NOKIA

Fabric Services System

FABRIC SERVICES SYSTEM CONNECT GUIDE

RELEASE 22.12.1

3HE 19140 AAAA TQZZA
Issue 1.0

December 2022

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 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 through 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. It is divided into two main components:

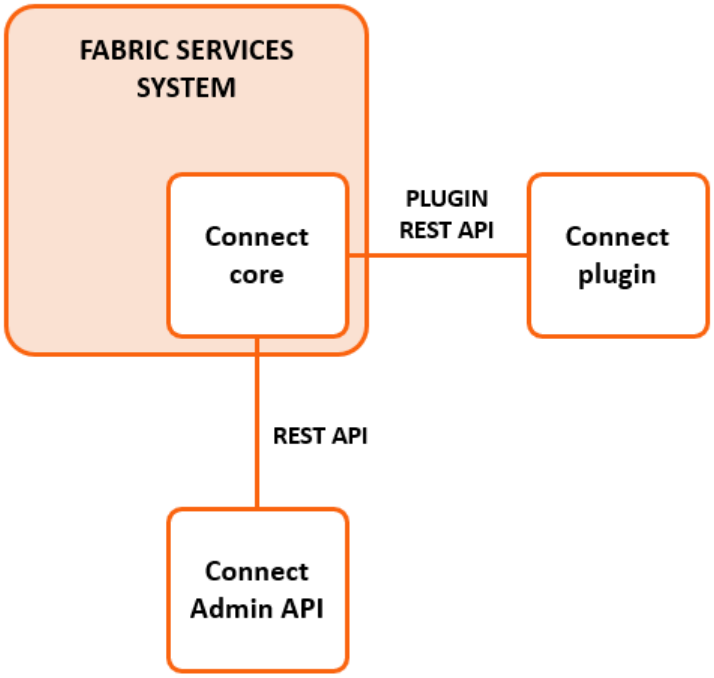
- the Connect service
- Connect plugins

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

3.1 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 are 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:

- OpenStack usage for Fabric Services System managed networks
- VMware Usage for Fabric Services System managed networks

3.2 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

4.1 Overview

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.

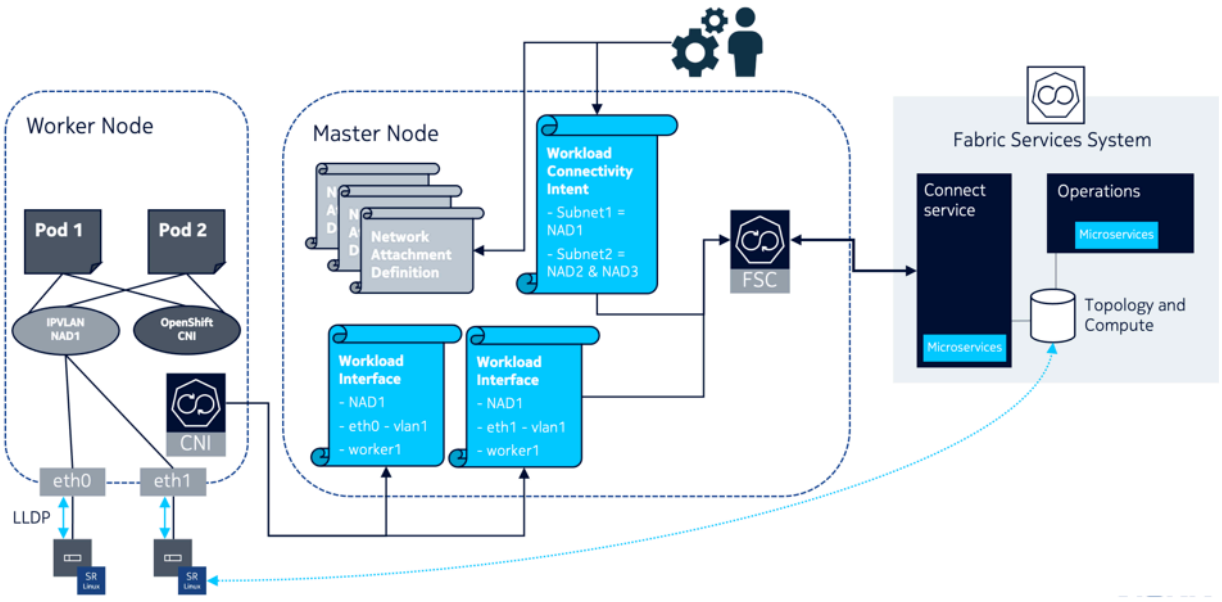
It provides the following advantages and capabilities:

