Fabric Services System

Release 23.4.1

Fabric Services System Connect Guide

3HE 19458 AAAA TQZZA
Edition 1
June 2023

Nokia is committed to diversity and inclusion. We are continuously reviewing our customer documentation and consulting with standards bodies to ensure that terminology is inclusive and aligned with the industry. Our future customer documentation will be updated accordingly.

Table of contents

1 Introduction

The Fabric Services System Connect solution (or "Connect") acts as a bridge between the Fabric Services System and different cloud environments.

Connect is aware of the different processes and workloads running on the servers that make up the cloud environment, while at the same time being aware of the fabric as configured on the Fabric Services System itself.

This dual awareness enables Connect to configure the fabric dynamically based on workloads coming and going on the cloud platform. It does this by inspecting the cloud itself and learning the compute server, network interface and VLAN on which a specific workload is scheduled. By also learning the topology based on the LLDP information arriving in the fabric switches, it connects those two information sources.

Components

The Connect solution is built on a modular platform to better support virtually any cloud environment. The solution is divided into two main components:

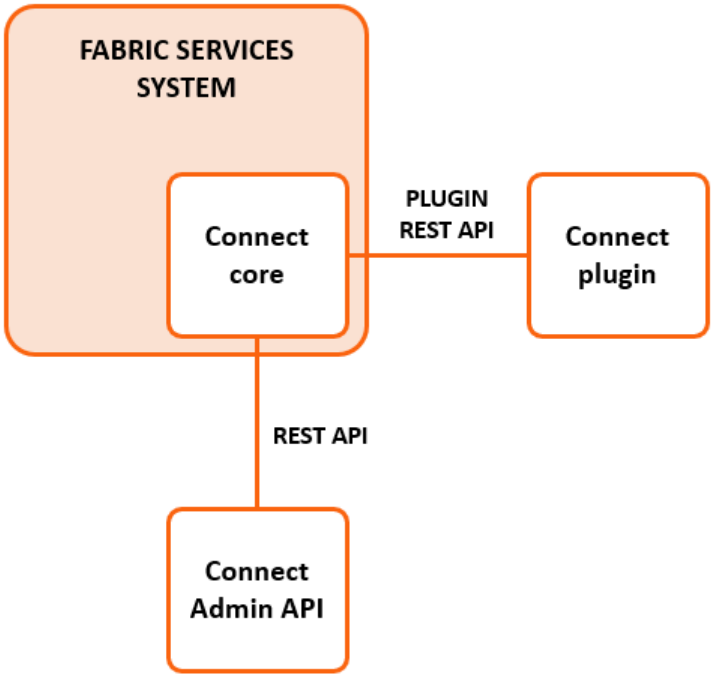
- the Connect service
- Connect plugins

The Connect service is a microservice with a REST API running inside the Fabric Services System environment. It is responsible for learning, through LLDP, how the cloud compute servers are connected to the fabric. It is also responsible for configuring the fabric for the different workloads of the cloud environment.

Connect plugins are responsible for learning on which compute servers and networks the various workloads in the cloud environment are scheduled, and passing that information to the Connect service.

This design allows plugins to be simple, handling only the integration with the cloud environment. All complexity pertaining to the fabric is hidden and centralized in Connect.

Figure 1: Fabric Services System Connect components



Connect overview

Connect is the component responsible for translating information that originates from plugins about cloud computes into information relating to the fabric.

Plugins overview

Connect plugins are specifically made to inspect one type of cloud environment. While these plugins can be developed specifically targeting a custom cloud environment, Connect comes with three Nokia supported plugins:

- the Connect Kubernetes Plugin
- the Connect OpenStack Plugin
- the Connect VMware Plugin

Feature overview

Connect supports the following features:

- Creating Layer 2 workload intents and workload subnets on the Fabric Services System (translating these to Tenant and Subnet in the Connect REST API).
- Automatically discovering the cloud compute resources from the fabric using LLDP.
- Automatically resolving inconsistent states between Connect and the fabric by performing an audit between Connect and the Fabric Services System.
- Using pre-existing LAGs in the fabric.
- Using the cloud management's standard network management tools to manage the fabric transparently.

- Using workloads managed by the Fabric Services System; this is the case in which an operator provisions a workload intent and workload subnets in the Fabric Services System before adding them to Connect through a tenant and subnet.

2 Compute requirements

In order for LLDP topology discovery from the fabric to function correctly, all computes must have LLDP enabled on their data interfaces. Only when LLDP is enabled on all computes can workloads be automatically scheduled on the fabric.

Both the OpenStack OSPD/CBIS and Vmware integration enable this for you. Manual intervention is needed when not using those integrations.

- On a Linux-based compute, enable LLDP:

```
NIC_DIR="/sys/class/net"
for itf in $(ls $NIC_DIR | grep -E 'eno|enp|ens')
do
    if [ -d "${NIC_DIR}/${itf}/device" -a ! -L "${NIC_DIR}/${itf}/device/physfn" ]
    then
        lldptool set-lldp -i $itf adminStatus=Tx
        lldptool -T -i $itf -V sysName enableTx=yes
        lldptool -T -i $itf -V portID subtype=PORT_ID_INTERFACE_NAME
    fi
done
```

- On VMware-based systems, ensure that LLDP is enabled to both send and receive LLDP in the Discovery protocol settings of the distributed vSwitch.
- For NICs that support hardware based LLDP (in-nic LLDP), make sure to disable this capability as it may send out conflicting LLDP information compared to what Linux or VMware would send out.
- The hostname returned by `hostnamectl` should correspond to hostname in output of `openstack host list`.
  If these are not the same, use the command `hostnamectl set-hostname` to align them.

3 The Connect core

The Connect core includes two key components:

- plugins
- deployments

Plugins

Plugins are a core component of the Fabric Services System Connect environment. In the Connect environment, a plugin represents the component that communicates with the external cloud services. The following plugins are supported by the Fabric Services System platform, and are further documented in their respective sections:

- OpenShift & Kubernetes plugin
- OpenStack plugin
- VMware plugin

Plugins are automatically registered within the Connect service when they are deployed. Each is stored in the database with the following main properties:

Table 1: Plugin properites

Property	Description	Values/ Range
Name	The name of the plugin.	String
Type	The type of plugin, related to the platform it supports.	String
External ID	An optional field to store an external reference.	String
Heartbeat support	Indicates whether the plugin sends regular heartbeat messages to signal its live state to Connect.	True/False
Heartbeat interval	Indicates how often the plugin should senda heartbeat message, in seconds.	Integer
Status	Indicates whether Connect: • is in the process of deploying a resources • has finished deploying it • has encountered an error during deployment	String

For more details about the full set of available properties, see the API documentation as described in the *Fabric Services System API Integration Guide*.

Heartbeat

When plugins register into the Connect core service, they can indicate that they support heartbeats. When a plugin supports heartbeats, the plugin is expected to send a heartbeat to the Connect core service at an

interval of the configured value (or more frequently). If the Connect core does not receive a heartbeat from the plugin after two intervals, it will raise an alarm in the Fabric Services System to indicate that there might be an issue with the plugin.

Deployments

Deployments represent the individual cloud environments that each plugin integrates with. For most plugins, there will only be a single deployment. The VMware plugin supports the integration of a single plugin with multiple vCenter servers, where each will be represented by a separate deployment.

Deployments have an Admin state, which indicates whether the plugin is allowed to make changes in the fabric for that deployment. This helps to prevent unwanted changes from plugins or deployments that have not been enabled, and are therefore in an Admin Down state.

Deployment states
A plugin can automatically create the deployments to which it belongs. Such deployments are created in an Admin Down state, and an administrator must update the deployment to an Admin Up state to enable it.

Deployments can also be created by an administrator, in which case the administrator can immediately enable the Deployment by setting it to Admin up.

Deployment properties
Deployments have the following main properties:

Table 2: Deployment properties

Property	Description	Values/ Range
Name	The name of the deployment	String
Description	A description of the deployment	String
Plugin	A reference to the plugin that owns the deployment	String
External ID	An optional field to store an external reference	String
Status	Indicates whether Connect: • :is in the process of deploying a resource • has finished deploying it • encountered an error occurred during deployment .	String
Settings	A collection of settings that depend on the type of the plugin. For a deployment of the VMware plugin, the settings will include information about the vCenter hostname, user name, and password, which are securely stored.	List

3.1 The Connect UI

The Fabric Services System UI contains a page from which you can manage the Connect deployments and view details about the Connect plugins.

The first page of the Connect UI displays a list of the Connect deployments and the relevant information for each deployment.

Clicking on the More menu for a deployment opens an action list from which you can:

- change the Admin state of the plugin in a single click
- open the deployment for more details
- delete the deployment.

You can also use the Views drop-down list to open the Plugins view. This opens a new page that lists Connect plugins and displays the relevant information for each plugin.

Clicking on the More menu for a plugin opens an action list from which you can:

- view a list of the deployment associated with that plugin
- delete the plugin

.

3.1.1 Connect UI parameters

Basic deployment parameters

The following parameters are used when managing a Deployment's configuration in the Fabric Services System UI. The set of parameters will vary depending on the type of plugin the Deployment is associated with.

Table 3: Basic deployment parameters

Parameter	Description	Values/Range
Admin Up	Indicates whether the deployment should be Administratively Up, or (if false) left Administratively Down. A deployment that is Administrtively Down is not functional.	True False
Name	The name used to refer to this deployment within the Fabric Services System.	A string value
Description	Optionally, a description for this deployment.	A string value

Deployment parameters based on plugin choice

The following parameters are used in the Connect page of the Fabric Services System UI to configure a Connect deployment associated with a Connect plugin.

The set of parameters can vary from one type of plugin to another.

Table 4: Deployment parameters based on plugin choice

Parameter	Open Shift	Open Stack	VMware	Description	Values/Range
Plugin	Yes	Yes	Yes	The type of plugin associated with this deployment.	OpenShift OpenStack VMware
Host	No	No	Yes	The vCenter host.	A valid host URL, in the form (for example) vmware.example.net.
Username	No	No	Yes	The username for the v Center user.	A string value, subject to vCenter constraints for user names.
Password	No	No	Yes	Password for the v Center user. **Note:** The password characters are obscured for security purposes.	A string value, subject to vCenter constraints for passwords.
TLS Verify	No	No	Yes	Indicates whether to verify TLS with vCenter.	True False
Certificate	No	No	Yes	If TLS Verify is set to True, a TLS certificate is required here. Only a single certificate is supported for a plugin. Failing to provide a certificate, or providing an invalid certificate, triggers an certificate validation failure alarm.	A valid TLS certificate

3.1.2 Creating Connect deployments and plugins

About this task

Follow this procedure to create a new Connect deployment using the Connect page of the Fabric Servcies System GUI.

For the coresponding process using the API instead of the GUI, see Managing plugins and deployments.

Procedure

Step 1. From the main Fabric Services System menu, select **Connect**.

Expected outcome

The Connect page displays, showing a list of all current Connect deployments.

Step 2. Click **CREATE DEPLOYMENT**.

Step 3. Enter the following information about the deployment as described in Basic deployment parameters:

- **Admin Up**
- **Name**
- **Description**

Step 4. Select a value for the **Plugin** parameter.

Step 5. Enter information about the associated plugin as described in Deployment parameters based on plugin choice:

Note: The remaining parameters may vary depending on the plugin you specified in step 4.

Step 6. Click **SAVE**.

3.1.3 Managing Connect deployments and plugins

About this task

Follow these steps to view, modify, or delete a Connect deployment, or to view a list of plugins or delete a plugin.

Procedure

Step 1. From the main menu, select **Connect**.

Expected outcome

The Connect page displays, showing a list of all current Connect deployments.

Step 2. Choose one of the following:

- To view details about a particular deployment, go to step 3.
- To change the administrative state of a deployment, go to step .
- To modify details about a deployment, go to step 5.
- To delete a deployment, go to step 6.
- To view a list of plugins, go to step 7.
- To delete a plugin, go to step 8.

Step 3. To view a deployment, do the following:

a. Find the deployment that you want to view and click ⋮ at the end of its row.

b. Select **Open** from the list of actions.

Details about the selected deployment are displayed in an overlay.

Step 4. To change the administrative state of a deployment, do the following:

a. Find the deployment in the list and click ⋮ at the end of its row.

b. Select **Set to Admin Up** or **Set to Admin Down** from the list of actions.

Step 5. To edit a deployment, do the following:

a. Find the deployment that you want to edit and click ⋮ at the end of its row.

b. Select **Edit** from the list of actions.

c. Modify the displayed properties as required.

Note: Not all properties can be modified.

d. Click **SAVE**.

Step 6. To delete a deployment, do the following:

a. Find the deployment that you want to delete and click ⋮ at the end of its row.

b. Select **Delete...** from the list of actions.

c. Click **OK** in the confirmation dialog.

Step 7. To view a list of plugins select Plugins from the **Views** drop-down list at the top of the page.

Expected outcome

The list of deployments is replaced by a list of current plugins.

Step 8. To delete a plugin, do the following:

a. Select Plugins from the **Views** drop-down list at the top of the page.

b. Find the plugin that you want to delete and click ⋮ at the end of its row.

c. Select **Delete...** from the list of actions.

d. Click **OK** in the confirmation dialog.

3.2 Fabric Services System Connect workflows

In the Cloud Management mode, Connect creates a workload intent for each tenant, and a Fabric Services System subnet for each subnet that is created in the Cloud Management system. In this mode, the changes in the Cloud Management system are transparently reflected into the Fabric Services System. The administrator of the Cloud Management system does not require any knowledge about how to use the Fabric Services System.

For more advanced use cases, another type of workload intent or Fabric Services System subnet might be required. In other advanced use cases some external peering must be configured with the workload intent, or special sub-interfaces are required.

In such cases Nokia recommends using the Fabric Services System Managed mode, which instructs Connect to associate tenants and subnets with existing workload intents and subnets in the Fabric Services System respectively, instead of creating these resources in the Fabric Services System based on the cloud management networking.

In this mode, an administrator (or orchestration engine) with knowledge of the Fabric Services System first creates the necessary resources in the Fabric Services System directly. They can create more complex configurations than the cloud management system itself would be able to do. When creating the networking constructs in the Cloud Management system, the administrator provides a set of unique identifiers referring to those pre-created networking constructs. This way the Connect plugin and Connect service know not to create their own Workloads and Subnets, but to use the pre-created items.

See also:

- Using the Fabric Services System Managed mode
- OpenStack usage for Fabric Services System managed networks
- VMware Usage for Fabric Services System managed networks

3.3 Managing the Connect core user

About this task

Connect uses a specific pre-created Connect user to access the Fabric Services System through an internal REST API. It is not necessary to change the password of this Connect user. However, if you do change the password of this user through the UI or API of the Fabric Services System, you must perform the following procedure which includes updating the Connect pod in the Fabric Services System Kubernetes cluster.

.

Procedure

Step 1. Obtain the base64 encoding value of the new password:

```
$ echo -n 'NewPassword' | base64
Tm9raWFDuZWN0MSE=
```

Step 2. Set the password with a new base64 encoded value in the Kubernetes secret file using following command and save the file:

```
$ kubectl edit secrets
                    dev-fss-connect-auth-secret
```

Expected outcome

Upon executing the above command you will find the following section in the file, which must be updated:

```
data:
  password: <New base64 encoded value>
```

Step 3. Delete the Connect pod so that Connect uses updated secret values to communicate with rest of the Fabric Services System services.

4 The OpenShift and Kubernetes plugin

The Fabric Services System integrates with OpenShift to provide fabric-level application networks for OpenShift pods and services. The Connect integration leverages the OpenShift Multus CNI solution to support managing the fabric directly from OpenShift and make the fabric dynamically respond to the networking needs of the application.

