RIFT.ware™ version 8.3.0.0.118520

Release Notes

January 2021

Notice

Table of Contents

RIFT.ware 8.3.0.0 Release Notes

This guide describes the RIFT.ware 8.3.0.0 release, including new features, fixed and known issues, with their workarounds.

New and Changed Features for 8.3.0.0

RIFT.ware version 8.3.0.0 introduces enhancements to improve management.

Feature	PFRs and JIRAs
Cloud Platform Enhancements	RIFT-31051 RIFT-30730
Google GKE Support	PFR-654 RIFT-30746
Kubernetes Enhancements	RIFT-30158
Map Existing Service Elements to Existing Resources Enhancements Support for CNF	PFR-665 RIFT-30748
Map Existing Service Elements to Existing Resources Enhancements Support for SOL003	PFR-620 RIFT-30745
Map Existing Service Elements to Existing Resources Improvements	PFR-676 RIFT-31437
Network Slicing Support	PFR-643 PFR-672 PFR-675 RIFT-30744 RIFT-28221
RIFT.ware Disaster Recovery Support	RIFT-29547

Cloud Platform Enhancements

Release 8.3 introduces the following enhancements to our cloud native platform support.

External Address and other Helm Chart Updates

In release 8.2, there were multiple external addresses (externalAddress). There was one address for the RIFT.ware UI and another for the RIFT.ware Grafana dashboards. The externalAddress could be both IP address and FQDN. In this release, there is only one externalAddress for all of RIFT.ware. The externalAddress can only be FQDN. If the FQDN is not resolvable in the network, then it is necessary to provide the mapping for the DNS resolution.

Note: When using externalAddress, RIFT.ware access will only work using the FQDN. Access will be rejected using a mapped IP Address.

The RIFT.ware Launchpad will not come up if the externalAddress is configured incorrectly. If Launchpad fails because the external address configure is incorrect, then the Launchpad pod goes into the 'Init:Error' or 'Init:CrashLoopBackOff' state.

NodePort mode: When Launchpad is installed in NodePort mode, it is necessary to configure the externalAddress in the values.yaml.

LoadBalancer mode: When Launchpad is installed in LoadBalancer mode, it is not necessary to set the externalAddress in the values.yaml. In this case, the IP address assigned by LoadBalancer can be used to access RIFT.ware. The RIFT_EXTERNAL_ADDRESS preference is externalAddress and then LoadBalancer IP.

Port Changes

In release 8.3, the RIFT.ware UI and Grafana can be accessed using the same port number. The UI, API server and the Auth Server are all behind a single proxy port. For example, if the LoadBalancer assigned IP to the ingress controller is 10.64.99.0, then the:

- RIFT.ware UI access is https://10.64.99.0
- API server access is https://10.64.99.0/api/... or whichever subpaths are currently proxied.
- Authorization access is https://10.64.99.0/token and other subpaths for auth service.
- Grafana access is https://10.64.99.0/grafana

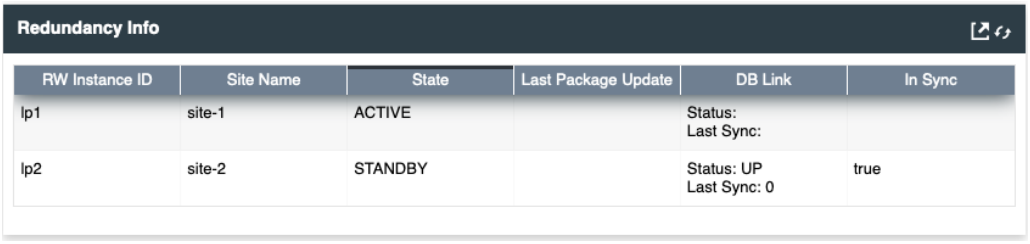
Certificate Changes

In release 8.3, RIFT.ware certificates types are now all external. The certificate type option still appears in the RIFT.ware UI and model. When configuring a certificate in the UI, choose External for the Certificate Type.

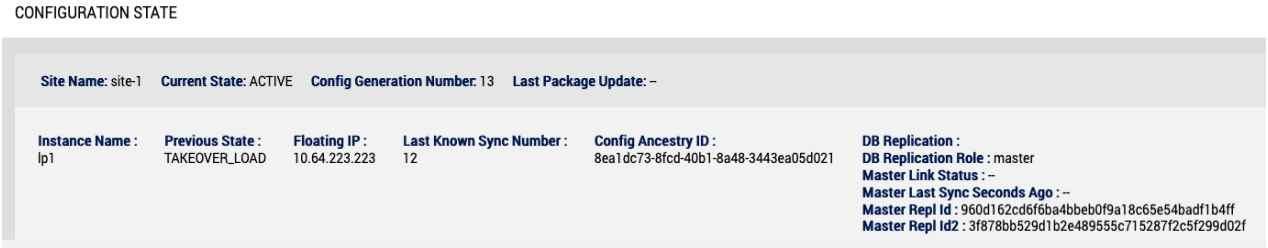
UI Enhancements

There are several enhancements on the RIFT.ware UI **HOME** page in release 8.3. The Redis Statistics and Nats Statistics widgets no longer appear on the **HOME** page of the RIFT.ware UI. The Redis and Nats Statistics can now be found on the Grafana RIFT.ware Dashboards.

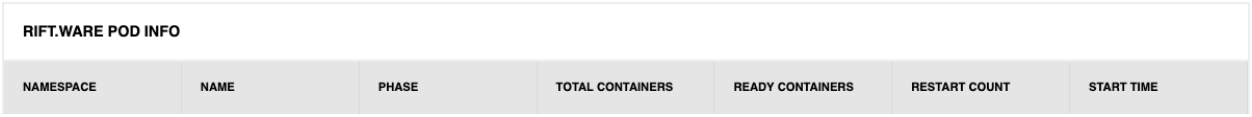
There is also a new DB Link column in the Redundancy widget.



Click  to open the Redundancy page. Choose the **STATUS** tab see the new DB Replication details. Select a Site Name to see the new DB Republication and DP Republic Role data.



This release also includes a change to the **ABOUT** section of the UI. The **ADMINISTRATION** > **ABOUT** page now displays the pod status (NAMPESPACE, NAME, PHASE, TOTAL CONTAINERS, READY CONTAINERS, RESTART COUNT and START TIME) for Launchpad.



RIFT.ware Observability Enhancements

This release includes additional Grafana Dashboards. It is now possible to use the dashboards to view HAProxy, Ingress Controller, RW.Redis, RW.NATS and RW.EventsDB metrics. These dashboards are not customized for RIFT.ware. For additional information on Grafana Dashboards see, RIFT.ware Observability.

Upgrade Process Changes

This release includes two new steps that must be completed prior to upgrading from release 8.2 to release 8.3.

- Services Deployment Changes
- New PVs and PVCs

Services Deployment Changes

Prior to upgrading from release 8.2 to 8.3, an operator must change the following modes in release 8.2 in the CLI:

- NodePort
- LoadBalancer
- ClusterIP

You do not interact with RIFT.ware from rift-shell. You interact with RIFT.ware through the RIFT UI or the RIFT CLI (RW.CLI). To enter the RIFT.ware CLI (rwcli) in a new terminal, run the following series of commands:

```
$ cd /usr/rift
$ sudo -H ./rift-shell -r -i /usr/rift -a /usr/rift/.artifacts
$ rwcli
```

The NETCONF username is `admin`. Contact RIFT Services and Support at support@riftio.com to obtain the proper password.

Note: You can safely ignore any errors output to the terminal, as long as you see the rift# prompt.