- 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

4.1.1 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 monitors for the management of 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 physical interface is present in the Kubernetes API for the combination of that information (Hostname, Network Attachment Definition name, physical interface 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 (WI) 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
- physical interface
- 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 done through 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 (example: fsc-v22.12.1-46.tar.gz) which contains the following files:

- `fsc-*-<release>-images.tar`: The container images for the different components.
- `fsc-charts-<release>.tgz`: A tar ball containing the Helm charts for the installation of the integration components.
- `fsc-installer.sh`: Utility that can be used to upload the images and charts to a registry.

Using the installer script to push the container images and charts

Through the fsc-installer.sh script, it is possible 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**: Username 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 OpenShift supporting objects

This section describes the following object that support the OpenShift plugin:

- Helm chart values
- Local values .yaml file

Helm chart values

The Helm charts for the Fabric Services System integration for OpenShift have the following structure and properties. Default values are provided.

```
fss-fsc:
  replicaCount: 1 # Number of replicas of FSC (do not change)
  image:
    repository: localhost # Container repository
    pullPolicy: IfNotPresent # Pull policy
    tag: v22.12.1-46 # Version to deploy
    mgrImageName: fsc-manager # FSC Manager image name
    cniImageName: fsc-cni # FSC CNI image name
    certImageName: fsc-cert # FSC Cert service image name
  nameOverride: "" # Override the (do not change)
  serviceAccount: # Do not change anything in this section
    name: "" # Set the name of the FSC service Account name
  global: # Do not change anything in this section
    crdGroup: ""
    openShift: true # Must be true
  fscInfo:
    dockerConfig: "" # base64 encoded secret for accessing the container registry
  cniInfo:
    log:
      level: info # trace, debug, info, warning, error, fatal, panic
      genFile: "true" # Whether to save a file or not
      maxAge: "7" # Max age in days for log file
      maxBackup: "3" # Number of files to keep
      maxSize: "100" # Max size of log file in MB
      path: /var/log/fsc-cni.log # Path of the log file
    injectCni: "true" # Whether to inject the CNI info in each NAD
    maxUnavailable: 3 # Max unavailable CNI daemonsets during upgrade
  fssInfo:
    hostName: "fss.nokia.com" # Fabric Services System hostname
    ipAddr: "127.0.0.1" # Fabric Services System IP
    port: "8090" # Fabric Services System port
    userId: "" # Fabric Services System username
    password: "" # Fabric Services System password
    tlsEnable: false # Whether Fabric Services System uses TLS
    pluginId: "k8s-plugin-id" # Must be a unique value across the environment
    pluginName: "k8s-plugin-name" # Human readable plugin name
    deploymentName: "k8s-deployment-name" # Human readable deployment name
```

```
    deploymentDescription: "k8s connect deployment" # Human readable description
    tlsSkipVerify: true # Whether to verify the server certificate
    tlsCertData: ""  # TLS Certificate data – valid only if tlsSkipVerify is true
    tlsCertKey: "" # TLS Certificate key – No longer used, but a value must be provided
  mgrInfo: # Do not change anything in this section, except for log if necessary
    heartbeatInterval: "3" # Keep-alive timer
    supportsHeartbeat: "true" # Whether Connect will generate heartbeat alarms
    actionables: | # Only supported value is DEPLOYMENT_UPDATED
      DEPLOYMENT_UPDATED
    log:
      level: info # trace, debug, info, warning, error, fatal, panic
      genFile: "true" # Whether to save a file or not
      maxAge: "7" # Max age in days for log file
      maxBackup: "3" # Number of files to keep
      maxSize: "100" # Max size of log file in MB
      path: /fss/data/fsc-data/logs/fsc-controller-manager.log # Log file path
  service: # Do not change anything in this section
    type: ClusterIP # Type of service created for FSC (do not change)
    port: 8443 # Port used by FSC (do not change)
    grpcPort: 50051 # Internal GRPC port of FSC (do not change)
    webHookPort: 443 # Webhook port for FSC (do not change)
```