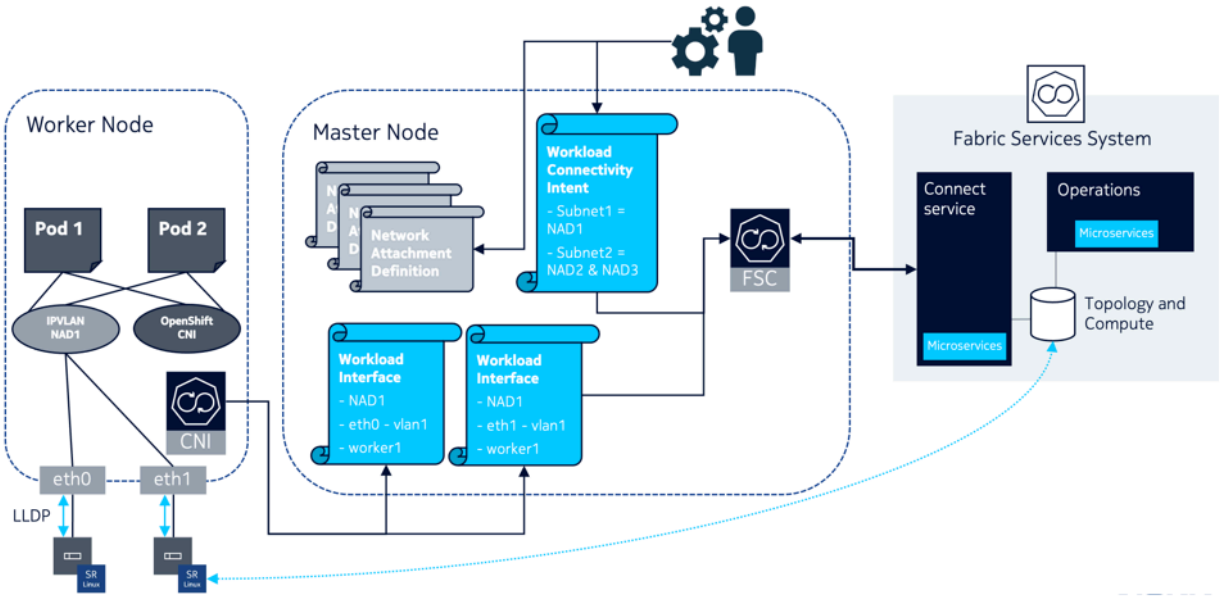
It provides the following advantages and capabilities:

- direct integration into the network management workflow of OpenShift
- use of the common CNIs used by Enterprise applications and CNFs like IPVLAN and SR-IOV
- automatic provisioning of the fabric based on where the application pods need the connectivity
- support for advanced workflows

4.1 Architecture

The Fabric Services System introduces some new components in an OpenShift environment to allow the management of the SR-Linux-based fabric using OpenShift. This section describes these components.

Figure 2: OpenShift architecture



The Fabric Services System Kubernetes Controller

The Fabric Services Kubernetes Controller (FSC) is a controller that is deployed in the master nodes, and allows the configuration of a fabric using the Connect service. It is responsible for monitoring the networking configuration of OpenShift, including the management of:

- Network Attachment Definitions
- Workload Connectivity Intents
- Workload Interfaces

The FSC uses Network Attachment Definitions and automatically updates Network Attachment Definitions with the information needed for the Fabric Services System CNI to function properly.

The FSC also monitors the creation of the Workload Connectivity Intents custom resource; and, based on the information stored in those Workload Connectivity Intents, creates the appropriate workload VPN intents and subnets inside the Fabric Services System. This allows the management of application networks through OpenShift.

Finally, the FSC monitors the Workload Interface custom resource and uses the information they store to create the appropriate sub-interfaces in the Fabric Services System. This provides the applications with connectivity to the subnets configured in the Workload Connectivity Intents.

The Fabric Services System Kubernetes Helper CNI

The Fabric Services System Kubernetes Helper CNI is a CNI that does not manipulate or change anything in the networking configuration of the pod or the worker node.

Its purpose is to learn about the relationship between a pod, a worker node the pod is running on, the Network Attachment Definition, and the physical interfaces used by that Network Attachment Definition.

When a pod is scheduled on a specific worker node, Kubelet will execute Multus for the networking of the pod. Multus in turn will look at the annotations of the pod to determine the Network Attachment Definitions to which the pod needs connectivity. When Multus processes these Network Attachment Definitions, it first executes the CNI mentioned in the Network Attachment Definition, which configures the pod networking. After that, Multus also executes the Fabric Services System Kubernetes Helper CNI.

When this CNI is executed, it learns:

- the hostname of the worker node it is being executed on
- the Network Attachment Definition for which it is being executed
- the master interface of that Network Attachment Definition
- the physical interface or interfaces and the VLAN used by that master interface

The CNI then makes sure that a Workload Interface custom resource for each interface is present in the Kubernetes API for the combination of that information (Hostname, Network Attachment Definition name and VLAN).

This Workload Interface in turn triggers the FSC to provision the appropriate sub-interfaces in the correct subnet in the Fabric Services System.

Workload Connectivity Intent CRD

The Workload Connectivity Intent (WCI) is a custom resource definition that the FSC registers in the Kubernetes API.

It is used to describe the relationship between Network Attachment Definitions and how they need to be connected to each other.

It allows the connection of multiple Network Attachment Definitions into the same subnet inside the Fabric Services System, and makes it possible to combine different types of Network Attachment Definitions into a single Layer-2 VRF (MAC-VRF).

Workload Interface CRD

The Workload Interface (WLI) is a custom resource definition that the FSC registers in the Kubernetes API.

It instructs the FSC to create the appropriate sub-interfaces for each combination of:

- worker node hostname
- VLAN
- Network Attachment Definition

The FSC uses this information to make sure the above combination is configured properly as sub-interface in the subnet in the Fabric Services System that is associated with that Network Attachment Definition. That association is learned through the Workload Connectivity Intent that references the Network Attachment Definition.

4.2 Installation

The installation of the Fabric Services System integration for OpenShift is performed using the Helm charts that are provided as part of each release of the Fabric Services System.

Package information

The Fabric Services System integration for OpenShift is provided as a tar ball (for example: fsc-v23.4.0-13.tar.gz) which contains the following files:

- `fsc-*-<release_tag>-images.tar`: The container images for the FSC Version $src_tag.
- `fsc-charts-<release_tag>.tgz`: A generic Helm package for FSC installation.
- `fsc-installer.sh`: A utility to store the container images and the charts in registries.

where <release-tag> represents the release version (such as v23.4.0-13).

Using the installer script to push the container images and charts

You can use the fsc-installer.sh script to push the container images to a container image registry and, optionally, push the Helm charts to a Helm repository.

Note: Currently, only an upload to a container image registry that requires authentication is supported.

The script accepts the following information:

- **User ID**: The user name to connect to the container image registry. (required)
- **Registry URL**: Path to the container image registry. (required)
- **Helm repository URL**: URL to the Helm repository to where the charts need to be uploaded. (optional).

The script can be run with the following command:

```
# ./fsc-installer.sh -u <user-id> -r <registry-url> -e <helm-repo>
```

For example:

```
# ./fsc-installer.sh -u imageuploader -r registry.domain.tld/fsc -e http://helm-
repo.domain.tld/fsc
```

4.2.1 Installing the OpenShift and Kubernetes Plugin

About this task

This procedure describes the installation and configuration of the OpenShift and Kubernetes plugin as might be performed on a baremetal OKD cluster version 4.10.
Some steps in this procedure are further illustrated with sample configurations in OpenShift supporting objects and examples

Prerequisites

1. Create a local values file as described in Sample local.yaml file. At minimum this file must contain:
   - the "dockerConfig" secret to access the container registry
   - the Fabric Services System server information
2. On the Top Of Rack node ports connected to the Kubernetes nodes, enable the following:
   - VLAN tagging
   - LLDP at the port and system levels
3. On the Kubenetes Linux servers, do the following:
   - enable LLDP
   - configure LLDP to advertise interface name
4. Pre-install the following on all nodes in the Kubernetes cluster:
   - CNI for Multus
   - IPVLAN
   - MACVLAN
   - IPAM
   - SR-IOV
   - SR-IOV device plugin
5. When using IPVLAN or MACVLAN, configure VLAN interfaces on the Linux system.

   Note: This VLAN interface is used as the master interface in the Network Attachment Definition (NAD). A common name for VLAN interfaces can be configured on all worker nodes so that Pods scheduled on that worker node can be use that interface. However, Pods can be scheduled on specific worker nodes using Node selector and master interfaces referred to in the Network Attachment Definition must be present on those worker nodes for Pods to come up correctly.

6. For SR-IOV, while using Virtual Function (VF), VLANs should not be present on the VF to be used by the Network Attachment Definition. These will be automatically configured when the NAD is deployed in a Pod. While using SR-IOV, only one Pod should be configured per VF.

7. For SR-IOV, while using the Port function (PF) for IP VLAN, configure VLAN interfaces before using them in the NAD and Pod deployment.

Procedure

Step 1. Add and update the helm repo, then run the installation script with the following commands:

```
# helm repo add <repoid> <repo URL> --username <username> --password <password>
# helm repo update
# helm install <RELEASE NAME> [-f <overrides file name>] <complete path to the chart>
 [--dry-run]
```

For example:

```
# helm repo add fsprepo https://artifacts.gitlabsr.nuq.ion.nokia.net/repository/fsp-
charts/ --username "fsp-charts-ro" --password  "******"
  "fsprepo" has been added to your repositories
# helm repo update
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "nfs-subdir-external-provisioner" chart
 repository
...Successfully got an update from the "traefik" chart repository
...Successfully got an update from the "fsprepo" chart repository
Update Complete. ??Happy Helming!??

# helm install prod fsprepo/fsc-charts --version v23.4.0-13 -f local.yaml
```

Step 2. Apply a Network Attachment Definition.

Note: In the local values file, the value of the injectCni parameter can be either true or false.

- the default value is true, which enbles automatic CNI injection.
- if the value is false, you must specify FSC-CNI plugin information in the NetworkAttachmentDefinitions that are referenced by WCI and require automatic fabric connectivity.
- For examples of NAD files, see REFERENCE.

For examples of NAD files, see Network Attachment Definitions (NADs),

Step 3. Apply a WorkloadConnectivityIntent (WCI) using the command `kubectl apply -f <file-name>`

Applying WCI results in the following:

- it creates tenants and subnets in the Fabric Services System
- it automatically injects fsc-cni plugin information in the NAD
- it creates a HostPortLabel for each NAD

Note: All NADs referenced by the WCI must be applied before WCI can be applied. Also, NADs referenced by the WCI should be unique across subnets and tenants. NADs should not be deleted before WCI deletion. Only one WCI can be created per tenant, and the tenant name must be unique.

For examples of WCI files, see WorkloadConnectivityIntent (WCI) examples

Step 4. Verify status in the Fabric Services System. Confirm that:

- a workload VPN intent has been created in the Fabric Services System for the WCI with tenant description as `WCI metadata.name`
- subnets with the name `spec.subnets.name` have been configured for this workload VPN intent (tenant)

Step 5. Configure Pod deployment with NADs in Annotations. Apply Pod deployment using the command `kubectl apply -f <file-name>`.

Note: Single or Multiple Annotations can be specified per Pod/NAD name

For examples of Pod deployments, see Pod configuration

Step 6. Verify the creation of objects related to the Workload Interface Verification (WLI) using `kubectl get workloadinterfaces.fsc.fss.nokia.com -n fsc-system`.

Upon Pod deployment, a WorkloadInterface (WLI) is created in the namespace “fsc-system” per worker node, per NAD on which Pods are deployed.

For examples of WLI verification, see Workload Interface (WLI)

Step 7. In the Fabric Services System, verify that sub-interfaces have been created with the specified VLAN (as in the NAD master-interface for NAD configured with Pod as Annotation) per NAD, per worker node..

Step 8. Validate data paths.

a. Log into the shell of one of the Pods and ping the other pod. THe ping should succeed.

b. Check the statistics of Top of Rack node using `show interface ethernet-1/<port-id> detail` and verify that the Tx and Rx statistics for sub-interface increments.

What to do next

Logging information is available for the FSC controller and FSC-CNI.

- FSC-Controller Logs are available at `/var/log/fsc-data/logs/ fsc-controller-manager.log` on master nodes

```
[root@master3 logs]# pwd
/var/log/fsc-data/logs
root@master3 logs]# ls -lrt
-rw-r--r--. 1 root root 6387577 Jul 28 06:46 fsc-controller-manager.log
```

- FSC-CNI Logs are available at `/var/log/fsc-cni.log` on worker nodes

```
[root@worker1 log]# ls -lrt
-rw-r--r--.  1 root        root            30881095 Jul 28 06:46 fsc-cni.log
[root@worker1 log]#
[root@worker1 log]# pwd
/var/log
```

4.2.2 OpenShift supporting objects and examples

Samples provided

This topic includes descriptions and examples of the following:

- Sample local.yaml file
- Network Attachment Definitions (NADs)
- WorkloadConnectivityIntent (WCI) examples
- Pod configuration
- Workload Interface (WLI)

Helm chart override and default values

This section describes:

- Helm chart values
- Helm chart default values

The Helm charts for the Fabric Services System integration for OpenShift includes the following overridable properties.

```
fss-fsc:
  image:
    repository: <image repository>
    pullPolicy:
    tag: <image tag>
    mgrImageName: <name of fsc pod controller image>
    cniImageName: <name of fsc cni image>
    certImageName: <name of fsc certmgr image>

  global:
    openShift: <boolean - whether the cluster is openShift based, For Rel 23.4 this is the only
 option>

  fscInfo:
    dockerConfig: <base64 encoded secret for accessing the container registry>

  cniInfo: <Specify values for FSC CNI operation>
    log:
      level: <trace, debug, info, warning, error, fatal, panic>
      genFile: <boolean - true, false>
      maxAge: <integer - Duration to persist the log files in days>
      maxBackup: <integer - Number of log files to be persisted>
      maxSize: <integer - Size of log file in MB>
      path: <The file name and path for the logs on the container>
    injectCni: <boolean - Specify true, false, whether the fsc-cni is to be added into the NAD
 definition automatically or not. Default is true >
    maxUnavailable: <integer - specify the maximum number of FSC CNI  DaemonSet pods that can
 be unavailable during an update>
  fssInfo:
    hostName: <hostname of the machine where FSS is running>
    ipAddr: <IP address of the host where FSS is running>
    userId: <User id to connect to the FSS>
    password: <Password to connect to the FSS>
    tlsEnable: <boolean - Whether the connection to FSS uses TLS>
    pluginId: <The Unique id across FSC plugins talking to the same FSS>
    pluginName: <User identifiable name for the plugin>
    deploymentName: < User identifiable name for the deployment, Max length is 79 characters>
    deploymentDescription: <Deployment description>
    tlsSkipVerify: <boolean - true/false - Whether to skip the verification of TLS certificates
 - valid only if tlsEnable is true)>
    tlsCertData: <TLS Certificate data – valid only if tlsEnable is true>
    heartbeatInterval: <The value in seconds for the keepalives between FSC plugin and FSS,
 recommended range 3 – 10 secs>
```

```
    supportsHeartbeat: <boolean - Supports generating alarms on FSS on plugin reachability when
 enabled, takes true/false>
    actionables: <FSS notifications that needs to be acted upon by FSC plugin. Allowed values
 are “DEPLOYMENT_UPDATED,AUDIT_REQUESTED”. DEPLOYMENT_UPDATED actionable support enables FSC to
 be aware of Deployment admin state (Admin Up or Admin Down) in Fabric Services System. AUDIT_
REQUESTED actionable support enabled on-demand audit for the plugin in the Fabric Services
 System.
  mgrInfo:
    log:
      level: <trace, debug, info, warning, error, fatal, panic>
      genFile: <boolean - true, false>
      maxAge: <integer - Duration to persist the log files in days>
      maxBackup: <integer - Number of log files to be persisted>
      maxSize: <integer - Size of log file in MB>
      path: <The file name and path for the logs on the container>
    connMapSize: <integer - Optimize memory utilization by providing the peak value of the
 number of concurrent pod interfaces created>
```

The default values for the helm chart properties, where <helm release name> release name given during helm install, are:

```
fss-fsc:
  global:
    openShift: true

  fscInfo:
    dockerConfig: ""

  cniInfo:
    log:
      level: info
      genFile: true
      maxAge: 7
      maxBackup: 3
      maxSize: 100
      path: /var/log/fsc-cni.log
    injectCni: true
    maxUnavailable: 3

  fssInfo:
    hostName: "fss.nokia.com"
    ipAddr: 127.0.0.1
    userId: ""
    password: ""
    tlsEnable: true
    tlsSkipVerify: true
    pluginId: "k8s-plugin-id"
    pluginName: "k8s-plugin-name"
    deploymentName: "k8s-deployment-name"
    deploymentDescription: "k8s connect deployment"
    tlsCertData: ""
    heartbeatInterval: 3
    supportsHeartbeat: true
    actionables: |
      DEPLOYMENT_UPDATED
      AUDIT_REQUESTED

  mgrInfo:
    log:
      level: info
      genFile: true
      maxAge: 7
      maxBackup: 3
```

```
      maxSize: 100
      path: /fss/data/fsc-data/logs/fsc-controller-manager.log
    connMapSize: 750
```