The above modes must be changed into ClusterIP with no NodePort configuration. See an 8.2 service deployment below.

```
xxx3-lp-alertmgr      ClusterIP      10.101.204.227   <none>
9093/TCP
16d
xxx3-lp-grafana       LoadBalancer   10.98.202.205    10.64.223.222
3000:30846/TCP
16d
```

```
xxx3-lp-launchpad    NodePort       10.98.49.154     <none>
8008:30108/TCP,8009:30109/TCP,8443:30143/TCP,8014:30114/TCP,8006:30163
/TCP   16d
```

In the above example, an operator must edit xxx3-lp-grafana and xxx3-lp-launchpad to Cluster IP using the `kubectl edit` command for release 8.3.

```
kubectl -n xxx3 edit svc xxx3-lp-launchpad
```

In the below example, remove the nodePort information and change the type from NodePort to ClusterIP. The step must be completed for all services with are not ClusterIP type.

```
spec:
  clusterIP: 10.98.49.154
  externalTrafficPolicy: Cluster
  ports:
  - name: rwrest
    ~~nodePort: 30108.~~
    port: 8008
    protocol: TCP
    targetPort: 8008
  - name: rwauth
    ~~nodePort: 30109.~~
    port: 8009
    protocol: TCP
    targetPort: 8009
  - name: rwui
    ~~nodePort: 30143.~~
    port: 8443
    protocol: TCP
    targetPort: 8443
  - name: rwredis
    ~~nodePort: 30114.~~
    port: 8014
    protocol: TCP
    targetPort: 8014
  - name: rwmongo
    ~~nodePort: 30163~~
    port: 8006
    protocol: TCP
    targetPort: 8006
  selector:
    app.kubernetes.io/instance: xxx3-lp
    app.kubernetes.io/name: launchpad
  sessionAffinity: None
  type: ClusterIP Changed from NodePort
```

New PVs and PVCs

It is not possible to reuse the existing PersistentVolume (PVs) or PersistentVolumeClaim (PVCs) so operators need to create new PVs manually and mount them into the pre-upgrade job.

For example, use the following API to create new PVs and PVCs.

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: task-pv-volume
  namespace: arn-test
  labels:
    type: local
spec:
  storageClassName: manual-lp
  capacity:
    storage: 80Gi
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  hostPath:
    path: "/localdisk/xxxxxxxx/K8_new"
---
apiVersion: v1
kind: PersistentVolume
metadata:
  name: task-pv-volume-grafana
  namespace: arn-test
  labels:
    type: local
spec:
  storageClassName: manual-grafana
  capacity:
    storage: 80Gi
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  hostPath:
    path: "/localdisk/xxxxxxxx/K8_new/grafana"
---
apiVersion: v1
kind: PersistentVolume
metadata:
  name: task-pv-volume-redis
  namespace: arn-test
  labels:
    type: local
spec:
  storageClassName: manual-redis
  capacity:
    storage: 80Gi
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  hostPath:
    path: "/localdisk/xxxxxxxx/K8_new/redis"
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: rift-var-root-arn-test-launchpad-0
```

```
  namespace: arn-test
spec:
  storageClassName: manual-lp
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 80Gi
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: data-arn-test-grafana-0
  namespace: arn-test
spec:
  storageClassName: manual-grafana
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 80Gi
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: data-arn-test-redis-0
  namespace: arn-test
spec:
  storageClassName: manual-redis
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 80Gi
```

For additional upgrade information see, RIFT.ware Installation and RIFT.ware Upgrade.

Rollback Support

This release introduces rollback support from 8.3.0 onward using Helm. This rollback does not support rollbacks to releases prior to 8.3.0.

Note: When an operator rolls back to a previous version, the previous version will not include any new data created in the newer version.

For example if an operator installs 8.3.0, then 9.1.1, and 9.2.0, it is possible to rollback to any of these previously installed versions. It is not possible to rollback to any version prior to release 8.2.0. The rollback process involves pointing “rift” to a previous release in the CLI.

In the CLI, review the current list of helm revisions for RIFT.ware installs. After installing release 8.3.0, it is possible to rollback to any of the previous listed versions in the example below.

```
xxxxxxxx@xxxxxxxx-master:/localdisk/amuralid/UB18-upgrade-rollback$
helm history arn-test
REVISION       UPDATED                        STATUS         CHART
       APP VERSION    DESCRIPTION
1              Wed Dec  9 11:22:29 2020      superseded
       launchpad-8.3.0.0      8.3.0.0.117928 Install complete
2              Wed Dec  9 11:39:47 2020      deployed
       launchpad-8.3.0.0      8.3.0.0.117936 Upgrade complete
```

This example demonstrates a rollback to Version A (8.3.0.0.117928) which is revision #1. Run the following command to rollback to a previous version.