Creating a local values yaml file

When deploying the Fabric Services System integration with OpenShift, you can use a local values .yaml file to overwrite the defaults of the `values.yaml` file provided in the charts.

The minimum changes are:

```
fss-fsc:
  image:
    repository: localhost # Container repository
    tag: v22.12.1-46 # Version to deploy
  fscInfo:
    dockerConfig: "<base64 encoded secret>" # today a secret is required
  fssInfo:
    hostName: "fss.domain.tld"
    ipAddr: "192.0.2.100"
    port: "443"
    userId: "fscuser"
    password: "secret-password"
    tlsEnable: true
    pluginId: "ocp-fsc-01"
    pluginName: "OpenShift FSC 01"
    deploymentName: "ocp-fsc-01-deployment"
    deploymentDescription: "First OpenShift Cluster with FSC"
    tlsSkipVerify: true
    tlsCertData: "" # the base64 encoded certificate file
    tlsCertKey: "Tm9uZQo="
```

Note: This release supports only a container registry that requires authentication through a secret.

Note: In this release, even with `tlsSkipVerify` set to true, the tlsCertData and tlsCertKey require a value. For the tlsCertKey, any valid base64 encoded value can be entered as it is not used at any time.

4.2.2 Installing and configuring the OpenShift and Kubernetes plugin

Prerequisites

Create a local values file as described in Creating a local values yaml file

About this task

This procedure describes how to install and configure the OpenShift and Kubernetes plugins for Connect.

Procedure

Step 1. Run the installation script with the following command:

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

Example

```
$ ./fsc-installer.sh -r registry.domain.tld/fsc -u registryuser -e https://
repository.domain.tld/fsc-charts/
Enter Registry Password:
WARNING! Using --password via the CLI is insecure. Use --password-stdin.
WARNING! Your password will be stored unencrypted in /home/fsc-helper/.docker/
config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
Docker Login success
Is the credentials for helm repo same as the docker registry?. y|n: n
Enter Helm Repo Userid: fsc-charts-rw
4ba5346beac5: Loading layer [==================================================>]
 1.536kB/1.536kB
00c259f593c6: Loading layer [==================================================>]
 14.98MB/14.98MB
14dcfc27fc3e: Loading layer [==================================================>]
 55.53MB/55.53MB
fdab6045c7d5: Loading layer [==================================================>]
 13.16MB/13.16MB
Loaded image: fsc-manager:v22.12.1-47
336eecb6d6f7: Loading layer [==================================================>]
 1.536kB/1.536kB
70e11503e7f0: Loading layer [==================================================>]
 11.39MB/11.39MB
352689f7e766: Loading layer [==================================================>]
 43.5MB/43.5MB
Loaded image: fsc-cni:v22.12.1-47
b3c73d8d2b99: Loading layer [==================================================>]
 1.536kB/1.536kB
057930989262: Loading layer [==================================================>]
 10.5MB/10.5MB
aa9b2d5cc420: Loading layer [==================================================>]
 40.21MB/40.21MB
Loaded image: fsc-cert:v22.12.1-47
The push refers to repository [registry.domain.tld/fsc/fsc-manager]
fdab6045c7d5: Pushed
14dcfc27fc3e: Pushed
00c259f593c6: Pushed
4ba5346beac5: Pushed
5510fa9aff79: Layer already exists
daeca4c64c94: Layer already exists
c31017bc1cae: Layer already exists
8bd900abc571: Layer already exists
```

```
994393dc58e7: Layer already exists
v22.12.1-47: digest:
 sha256:d25836c54943ce2c5b50d0265e941d13b89172e9840551e1be459fbdbc649ad0 size: 2205
The push refers to repository [registry.domain.tld/fsc/fsc-cni]
352689f7e766: Pushed
70e11503e7f0: Pushed
336eecb6d6f7: Pushed
5510fa9aff79: Layer already exists
daeca4c64c94: Layer already exists
c31017bc1cae: Layer already exists
8bd900abc571: Layer already exists
994393dc58e7: Layer already exists
v22.12.1-47: digest:
 sha256:21b3632b9bb134409a79dbbd6f8a158d52a13d483a79257a45f1fefb6009f5d0 size: 1994
The push refers to repository [registry.domain.tld/fsc/fsc-cert]
aa9b2d5cc420: Pushed
057930989262: Pushed
b3c73d8d2b99: Pushed
5510fa9aff79: Layer already exists
daeca4c64c94: Layer already exists
c31017bc1cae: Layer already exists
8bd900abc571: Layer already exists
994393dc58e7: Layer already exists
v22.12.1-47: digest:
 sha256:e4ff6b4d00e69b237161c476e7b6dcbadf3fd60e723fa696c2de74e2fd855398 size: 1994
Untagged: fsc-manager:v22.12.1-47
Untagged: fsc-cni:v22.12.1-47
Untagged: fsc-cert:v22.12.1-47
fsc-charts/Chart.yaml
fsc-charts/values.yaml
fsc-charts/.helmignore
fsc-charts/charts/fss-fsc/Chart.yaml
fsc-charts/charts/fss-fsc/values.yaml
fsc-charts/charts/fss-fsc/templates/_helpers.tpl
fsc-charts/charts/fss-fsc/templates/configmap.yaml
fsc-charts/charts/fss-fsc/templates/crd.yaml
fsc-charts/charts/fss-fsc/templates/deployment.yaml
fsc-charts/charts/fss-fsc/templates/scc.yaml
fsc-charts/charts/fss-fsc/templates/secret.yaml
fsc-charts/charts/fss-fsc/templates/service.yaml
fsc-charts/charts/fss-fsc/templates/serviceaccount.yaml
```

Step 2. Do one of the following:

If	Then
You have created a local values file (see Prerequisites),	Go to step 3.
You used the fsc-installer.sh script to upload the charts to a Helm repository, or another tool was used to do so,	Go to step 5.

Step 3. Use the following command to deploy the Fabric Services System integration with OpenShift:

```
helm install -f local-values.yaml fsc-deployment
          fsc-charts-v22.12.1-46.tgz
```

Step 4. Go to step 6.

Step 5. Use the following commands to deploy the Fabric Services System integration with OpenShift

```
# helm repo add fscrepo https://helm-repo.domain.tld/repository/fsp-charts/ --username
 "fsp-charts-ro" --password  "******"
  "fscrepo" 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 "fscrepo" chart repository
Update Complete. ??Happy Helming!??

# helm install prod fsprepo/fsc-charts --version v22.12.1-46 -f local.yaml
```

Step 6. Verify that the installation was successful by checking that the required pods are in the Running or Completed state as shown below:

```
# oc get pods -n fsc-system
NAME                                              READY STATUS    RESTARTS AGE
prod-fss-fsc-cert-manager                         0/1   Completed 0        35s
prod-fss-fsc-cni-ds-wrqrs                         1/1   Running   0        35s
prod-fss-fsc-cni-ds-wtlf7                         1/1   Running   0        35s
prod-fss-fsc-cni-ds-xt9qj                         1/1   Running   0        35s
prod-fss-fsc-controller-manager-5898c6b5b9-xh459 2/2   Running   0        35s
```

Note: The Cert Manager pod should be in the Completed state for a successful deployment.

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 during the creation of a Network Attachment Definition. If a change is applied that removes the injected Helper CNI plugin configuration, it is not added again.

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.

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 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 Config resource

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

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

```
apiVersion: fsc.fss.nokia.com/v1
kind: FscGlobalConfig
metadata:
  name: fscglobal
  namespace: fsc-system
status:
  ControllerAuditStatus: Done
```

```
ControllerConnectStatus: StateClientRunning
FscCniInjectionMode: Automatic
```

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

5.1 Overview

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.