Sample local.yaml file

In the following sample local.yaml file, the dockerConfig is the base64 encoded pull secret for downloading the FSC container images from the registry.

```
fss-fsc:
  image:
    repository: registry.gitlabsr.nuq.ion.nokia.net/sr/linux/fsp/fsc
    pullPolicy: IfNotPresent
    tag: v23.4.0-47
    mgrImageName: fsc-manager
    cniImageName: fsc-cni
    certImageName: fsc-cert
  global:
    openShift: true
  fscInfo:
    dockerConfig: ewoJImF1dGhz
IjogewoJCSJyZWdpc3RyeS5naXRs**********************************mxjanA0YlRKaFVHbGlkMWR4VlVKWWEz
RklRbkp4WXc9PSIKCQl9Cgl9Cn0KCg==
  cniInfo:
    log:
      level: info
      genFile: true
      maxAge: 7
      maxBackup: 3
      maxSize: 100
      path: /var/log/fsc-cni.log
    injectCni: true
    maxUnavailable: 3
  fssInfo:
    hostName: "fss.nokia.com"
    ipAddr: 127.0.0.1
    userId: ""
    password: ""
    tlsEnable: true
    tlsSkipVerify: true
    pluginId: "k8s-plugin-id"
    pluginName: "k8s-plugin-name"
    deploymentName: "k8s-deployment-name"
    deploymentDescription: "k8s connect deployment"
    heartbeatInterval: 3
    supportsHeartbeat: true
    actionables: |
      DEPLOYMENT_UPDATED
      AUDIT_REQUESTED
  mgrInfo:
    log:
      level: info
      genFile: true
      maxAge: 7
      maxBackup: 3
      maxSize: 100
      path: /fss/data/fsc-data/logs/fsc-controller-manager.log
    connMapSize: 750
```

Network Attachment Definitions (NADs)

A Network Attachment Definition (NAD)

The following NAD includes the FSC-CNI plugin information.

```
[fsc-helper@blrfsctb01-fsc-helper default]$ cat defnad9port2_cni.yaml
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: def-nad9-port2
spec:
  config: '{
     "cniVersion": "0.3.1",
     "name": "def-nad9-port2",
     "plugins": [
       {
         "type": "ipvlan",
         "master": "fscintf2.2709",
         "mode": "l2",
         "ipam": {
           "type": "whereabouts",
           "range": "29.1.1.1/24",
           "gateway": "29.1.1.254"
         }
       },
       {
         "type": "fsc-cni",
         "args": {
           "parent": "default/def-nad9-port2",
           "cnicache": "/var/lib/cni/fsc-cni"
         }
       }
     ]
    }'
```

Note:

- “type” should always be “fsc-cni” for FSC plugin.
- “parent” in the fsc-cni follows the format “namespace/<network-attachment-definition-name>”. For the default namespace, it should be “default/ <network-attachment-definition-name>”
- <network-attachment-definition-name> must match the metadata.name of the NetworkAttachmentDefinition
- cnicache is the location to store the prevresult plugin config provided by multus. The configuration from this location is retrieved later for deletion purpose

The following is an example of an SR-IOV NAD:

```
[fsc-helper@blrfsctb02-fsc-helper NAD]$ cat nad-sriov_vf0_516.yaml

apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: blrone-sriov-vf0-516
  namespace: blrone
  annotations:
    k8s.v1.cni.cncf.io/resourceName: intel.com/intel_sriov_netdevice
spec:
  config: '{
  "cniVersion": "0.3.1",
  "name": "blrone-sriovnet-vf0-516",
  "plugins": [
    {
```

```
      "type": "sriov",
      "cniVersion": "0.3.1",
      "name": "blrone-sriovnet-vf0-516",
      "vlan": 516,
      "deviceID": "0000:01:03.6",
      "ipam": {
        "type": "whereabouts",
        "range": "40.9.1.1/24",
        "gateway": "40.9.1.254"
      }
    }
  ]
}'
```

The following is an example of an SR-IOV NAD that is untagged, and does not speicfy VLAN:

```
[fsc-helper@blrfsctb02-fsc-helper NAD]$ cat nad-sriov_vf0_517_untagged.yaml

apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: blrone-sriov-vf0-517-untagged
  namespace: blrone
  annotations:
    k8s.v1.cni.cncf.io/resourceName: intel.com/intel_sriov_netdevice
spec:
  config: '{
  "cniVersion": "0.3.1",
  "name": "blrone-sriovnet-vf0-517-untagged",
  "plugins": [
    {
      "type": "sriov",
      "cniVersion": "0.3.1",
      "name": "blrone-sriovnet-vf0-517-untagged",
      "deviceID": "0000:01:03.7",
      "ipam": {
        "type": "whereabouts",
        "range": "40.9.1.1/24",
        "gateway": "40.9.1.254"
      }
    }
  ]
}'
```

The following is an example of an SR-IOV NAD that specifies VLAN as 0:

```
[fsc-helper@blrfsctb02-fsc-helper NAD]$ cat nad-sriov_vf0_513_untagged.yaml
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: blrone-sriov-vf0-513-untagged
  namespace: blrone
  annotations:
    k8s.v1.cni.cncf.io/resourceName: intel.com/intel_sriov_netdevice
spec:
  config: '{
  "cniVersion": "0.3.1",
  "name": "blrone-sriovnet-vf0-513-untagged",
  "plugins": [
    {
      "type": "sriov",
      "cniVersion": "0.3.1",
      "name": "blrone-sriovnet-vf0-513-untagged",
```

```
      "vlan": 0,
      "deviceID": "0000:01:03.3",
      "ipam": {
        "type": "whereabouts",
        "range": "40.7.1.1/24",
        "gateway": "40.7.1.254"
      }
    }
  ]
}'
```

After FSC-CNI injection on a WCI Deployment, FSC-CNI plugin information is available in the Network Attachment Definition file after automatic injection, as shown in the example below.

```
[fsc-helper@blrfsctb01-fsc-helper default]$ kubectl describe network-attachment-
definitions.k8s.cni.cncf.io def-nad11-port2 -n blrtwo
Name:         def-nad11-port2
Namespace:    blrtwo
Labels:       <none>
Annotations:  <none>
API Version:  k8s.cni.cncf.io/v1
Kind:         NetworkAttachmentDefinition
Metadata:
  Creation Timestamp:  2022-07-19T13:55:40Z
  Generation:          50
  Managed Fields:
    API Version:  k8s.cni.cncf.io/v1
    Fields Type:  FieldsV1
    fieldsV1:
      f:metadata:
        f:annotations:
          .:
          f:kubectl.kubernetes.io/last-applied-configuration:

      f:spec:
    Manager:      kubectl-client-side-apply
    Operation:    Update
    Time:         2022-07-27T08:59:23Z
    API Version:  k8s.cni.cncf.io/v1
    Fields Type:  FieldsV1
    fieldsV1:
      f:spec:
        f:config:
    Manager:         fsc-manager
    Operation:       Update
    Time:            2022-07-28T06:09:20Z
  Resource Version:  18598415
  UID:               ac661ad5-44e3-47ac-a658-e099b509d2b8
Spec:
  Config:  {"cniVersion":"0.3.1","name":"def-nad11-port2","plugins":[{"ipam":
{"gateway":"27.1.1.254","range":"27.1.1.1/
24","type":"whereabouts"},"master":"fscintf2.2706","mode":"l2","type":"ipvlan"},{"args":
{"cnicache":"/var/lib/cni/fsc-cni","parent":"blrtwo/def-nad11-port2"},"type":"fsc-cni"}]}
Events:    <none>
```

Note:

- After a WorkloadConnectivityIntent (WCI) is deployed, if you reapply a NAD definition that is referenced by th WCI and that NAD definition does not include fsc-cni plugin information, it NOT injected again.

- When modifying a NAD referenced by a WCI, you must add fsc-cni plugin information in the Network Attachment Definition file and the fsc-cni plugin should be the last one. To obtain plugin information, use `Kubectl get network-attachment-definitions <NAD-Name> -o yaml”` of NAD before modification.
- The “Parent name” in the fsc-cni should match the NAD name and must be in the format `<namespace/NAD-name>` when you specify fsc-cni plugin information in the Network Attachment Definition file during modification.

The following sample shows a NAD configuration for IPVLAN, for modification after WCI deployment.

```
[fsc-helper@blrfsctb01-fsc-helper default]$ cat blrtwodefnad11port2_cni.yaml
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: def-nad11-port2
  namespace: blrtwo
spec:
  config: '{
       "cniVersion": "0.3.1",
       "name": "def-nad11-port2",
       "plugins": [
         {
           "type": "ipvlan",
           "master": "fscintf2.2706",
           "mode": "l2",
           "ipam": {
             "type": "whereabouts",
             "range": "27.1.1.1/24",
             "gateway": "27.1.1.254"
           }
         },
         {
           "type": "fsc-cni",
           "args": {
             "parent": "blrtwo/def-nad11-port2",
             "cnicache": "/var/lib/cni/fsc-cni"
           }
         }
       ]
      }'
```

WorkloadConnectivityIntent (WCI) examples

A WorkloadConnectivityIntent file includes the following information:

- Metadata.name = name of tenant to be created
- Metadata.namespace is always fsc-system
- Spec.namespace = this is not mandatory. However, if specified the all NADs must be in same namespace as spec.namespace. If this is not specified, subnets can reference NADs from any namespace
- Spec.subnets: This is list of subnets to be created for this tenant.
- Spec.subnets.cni : This provides a list of NADs that would be part of this subnet
- NAD can be specified as <NAD-name> or <namespace/NAD-name>.

  - When spec.namespace is “Not configured”, specify the NAD as <NAD_name> is considered as “default/<NAD-name>”.
  - When spec.namespace is “configured”, specify the NAD as <NAD_name> is considered as “spec.namespace/<NAD-name>”.
- Spec.type is “IRB”, for Integrated Routing and Bridging
- Spec.subnets.type: only "bridged" is currently supported

The following WCI uses the spec.namespqace "Configured".

```
[fsc-helper@blrfsctb01-fsc-helper default]$ cat copy_blrtwointentNS.yaml
apiVersion: fsc.fss.nokia.com/v1
kind: WorkloadConnectivityIntent
metadata:
  name: blrtwotenant10
  namespace: fsc-system
spec:
  namespace: blrtwo
  type: "IRB"
  subnets:
   - name: "blrtwot1sub1"
     type: "bridged"
     cni:
       - "def-nad10"
       - "blrtwo/def-nad9"
   - name: "blrtwot1sub2"
     type: "bridged"
     cni:
       - "blrtwo/def-nad9-port2"
       - "def-nad10-port2"
   - name: "blrtwot1sub4"
     type: "bridged"
      cni:
       - "blrtwo/def-nad11-port2"
        - "def-nad11"
```

The following WCI uses the spec.namespqace "Not Configured".

```
[fsc-helper@blrfsctb01-fsc-helper default]$ cat defintent_multiNamespace.yaml
apiVersion: fsc.fss.nokia.com/v1
kind: WorkloadConnectivityIntent
metadata:
  name: deftenant1
  namespace: fsc-system
spec:
  type: "IRB"
  subnets:
   - name: "deft1sub1"
     type: "bridged"
     cni:
       - "def-nad1"
       - "def-nad2"
       - "def-nad3"
       - "def-nad4"
       - "def-nad5"
       - "def-nad9"
       - "default/def-nad6"
       - "blrone/blrone-nad2"
       - "blrone/blrone-nad1"
       - "def-nad9-port2"
   - name: "deft1sub2"
```

```
    type: "bridged"
    cni:
      - "def-nad1-port2"
      - "def-nad2-port2"
      - "def-nad3-port2"
      - "def-nad4-port2"
      - "def-nad5-port2"
      - "default/def-nad6-port2"
      - "default/def-nad7-port2"
      - "blrone/blrone-nad1-port2"
      - "blrone/blrone-nad2-port2"
      - "blrtwo/blrtwo-nad6"
      - "default/def-nad7"
  - name: "deft1sub3"
    type: "bridged"
    cni:
      - "blrtwo/blrtwo-nad6-port2"
      - "blrtwo/blrtwo-nad7-port2"
      - "blrtwo/blrtwo-nad7"
      - "default/def-nad10-port2"
      - "default/def-nad10"
  - name: "deft1sub4"
    type: "bridged"
    cni:
      - "default/def-nad8-port2"
       - "default/def-nad8"
```

You can view the workload connectivity intent contents using `kubectl describe workloadconnectivityintents.fsc.fss.nokia.com blrtwotenant10 -n fsc-system`

WCI output after WCI deployment includes the following fields:

- Connectstatus: Sync-Done in this field indicates whether WCI is deployed on FSS through connect.
- Cnistatus: Connectedpods is “true” when there are Pods are available using this NAD are available. If there are no Pods using this, status is “false”.
- Crdstatus:
  - "CNI-Validation-Failed" - Failed to validate NAD presence.
  - "Queued-update-FSS" - Queued Update to FSS. FSS status will be monitored by Connect-Status.
  - "Queued-delete-FSS" - Queued delete to FSS. FSS status will be monitored by Connect Status.
- Connectstatus:
  - “Sync-Done” in this field indicates whether NAD is deployed in FSS through connect.
  - "Reg-Failed" in this field indicates FSS registration had failed. There is no retries to connect to FSS.
  - "Sync-Pending" in this field indicates request has been send to FSS to create resources however response if yet to be received. Retry will be done to achieve Sync-done with FSS.
  - "Sync-Deleted" in this field indicates request to delete resources has been sent to FSS and successful response is received.
  - "Sync-Mark-Delete " in this field indicates request to delete resources has been sent to FSS but response is yet to be received
  - "Sync-Add-Failed " in this field indicates request to add or create resources has been sent to FSS and failure is received from FSS for some reason. No retries are done in this case.
  - "Sync-Del-Failed " in this field indicates request to delete resources has been sent to FSS and failure is received from FSS for some reason. No retries are done in this case.

The sample below shows the output of a workload connectivity intent query after WCI deployment.

```
[fsc-helper@blrfsctb01-fsc-helper default]$ kubectl describe
 workloadconnectivityintents.fsc.fss.nokia.com blrtwotenant10 -n fsc-system
Name:         blrtwotenant10
Namespace:    fsc-system
Labels:       <none>
Annotations:  <none>
API Version:  fsc.fss.nokia.com/v1
Kind:         WorkloadConnectivityIntent
Metadata:
  Creation Timestamp:  2022-07-28T06:09:19Z
  Finalizers:
    fsc.io/gWCFinalizer
  Generation:  2
  Managed Fields:
    API Version:  fsc.fss.nokia.com/v1
    Fields Type:  FieldsV1
    fieldsV1:
      f:metadata:
        f:finalizers:
          .:
          v:"fsc.io/gWCFinalizer":
      f:spec:
        f:managedid:
        f:subnets:
    Manager:      fsc-manager
    Operation:    Update
    Time:         2022-07-28T06:09:19Z
    API Version:  fsc.fss.nokia.com/v1
    Fields Type:  FieldsV1
    fieldsV1:
      f:metadata:
        f:annotations:
          .:
          f:kubectl.kubernetes.io/last-applied-configuration:
      f:spec:
        .:
        f:namespace:
        f:type:
    Manager:      kubectl-client-side-apply
    Operation:    Update
    Time:         2022-07-28T06:09:19Z
    API Version:  fsc.fss.nokia.com/v1
    Fields Type:  FieldsV1
    fieldsV1:
      f:status:
        .:
        f:connectstatus:
        f:crdstatus:
        f:subnetstatus:
    Manager:         fsc-manager
    Operation:       Update
    Subresource:     status
    Time:            2022-07-28T06:09:20Z
  Resource Version:  18598421
  UID:               cfdf73bb-ff85-45e2-aca0-0b1f3d7643e1
Spec:
  Managedid:
  Namespace:  blrtwo
  Subnets:
    Cni:
      def-nad10
    Managedid:
```

```
    Name:       blrtwot1sub1
    Type:       bridged
    Cni:
      blrtwo/def-nad9-port2
    Managedid:
    Name:       blrtwot1sub2
    Type:       bridged
    Cni:
      blrtwo/def-nad11-port2
      def-nad11
      def-nad10-port2
    Managedid:
    Name:       blrtwot1sub4
    Type:       bridged
    Cni:
      blrtwo/def-nad9
    Managedid:
    Name:       blrtwot1sub5
    Type:       bridged
  Type:         IRB
Status:
  Connectstatus:  Sync-Done
  Crdstatus:      Queued-update-FSS
  Subnetstatus:
    Cnistatus:
      Connectedpods:  false
      Connectstatus:  Sync-Done
      Name:           def-nad10
      Namespace:      blrtwo
    Connectstatus:    Sync-Done
    Name:             blrtwot1sub1
    Cnistatus:
      Connectedpods:  false
      Connectstatus:  Sync-Done
      Name:           blrtwo/def-nad9-port2
      Namespace:      blrtwo
    Connectstatus:    Sync-Done
    Name:             blrtwot1sub2
    Cnistatus:
      Connectedpods:  false
      Connectstatus:  Sync-Done
      Name:           blrtwo/def-nad11-port2
      Namespace:      blrtwo
      Connectedpods:  false
      Connectstatus:  Sync-Done
      Name:           def-nad11
      Namespace:      blrtwo
      Connectedpods:  false
      Connectstatus:  Sync-Done
      Name:           def-nad10-port2
      Namespace:      blrtwo
    Connectstatus:    Sync-Done
    Name:             blrtwot1sub4
    Cnistatus:
      Connectedpods:  false
      Connectstatus:  Sync-Done
      Name:           blrtwo/def-nad9
      Namespace:      blrtwo
    Connectstatus:    Sync-Done
    Name:             blrtwot1sub5
Events:               <none>
```