```
xxxxxxxx@xxxxxxxx-master:/localdisk/amuralid/UB18-upgrade-rollback$
helm rollback arn-test 1
Rollback was a success! Happy Helming!
```

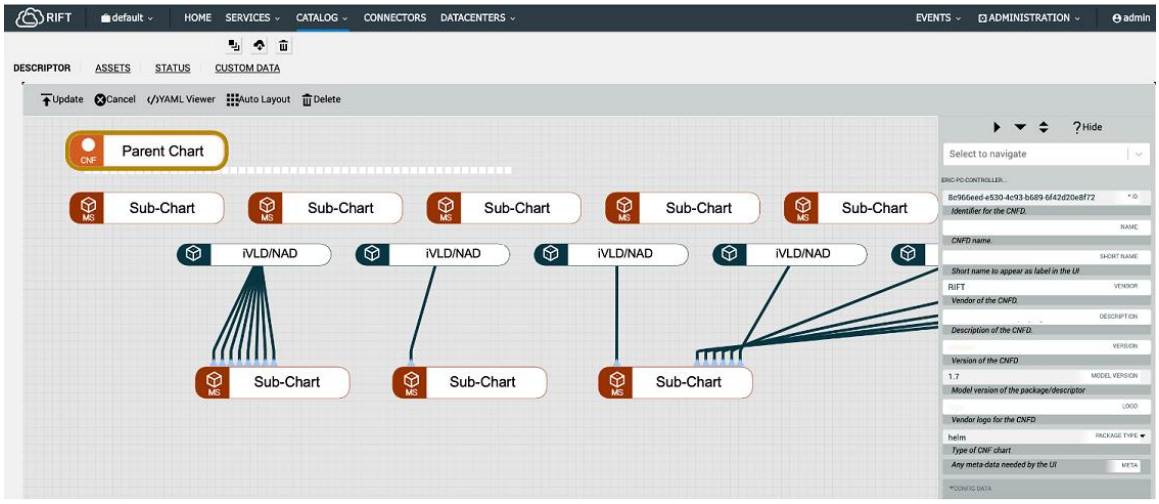
When the rollback is successful, the Launchpad pod is recreated with the older version. It also restores operational data and configuration in Redis and Confd.

CNFD UI Enhancements

In release 8.3, RIFT.ware now displays **internal-vlds** and the corresponding connections in a tiered layout on the CNFD composer section of the UI.

On the **Launchpad** menu, click **CATALOG > CNFD LIST** and select the appropriate CNFD descriptor from the catalog. Double-click the descriptor. The composer canvas displays **internal-vlds** and their corresponding connections to some microservices. On this page, an operator can make connections between the boxes.

The screenshot below captures how the UI displays **internal-vlds** and their connections to some microservices in a tiered layout.



Google GKE Support

RIFT.ware now supports orchestration of CNFs on Google Kubernetes Engine (GKE). A GKE account provides a management environment for deploying, managing and scaling containerized applications using the Google architecture. See Google Kubernetes Engine for more details on this account type.

Prior to adding a GKE account, you must have an IAM account in GCP. See Identify and Access Management documentation and Using the Google Cloud SDK installer.

Note: Audit reports are not supported for GKE accounts in release 8.3.

- Add a GKE Account
- Model Changes

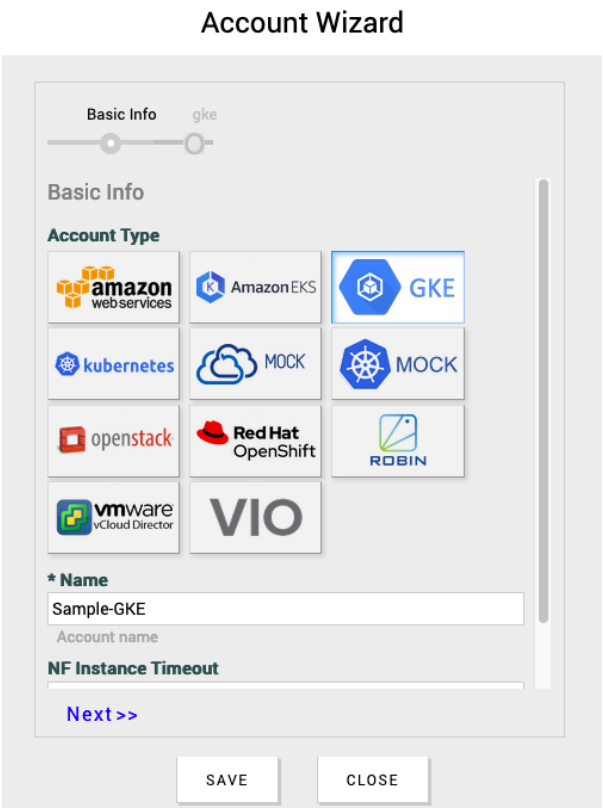
Add a GKE Account

Prior to setting up your account in the UI, you must ensure that you have access to a GKE Account. Obtain access to this account type from a GKE_admin. After setting up your access, you will receive a system password. Verify that you can log into console.cloud.google.com.

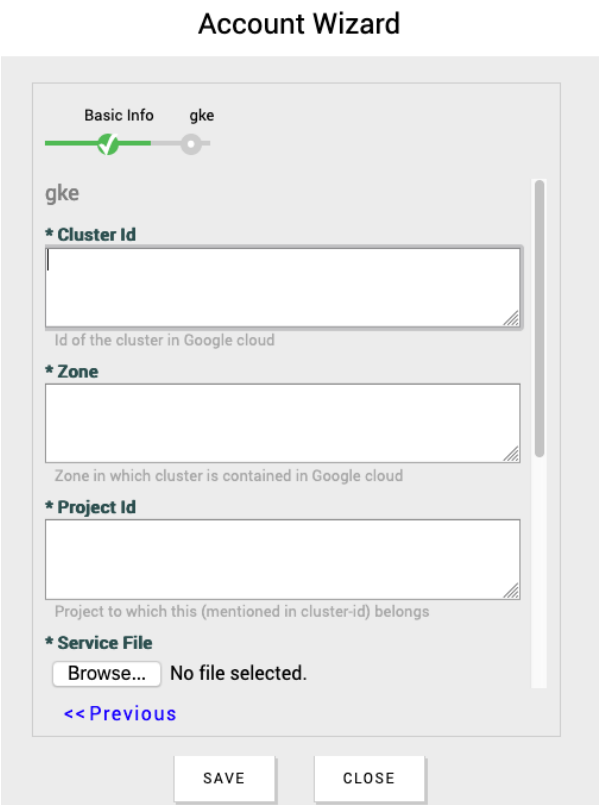
Note: The number of GKE services that an operator can instantiate at one time is limited by the resource quota in GKE for a particular project.

There is a new GKE account type in the Datacenter section of the UI.

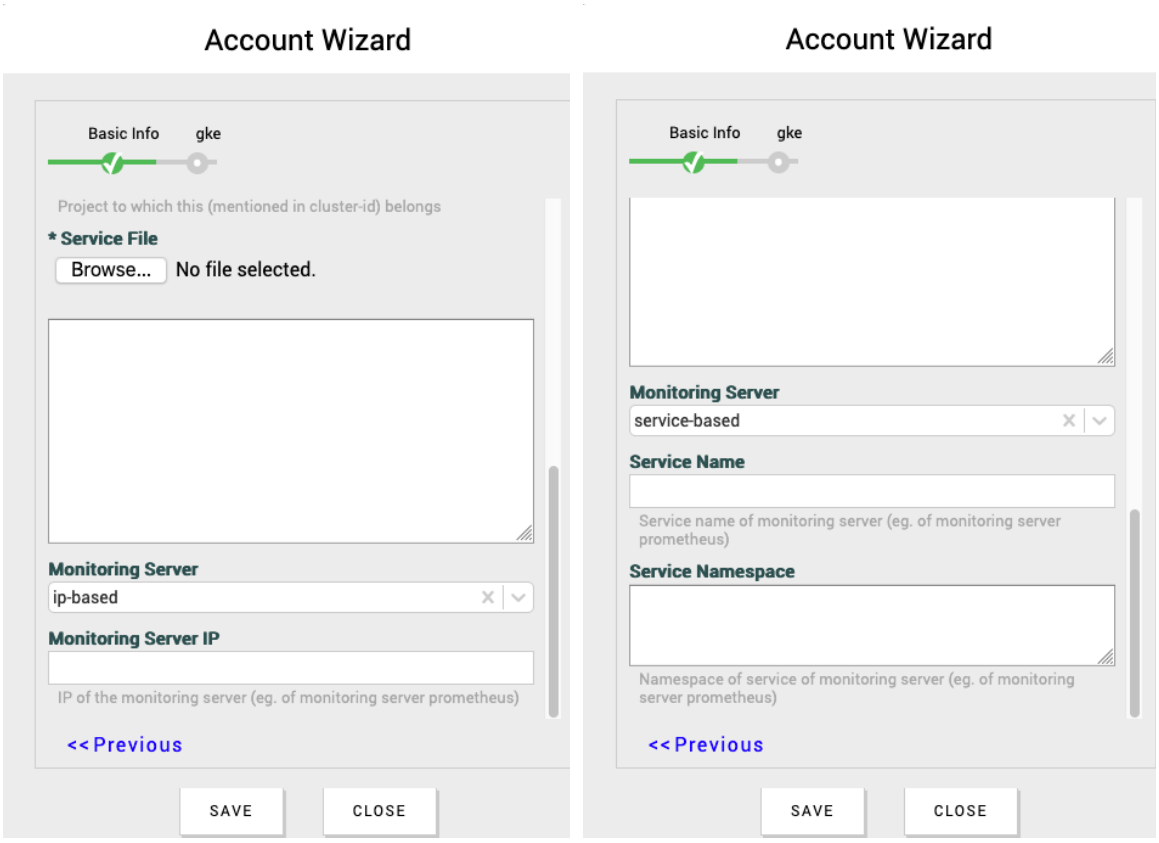
1. On the **Launchpad** menu, click **DATACENTERS**.
2. Click + to add a new datacenter. An account wizard appears.
3. Choose GKE in the **Account Type** section.
4. Add a unique account name in the ***Name** field.
5. Add the **Nf Instance Timeout** (SECONDS) information. This is the maximum time allocated for resource instantiation in the GKE account. Default is 900.



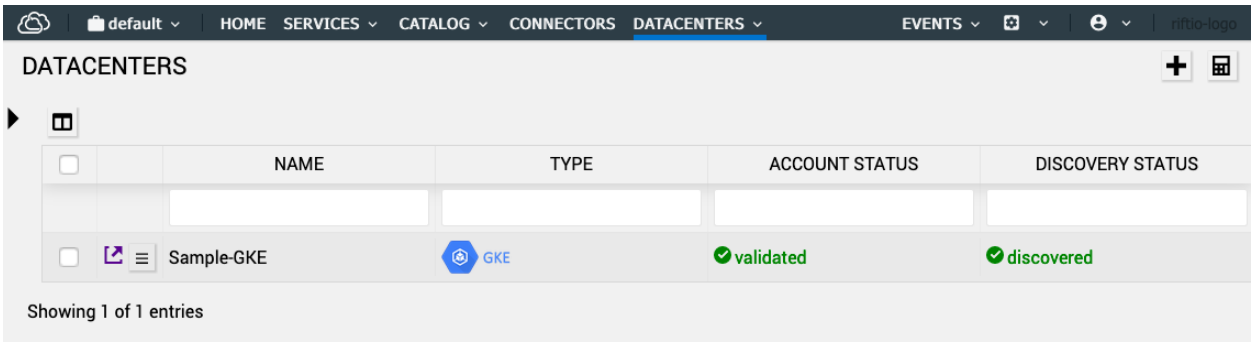
6. Click **Next >>** and provide the following GKE account details. You can find the google specific information at console.cloud.google.com.
   - ***Cluster Id**: Name of the cluster in Google Cloud.
   - ***Zone**: Location in which cluster is contained in Google cloud.
   - ***Project Id**: Project in GCP to which this (mentioned in cluster-id) belongs (see Project info –> Project ID)
   - ***Service File**: Service file is used for authentication. This service file must be provided by the operator from the IAM at console.cloud.google.com.



- **Monitoring Server**: Select ip-based or server-based.
- **ip-based**: If you choose an ip-based monitoring server, then add:
  - **Monitoring Server IP address**: IP of the monitoring server.
- **server-based**: If you choose a server-based monitoring server, then add:
  - **Service Name**: Service name of the monitoring server.
  - **Service Namespace**: Namespace of service of monitoring server.