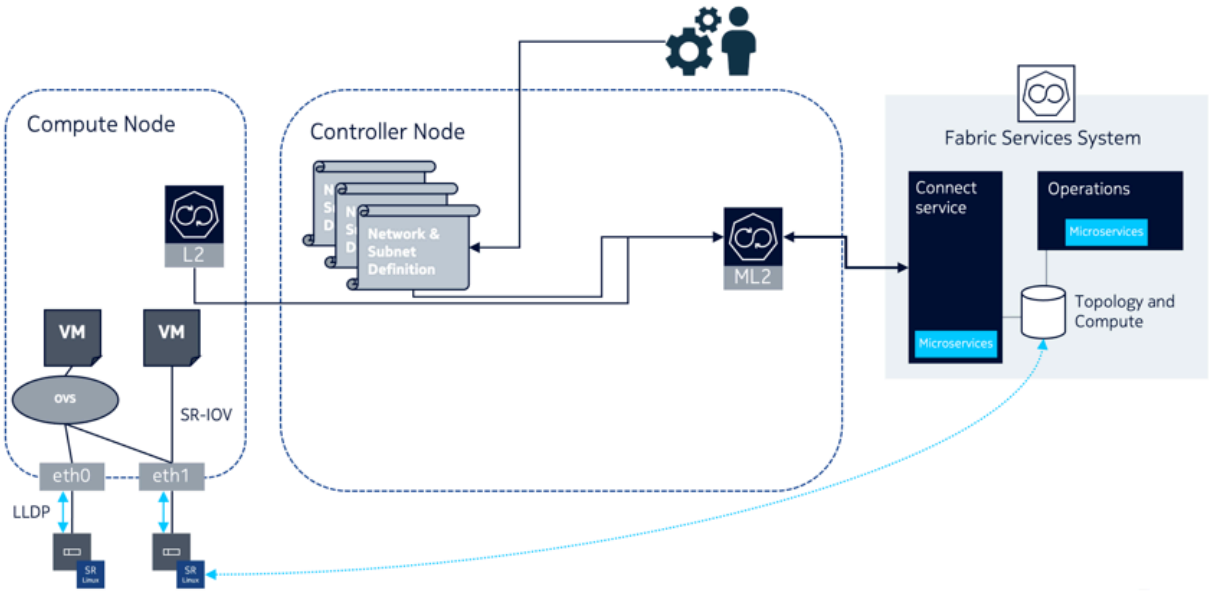
It provides the following advantages and capabilities:

- 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.

5.1.1 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.

5.1.2 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.

5.2 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.

5.3 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.

In order 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 additional information, see Setting the AdminUp status for a deployment.

See also Managing plugins and deployments.

5.4 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.

5.5 Usage

5.5.1 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.

5.5.2 OpenStack usage for Fabric Services System managed networks

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

In order to support Fabric Services System managed networking, two proprietary extensions have been added to Network:

- `fss-workload-evpn`
- `fss-subnet`

Fabric Services System managed networking uses the following management flow:

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

   As an example, assume the Workload VPN has the UUID `410559942890618880` and the subnet with in it has the UUID `410560233203564544`.
3. 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 demo-project --fss-workload-evpn
410559942890618880 --fss-subnet 410560233203564544 demo-network
```

Creating a subnet within this network follows the standard API syntax. For example:

```
openstack subnet create --network demo-network --subnet-range 10.10.1.0/24
       demo-subnet
```

Neutron port and Nova server creation within this network and subnet also use the 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` parameter.
  This means that a new Connect tenant is created for every `fss-workload-evpn` passed to network create.
- 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 VPN can by consumed by OpenStack by an 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.

5.5.3 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.

5.5.4 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.

5.5.5 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.

5.5.6 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.

5.5.7 Trunk ports

Trunk ports are supported the OpenStack/Connect/Fabric Services System solution. They follow the standard OpenStack model.

6 VMware plugin

6.1 VMware overview

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.

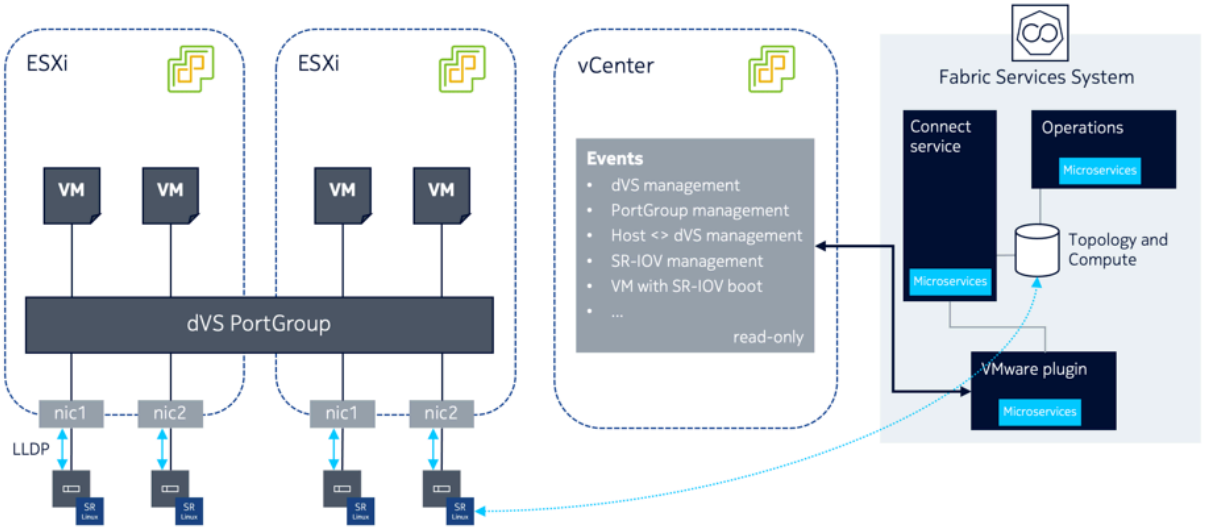
It provides the following advantages and capabilities:

- 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.

6.1.1 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

In order 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 is done using the following API call :

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

Within the POST body of this request, you must include the settings that the plugins require. The possible settings can be retrieved by interrogating the plugin..

Some settings are mandatory; others are optional.

Specifically for the VMware plugin, these settings are :