Pod configuration

Single or Multiple Annotations can be specified per POD/NAD name

Apply a Pod/Deployment using `kubectl apply -f <file-name>`

The following is an example of a Pod with a single NAD annotation.

```
[fsc-helper@blrfsctb01-fsc-helper default]$ cat blrtwodefdepPod10_port2.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: blrtwo-def-nad10-port2-dep
  namespace: blrtwo
  labels:
    project: fsc
spec:
  replicas: 7
  selector:
    matchLabels:
      project: fsc
  template:
    metadata:
      labels:
        project: fsc
      annotations:
        k8s.v1.cni.cncf.io/networks: blrtwo/def-nad10-port2
    spec:
      imagePullSecrets:
      - name: regcred
      containers:
      - name: centos1
        imagePullPolicy: IfNotPresent
        image: centos/tools
        command: ["/bin/bash"]
        args: ["-c", "while true; do echo hello; sleep 10;done"]
        ports:
        - containerPort: 8080
```

The following is a sample Pod configuration with multiple NAD annotations

```
[fsc-helper@blrfsctb01-fsc-helper default]$ cat defdepPod7_select.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: def-nad7-8-9-10-dep-select
  labels:
    project: fsc
spec:
  replicas: 5
  selector:
    matchLabels:
      project: fsc
  template:
    metadata:
      labels:
        project: fsc
      annotations:
        k8s.v1.cni.cncf.io/networks: def-nad7, def-nad8, def-nad9, def-nad10
    spec:
      nodeSelector:
        sriovnic: present
      imagePullSecrets:
      - name: regcred
```

```
      containers:
      - name: centos1
        imagePullPolicy: IfNotPresent
        image: centos/tools
        command: ["/bin/bash"]
        args: ["-c", "while true; do echo hello; sleep 10;done"]
        ports:
        - containerPort: 8080
```

Workload Interface (WLI)

Upon Pod deployment, a WorkloadInterface (WLI) is created in the namespace “fsc-system” per worker node, per NAD on which Pods are deployed. The WLI data can be obtained using the command `kubectl get workloadinterfaces.fsc.fss.nokia.com -n fsc-system`

For example:

```
[fsc-helper@blrfsctb01-fsc-helper default]$ kubectl get workloadinterfaces.fsc.fss.nokia.com -n
 fsc-system
NAME                                  AGE
Worker1.lab.fsc.io-blrtwo-def-nad10   17m
Worker2.lab.fsc.io-blrtwo-def-nad10   17m
```

A sub-interface is created on deployment of the first Pod on a worker node for each NAD. However, when deploying multiple Pods using the same NAD on the same worker node, the same WLI keeps track of all Pods sharing the sub-interface. Upon removal of a Pod from a worker node, an entry is removed from the WLI. Upon removal of the last Pod sharing a specific WLI, the WLI itself is deleted for that worker node and a deletion message to delete a sub-interface is sent to the Fabric Services System.

To obtain details about the pods using a WLI, use the command `kubectl describe workloadinterfaces.fsc.fss.nokia.com <wli-name> -n fsc-system`

In the resulting output:

- “Sync-Done” in this field indicates successful creation of sub-interfaces through Connect.
- " Sync-Reg-Failed " in this field indicates that Fabric Services System registration has failed. No further attempts are made to connect to the Fabric Services System.
- "Sync-Pending" in this field indicates the request has been send to the Fabric Services System to create resources however response if yet to be received. Another attempt will be made to achieve Sync-done with the Fabric Services System.
- "Sync-Deleted" in this field indicates request to delete resources has been sent to the Fabric Services System and a successful response was received.
- "Sync-Mark-Delete " in this field indicates request to delete resources has been sent to the Fabric Services System but no response has been received.
- "Sync-Add-Failed " in this field indicates request to add or create resources has been sent to the Fabric Services System and failure is received from the Fabric Services System for some reason. No further attempts are made in this case.
- "Sync-Del-Failed " in this field indicates request to delete resources has been sent to the Fabric Services System and failure is received from the Fabric Services System for some reason. No further attempts are made in this case.

The following is a sample output from a workload description command.

```
[fsc-helper@blrfsctb01-fsc-helper default]$ kubectl describe
 workloadinterfaces.fsc.fss.nokia.com worker1.lab.fsc.io-blrtwo-def-nad10 -n fsc-system
Name:         worker1.lab.fsc.io-blrtwo-def-nad10
Namespace:    fsc-system
Labels:       <none>
Annotations:  fsc/metadata: {"uid":"58542c1f-4a3c-4987-9140-8d3029a05e37","creation
Timestamp":"2022-07-28T06:46:09Z"}
API Version:  fsc.fss.nokia.com/v1
Kind:         WorkloadInterface
Metadata:
  Creation Timestamp:  2022-07-28T06:46:09Z
  Generation:          1
  Managed Fields:
    API Version:  fsc.fss.nokia.com/v1
    Fields Type:  FieldsV1
    fieldsV1:
      f:spec:
        .:
        f:cni:
        f:server-interface:
          .:
          f:interface:
          f:node:
          f:vlan-end:
          f:vlan-start:
          f:vlan-type:
    Manager:      Go-http-client
    Operation:    Update
    Time:         2022-07-28T06:46:09Z
    API Version:  fsc.fss.nokia.com/v1
    Fields Type:  FieldsV1
    fieldsV1:
      f:status:
        .:
        f:connectstatus:
        f:pending-podkeys:
        f:synced-podkeys:
          .:

 f:worker1.lab.fsc.io:37721f66056e9e87038cf39a73e023b27046dabb7ec1b0a5d59dcfe6d53081a0-net1:

 f:worker1.lab.fsc.io:4df27e16cc53a660f3841a10522c02e254f722907c7be9f193965107750cca4f-net1:

 f:worker1.lab.fsc.io:c7acdb797dae90fa2318b47063493dbd24ca46c300ca7560cebdb0448c5dc74d-net1:

 f:worker1.lab.fsc.io:e21de0ba145d2576f8f1423b9d5dfece6fe399ec9db8996c05afbce837a36533-net1:
    Manager:         fsc-manager
    Operation:       Update
    Subresource:     status
    Time:            2022-07-28T06:46:09Z
  Resource Version:  18611818
  UID:               21fbed90-43fb-465f-bbfb-e5de7ee9fe9d
Spec:
  Cni:  blrtwo/def-nad10
  Server - Interface:
    Interface:     enp6s0
    Node:          worker1.lab.fsc.io
    Vlan - End:    2006
    Vlan - Start:  2006
    Vlan - Type:   VLANTYPE_VALUE
Status:
  Connectstatus:  Sync-Done
```

```
  Pending - Podkeys:
  Synced - Podkeys:
    Worker1.lab.fsc.io:37721f66056e9e87038cf39a73e023b27046dabb7ec1b0a5d59dcfe6d53081a0-net1:
 blrtwo/blrtwo-def-nad10-dep-7bd5f9f9c4-ptmnx
    worker1.lab.fsc.io:4df27e16cc53a660f3841a10522c02e254f722907c7be9f193965107750cca4f-net1:
 blrtwo/blrtwo-def-nad10-dep-7bd5f9f9c4-lzjm7
    worker1.lab.fsc.io:c7acdb797dae90fa2318b47063493dbd24ca46c300ca7560cebdb0448c5dc74d-net1:
 blrtwo/blrtwo-def-nad10-dep-7bd5f9f9c4-dg6kt
    worker1.lab.fsc.io:e21de0ba145d2576f8f1423b9d5dfece6fe399ec9db8996c05afbce837a36533-net1:
 blrtwo/blrtwo-def-nad10-dep-7bd5f9f9c4-2xvnq
Events:
<none>
```

4.3 Usage

This section lists commands available for managing the OpenShift plugin.

Helper CNI injection

If the `injectCni` is set to `true` (the default value), FSC automatically injects the Helper CNI configuration in any Network Attachment Definition that is created in the platform. This assures the correct functionality of the Fabric Services System integration.

If the injection was disabled during the installation, add the following CNI configuration to the list of plugins for each Network Attachment Definition that requires fabric management:

```
{
  "type": "fsc-cni",
  "args": {
    "parent": "<nad-namespace>/<nad-name>",
    "cnicache": "/var/lib/cni/fsc-cni"
  }
}
```

In this plugin definition, the `<nad-namespace>/<nad-name>` value must be changed to the actual namespace and name of the Network Attachment Definition. A complete example of an IPVLAN Network Attachment Definition looks like the following:

```
---
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: def-nad9-port2
spec:
  config: '{
     "cniVersion": "0.3.1",
     "name": "def-nad9-port2",
     "plugins": [
       {
         "type": "ipvlan",
         "master": "fscintf2.2709",
         "mode": "l2",
         "ipam": {
           "type": "whereabouts",
           "range": "29.1.1.1/24",
           "gateway": "29.1.1.254"
         }
       },
```

```
      {
        "type": "fsc-cni",
        "args": {
          "parent": "default/def-nad9-port2",
          "cnicache": "/var/lib/cni/fsc-cni"
        }
      }
    ]
  }'
```

Note: When `injectCni` is set to `true`, FSC only injects the Helper CNI definition in the Network Attachment Definitions configured in the WorkloadConnectivityIntent (WCI) when the WCI is applied. Also, when the WCI is deleted, FSC removed the Helper CNI definition in the Network-Attachment-Definitions configured in the WCI. If a change is applied to a Network AttachmentdDefinition that is already present in the applied WCI, it is not added again; you must inject the Helper CNI in the updated Network AttachmentdDefinition.

Defining Workload Connectivity Intent resources

A Workload Connectivty Intent contains the network design for an application. Each Workload Connectivity Intent matches with a Workload VPN Intent inside the Fabric Services System.

Below is a n overview of the definition of a Workload Connectivity Intent.

```
---
apiVersion: fsc.fss.nokia.com/v1
kind: WorkloadConnectivityIntent
metadata:
  name: app01 # A name for this Workload Connectivity Intent
  namespace: fsc-system # Should always be the fsc-system namespace
spec:
  namespace: app01 # (Optional) The namespace to find the NADs
  type: "IRB" # Should always be IRB
  subnets: # List of Subnets (MAC VRFs) to create in the fabric
   - name: "frontend" # Name of the subnet
     type: "bridged" # Should always be bridged
     cni: # List of NADs that need to connect into this subnet
       - "frontend-nad01" # A NAD name in the 'app01' namespace
       - "global-ns/frontend-shared" # A NAD in a different namespace
   - name: "backend"
     type: "bridged"
     cni:
       - "backend-nad01"
   - name: "database"
     type: "bridged"
      cni:
       - "db-nad01"
```

The above Workload Connectivity Intent results in a network design in the Fabric Services System that has a Workload VPN Intent named “app01” with three subnets: “frontend”, “backend” and “database”.

When a pod starts that refers to any of the referenced Network Attachment Definitions, the helper CNI and FSC ensure that the fabric is properly configured to provide connectivity for that pod on the specific worker node on which it is started.

Note: A Network Attachment Definition must exist before it can be referenced in a Workload Connectivity Intent, and it can only be referenced by one Workload Connectivity Intent.

Note: If you configures non-default namespace under the spec context, Network Attachment Definitions for that namespace only can be specified in the list of CNI under any subnet of that WorkloadConnectivityIntent.

Using the Fabric Services System Managed mode

The example Workload Connectivity Intent in Defining Workload Connectivity Intent resources is an example representing the use of the Cloud Managed mode (for more details, see Fabric Services System Connect workflows).

The OpenShift integration also supports the use of the Fabric Services System mode. In this scenario, the Workload Intents and Subnets are created in the Fabric Services System, without sub-interfaces, before the Workload Connectivity Intent gets created.

After the creation of the Workload Intents and Subnets in the Fabric Services System, the operator or administrator of OpenShift creates a Workload Connectivity Intent where the workload and each subnet refers to the pre-created resources by using the appropriate unique identifier (UUID).

An example of a Workload Connectivity Intent when using the Fabric Services System Managed mode:

```
apiVersion: fsc.fss.nokia.com/v1
kind: WorkloadConnectivityIntent
metadata:
  name: tenant1
  namespace: fsc-system
spec:
  type: "IRB"
  namespace: fsc-system
  fssWorkloadEvpnID: "433478043755872256" # The Workload Intent ID from FSS
  subnets:
    - name: "subnet1"
      type: "bridged"
      fssSubnetID: "433478096436330496" # The Subnet ID from FSS
      cni:
        - "nad1"
```

When using the Fabric Services System Manage mode, the following extra fields are mandatory in the Workload Connectivity Intent:

- fssWorkloadEvpnID: The ID of the Workload Intent inside the Fabric Services System that was created. It must be configured at the root of the spec of the Workload Connectivity Intent.
- fssSubnetID: The ID of a Subnet that is part of the Workload Intent referred to by the fssWorkloadEvpnID. This needs to be defined for each subnet in the Workload Connectivity Intent if a fssWorkloadEvpnID is configured.

Listing Workload Connectivity Intents

You can retrieve a list of existing Workload Connectivity Intents with the following command:

```
$ kubectl get workloadconnectivityintents.fsc.fss.nokia.com -n fsc-system
NAME    AGE
app01   17m
app02   17m
```

Inspecting a Workload Connectivity Intent

To retrieve all the details of a Workload Connectivity intent, run the following command:

```
$ kubectl describe workloadconnectivityintents.fsc.fss.nokia.com app01 -n fsc-system
Name:         app01
Namespace:    fsc-system
Labels:       <none>
Annotations:  <none>
API Version:  fsc.fss.nokia.com/v1
Kind:         WorkloadConnectivityIntent
Metadata:
  Creation Timestamp:  2022-07-28T06:09:19Z
  Finalizers:
    fsc.io/gWCFinalizer
  Generation:  2
  Managed Fields:
    API Version:  fsc.fss.nokia.com/v1
    Fields Type:  FieldsV1
    fieldsV1:
      f:metadata:
        f:finalizers:
          .:
          v:"fsc.io/gWCFinalizer":
      f:spec:
        f:managedid:
        f:subnets:
    Manager:      fsc-manager
    Operation:    Update
    Time:         2022-07-28T06:09:19Z
    API Version:  fsc.fss.nokia.com/v1
    Fields Type:  FieldsV1
    fieldsV1:
      f:metadata:
        f:annotations:
          .:
          f:kubectl.kubernetes.io/last-applied-configuration:
      f:spec:
        .:
        f:namespace:
        f:type:
    Manager:      kubectl-client-side-apply
    Operation:    Update
    Time:         2022-07-28T06:09:19Z
    API Version:  fsc.fss.nokia.com/v1
    Fields Type:  FieldsV1
    fieldsV1:
      f:status:
        .:
        f:connectstatus:
        f:crdstatus:
        f:subnetstatus:
    Manager:         fsc-manager
    Operation:       Update
    Subresource:     status
    Time:            2022-07-28T06:09:20Z
  Resource Version:  18598421
  UID:               cfdf73bb-ff85-45e2-aca0-0b1f3d7643e1
Spec:
  Managedid:
  Namespace:  app01
  Subnets:
   - name: "frontend" # Name of the subnet
     type: "bridged" # Should always be bridged
```

```
      cni: # List of NADs that need to connect into this subnet
        - "frontend-nad01" # A NAD name in the 'app01' namespace
        - "global-ns/frontend-shared" # A NAD in a different namespace
    - name: "backend"
      type: "bridged"
      cni:
        - "backend-nad01"
    - name: "database"
      type: "bridged"
       cni:
        - "db-nad01"
     Cni:
       frontend-nad01
       global-ns/frontend-shared
     Managedid:
     Name:       frontend
     Type:       bridged
     Cni:
       backend-nad01
     Managedid:
     Name:       backend
     Type:       bridged
     Cni:
       db-nad01
     Managedid:
     Name:       database
     Type:       bridged
   Type:         IRB
Status:
  Connectstatus:  Sync-Done
  Crdstatus:      Queued-update-FSS
  Subnetstatus:
    Cnistatus:
      Connectedpods:  false
      Connectstatus:  Sync-Done
      Name:           frontend-nad01
      Namespace:      app01
      Connectedpods:  false
      Connectstatus:  Sync-Done
      Name:           global-ns/frontend-shared
      Namespace:      global-ns
    Connectstatus:    Sync-Done
    Name:             frontend
    Cnistatus:
      Connectedpods:  false
      Connectstatus:  Sync-Done
      Name:           backend-nad01
      Namespace:      app01
    Connectstatus:    Sync-Done
    Name:             backend
    Cnistatus:
      Connectedpods:  false
      Connectstatus:  Sync-Done
      Name:           db-nad01
      Namespace:      app01
    Connectstatus:    Sync-Done
    Name:             database
Events:               <none>
```

Potential Status fields for a Workload Connectivity Intent

The following status values can be returned in the different status fields of a Workload Connectivity Intent:

- **Connectedpods**: Is “true” when there are Pods are running using this NAD. If there are no Pods using this NAD, the status is “false”.
- **Crdstatus**: Can have the following values:
  - "CNI-Validation-Failed" - Failed to validate the NAD presence.
  - "Queued-update-FSS" – An update is queued to the Fabric Services System and the status is monitored by the Connect status.
  - "Queued-delete-FSS" – A delete is queued to the Fabric Services System and the status is monitored by the Connect status.
- **Connectstatus**: Can have the following values:
  - **Sync-Done** indicates that a NAD is deployed in the Fabric Services System through the Connect service.
  - **Reg-Failed** indicates the registration has failed.
  - **Sync-Pending** indicates a request was sent to the Fabric Services System to create resources, but no response was received. A new attempt is made to achieve Sync-done at a regular interval.
  - **Sync-Deleted** indicates a request to delete the resources was sent to the Fabric Services System and a successful response was received.
  - **Sync-Mark-Delete** indicates a request to delete the resources was sent to the Fabric Services System, but no response was received.
  - **Sync-Add-Failed** indicates a request to add or create the resources was sent to the Fabric Services System, and a failure response was received for some reason. No further attempts are made in this case.
  - **Sync-Del-Failed** indicates a request to delete the resources was sent to the Fabric Services System and a failure response was received for some reason. No further attempts are made in this case.