7. Carefully check your entries and click **SAVE**. After successfully creating the account, it will appear in the **DATACENTERS** list page as validated.



Before you can instantiate a service with CNFs on a GKE account, you must add NSD and CNFD descriptions on the **Catalog** > **NSD/CNFD** List page.

Note: RIFT.ware requires that you add an ip profile and segment profile. However, GKE does not support two interfaces. After you add a GKE account, you can add generic information in the ip profile section. The segment profile requires the **VIM NETWORK NAME** but the **PROVIDER NETWORK** information is not needed for a GKE account.

Model Changes

Added a new `rw-cloud:gke` account type in `rw-cloud:account` in the `rw-cloud:cloud` Data Model. This field includes the following variables:

Name	Type	Cardinality	Description
cluster-id	string	1	Id of the cluster in the Google cloud.
zone	string	1	Zone in which a cluster is contained in the Google cloud.
project-id	string	1	Project to which this (mentioned in the cluster-id) belongs.
service-file	string	1	Service file is used for authentication.
monitoring-server-ip	union	1	IP of the monitoring server
service-name	string	1	Service name of monitoring server.
service-namespace	string	1	Namespace of service of the monitoring server.
plugin-name	string	1	Plugin name associated with an account.

Map Service Elements to Existing Resources Enhancements

This feature introduces additional options to discover resources and map them to service elements during instantiation. In release 8.3 an operator can map existing resources during instantiation with CNFs or with SOL003 based VNFs. This feature also includes more information on the **DATACENTERS** > **RESOURCE** tab to improve the process of discovering resources when mapping them to service elements during instantiation.

Map Service Elements to Existing Resources during Instantiation with CNFs

RIFT.ware now supports the ability to map existing resources during instantiation with CNFs. A user can now run Discovery on a Kubernetes Datacenter account to identify existing resources. An existing resource can then be mapped to an appropriate CNF that is part of a Network Service being instantiated by RIFT.

This release introduces a new input-parameter-xpath to customize the CNFR. There are two new fields to allow an operator to modify the CNF.

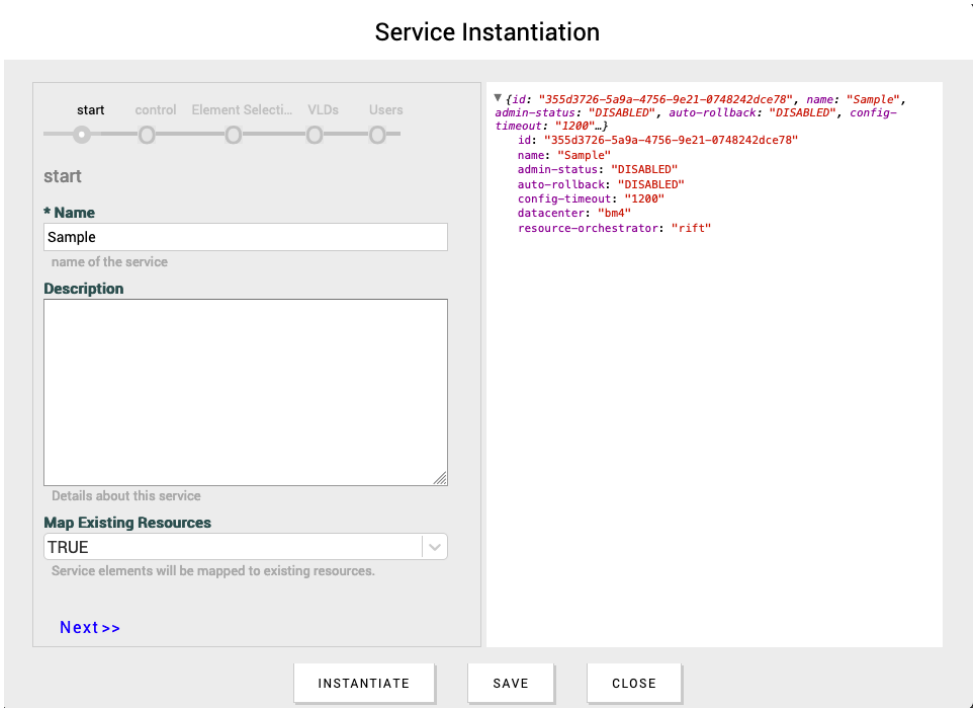
- **Preexisting-release-name**: The preexisting helm release which is used for this CNFR.
- **Delete-on-terminate**: If you mark this field as true, then the helm releases are deleted when you terminate the Network Service from RIFT.ware.

UI Enhancements

Prior to mapping service element to existing resources, an operator must import a helm chart to automatically create CNFDs and create a service.

1. On the Launchpad menu, select **CATALOG** > **NSD LIST**. The NSD CATALOG page opens.

2. Select a network service descriptor from the catalog and click to the left of the descriptor to edit parameters for the NSD.
3. Click **Update**.
4. Next, click to create the Service.
5. On the Configuration page, click and the Service Instantiation wizard appears. Fill in the appropriate details for the specific service.
6. Provide the following details in the Service Instantiation step:
   - ***Name**: Name of the service
   - **Description**: Details about this service
   - **Map Existing Resources**: Service elements will be mapped to existing resources. Choose **True** from the drop-down menu in this field to map service elements to existing resources.