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 Usage

6.4.1 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.

6.4.2 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.

In order 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 attribute(s) 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.

6.5 VMware audit

The VMware plugin has automated auditing and correcting support from vSphere to Connect. In this case, the vSphere config is all times the master config.

This means:

- Any missing config in Connect for a config present in vSphere is automatically added to Connect
- Any dangling config in Connect for a config not present in vSphere is automatically removed from Connect
- Any incorrect config in Connect for a config present in vSphere will be automatically updated in Connect

Fabric Services System managed resources, though, 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 upon any of the following triggers :

- Upon startup of the VMware plugin (specifically, on 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 creation of a new deployment as administratively Up.
- Upon updating an existing administratively down deployment to administratively up (that is, setting the `adminUp` field to True)

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**

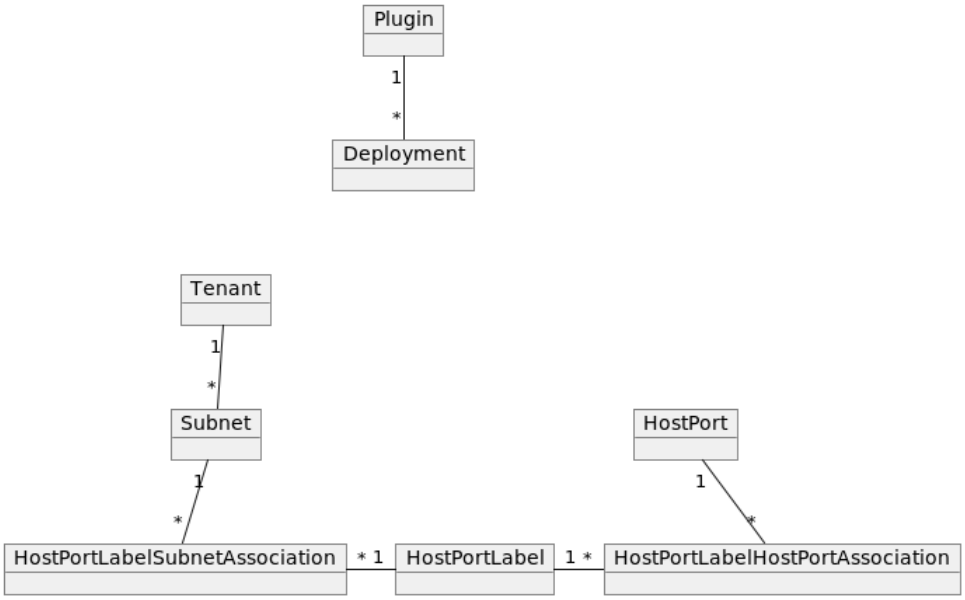
Version is incremented after every Create or Update operation on a resource.

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 rather than 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. Rather than 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 regarding the connected hypervisors or servers to the leaf fabric.