Deleting a Workload Connectivity Intent

A Workload Connectivity Intent cannot be deleted if any pods are using the Network Attachment Definitions that are referenced in the Workload Connectivity Intent.

To delete a Workload Connectivity Intent, run the following command:

```
$ kubectl delete workloadconnectivityintents.fsc.fss.nokia.com app01 -n fsc-system
```

Working with Workload Interface resources

When a pod is started on a worker node, the Helper CNI creates Workload Interfaces to indicate which worker nodes, physical interfaces, and VLANs on those interfaces must be added as sub-interfaces for a specific Network Attachment Definition.

These Workload Interface resources should not be manipulated by the operator of the OpenShift cluster, and are under the full control of the Helper CNI and the FSC.

You can retrieve a list of Workload Interfaces with the following commands.

```
$ kubectl get workloadinterfaces.fsc.fss.nokia.com -n fsc-system
NAME                                      AGE
Worker1.lab.fsc.io-app01-frontend-nad01   17m
```

```
Worker2.lab.fsc.io-app01-frontend-nad01   17m
```

```
$ kubectl describe workloadinterfaces.fsc.fss.nokia.com worker1.lab.fsc.io-app01-frontend-nad01
 -n fsc-system
Name:         worker1.lab.fsc.io-app01-frontend-nad01
Namespace:    fsc-system
Labels:       <none>
Annotations:  fsc/metadata: {"uid":"58542c1f-4a3c-4987-9140-8d3029a05e37","creation
Timestamp":"2022-07-28T06:46:09Z"}
API Version:  fsc.fss.nokia.com/v1
Kind:         WorkloadInterface
Metadata:
  Creation Timestamp:  2022-07-28T06:46:09Z
  Generation:          1
  Managed Fields:
    API Version:  fsc.fss.nokia.com/v1
    Fields Type:  FieldsV1
    fieldsV1:
      f:spec:
        .:
        f:cni:
        f:server-interface:
          .:
          f:interface:
          f:node:
          f:vlan-end:
          f:vlan-start:
          f:vlan-type:
    Manager:      Go-http-client
    Operation:    Update
    Time:         2022-07-28T06:46:09Z
    API Version:  fsc.fss.nokia.com/v1
    Fields Type:  FieldsV1
    fieldsV1:
      f:status:
        .:
        f:connectstatus:
        f:pending-podkeys:
        f:synced-podkeys:
          .:

 f:worker1.lab.fsc.io:37721f66056e9e87038cf39a73e023b27046dabb7ec1b0a5d59dcfe6d53081a0-net1:

 f:worker1.lab.fsc.io:4df27e16cc53a660f3841a10522c02e254f722907c7be9f193965107750cca4f-net1:

 f:worker1.lab.fsc.io:c7acdb797dae90fa2318b47063493dbd24ca46c300ca7560cebdb0448c5dc74d-net1:

 f:worker1.lab.fsc.io:e21de0ba145d2576f8f1423b9d5dfece6fe399ec9db8996c05afbce837a36533-net1:
    Manager:         fsc-manager
    Operation:       Update
    Subresource:     status
    Time:            2022-07-28T06:46:09Z
  Resource Version:  18611818
  UID:               21fbed90-43fb-465f-bbfb-e5de7ee9fe9d
Spec:
  Cni:  app01/frontend-nad01
  Server - Interface:
    Interface:     enp6s0
    Node:          worker1.lab.fsc.io
    Vlan - End:    2006
    Vlan - Start:  2006
    Vlan - Type:   VLANTYPE_VALUE
Status:
```

```
  Connectstatus:  Sync-Done
  Pending - Podkeys:
  Synced - Podkeys:
    Worker1.lab.fsc.io:37721f66056e9e87038cf39a73e023b27046dabb7ec1b0a5d59dcfe6d53081a0-net1:
 blrtwo/blrtwo-def-nad10-dep-7bd5f9f9c4-ptmnx
    worker1.lab.fsc.io:4df27e16cc53a660f3841a10522c02e254f722907c7be9f193965107750cca4f-net1:
 blrtwo/blrtwo-def-nad10-dep-7bd5f9f9c4-lzjm7
    worker1.lab.fsc.io:c7acdb797dae90fa2318b47063493dbd24ca46c300ca7560cebdb0448c5dc74d-net1:
 blrtwo/blrtwo-def-nad10-dep-7bd5f9f9c4-dg6kt
    worker1.lab.fsc.io:e21de0ba145d2576f8f1423b9d5dfece6fe399ec9db8996c05afbce837a36533-net1:
 blrtwo/blrtwo-def-nad10-dep-7bd5f9f9c4-2xvnq
Events:
<none>
```

4.4 LAG support

For the OpenShift and Kubernetes plugin, two types of bond are supported for LAGs:

Bond configuration examples

- Mode-1 bond - Active-Backup. This bond:
  - requires autonomous ports
  - must not have a LAG created inside the Edge-Links of the Fabric in the Fabric Services System
  - will create sub-interfaces for each individual leaf port in the Fabric
- Mode-4 bond - 802.3ad LACP (Active/Active). This bond:
  - requires an LACP enabled bond
  - must have an LACP LAG created inside the Edge-Links of the Fabric in the Fabric Services System
  - will create sub-interfaces for the LACP LAG edge-link

Bond configuration on the OpenShift compute nodes is performed following the OpenShift recommended method of using the NMState operator.

The following is an example of a mode-4 bond with four member ports:

```
apiVersion: nmstate.io/v1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: bond0-worker1
spec:
  nodeSelector:
    kubernetes.io/hostname: blrfsctb01-worker1.lab.fsc.fss.nokia.com
  desiredState:
    interfaces:
    - name: bond0
      description: Bond0 config on worker1
      type: bond
      state: up
      link-aggregation:
        mode: 802.3ad
        options:
          miimon: '140'
        port:
          - enp1s0f0
          - enp1s0f1
          - enp1s0f2
```

```
          - enp1s0f3
      mtu: 1450
```

The following is an example of a mode 1 bond with four member ports:

```
apiVersion: nmstate.io/v1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: bond0-worker1
spec:
  nodeSelector:
    kubernetes.io/hostname: blrfsctb01-worker1.lab.fsc.fss.nokia.com
  desiredState:
    interfaces:
    - name: bond0
      description: Bond0 config on worker1
      type: bond
      state: up
      link-aggregation:
        mode: active-backup
        options:
          miimon: '140'
        port:
          - enp1s0f0
          - enp1s0f1
          - enp1s0f2
          - enp1s0f3
      mtu: 1450
```

Supported LAG topologies

FSC supports two types of LAG topologies:

1. Single-homed LAG
2. Multi-homed LAG

Multi-homed (MH-) LAG support is currently a beta-quality feature. For an MH-LAG, the only configuration that FSC supports is the a mode-4 bond with Active-Active MH-LAG(LACP) on the Fabric Services System

See the *Fabric Servcies System User Guide* for further details about LAG support within the Fabric Services System.

4.5 Triggered audit

If the OpenShift plugin has been configured with the required actionables (the default values of DEPLOYMENT_UPDATED and AUDIT_REQUESTED), the plugin supports the Audit capabilities as described in Audit.

The OpenShift plugin has automated auditing and correcting support from OpenShift to Connect. For this purpose, the OpenShift configuration is considered the master configuration. This means:

- if a configuration is missing in Connect but present in OpenShift, it is automatically added to Connect
- if a configuration is dangling in Connect and is not present in OpenShfit, it is automatically removed from Connect
- if a configuration is incorrect in Connect and is present in OpenShift, it will be automatically updated in Connect

Resources managed by the Fabric Services System are an exception:

- Fabric Services System managed resources are never deleted from Connect
- Fabric Services System managed resources are never updated in Connect

An audit is executed automatically upon any of the following triggers :

- upon starting up the OpenShift plugin (specifically, upon startup of the FSC pod in the Kubernetes cluster) for all deployments that are administratively Up (that is, the adminUp field is set to True)
- upon updating an the deployment that is administratively Down to be administratively Up (that is, setting the adminUp field to True)

An audit can be triggered manually by sending a POST request to the audits API endpoint (see Audit). If the scope of the request is either PLUGIN_ONLY or FULL, this will include an audit of OpenShift by the plugin.

4.6 Troubleshooting

Logs

By default the following logs are available:

- FSC Controller/Manager log file on the master nodes: `/var/log/fsc-data/logs/fsc-controller-manager.log`
- FSC Helper CNI log file on all nodes: `/var/log/fsc-cni.log`

FSC Global resource

A new custom resource is defined called FSC Global (fscGlobal) which contains information about the state of the Fabric Services System OpenShift integration.

It includes only a set of status fields, as shown below:

```
[fsc-helper@blrfsctb02-fsc-helper ~]$ oc get fscGlobal fscglobal -n fsc-system
NAME        AGE
fscglobal   69s
[fsc-helper@blrfsctb02-fsc-helper ~]$ oc describe fscglobal fscglobal -n fsc-system
Name:         fscglobal
Namespace:    fsc-system
Labels:       <none>
Annotations:  <none>
API Version:  fsc.fss.nokia.com/v1
Kind:         FscGlobal
Metadata:
  Creation Timestamp:  2023-05-22T09:49:55Z
  Generation:          1
  Resource Version:    36895377
  UID:                 91f5b160-d76c-4fcc-8e71-c06d91ad750b
Status:
  Controller Audit Status:    Done
  Controller Connect Status:  StateClientRunning
  Deployment Admin State:     AdminUp
  Deployment Id:              461741525005517023
  Deployment Name:            k8s-deployment-name
  Fsc Cni Injection Mode:     automatic
  Fsc Controller Replicas:    1
  Plugin Id:                  k8s-plugin-id
```

```
  Pod Name:                  prod-fss-fsc-controller-manager-86c48fb667-cp2mm
Events:                      <none>
[fsc-helper@blrfsctb02-fsc-helper ~]$
```

The ControllerAuditStatus highlights the latest state of the Audit mechanism of the FSC Controller, which can be any of the following:

- Pending
- InProgress
- Done

The ControllerConnectStatus represents the status of the FSC Controller connectivity to the Fabric Services System Connect service. Possible values are:

- StateClientInit
- StateClientRestart
- StateClientRegister
- StateClientAudit
- StateClientDisconnect
- StateClientRunning

5 The OpenStack plugin

The Fabric Services System integrates with OpenStack to provide fabric level application networks for OpenStack Virtual Machines. The Connect integration leverages the OpenStack Neutron architecture to support managing the fabric directly from OpenStack and make the fabric dynamically respond to the networking needs of the application.

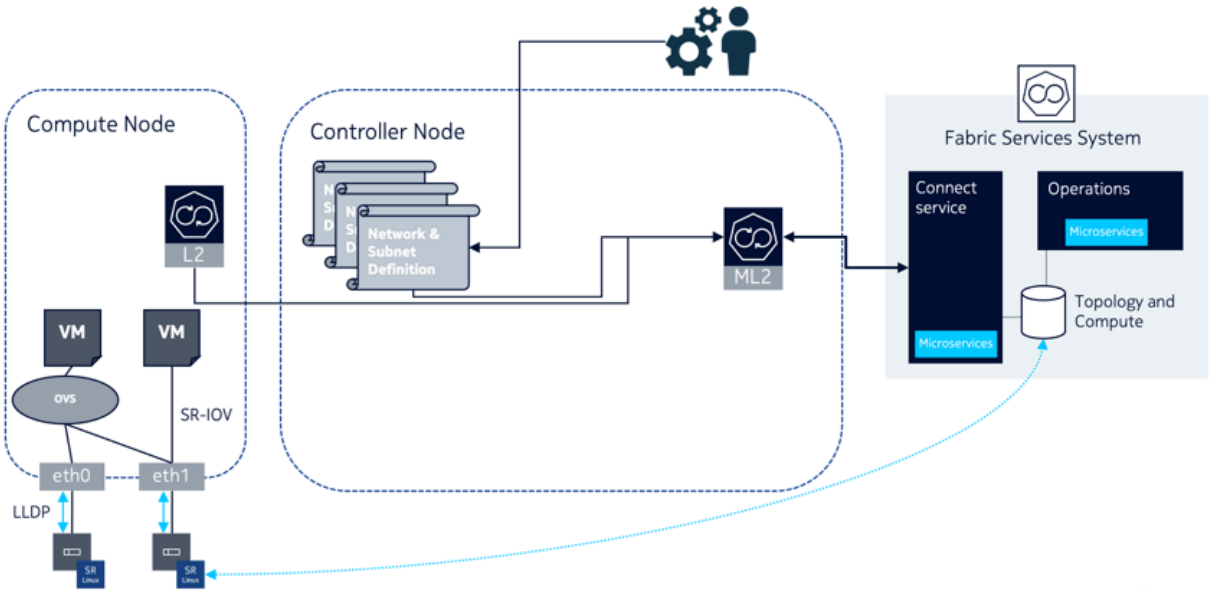
It provides the following advantages and capabilities:

- Direct integration into the network management workflow of OpenStack.
- The use of the common ML2 Plugins used by Enterprise applications and VNFs like OVS, OVS-DPDK and SR-IOV.
- Automatic provisioning of the fabric based on where the Virtual Machines need the connectivity.
- Support advanced workflows through the Fabric Service System Managed solution. Including for VNF use cases with features like QoS, ACLs, BGP PE-CE, ...
- Interconnectivity between different cloud environments, allowing for flexible network configurations.

5.1 Architecture

The Fabric Services System introduces a few new components in an OpenStack environment to allow the management of the SR Linux based fabric through OpenStack. Below is an overview of these components.

Figure 3: OpenStack Architecture



The Connect ML2 Plugin

The Connect ML2 Plugin is the heart of the integration between OpenStack and the Fabric Services System. This plugin integrates with OpenStack Neutron and will react to the creation of projects (tenants), Networks (and Subnets) and VM ports.

Whenever a project is created, a matching Workload VPN Intent is created in the Fabric Services System. This Workload VPN Intent will then be used for each of the Subnets created for that project, which will become Subnets inside the Workload VPN Intent.

When a VM port is created inside a Neutron Subnet and the VM is started on a OpenStack compute node, the ML2 Plugin will learn on which compute node the VM is deployed and through the internal topology will make sure the necessary sub-interfaces are configured as part of the Subnet in the fabric.

This information is learned from the L2 Agent extension which stores the Neutron Network to physical interfaces topology in the Neutron database. This information is then provided to the Connect service and together with the LLDP information of the fabric, the Connect service knows which edge-links in the fabrics need to be configured as sub-interfaces.

The Connect L2 Agent Extensions

The Connect L2 Agent Extensions extend the already existing L2 Agent that is present on every OpenStack compute. These extensions are responsible of mapping the relation between the physical NICs and the different networking constructs setup for the Neutron networks.

5.2 Supported versions

The Fabric Services System Connect plugin supports the integration with OpenStack Train. Official support is provided for Red Hat OSPd 16.2. The full certification of this platform is ongoing.

5.3 Installation

About this task

Follow this procedure to install the Fabric Services System with OpenStack.

Procedure

Step 1. Deploy the Fabric Services System normally.

Step 2. Create an OpenStack user within the Fabric Services System.

Step 3. Deploy OpenStack Train (such as RHOSP 16.2)

Step 4. Patch the Neutron container images with the following Fabric Services System openstack RPMs for this release:

a. On the controllers, install:

- `networking-nokia-fss-16.0-22.12.1.noarch.rpm`
- `fss-connect-pythonsdk-16.0-22.12.1.noarch.rpm`
- `python-fss-openstackclient-16.0-22.12.1.noarch.rpm`

b. on the computes, install `networking-nokia-fss-16.0-22.12.1.noarch.rpm`.

