Setting of Default SecureRandom
    permission com.safelogic.cryptocomply.crypto.CryptoService
sPermission "defaultRandomConfig";
    //CCJ Setting CryptoServicesRegistrar Properties
    permission com.safelogic.cryptocomply.crypto.CryptoServicesP
ermission "globalConfig";
    //CCJ Enable JKS
    permission com.safelogic.cryptocomply.jca.enable_jks "true";
};
```

A path reference to this file must be provided with an environment variable JAVA_POLICY_PATH.

Configuration

The Cloudera Flow Management Kubernetes Operator for Apache NiFi has two methods of providing FIPS compliant security providers to the NiFi JVM: image rebuild or with volumes.

Image rebuild

Note:

This option requires access to an internal container registry.

This is the recommended method of enabling FIPS if you've got the infrastructure to utilize, as this requires no runtime configuration, Flow developer teams will simply reference the new FIPS enabled image.

You can provide all required JVM Security Provider Information directly to the cfm-nifi-k8s and cfm-nifiregistry-k8s images via an image rebuild. With this method, you will create a Dockerfile that modifies the images you've pulled from Cloudera prior to pushing them to your internal registries.

1. In a directory, place the provider jars, provider definition file, and optional java policy file.

```
$ ls
additional-java-policy.txt  additional-security-providers.txt  bctls.jar
 ccj-3.0.2.1.jar
```

2. Create a Dockerfile.

```
# Use args to parameterize this Dockerfile for reuse
ARG CFM_NIFI_K8S_BASE_IMAGE=container.repository.cloudera.com/cloudera/
cfm-nifi-k8s
ARG CFM_NIFI_K8S_BASE_TAG=2.9.0-b96-nifi_1.27.0.2.3.14.0-14

FROM ${CFM_NIFI_K8S_BASE_IMAGE}:${CFM_NIFI_K8S_BASE_TAG} AS nifi-k8s

# Copy the required files
COPY bctls.jar ccj-3.0.2.1.jar $NIFI_HOME/lib/
COPY additional-java-policy.txt additional-security-providers.txt $NIFI
_HOME/conf/
# Configure environment variables to point to the provided files
ENV PROVIDER_JAR_PATH="$NIFI_HOME/lib/ccj-3.0.2.1.jar:$NIFI_HOME/lib/bctls
.jar"
ENV JAVA_POLICY_PATH="$NIFI_HOME/conf/additional-java-policy.txt"
ENV SECURITY_PROVIDERS_PATH="$NIFI_HOME/conf/additional-security-provide
rs.txt"
# Configure the keystore type
ENV KEYSTORE_TYPE=BCFKS

# Specify the security provider classe
```

```
ENV KEYSTORE_PROVIDER_CLASS=com.safelogic.cryptocomply.jcajce.provider.Cr
yptoComplyFipsProvider
```

3. Build the new image.

```
docker build -t <your-registry>/cloudera/cfm-nifi-k8s:2.9.0-b96-nifi_1.2
7.0.2.3.14.0-14-fips .
docker push <your-registry>/cloudera/cfm-nifi-k8s:2.9.0-b96-nifi_1.27.0.2
.3.14.0-14-fips
```

Using volumes

Using volumes, Security Providers can be configured at deploy time using the standard cfm-nifi-k8s and cfm-nifiregistry-k8s images provided by Cloudera. Prior to deploying NiFi or NiFi Registry, a volume that supports RWX should be created and populated with the required files:

- Security provider jars
- Security provider definition file
- Additional Java policy

1. In your Nifi or NifiRegistry yamls, add the following to mount the volume:

```
spec:
  statefulset:
    volumes:
    - name: fips-providers
      persistentVolumeClaim:
        claimName: [***RWX VOLUME CLAIM***]
    volumeMounts:
    - name: fips-providers
      mountPath: /opt/nifi/fips-providers
```

2. Reference the provided files, keystore type, and keystore provider class:

```
spec:
  security:
    jvmSecurityProviderInfo:
 # List of provider jars in classpath format
      providerJarPath: "/opt/nifi/fips-providers/ccj-3.0.2.1.jar:/opt/ni
fi/fips-providers/bctls.jar"
 # Class providing the keystore implementation
      providerClass: com.safelogic.cryptocomply.jcajce.provider.CryptoCo
mplyFipsProvider
 # Keystore format
      keystoreType: BCFKS
 # Path to security providers definition
      securityProvidersPath: /opt/nifi/fips-providers/additional-security-
providers.txt
 # Path to additional Java policy
      javaPolicyPath: /opt/nifi/fips-providers/additional-java-policy.txt
```

Example CR

The following example NiFi CR deploys a 3 node cluster with Kubernetes-based state management and leader election, and a Route to access the NiFi UI.

```
apiVersion: cfm.cloudera.com/v1alpha1
kind: Nifi
metadata:
  name: mynifi
spec:
  nifiVersion: [***NIFI VERSION***]
  replicas: 3
  image:
    repository: container.repository.cloudera.com/cloudera/cfm-nifi-k8s
    tag: [***NIFI TAG***]
    pullSecret: docker-pull-secret
  tiniImage:
    repository: container.repository.cloudera.com/cloudera/cfm-tini
    tag: [***CFM TINI TAG***]
    pullSecret: docker-pull-secret
  hostName: mynifi.[***OPENSHIFT ROUTER DOMAIN***]
  uiConnection:
    type: Route
    serviceConfig:
      sessionAffinity: ClientIP
```