The steps to instantiate a service that will use existing resources mapped to elements defined in the service are customized for your NSD. The Network Service Instantiation section includes more information on designing and editing specific variables in an NS prior to instantiation. The required steps are to discover resources and then select the NFs to which an existing resource will be attached. Once a NF is selected a new step will be added to the Service

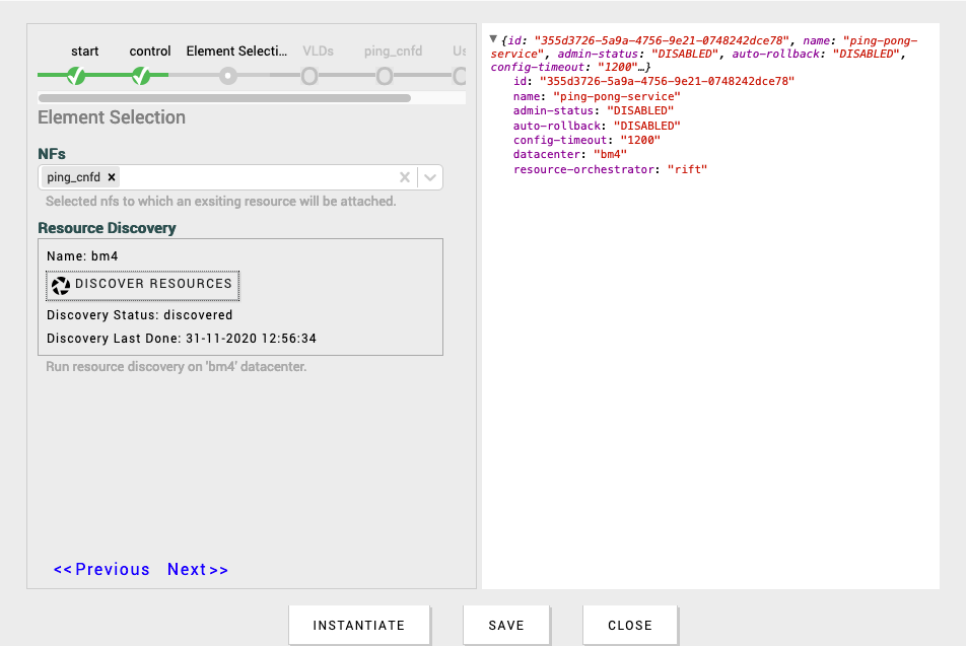
Instantiation Wizard for each NF. When you get to this step you will identify the preexisting release name and determine if the release is removed when terminating the service.

Note: There are steps to configure each CNF elected in the Element selection step.

7. Click **Next >>** and provide appropriate details for each step. Once you reach the Element Selection Step, provide the following details for your service.
   - **NFs**: Selected nfs to which an existing helm release will be attached. You can select an option for each CNFD that is part of the NSD. The drop-down menu will only show available resources if you run discovery on a datacenter.
   - In the **Resources Discovery** section click DISCOVER RESOURCES and the Discovery Status changes to discovering. The discover process can take several minutes. An operator can continue to perform other functions such as save, close, etc. while the discovery process takes place. If the Kubernetes resources under the preexisting release are not ready, then the CNF instantiation fails.

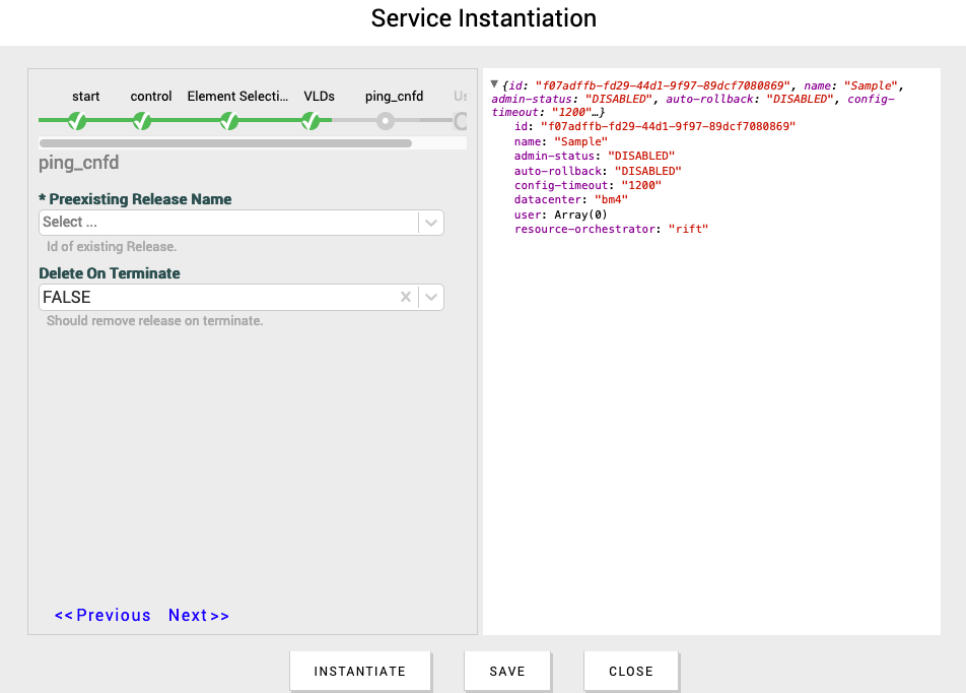
Note: If discovery is already done at the Datacenter level, then you can skip this step.

Service Instantiation



8. After selecting an NF, a NF section appears in the Service Instantiation Wizard. On that screen, provide the following details:
   - ***Preexisting Release Name**: Name of the existing helm release.

- **Delete On Terminate**: Select True or False to identify if the helm releases are deleted when you terminate the Network Service.



9. Click **Save**. See Create and Instantiate a Network Service for more information on how to instantiate a network service.

Model Changes

This release introduces two new input-parameter fields in the `cnfr:release-info` in the CNFR Data Model (cnfr:cnfr).

- `preexisting-release-name`: Preexisting helm release which is used for this CNFR.
- `delete-on-terminate`: Controls the deletion of the preexisting release.

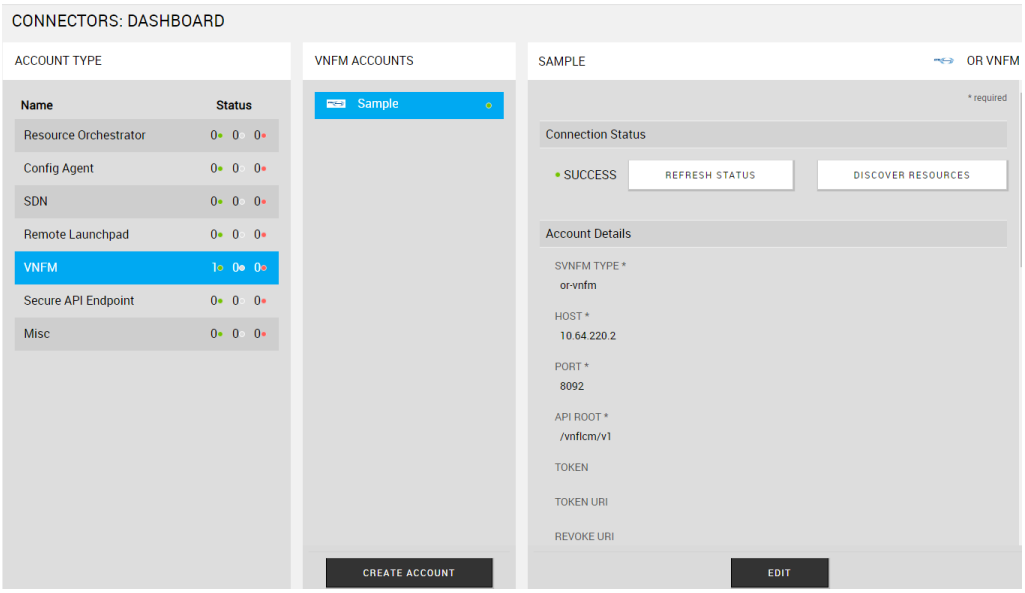
Map Service Elements to Existing Resources during Instantiation with SOL003 based VNFs

RIFT.ware now supports the ability to map existing resources, during service instantiation, to a functioning VNFM. This VNFM must support the Or-VNFM interface defined by the ETSI specs SOL003/IFA 007. At the time of instantiation, the VNFM account needs to be selected to manage an associated VNF. See Add a VNFM for specifics on creating an Or-VNFM Account. The Or-VNFM account name is specified as part of the instantiation process. After creating an account, an operator can run an account discovery on a VNFM account to identify resources and map the specific resources to a VNFM.

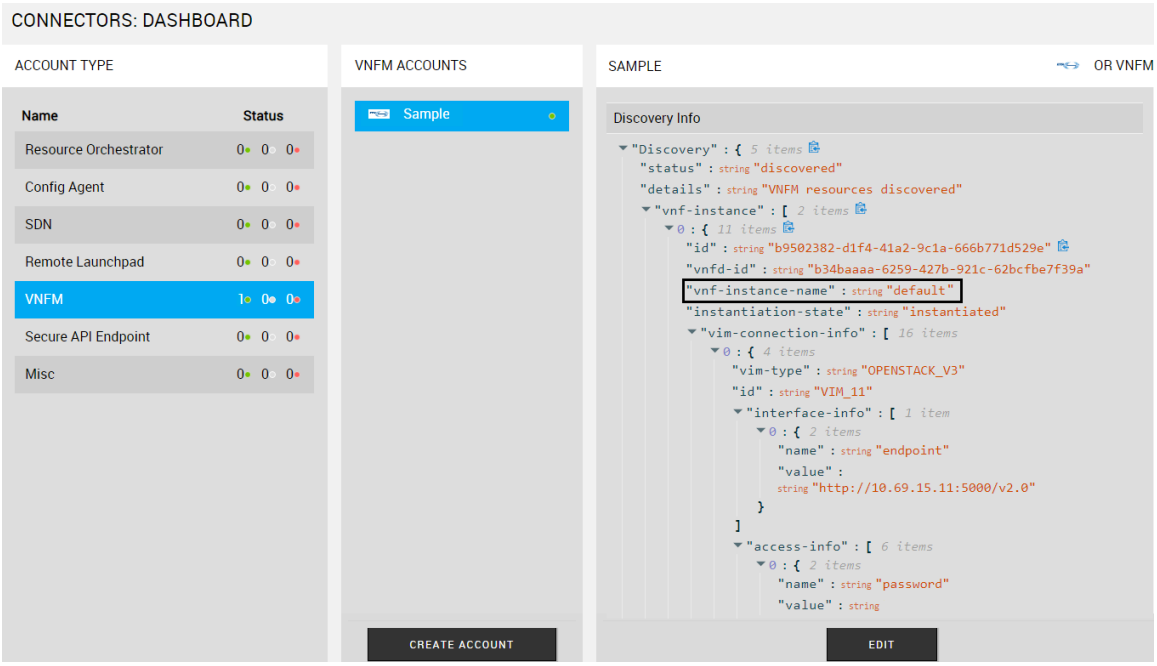
UI Enhancements

This feature includes a new 

button in the **CONNECTORS** section of the Launchpad menu for VNFM accounts.



Scroll down to the bottom of the screen to see the status and discovery details.



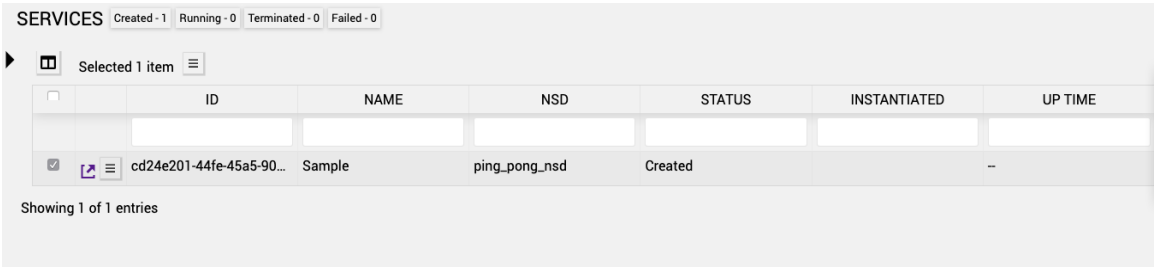
Note the **vnf-instance-name** from the Discovery Info. This name is necessary to add in the vnf-instance-id section in the Service Instantiation Wizard during Instantiation.