Step 5. Adjust config files:

a. On the OpenStack controller nodes, add sections to the following config files:

```
# neutron.conf
[DEFAULT]
service_plugins = ...,fss_nic_mapping
```

```
# ml2_conf.ini
[DEFAULT]
nic_mapping_provisioning = True  # False if operator should not be allowed to
 provision mappings
```

```
[ml2]
extension_drivers = ...,fss_network
mechanism_drivers = fss_connect,openvswitch
```

```
[ml2_fss_connect]
uri = http://<fss ip>/rest
username = <openstack user name>
password = <openstack user password>
deployment_name = openstack  # string, unique name to identify current deployment in
 Fss
```

```
# openvswitch_agent.ini
[agent]
extensions = nic-mapping,...
```

b. On the OpenStack compute nodes, add sections to the following config file:

```
# openvswitch_agent.ini
[agent]
extensions = nic-mapping,...
```

Step 6. Restart the Neutron containers on controllers and computes.

5.4 Configuring the OpenStack plugin

When the OpenStack Neutron ML2 mech driver connects to the Fabric Services System and the Connect service, it will automatically create its own Connect deployment.

However, this deployment will be administratively Down, and so will not be functional.

Deployments are set to be administratively Down to prevent unwanted changes in the fabric and serve as another check for an Administrator to approve the specific OpenStack cluster to integrate with, and control, the fabric.

To bring this deployment administratively Up, update the deployment. To do this, fetch the deployment by name based on the plugin's `deployment_name` setting, and set the `adminUp` field to True. For more information, see Setting the AdminUp status for a deployment.

See also Managing plugins and deployments.

5.5 Managing OpenStack users

About this task

Follow this procedure to update the OpenStack plugin's Fabric Services System user account with a new password.

Procedure

Step 1. Update the plugin .ini file (likely at /etc/neutron/plugins/ml2/ml2_conf_fss_connect.ini) with the new password:

```
[ml2_fss_connect]
uri = <fss uri>
username = <username>
password = <new password>
deployment_name = <deployment name>
```

Step 2. Restart Neutron.

5.6 Usage

5.6.1 OpenStack usage for OpenStack managed networks

The Connect-based OpenStack solution is fully compatible with OpenStack. The standard API commands for creating networks and subnets remain valid.

When mapping to Connect:

- The network in OpenStack is created within the user's project. If the network is created by an administrator, the administrator can optionally select the project to which the network belongs.

  This OpenStack project appears as a tenant in Connect. This tenant is created when the first OpenStack network is created in this project, and it is deleted when the last network in this project is deleted
- Only VLAN-based networks (where segmentation_type == 'vlan') are handled by the Fabric Services System ML2 plugin, meaning only VLAN networks are HW-offloaded to the fabric managed by the Fabric Services System.
  Neutron defines the default segmentation type for tenant networks. If you want non-administrators to create their own networks that are offloaded to the Fabric Services System fabric, the segmentation type must be set to 'vlan'.

  However, if only administrators need this capability, then the segmentation type can be left to its default, since administrators can specify the segmentation type as part of network creation.
- The plugin only supports Layer 2 Fabric Services System fabric offloaded networking. When a Layer 3 model is defined in OpenStack (using router and router attachments), this model will work; but Layer 3 is software-based according the native OpenStack implementation.

  Layer-3-associated features, like floating IPs, will also work; but these are only software-based according to the native OpenStack implementation.

Note: This is tested with the traditional non-Distributed Virtual Routing (DVR) implementation only; DVR has not been tested, and might not work.

- Untagged networks are not supported in this release.
- MTU provisioning is not supported in this release. That is, MTU information is not programmed into the fabric.

5.6.2 OpenStack usage for Fabric Services System managed networks

About this task

In addition to the OpenStack managed networking model, Connect supports networking managed by the Fabric Services System. In this case, the Workload EVPN and subnet resources are created first in the Fabric Services System directly, and then are consumed in OpenStack.
To support Fabric Services System managed networking, two pairs of proprietary extensions have been added to Network:

- when using the UUIDs of the Workload Intent and Subnet: `fss-workload-evpn` and `fss-subnet`
- when using the names of the Workload Intent and Subnet: `fss-workload-evpn-name` and `fss-subnet-name`

Procedure

Step 1. Using the Fabric Services System intent-based REST API or the UI, create a Workload VPN and one or more bridged subnets within that workload.

Step 2. Obtain the UUIDs of these resource.

As an example, assume the Workload Intent has the name `fss-evpn-1` and the UID `410559942890618880`, and the Subnet within it has the name `fss-subnet-1` and the UUID `410560233203564544`.

Step 3. Choose one of the following:

- To complete the configuration using resource UUIDs, go to step 4.
- To complete the configuration using object names, go to step 5.

Step 4. Complete the configuration using object UUIDs:

a. In OpenStack, create the consuming network, linking to these pre-created entities.

To continue the example, assume a network named `demo-network` is created within a project named `demo-project`. Using the previous example values for the VPN and subnet, the command for this step would be:

```
openstack network create --project os-project --fss-workload-evpn 410559942890618880
 --fss-subnet 410560233203564544 os-network-1
```

b. Create a subnet within this network using the standard API syntax. For example:

```
openstack subnet create --network os-network-1 --subnet-range 10.10.1.0/24 os-
subnet-1
```

Step 5. Complete the configuration using object names:

a. In OpenStack, create the consuming network, linking to these pre-created entities.

Assume the Workload EVPN is named `fss-evpn-1` and the Subnet underneath is named `fss-subnet-1`. Further assume a network named `demo-network` is created within a project named `demo-project`. Using the previous example values for the VPN and subnet, the command for this step would be:

```
openstack network create --project os-project --fss-workload-evpn-name fss-evpn-1 --
fss-subnet-name fss-subnet-1 os-network-1
```

b. Create a subnet within this network using the standard API syntax. For example:

```
openstack subnet create --network os-network-1 --subnet-range 10.10.1.0/24 os-
subnet-1
```

What to do next

Creating a Neutron port and Nova server within this network and subnet also uses standard APIs.

When mapping these objects to Connect:

- In Fabric Services System managed networking, the relationship between OpenStack projects and the Connect tenant disappears because the Connect tenant is dictated by the **`--fss-workload-evpn/fss-workload-evpn-name`** parameter set.
  This means that a new Connect tenant is created for every `--fss-workload-evpn/fss-workload-evpn-name` parameter passed to network create.

  For identical **`--fss-workload-evpn/fss-workload-evpn-name`** parameters passed to different network create commands, the Connect tenant resource is re-used.
- From the OpenStack point of view, Layer 3 is not supported in Fabric Services System managed networking. A subnet within a Fabric Services System managed network cannot be router-associated. From the Fabric Services System point of view, routed subnets defined within a Workload EVPN can by consumed by OpenStack by a Fabric Services System managed network.

  In this case, the Layer 3 forwarding and routing is offloaded to the Fabric Services System managed fabric, unlike the software-based model that applies for Layer 3 in OpenStack managed networking.

5.6.3 Edge topology introspection

Edge topology information is introspected using LLDP. LLDP must be enabled on the OpenStack computes. The enabling of LLDP is performed automatically by the nic-mapping L2 agent extension for OVS computes. For more on the nic-mapping agent, see NIC mapping.

5.6.4 NIC mapping

The following cases can apply for managing NIC mapping:

- Automated NIC mapping as of introspection by extended Layer 2 agent on the compute:
  When the nic-mapping agent extension is configured, the Neutron OVS agent will inspect its configuration on startup and enable LLDP Tx on the interfaces under any OVS bridges it finds.

  It will also persist the <physnet> ↔ <compute, interface> relation in the Neutron database so it can wire Neutron ports properly in the fabric.

Known interface mappings can be consulted using the CLI command `openstack fss interface mapping list`.

Note: Enabling LLDP Tx from the OVS agent extension is supported only for DPDK interfaces. For enabling LLDP in other cases, see Compute requirements.

- Programmed NIC mapping via Neutron API
  If you do not want to enable the OVS Layer 2 agent extension, you can provision interface mappings manually using the command `openstack fss interface mapping create`.

  Note: To support this operation, the following configuration should be present in the ml2 plugin section:

  ```
  [DEFAULT]
  nic_mapping_provisioning = True
  ```

Note: Nokia does not recommend combining the Layer 2 agent extension with manual provisioning, because when the agent starts or re-starts, it clears entries that were provisioned manually.

5.6.5 SR-IOV NICs

SR-IOV NICs are supported the OpenStack/Connect/Fabric Services System solution. They follow the standard OpenStack model, for example using the vnic_type 'direct'.

If there are physical NICs on the compute used by SR-IOV only, and so PF is not in the OVS agent configuration, an LLDP agent such as lldpad or lldpd must be installed and running on the compute to provide neighbour information to fabric. This does not occur automatically.

5.6.6 Bonded interfaces

There are two types of bonded interfaces:

- Linux bonds
- OVS bonds

Linux bonds can be applied on VIRTIO computes. In the case of DPDK computes, OVS bonds can be applied.

Both Linux and OVS bonds come as Active/Active (A/A) or Active/Standby (A/S).

Linux bonds can be deployed as static LAGs or dynamic LAGs using LACP.

OVS DPDK can only work with dynamic LAGs using LACP.

For computes configured OVS DPDK Active/Standby (A/S), the preferred_active should select the device which has the active DPDK port.

Note: In the current release, only Active/Active LACP LAGs are supported. Linux mode-1 bonds (Active/Standby) are not yet supported. LACP LAGs require the appropriate configuration to be

present on the fabric configuration in the Fabric Services System as well as described in the LACP LAGs.

5.6.7 Trunk ports

The network trunk service allows multiple networks to be connected to an instance using a single virtual NIC (vNIC). Multiple networks can be presented to an instance by connecting it to a single port.

For information about configuring and operating the network trunk service, please see https://docs.openstack.org/neutron/train/admin/config-trunking.html

Trunking is supported for VRTIO, DPDK and SRIOV.

- When the vnic_type is set to "normal ports" (VIRTIO/DPDK), trunking is supported through an upstream openvswitch trunk driver.
- When the vnic_type is set to "direct ports" (SRIOV), trunking is supported by the Fabric Services System ML2 plugin trunk driver.

When using trunks with SRIOV, some limitations apply to the upstream OVS trunk model:

- The parent port of the trunk and all subports must be created with the vnic_type "direct".
- To avoid the need for QinQ on switches, trunks for SRIOV instance must be created with a parent port belonging to a flat network (untagged).
- If multiple projects within a deployment must be able to use trunks, the neutron network must be created as shared (using the --share attribute)
- When adding subports to a trunk:
  - you must specify their segmentation-type as VLAN
  - their segmentation-id must match a segmentation-id of the neutron network to which the subport belongs

6 VMware plugin

The Fabric Services System integrates with VMware vSphere to provide fabric level application networks for VMware Virtual Machines. The Connect integration leverages the VMware vSphere distributed vSwitch architecture to support managing the fabric directly from VMware vCenter and make the fabric respond to the networking needs of the environment.

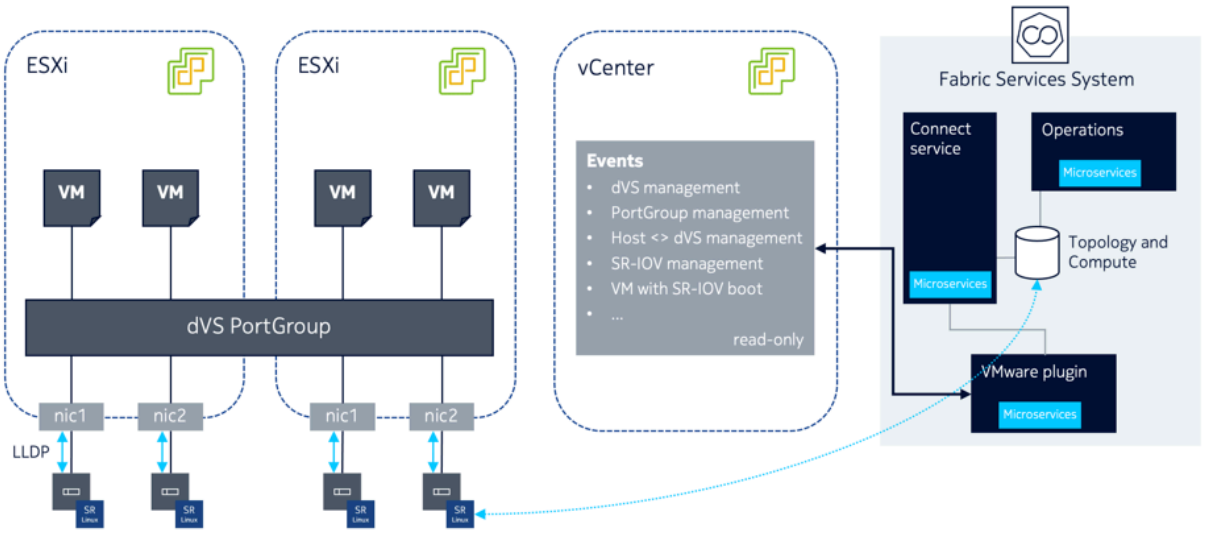
It provides the following advantages and capabilities:

- Direct integration into the network management workflow of VMware vCenter.
- The use of the common distributed vSwitches and Port Groups for both regular Virtual Machine NICs as well as SR-IOV use cases.
- Automatic provisioning of the fabric based on where the Virtual Machines need the connectivity.
- Support advanced workflows through the Fabric Service System Managed solution. Including for VNF use cases with features like QoS, ACLs, BGP PE-CE, ...
- Interconnectivity between different cloud environments, allowing for flexible network configurations.

6.1 VMware architecture

The Fabric Services System introduces a new component in a VMware vSphere environment to allow the management of the SR Linux based fabric through VMware vCenter. Below is an overview of the architecture.

Figure 4: VMware architecture



The Connect VMware vSphere plugin

The Connect VMware vSphere plugin is a micro-service which is deployed inside the Fabric Services System environment itself. After deploying the plugin itself, connections to different VMware vCenter environments can be set up through the creation of Deployments within the plugin. This is done through the Connect Service API.

Once a VMware Plugin has been setup to a VMware vCenter environment, it will connect to that vCenter and listen for events for the following objects in VMware vCenter:

- Distributed vSwitch management
- Port Group management
- Host to dVS associations
- SR-IOV Configuration
- SR-IOV enabled Virtual Machine boot and shutdown

The plugin only uses a read-only user for these activities.

Note:
The Connect VMware vSphere plugin supports up to a maximum of 2 different VMware vCenter deployments. This will be increased in the future.

6.2 Installation

About this task

Because the VMware plugin for Connect is deployed within the Fabric Services System's Kubernetes cluster, its installation is integrated in the Fabric Services System installer.

Procedure

Step 1. Install the Fabric Services System normally using its installer.

Step 2. Once the Fabric Services System is running, create a Fabric Services System user that will be used by the VMware plugin. The VMware plugin will use this account (specifying its username and password) when communicating with the Fabric Services System and the Connect service.

Step 3. Install the VMware plugin by executing the `fss-vmware-plugin-install.sh` script.

This scripts is executed with the username and password you configured in step 2. If the script does not include these credentials initially, they can be provided interactively. See the script's help information:

```
$ /root/binfss-vmware-plugin-install.sh -h

Usage: ./fss-vmware-plugin-install.sh -u <fss-username> -p <fss-password>
        -u Fabric Services System username to be used by vmware plugin
        -p Fabric Services System password for the user to be used by vmware plugin
        -h help
```

Expected outcome

When the plugin is installed, it will register itself to the Connect service. If all goes well, the pod will be in the Running state when queried using `kubectl`.

You can also verify that the plugin is registered using the following API call:`GET <fss-server>/connect/api/v1/plugins/plugins`

The reply will resemble the following:

```
[{"id":"422770209502265344","connectType":"vmware","callback
Url":"connect01.lab01.nokia:80","apiKey":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey
JhdXRob3JpemVkIjp0cnVlLCJleHRlcm5hbElEIjoidm13YXJlIn0.ZC0YpfyHzHqv6oPuU-aJhn
MnAGtLCVDlQFPjfRNkPm4","name":"vmware","externalId":"vmware","possibleSettings":
[{"name":"host","required":true,"description":"vCenter
          host","unique":true,"example":"vmware.example.net","encrypted":false},
{"name":"username","required":true,"description":"vCenter
        user","unique":false,"example":"admin","encrypted":false},
{"name":"password","required":true,"description":"Password
        of the vCenter
        user","unique":false,"example":"secret","encrypted":true}],"supportsNew
Deployments":true}]
```

The plugin is now operational and ready for the creation of a deployment.

6.3 Configuring VMware

To manage a given vSphere deployment with the VMware Connect plugin, create an Connect deployment for that VMware Connect plugin. This process can be repeated for multiple vSphere deployments.