7.1 Key concepts

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*.
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.

Concept	Definition
	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

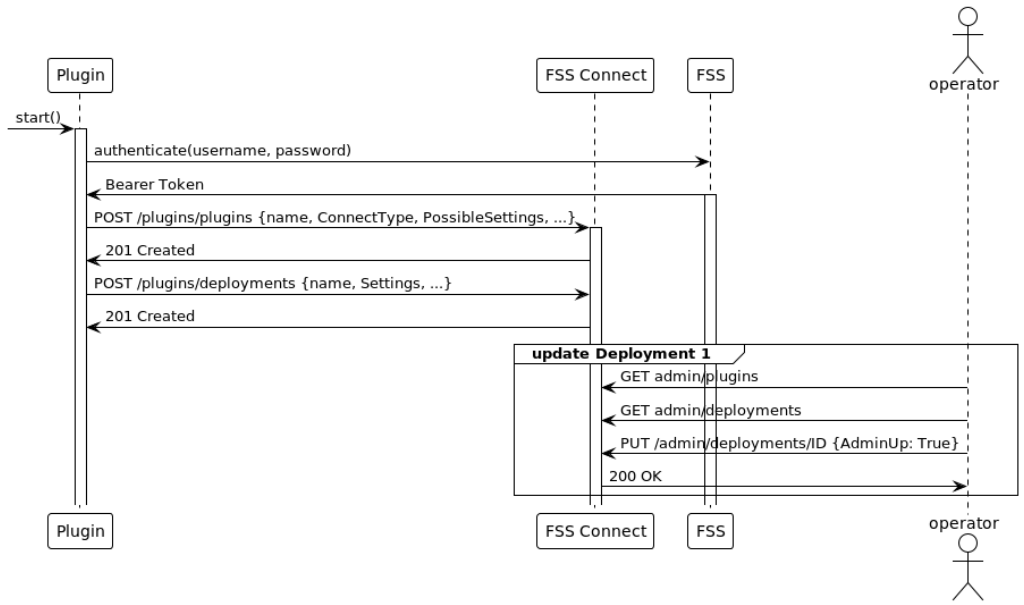
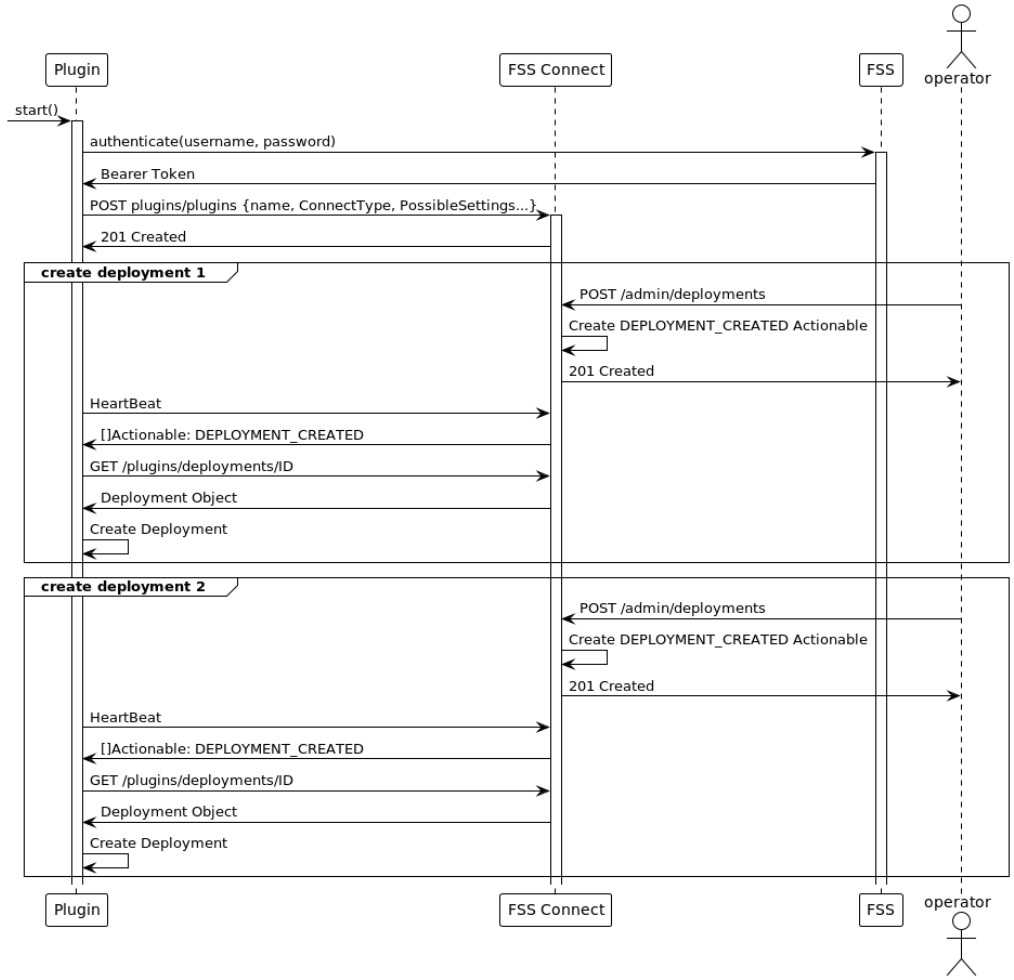


Figure 7: Deployment-agnostic plugin



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

Connect supports the following audit scopes:

- **CONNECT_ONLY**: The audit examines the full relationship between Connect and the Fabric Services System.
- **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 due to unforeseen circumstances. Note that this kind of audit will not detect DANGLING_RESOURCE deviations on the Fabric Services System.

```
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.
- `dryRun` is an optional boolean, indicating that only a report must be printed instead of executing the audit actions.
- `Report` is a list of actions taken by the audit.
- `Report` entries can be of type:
  - `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.

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" }
]
```

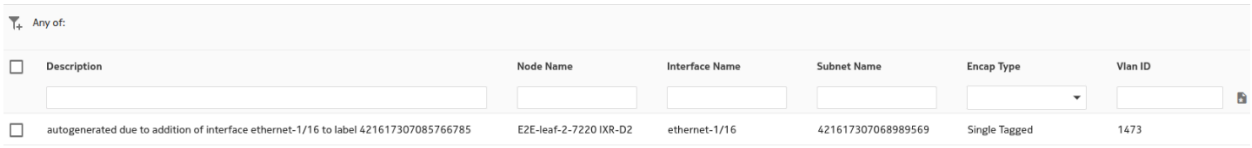
Based on the information contained in the plugin, we can create a Deployment. To create a Deployment, we need to follow the rules the plugin specified in the settings field. In this example we need the "host", "username" and "password" settings.

```
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 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:

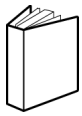


If this does not occur, check the following:

- 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