After discovery is complete, the VNFM instance only display the VNF instances that are present on the VNFM. Prior to instantiation, an operator must:

- Create or onboard a VNFD. This VNFD represents a VNF to be mapped to an existing resource. In this VNFD, the VNFM property must be set to “or-vnfm”. See Onboard a Preconfigured CNF or VNF Package or Create a Descriptor Package.

- Create an NSD using the above VNFD and then a Service created from this NSD. See Onboard a Preconfigured NSD Package.

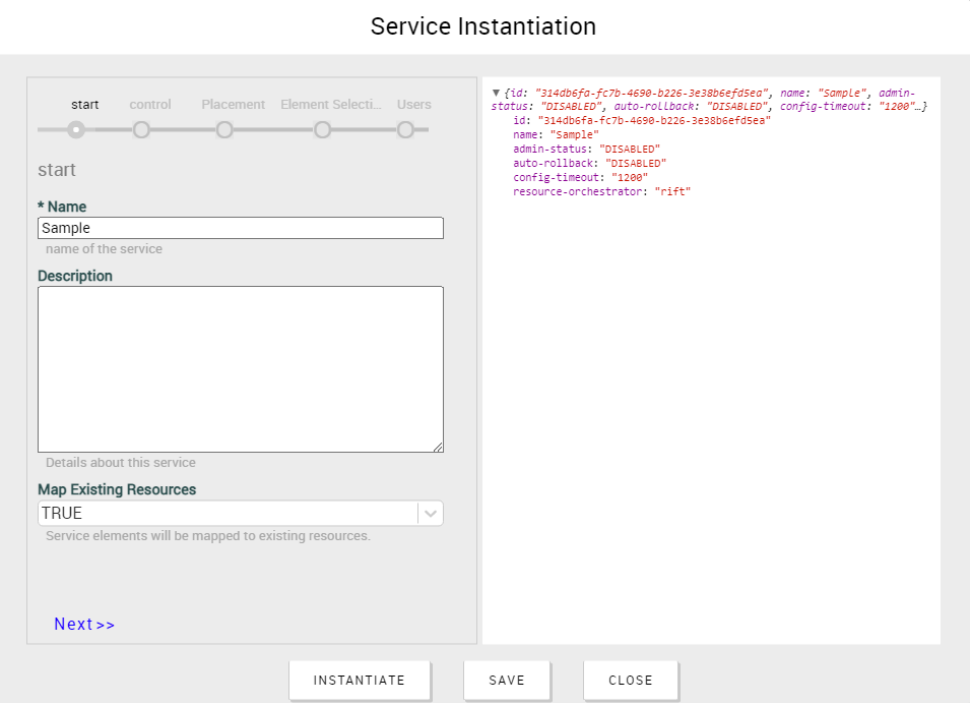
1. On the **Launchpad** menu, click **SERVICES > SERVICE LIST**.
2. Click to open a service and then click to edit the service.



A Service Instantiation wizard appears. Fill in the appropriate details for the specific service. The right side of the wizard screen displays what can be changed using this edit option.

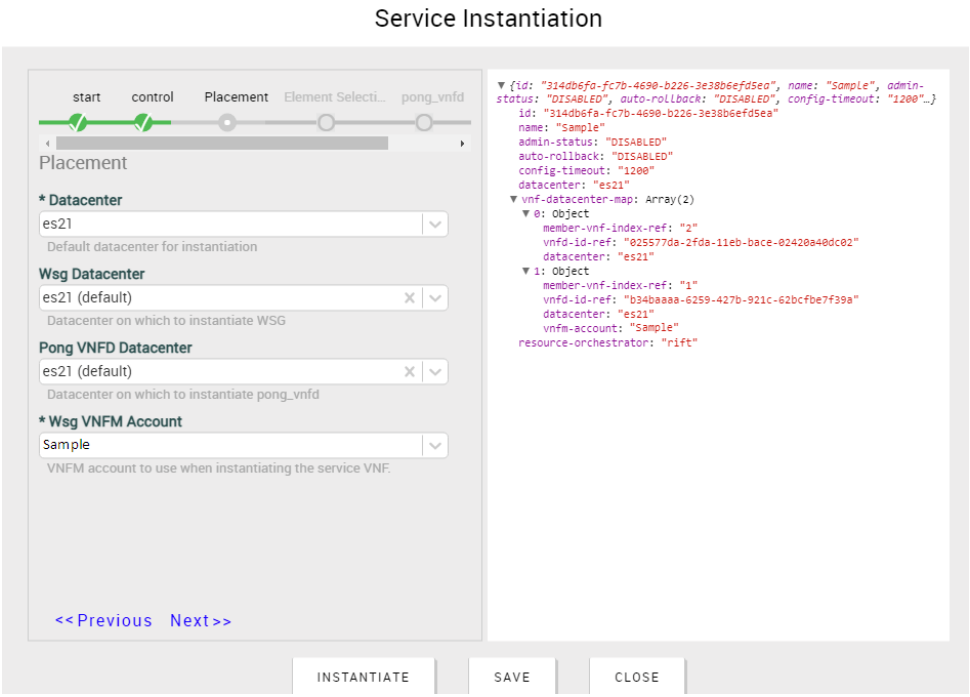
Note: You cannot navigate forward without filling in required fields (*). The **Next >>** button will not function if required fields are not populated. The Save button allows you to save your NSR config content and edit it later if you are not ready to Instantiate.

3. Provide the following details in the Service Instantiation step:
   - ***Name**: Name of the service
   - **Description**: Details about this service
   - **Map Existing Resources**: Service elements will be mapped to existing resources. Choose **True** from the drop-down menu in this field to map service elements to existing resources.



The steps to instantiate a service that will use existing resources mapped to elements defined in the service are customized for your NSD. The Network Service Instantiation section includes more information on designing and editing specific variables in an NS prior to instantiation. The required steps are to discover resources and then select the NFs to which an existing resource will be attached. Once the NF is select, then the NF will appear in the Service Instantiation Wizard.

4. Click **Next >>** to provide the default datacenter, the datacenter on which to instantiate the vnf and the VNFM account to use when instantiating the service VNF in the Placement step. The Placement step does not appear if there is only one Datacenter option in the Datacenter Account.



5. Click **Next >>** to provide the following details in the Element Selection step:
   - **NFs**: Selected nfs to which an existing resource will be attached. You can select an option for each VNFD that is part of the NSD. The drop-down menu will only show available resources if you run discovery on a datacenter.
   - In the **Resources Discovery** section click DISCOVER RESOURCES and the Discovery Status changes to discovering. The discover process can take several minutes. An operator can continue to perform other functions such as save, close, etc. while the discovery process takes place.
6. Click **Next >>** to provide the appropriate details in the pong_vnfd step. There are no changes to this step to instantiate on a VNFM account.
7. Click **Next >>** to provide the following details in the WSG step:
   - ***VNF Instance ID**: ID of existing VNF. If the vnf instance id does not exist, then the instantiation fails. The VNF Instance ID maps to the vnf-instance-name value found in the VNFM Account details on the **CONNECTORS** tab. See the beginning of Map Service Elements to Existing Resources during Instantiation with SOL003 based VNFs.
   - **Delete On Terminate**: Select True or False to identify if the resources are deleted when you terminate the Network Service.
8. Click **Save**. See Create and Instantiate a Network Service for more information on how to instantiate a network service.