This can be done through the Connect UI following the "Creating Connect Deployments" process, or using the following API call:

`POST <fss-server>/connect/api/v1/admin/deployments`

For more details on the API POST request, refer to Creating a Deployment (for VMware Plugin) or to the API documentation as described in the *API Integration Guide*.

Some settings are mandatory; others are optional. Specifically for the VMware plugin, these settings are :

Table 5: VMware settings

Setting	Description	Unique?	Required?	Stored encrypted?
`host`	The hostname of the vCenter.	True	True	False
`username`	The user name to use with vCenter.	False	True	False
`password`	The password to the vCenter.	False	True	True
`location`	The location of the vCenter.	False	False	False

- The `host` parameter must be unique across all deployments.
- The `password` parameter will be encrypted in the database.
- Note that `location` is an optional parameter.

Setting	Description	Unique?	Required?	Stored encrypted?
• The username and password are for a user in vCenter which has read-only access to the vCenter environment				

For details about creating a Deployment for the VMware Plugin, see Creating a Deployment (for VMware Plugin) in the Common API Examples section.

6.4 vCenter certificate validation

About this task

The connection of the VMware plugin to the vCenter server always uses HTTPS so that communication is secured. However, to fully secure this communication, the best practice is to also enable certificate validation for the vCenter server certificate.
The process below forces the plugin to verify the server certificate. To verify the certificate, it must either be signed by a well-known public Certificate Authority, or the server certificate to trust must be provided in the configuration.

Procedure

Step 1. To configure the certificate verification, you must obtain the certificate of the vCenter server in PEM format. To obtain the certificate, do either of the following:

- Use your browser to export the certificate to the PEM format and store it on your local system. You can do this by opening the vCenter UI and use the standard browser capabilities to view the certificate details and export the certificate. The exact procedure depends on which browser you use and can be found in the documentation for browser.
- Use the vCenter API to fetch the TLS details of the server and use the "cert" field of the output.
  For more details about the API, check the documentation for the vCenter Server version. For example, for version v7.0u3, see https://developer.vmware.com/apis/vsphere-automation/v7.0U3/vcenter/api/vcenter/certificate-management/vcenter/tls/get/

Step 2. Update the Connect Deployment configuration for the vCenter and update the following two settings:

- Tls Verify: set this field to "true".
- Certificate: provide the certificate PEM content here, starting with the `-----BEGIN CERTIFICATE-----` and ending with the `-----END CERTIFICATE-----` text

Expected outcome

After you have made these changes, the vCenter plugin will start validating the certificate of vCenter and will fail the connection if the vCenter certificate does not match the provided details. An alarm will be raised in case there is a connectivity issue.

6.5 Usage

6.5.1 VMware usage for VMware managed networks

The VMware plugin acts on the operator actions in vSphere by listening in on the vCenter event bus. The standard VMware method of creating networks is supported by the VMware plugin for Connect:

- Create dvSwitches
- Add uplinks
- Create dvPortGroups

In the standard VMware managed networks model, all Fabric Services System subnets are created within a single Fabric Services System Workload VPN, which represents one particular vSphere deployment. Within Connect, when inspected using the API, this corresponds to a single tenant.

6.5.2 VMware Usage for Fabric Services System managed networks

In addition to the VMware managed networking model, Connect supports advanced Fabric Services System managed networking. In this model, Workload VPN and subnet resources are first created directly in the Fabric Services System and are then consumed in VMware.

To support this model, Connect uses custom attributes in vSphere.

The management flow is :

- In the Fabric Services System:
  1. Create a Workload VPN and one or more bridged subnets within it using the Fabric Services System intent-based REST API or UI.
  2. Obtain the UUIDs of these resource.

  For this example, we will assume that the Workload VPN has the UUID `410559942890618880` and the subnet within it has the UUID `410560233203564544`.
- In VMware:
  1. Create a dvSwitch.
  2. Specify a custom attribute "FSSWorkloadEVPNID" on that dvSwitch, and set it to the UUID of the previously created Workload VPN in the Fabric Services System.
     In our example, that would be `410559942890618880.`
  3. Create a dvPortGroup.
  4. Specify a custom attribute FSSSubnetID on that dvPortGroup and set it to the UUID of the operator-precreated subnet in the Fabric Services System.
     In this example, that would be `410560233203564544`.

From here, all standard VMware operations continue.

You can undo the Fabric Services System managed characteristic of a dvSwitch or dvPortGroup by re-defining the custom attributes to the value None. The value is not case sensitive.

Note: During step 3, you will see a temporary, new, and identical subnet being created within the previously created Workload VPN. This is because during step 3, a VMware managed subnet is created under a Fabric Services System managed Workload VPN. If the scenario were to stop at

step 3, this would be the intended state. When performing step 4 however, this temporary subnet is deleted, and only the operator created subnet remains.

Fabric Services System managed dvSwitches with VMware managed dvPortGroups

A particular case exists for a Fabric Services System managed networks in which the dvSwitch in VMware is managed by the Fabric Services System, but the dvPortGroups remain managed by VMware.

The opposite model does not exist; there is no support for a Fabric Services System managed dvPortGroups within VMware managed dvSwitches.

6.6 LACP support

Prerequisites

- a VMware plugin must be running
- a deployment mapping to the participating vCenter must exist

About this task

You can integrate the functionality of LACP LAGs in vCenter by configuring the underlying fabric in response to changes in vCenter.

Procedure

Step 1. Create an LACP LAG on the DVS in vCenter. Specify parameters for the LAG, including:

- LAG name
- number of LAG members
- LAG mode and load balancing mode

Note: This does not trigger any action in the VMware plugin.

Step 2. Ensure that the LAG exists on the Fabric Services System fabric and that its members map to the correct SubHostPorts in Connect.

Step 3. Attach the preferred host NICs to the preferred LAG members.

The VMware plugin will use this information to determine the correct SRL Leaf ports for when the LACP LAG is used as an uplink for a Port Group.

Step 4. Create one or more Port Groups on the distributed vSwitch and use the LACP LAG as the Active uplink.

The VMware plugin will use the Connect service to provision the Leaf switches with the appropriate network connectivity for the Port Groups.

Expected outcome

Sub-Interfaces corresponding to the LAG members are automatically created in the appropriate Subnets in the Fabric Services System.

Note:

- To get LLDP information from vCenter, ensure that the discovery protocol in the distributed vSwitch advanced settings is set to Link Layer Discovery Protocol, and the operation is either advertise or both.
- The total number of LAG members and individual uplinks per distributed vSwitch cannot exceed 32.

6.7 VMware audit

The VMware plugin has automated auditing and correcting support from vSphere to Connect. For this purpose, the vSphere configuration is considered the master configuration.

This means:

- if a configuration is missing in Connect but present in vSphere, it is automatically added to Connect
- if a configuration is dangling in Connect and is not present in vSphere, it is automatically removed from Connect
- if a configuration is incorrect in Connect and is present in vSphere, it will be automatically updated in Connect

Resources managed by the Fabric Services System are an exception:

- Fabric Services System managed resources are never deleted from Connect
- Fabric Services System managed resources are never updated in Connect

An audit is executed automatically upon any of the following triggers :

- upon starting up the VMware plugin (specifically, upon startup of the pod in the Kubernetes cluster) for all deployments that are administratively Up (that is, the `adminUp` field is set to True)
- upon creating a new deployment as administratively Up
- upon updating an existing deployment that is administratively Down to be administratively Up (that is, setting the `adminUp` field to True)

An audit can be triggered manually by sending a POST request to the audits API endpoint (see Audit). If the scope of the request is either PLUGIN_ONLY or FULL, this will include an audit of VMware by the plugin.

7 The Connect service architecture

Connect acts as a bridge between the Fabric Services System and different cloud environments. It is aware of the different processes and workloads running on the servers that make up the cloud environment while at the same time being aware of the fabric as configured on the Fabric Services System.

To ensure a fast and consistent API, the operations on Connect are decoupled from the Fabric Services System itself. Each resource in Connect corresponds to one or more resources in the Fabric Services System. By being decoupled, the successful creation or update of a Connect resource is only a promise that an action will later be taken on the Fabric Services System. To expose this concept on the API, three fields are available:

- **version**
- **deployedVersion**
- **Status**

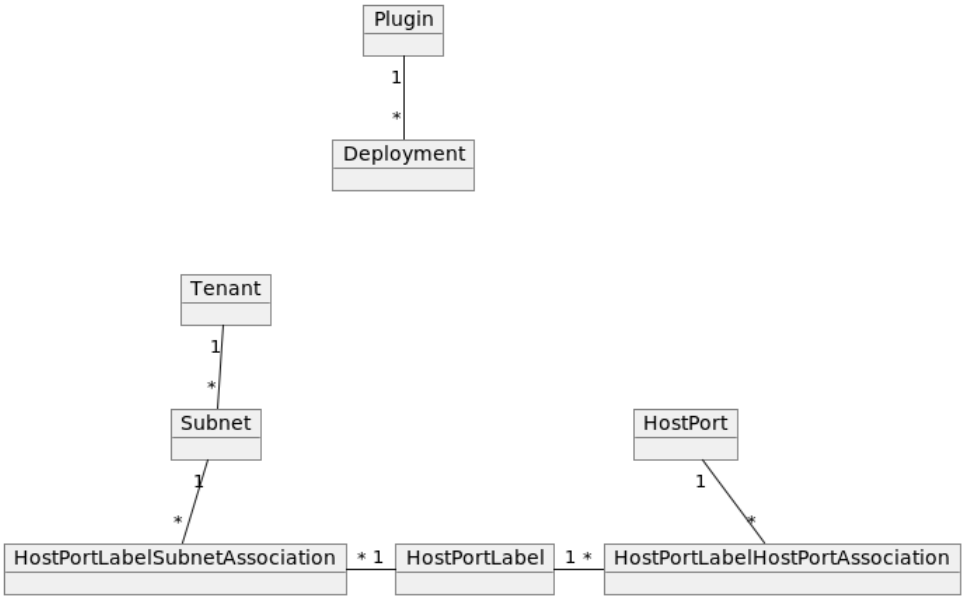
Version is incremented after every Create or Update operation on a resource.

deployedVersion is incremented once those changes are deployed to the Fabric Services System.

For convenience, Connect maintains the **Status** field. This field indicates whether a resource is Deploying, Deployed, or in an Error state. It is this field that will be important when troubleshooting failures. While the field value is Deploying, the Fabric Services System itself is not yet configured based on the resource in Connect.

Connect maintains the objects shown in Figure 5: Connect component objects.

Figure 5: Connect component objects



- *Plugin*: An instance of a Connect plugin, registered on Connect by the plugin itself. (See also Managing plugins and deployments.)

- *Deployment*: A cloud environment that is monitored by a specific plugin. A Deployment is created on a plugin by either the operator or the Plugin itself (see Managing Plugins & Deployments further on). All further Connect resources will belong to one specific Deployment.
  For more information about deployment, see Managing plugins and deployments.
- *Tenant*: A container intended for the separation of concern between different tenants or customers on the cloud environment. A tenant will result in a workload intent on the Fabric Services System. The ID of this workload intent is populated into the fssWorkloadEvpnId field of the tenant.
- *Subnet*: A networking subnet. A subnet will result in a bridged workload subnet on the Fabric Services System.
- *HostPort*: A server compute NIC. A HostPort does not have any representation on the Fabric Services System as its connection is to the fabric leaf node interface, not a direct translation. The HostPort's <HostName, PortName> should be globally unique, typically with a fully qualified domain names. For example, <compute-01.datacenter01.company.com, eth0>.
  The status of a HostPort is also dependent on whether an EdgeMap with the corresponding host information was found.
- *HostPortLabel*: A container connecting HostPort and subnet. Instead of configuring every single VLAN we want to configure on a HostPort (actually on the EdgeLink Port on the Leaf where the HostPort is connected to on the fabric), Connect works with an indirection object. This allows Connect to specify a group of HostPorts with a similar configuration: for example, all compute HostPorts that serve the same overlay network through VLAN 200. A HostPortLabel is represented by a label in the Fabric Services System.
- *HostPortLabelHostPortAssociation*: HPLHPA for short, this is the association between the container object HostPortLabel and a HostPort. It ensures that all VLANs scheduled on the HostPortLabel are connected to this HostPort. An HPLHPA is represented by an association between the EdgeLink that the HostPort corresponds to, and the label of the HostPortLabel.
- *HostPortLabelSubnetAssociation*: HPLSA for short, this is the association between the container object HostPortLabel and a subnet. The association contains a VLAN to be scheduled on the sub-interfaces that follow from this HPLSA.
- *EdgeMap*: An EdgeMap contains the neighbor information the fabric has learned through LLDP. It contains the information Connect has learned about the connected hypervisors or servers to the leaf fabric.

7.1 Key concepts

Table 6: Key concepts for the connect service

Concept	Definition
Decoupled	Connect is *decoupled* from the Fabric Services System. Operations in Connect resources are only later reflected in the Fabric Services system. The *status* of a resource indicates whether it is fully deployed. See the related key concepts *Version*, *DeployedVersion*, and *Status*.
Deployed Version	The *DeployedVersion* of a Connect resource indicates which *Version* of the resource was actually deployed to the Fabric Services System. See the related key concepts *Version*, *Decoupled*, and *Status*.

Concept	Definition
Deployment	A Connect *deployment*represents a cloud Deployment. It can be created by the plugin or by the operator, depending on the type of plugin. See the related key concepts *Managing Plugins* and *Deployments*.
EdgeMap	A Connect *EdgeMap* represents the edge neighbor information learned through LLDP. It shows the compute or server physical ports that are attached to the leaf nodes of the fabric.
Plugin	A Connect *plugin* is responsible for collecting information about the workloads that are running in a cloud environment and for configuring Connect accordingly.
Status	The *Status* of a Connect resource indicates whether it is Deploying, Deployed, or in an Error state. When the resource is in an Error state, check the Fabric Services System alarms for further information.
Tenant	A Connect *tenant* is a concept for separation of concern between different *tenants* of the cloud environment. It translates into a workload intent in the Fabric Services System.
Version	The *version* of a Connect resource indicates how many create or update operations have been performed on this resource. See the related key concepts *DeployedVersion*, *Decoupled*, and *Status*.

7.2 The Connect API

Connect provides two different APIs:

- the plugins API, which can be accessed by plugins and administrators
- the admin API, which is meant for operators to manage or troubleshoot a deployment

Full API documentation can be found in the Swagger documentation.

- Typically, all resources support single GET and GET of all resources, including filtering.
- PUT and DELETE operations are supported for most resources.
- PATCH operations are not supported.

7.3 Managing plugins and deployments

Plugins register themselves to Connect.

There are two types of plugins:

- deployment-agnostic plugins: these support multiple cloud deployments and are not bound to a specific deployment. Deployments are created by the operator using the Admin API. The Nokia provided VMware plugin is a deployment-agnostic plugin.
- deployment-specific plugins: can only support one Deployment, typically the Plugin runs within the cloud environment. Deployments are self reported to the Connect service. The Nokia provided Kubernetes and OpenStack plugins are deployment-specific plugins.

Deployments can be in AdminUp true or false state (AdminUp or AdminDown). The admin state of a deployment indicates whether changes can be made on the resources (tenants, subnets, …) within that deployment.

Deployment-specific plugins will create their deployment in the Admin Down state by default. The operator must then manually enable this deployment using the Admin API. This way the operator has full control over which plugins and deployments can actually create resources.

