Cloudera Flow Management Operator 2.8.0

Configuring NiFi CR

Date published: 2024-06-11
Date modified: 2024-06-11

https://docs.cloudera.com/

Legal Notice

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	9
Services	1 + #Connections
Pods	1 min, 3+ recommended
StatefulSet	1
Deployment	0
Ingress	1 + #IngressConnection

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 NiFi Image

Specify locaction of the image used for deployment.

This is how you specify the NiFi image repository and image 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:
  image:
    repository: container.repository.cloudera.com/cdp-private/cfm-nifi-k8s
    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.

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 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 CFM Operator 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, such as custom processor JARs, or 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. CFM Operator 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.

```
spec:
  assets:
    mountPath: “/opt/nifi/nifi-assets”
    persistentVolumeClaim: my-nifi-assets-volume-claim
```

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 CFM Operator. However, as it is not the default state management provider set by CFM Operator, 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.

CFM Operator 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 NiFi properties

Learn how to override default NiFi configuration settings provided by CFM Operator 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 CFM Operator, 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 connections to NiFi and NiFi Registry

Learn about configuring connections for your NiFi cluster.

CFM Operator 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 CFM Operator.

This YAML snippet configures an Ingress resource for accessing the NiFi Web UI. It uses the TLS files generated by a CFM Operator 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 authentication for NiFi and NiFi Registry

Learn about configuring single user authentication for development purposes and TLS authentication more suited for porduction environments.

Configuring single user authentication

Single user authentication is NiFi's most basic authentication option, sufficient for development clusters. A single user is granted all permissions on the NiFi cluster, no other users can be configured.

To configure single user authentication, you need to specify it in loginIdentityProviders and you need to make overrides to the nifi.properties configuration file:

```
spec:
  security:
    customAuthorizer:
      identifier: single-user-authorizer
      className: org.apache.nifi.authorization.single.user.SingleUserAuthor
izer
```

```
  configOverride:
    loginIdentityProviders: |
      <loginIdentityProviders>
        <provider>
          <identifier>single-user-provider</identifier>
          <class>org.apache.nifi.authentication.single.user.SingleUserLog
inIdentityProvider</class>
          <property name="Username">[***SINGLE USER NAME***]</property>
          <property name="Password">[***HASHED PASSWORD**]</property>
        </provider>
      </loginIdentityProviders>
    nifiProperties:
      upsert:
        nifi.security.user.authorizer: single-user-authorizer
        nifi.security.user.login.identity.provider: single-user-provider
```

Replace:

- *[***SINGLE USER NAME***]* with your desired username
- *[***HASHED PASSWORD***]* with a hash of your password. You can use the cfmctl CLI tool to generate this hash using the `single-user-credential` command.

Related Information

Single user authentication

Generating password hash using cfmctl

You can generate a NiFi password hash using the single-user-credentials cfmctl command.

Procedure

1. Run the `singe-user-credentials` command.

   ```
   bin/cfmctl single-user-credentials
   ```

2. Enter a password when prompted.

   ```
   Enter password:
   ```

   The property and the hash value are printed to stdout:

   ```
   <property name="Password">$2b$10$FdOuiIvdvrcF3dG9YMSa3u7VPLXOLQUdLbJAML
   BdeM4S6tbDMg322</property>
   $2b$10$FdOuiIvdvrcF3dG9Y1234u7VPLXOLQUdLbJAMLB6AXQS6tbDMg322
   ```

   Add one or the other to your loginIdentityProviders.xml override to enable single user login using the desired credentials.

Configuring LDAP authentication

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

CFM Operator 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-openldap-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, CFM Operator 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
```

Example CR

The following example NiFi CR deploys a 3 node cluster with single user authentication (admin/admin), 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:
  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
  configOverride:
    nifiProperties:
      upsert:
        nifi.cluster.leader.election.implementation: "KubernetesLeaderElecti
onManager"
    authorizers: |
      <authorizers>
        <authorizer>
          <identifier>single-user-authorizer</identifier>
          <class>org.apache.nifi.authorization.single.user.SingleUserAuthor
izer</class>
        </authorizer>
      </authorizers>
    loginIdentityProviders: |
      <loginIdentityProviders>
        <provider>
          <identifier>single-user-provider</identifier>
          <class>org.apache.nifi.authentication.single.user.SingleUserLoginI
dentityProvider</class>
          <property name="Username">admin</property>
          <property name="Password">$2b$10$GRa8g9Z5rBENXPFNHFBosev9XmY6CSk0
SdcBi5sQMRX92KD73asGG</property>
        </provider>
      </loginIdentityProviders>
  stateManagement:
   clusterProvider:
     id: kubernetes-provider
     class: org.apache.nifi.kubernetes.state.provider.KubernetesConfigMapSt
ateProvider
```