Model Changes

This release includes changes to the RESTful protocols specification for the Or-Vnfm Reference Point (ETSI GS NFV-SOL 003). There are two new APIs with VNF_Instance_ID as the identifier.

- Patch: `/vnflcm/v1/vnf_instances/{VNF_Instance_Id}`
  Upon successful brownfield discovery, RIFT assumes the responsibility of adding the Rift_system_id, Rift_Project_Name and Federation_Id under the metadata section of the VNF instance using the above patch request API.
- Get: `/vnflcm/v1/vnf_instances`
  This API is used to get a list of running VNFs managed by the respective VNFM.

Additional Mapping Service Elements to Existing Resource Improvements

Release 8.3 includes more information on the **DATACENTERS** > **RESOURCE** tab to improve the process of discovering resources when mapping them to service elements during instantiation. The additional information allows a user to map resources to service elements when an operator does not have direct access to a VIM.

This information includes:

- VM image name, if available
- The IP address(es) for the server interfaces.
- The Management IP of the server.
- Router details if accessible.
- Additional details in the container resources and added services details.

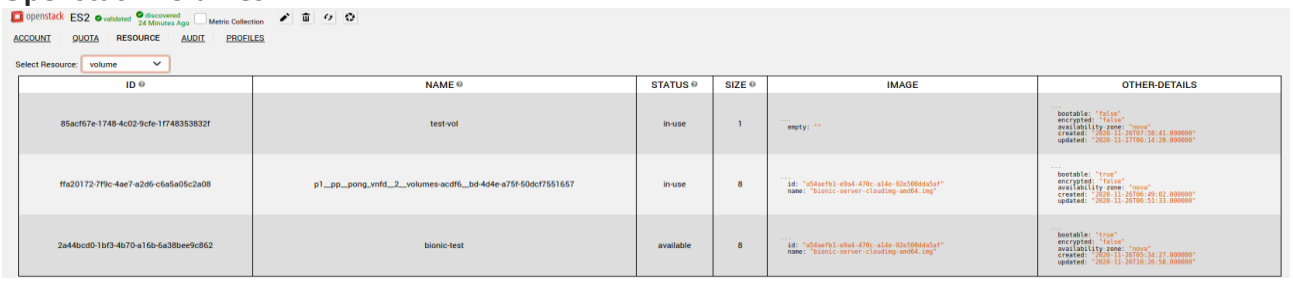
The networks in the server details are now part of the Interface resource field to clarify the link between the interface and the network.

The examples below display the main resources used for discovery design (server, volume, network and helm).

- Openstack VMs
- Openstack Volumes
- Openstack Networks
- Kubernetes Helm
- Kubernetes Pods
- Kubernetes Networks

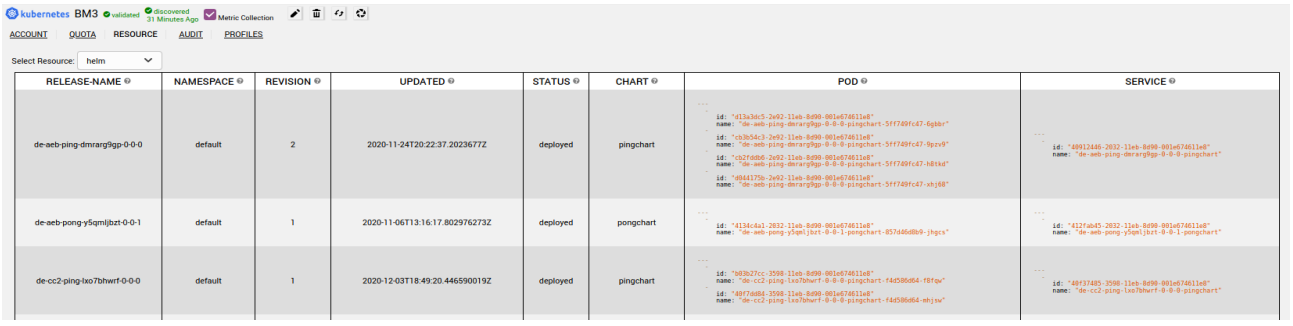
Openstack VMs

Openstack Volumes



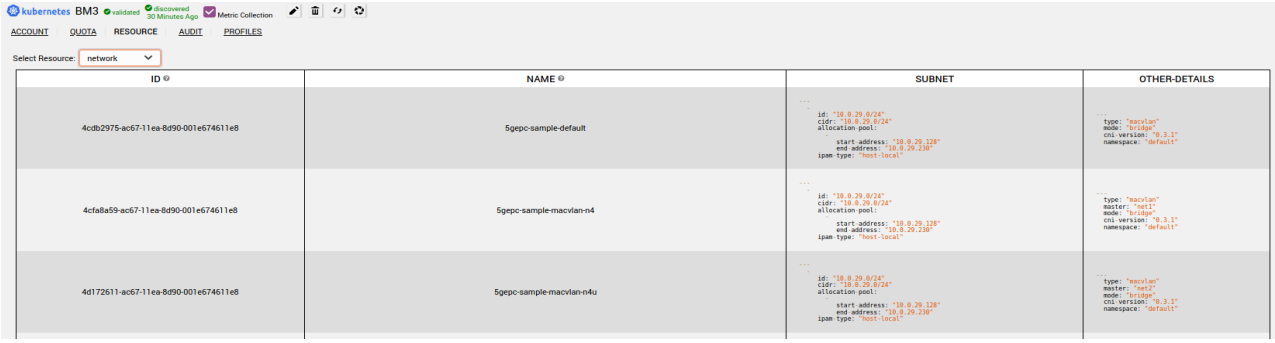
Openstack Networks

Kubernetes Helm



Kubernetes Pods

Kubernetes Networks



Network Slicing Support

This release introduces 5G slice management. 5G network slicing is a network architecture that enables the multiplexing of virtualized and independent logical networks on the same physical network infrastructure. Each network slice is an isolated end-to-end network tailored to fulfill diverse requirements requested by a particular application.

This RIFT.ware approach includes automation and service-level life cycle management capabilities to catalog, deploy and manage end-to-end 5G Network Slices. RIFT's network slice management method allows users to instantiate 5G slice subnets and constituent 5G Network Functions (NFs) using RIFT.ware.

Operators can associate RIFT's network services with slice meta-data which connects the ETSI Network Service with 3GPP end-to-end slices or slice subnets. This allows 5G suppliers to retain control of the selection, placement and discovery process. After instantiation, cloud-native NFs self-monitor the service to ensure that customer SLA's are being met. If KPI degradation occurs, then the NF with RIFT.ware assistance will automatically scale to meet the demands or self-heal in case of faults.

An operator builds network slices using three specifications: CFS Specification, RFS Specification and the NEST Specification.

- **CFS Specification**: This specification defines the customer services requirements for a Network Slice.
- **RFS Specification**: This specification defines, how a Network Slice is realized using the southbound resources.
- **NEST Specification**: This specification describes the set of characteristics of a Network Slice and is defined in NG.116-Generic Network Slice Template.

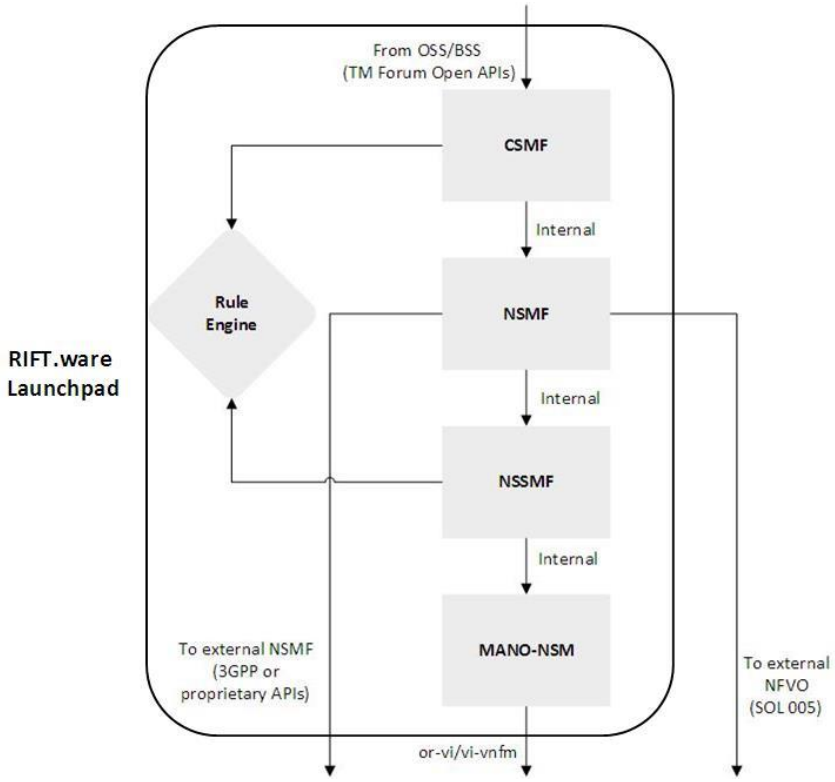
Glossary of Terms