Figure 6: Deployment-specific plugin

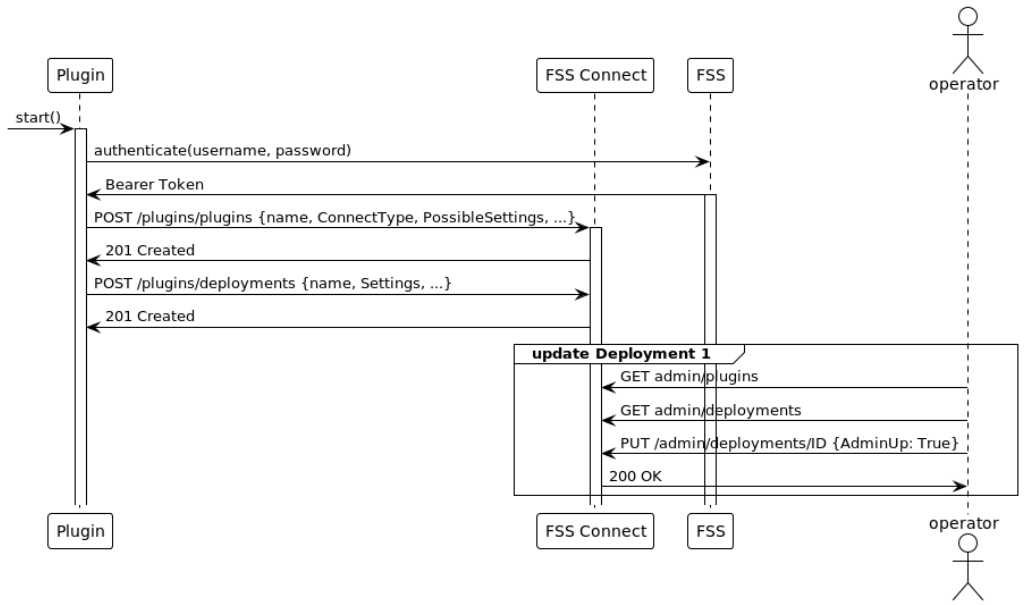
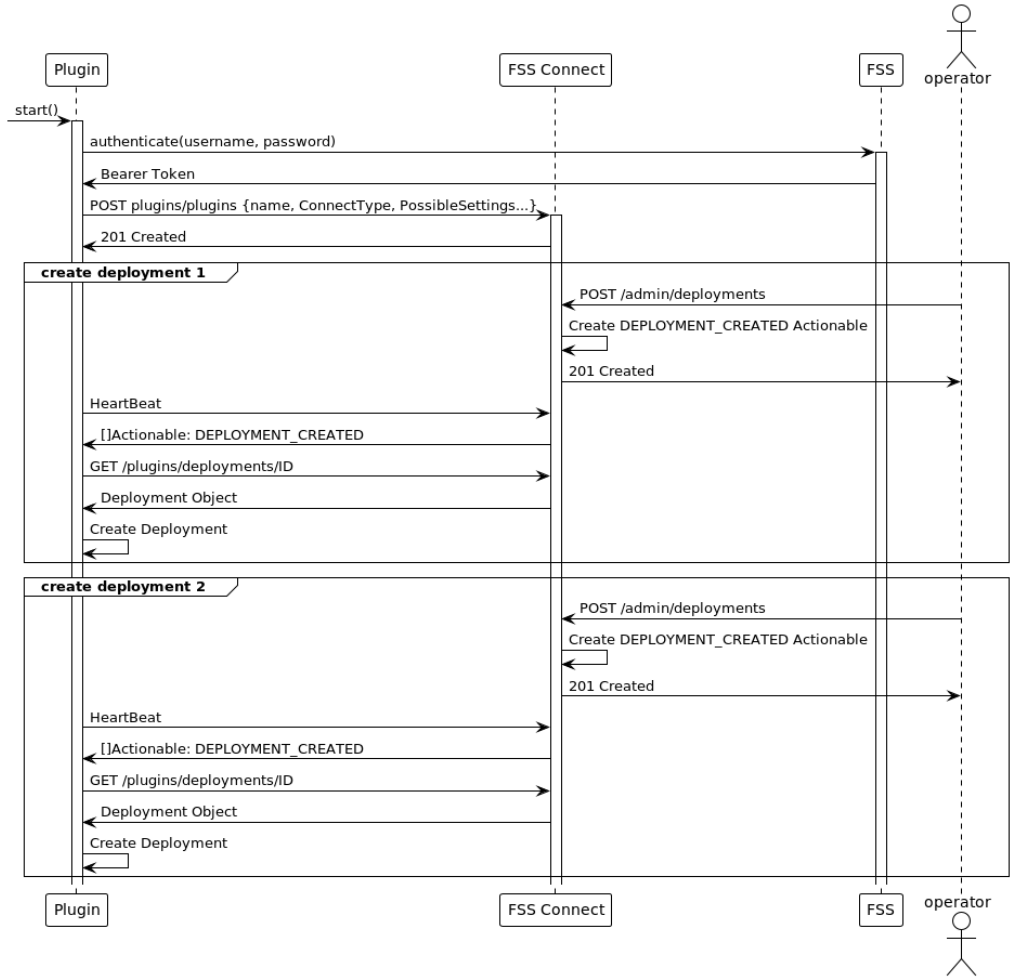


Figure 7: Deployment-agnostic plugin



7.4 LACP LAGs

Connect can create sub-interfaces using LAGs. To do this, a HostPort must have its `isLag` attribute set to `true`.

SubHostPorts (other HostPorts that are part of the LAG) must also be created for each member of the Bond on compute. This is indicated by setting a HostPort's `ParentHostPortID` field to the ID of the LAG HostPort.

Before associating a LAG-enabled HostPort to a HostPortLabel, make sure that the LAG exists on the Fabric Services System fabric and that its members map to the SubHostPorts in Connect.

If a LAG mapping to the ParentHostPort and its SubHostPort members are not found on the fabric, or if the SubHostPorts are scattered across multiple LAGs on the fabric, Connect will raise an alarm.

Note: This section only applies to LACP LAGs (Active/Active). Linux mode-1 bonds (Active/Standby) are not yet supported. In the future, these Active/Standby lags should not be created as LAGs on the fabric.

7.5 Audit

Connect provides a mechanism to audit the state of the Fabric Services System against the state of the Connect service. The state of these two components can become disconnected as as result of fabric deviations, general disconnects in the Kubernetes cluster, or manual intervention.

To recover from these scenarios, Connect introduces the audit mechanism.

To create an audit request, send a POST request including at least the deploymentID to be audited.

Connect supports the following audit scopes:

- **CONNECT_ONLY**: The audit examines the full relationship between Connect and the Fabric Services System. It also examines the LLDP information in all nodes and consolidates that with the information currently contained in Connect (Note: only for the HostPorts that are currently active in the Deployment).
- **ERROR_ONLY**: The audit examines only Connect resources that are in an ERROR state. Resources can enter an ERROR state when Connect cannot create or update their equivalent on the Fabric Services System because of unforeseen circumstances. Note that this kind of audit will not detect DANGLING_RESOURCE deviations on the Fabric Services System. If there are few resources in ERROR this audit will conclude much faster than a CONNECT_ONLY scope. However if there are a lot of resources in ERROR, use the CONNECT_ONLY scope.
- **PLUGIN_ONLY:** The audit only examines the relationship between the Connect Plugin and Connect itself. This type of Audit can be used when the connection was lost between Plugin and Connect or when a backup is restored.
- **FULL:** A full Audit covers the Plugin to Connect relation, as well as the Connect to FSS relation. It Audits the full system, as a result this type of Audit can take quite some time.

By default, the scope of the audit is CONNECT_ONLY. To change the scope, you need to provide the scope in the POST request.

```
REQUEST: POST http://localhost/rest/connect/api/v1/admin/audits

{"deploymentId": "422199394960411946"}

RESPONSE:
{
  "id": "422199984495070506",
  "enqueueTime": "2022-08-22T17:00:01.005355194+02:00",
  "endTime": "0001-01-01T00:00:00Z",
  "status": "InProgress",
  "failureReason": "",
  "dryRun": false,
  "report": [],
  "deploymentId": "422199394960411946",
  "scope": "CONNECT_ONLY",
  "totalNumberOfDiscrepancies": 0,
  "totalNumberOfSuccessfulDiscrepancies": 0,
  "totalNumberOfFailedDiscrepancies": 0
```

```
}
```

```
REQUEST: GET http://localhost/rest/connect/api/v1/admin/audits/422199984495070506

RESPONSE:

{
  "id": "422199984495070506",
  "enqueueTime": "2022-08-22T15:00:01.005Z",
  "endTime": "2022-08-22T15:00:01.578Z",
  "status": "Success",
  "failureReason": "",
  "dryRun": false,
  "report": [],
  "deploymentId": "422199394960411946",
  "scope": "CONNECT_ONLY",
  "totalNumberOfDiscrepancies": 0,
  "totalNumberOfSuccessfulDiscrepancies": 0,
  "totalNumberOfFailedDiscrepancies": 0
}
```

- `EnqueueTime` and `Endtime` allow you to monitor job execution time.
- `Status` indicates whether the audit was successful, errored, or is still in progress.
- `failureReason` is only populated when the audit is in an error state, indicating which error occurred during the audit.
- `Report` is a list of actions taken by the audit.
- `Report` entries can be of the types:
  - `MISSING_RESOURCE`: the Fabric Services System is missing a resource that is configured in Connect. To correct this, re-create the Fabric Services System resource.
  - `DANGLING_RESOURCE`: Connect is no longer aware of a resource that is configured in the Fabric Services System. To correct this, delete the Fabric Services System resource.
  - `MISCONFIGURED_RESOURCE`: Connect has a different configuration than the Fabric Services System. To correct this, update the Fabric Services System resource.
  - `EVPN_UNDEPLOYED:` Connect detected that one of the tenants/EVPNs it manages is not deployed correctly.

SubAudits

With the introduction of the PLUGIN_ONLY and FULL scope, Connect offers a new system to report SubAudits. The Audit object is calculated from several SubAudit objects. In case of a FULL Audit this is:

- the Plugin-performed Audit and report
- the Connect-performed Audit and report for the Fabric Services System REST layer.
- the Connect-performed Audit for LLDP information (or Topology Audit)

Currently the following types of SubAudit are available in Connect:

- **CONNECT:** The SubAudit process that concerns the data consistency between Connect and FSS.
- **TOPOLOGY:** The SubAudit process that concerns the data consistency between Connect and the LLDP information in SRL.
- **PLUGIN:** The SubAudit process that concerns the data consistency between Plugin and Connect.

The SubAudits of a particular audit can be queried using a GET request on the plugin API of Connect. An example can be found below:

```
REQUEST: GET http://localhost/rest/connect/api/v1/plugins/subaudits?parentAuditID=
454820455069516076

RESPONSE:

[
  {
    "parentAuditID":"454820455069516076",
    "type":"CONNECT",
    "id":"454820455086358828",
    "enqueueTime":"2023-04-04T15:55:16.62Z",
    "endTime":"2023-04-04T15:55:20.198Z",
    "status":"Success",
    "failureReason":"",
    "dryRun":false,
    "report":[],
    "deploymentId":"454816541532225836",
    "scope":"ERROR_ONLY",
    "totalNumberOfDiscrepancies":2,
    "totalNumberOfSuccessfulDiscrepancies":2,
    "totalNumberOfFailedDiscrepancies":0
  }
]
```

7.6 Common API examples

Setting the AdminUp status for a deployment

The following example shows the API call used to setting a deployment's AdminUp state to True.

First, query the deployments to find the relevant deployment:

```
REQUEST: GET http://fss.nokia.com/rest/connect/api/v1/admin/deployments?name=test

RESPONSE:
[{
  "id": "421602784727531649",
  "name": "test",
  "pluginID": "421602784693977217",
  "description": "test:description",
  "adminUp": false,
  "settings": {},
  "externalId": "test:deployment:ext_id",
  "status": "Deployed"
}]
```

Alternatively, query the deployment with a specific ID to verify the current AdminUp status:

```
REQUEST: GET http://fss.nokia.com/rest/connect/api/v1/admin/deployments/421602784727531649

RESPONSE:
{
  "id": "421602784727531649",
  "name": "test",
  "pluginID": "421602784693977217",
```

```
  "description": "test:description",
  "adminUp": false,
  "settings": {},
  "externalId": "test:deployment:ext_id",
  "status": "Deployed"
}
```

Then, set the AdminUp status to True:

```
REQUEST: PUT http://fss.nokia.com/rest/connect/api/v1/admin/deployments/421602784727531649

{
  "id": "421602784727531649",
  "name": "Updated:test:deployment",
  "pluginID": "421602784693977217",
  "description": "Updated:test:deployment:description",
  "adminUp": true,
  "settings": {},
  "externalId": "test:deployment:ext_id",
  "status": "Deployed"
}

RESPONSE:
{
  "id": "421602784727531649",
  "name": "Updated:test:deployment",
  "pluginID": "421602784693977217",
  "description": "Updated:test:deployment:description",
  "adminUp": true,
  "settings": {},
  "externalId": "test:deployment:ext_id",
  "status": "Deployed"
}
```

Creating a Deployment (for VMware Plugin)

The VMware plugin requires the operator to create a Deployment, as it is not tied to the VCenter itself, but rather runs in the Fabric Services System environment.

First, get the Plugin for which you are creating a Deployment:

```
REQUEST: GET http://fss.nokia.com/rest/connect/api/v1/admin/connecttypes

RESPONSE:
["OpenStack", "k8s-connect", "vmware"]
```

In this example, we are trying to create a Deployment on VMware. To do this, we first GET all available VMware plugins:

```
REQUEST: GET http://fss.nokia.com/rest/connect/api/v1/admin/plugins?connecttype=vmware

RESPONSE:
[
{ "id": "436378098703794176",
   "connectType": "vmware",
   "name": "vmware",
   "externalId": "vmware",
   "possibleSettings": [
```

```
      { "name": "host", "required": true, "description": "vCenter host", "unique": true,
 "example": "vmware.example.net", "encrypted": false },
      { "name": "username", "required": true, "description": "vCenter user", "unique": false,
 "example": "admin", "encrypted": false },
      { "name": "password", "required": true, "description": "Password of the vCenter user",
 "unique": false, "example": "secret", "encrypted": true }
   ],
   "supportsHeartbeat": true,
   "heartbeatInterval": 1,
   "supportedActionables": [ "DEPLOYMENT_CREATED", "DEPLOYMENT_DELETED", "DEPLOYMENT_
UPDATED" ],
    "status": "Deployed" }
]
```

Based on the information contained in the plugin, we can create a Deployment. To create a Deployment, we need to follow the rules the plugin specified in the settings field. In this example we need the "host", "username" and "password" settings.

```
REQUEST: POST http://fss.nokia.com/rest/connect/api/v1/admin/deployments/

{ "adminUp": true,
  "description": "demo-deployment",
  "externalId": "demo-external-id",
  "name": "demo-deployment",
  "pluginID": "436378098703794176",
  "settings": { "password": "PASSWORD", "host": "vsphere", "username":
 "administrator@vsphere.local" }
}

RESPONSE:
{ "id": "436401371051196416",
  "name": "demo-deployment",
  "pluginID": "436378098703794176",
  "description": "demo-deployment",
  "adminUp": true,
  "settings": { "host": "vsphere.local", "password": "PASSWORD", "username":
 "administrator@vsphere.local" },
  "externalId": "demo-external-id",
  "status": "Deployed"
}
```

7.7 Connect service and plugin alarms

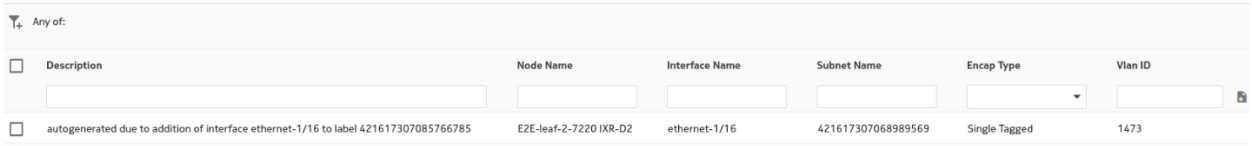
The Connect service and the different Connect plugins can raise alarms if something is wrong in the environment that needs to be reported to the network or cloud operators. These alarms will typically need an intervention from an operator to resolve the situation, sometimes needing one of the audit capabilities to be executed.

See the Alarms section of the *Fabric Services System User Guide* for more information about the supported alarms.

7.8 Common troubleshooting

Sub-interfaces not created on the Fabric Services System

When a plugin creates HostPortLabelSubnetAssociations and HostPortLabelHostPortAssociations for the same HostPort, you should see automatically generated sub-interfaces in the Fabric Services System:



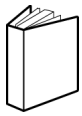
If this does not occur, check the following:

- Have alarms been created? This could indicate that the tenant did not deploy.
- Is the relevant tenant in an error state? This could indicate that the tenant did not deploy.
- Are the HostPortLabelSubnetAssociations and HostPortLabelHostPortAssociations deployed? If not, check the HostPort related to this:
- Is the HostPort in the Deployed state?
  - If the HostPort is in the Error state, something went wrong while creating the edgeLink association.
  - If the HostPort in the Deploying state, there is no EdgeMap detected. This means the fabric LLDP service did not discover the relevant <Host,Nic> pair attached to the leaf nodes. This could indicate a problem in LLDP discovery or a disconnect between the information found in the LLDP information coming from the fabric and the HostPort information coming from the plugin.
- List all EdgeMaps and check which ones are missing for your Host:

```
GET http://fss.nokia.com/rest/connect/api/v1/admin/edgemaps?hostName=fakeHostName

[
  {
    "id": "419576130018741546",
    "hostName": "fakeHostName",
    "hostPortName": "eth135",
    "edgeIntentId": "419112045749731328",
    "edgeFabricId": "419112045766508544",
    "edgeNodeId": "419112047393898496",
    "edgeNodeName": "E2E-leaf-1-7220 IXR-D2",
    "edgeIfName": "ethernet-1/35",
    "edgeNeighbor": "fakeHostName"
  }
]
```

Customer document and product support



Customer documentation

Customer documentation welcome page

Technical support

Product support portal

Documentation feedback

Customer documentation feedback