Term	Abbreviation	Description
Communications Service Customers	CSC	N/A
Communication Services Management Function	CSMF	This Management Function is responsible for translating Communication Services requirements to Network Slice related requirement. This interfaces with the OSS/BSS. TMForum like Open APIs are used. An example of a CS requirement is eMBB service type requiring high data rate and high traffic density.
Customer Facing Service Specification	CFS Specification	This specification defines the customer services requirements for a Network Slicing. A CFS Specification refers can refer to multiple NESTs and multiple RFS Specifications. When multiple RFS Specs are present, the selectionRule in the RFS Specification is evaluated to select the RFS Specification to be used.
Generic Slice Template	GST	The Generic Slice Template provides a standardized list of attributes that can be used to characterize different types of network slice.
Network Service Descriptors	NSD	N/A
Network Service Instance	NSI	This instance is a manageable abstraction for all the managed function instances with their supporting resources to provide a certain set of communication services to serve a certain business purpose. An instantiated SD is an Network Service Instance (NSI).
Network Service Subnet Instance	NSSI	This instance represents a group of network function instances (including their corresponding resources) that form part or complete constituents of a Network Slice Instance (NSI).

		Some examples are Core Network NSSI and Specific Area RAN Network NSSI.
Network Slice Management Function	NSMF	This function is responsible for the management and orchestration of an NSI. It derives requirements for Network Slice Subnet from the Network Slice requirements. It has a CSMF in the northbound and NSSMF in the southbound. For external access, it uses use 3GPP APIs and extensions.
Network Slice Subnet Management Function	NSSMF	This function is responsible for the management and orchestration of NSSI. It has NSMF in the northbound and NFV-O in the southbound. For external access, it uses 3GPP APIs and extensions.
Network Slice Type	NEST	This specification defines the set of characteristics of a Network Slice.
Private NESTs	P-NEST	NESTs which character attributes are assigned values by the Network Slice Providers, which are different from those assigned in S-NESTs.
Resource Facing Service Specification	RFS	This specification defines how a Network Slice is realized using the southbound resources. For example, the NFVO to be used, the Network Service descriptors to be used, etc.
Slice Descriptor	SD	A slice descriptor is used to define a Network slice.
Slice Subnet Descriptor	SSD	N/A
Standardized NESTs	S-NEST	Nests which character attributes are assigned values by SDOs, working groups, foras, etc.

Network Slice Components

The core components of a network slice are CSMF, NSMF and NSSMF.



CSMF

Communication Services Management Function (CSMF) is a 3GPP defined node responsible for translating Communication Services requirements to Network Slice related requirements. It interfaces with OSS/BSS using TM Forum Open APIs and southbound systems such as the NSMF, NFVO, or other Domain Orchestrators using the API corresponding to the southbound domain. The CSMF enables CSC to order any form of end-to-end communications service such as Network Slices, multi-site SDWAN, and others. A communications service is defined in the CSMF using two models:

- Customer Facing Service Specification (CFS Specification)
- Resource Facing Service Specification (RFS Specification)

To create an end-to-end slice, a CFS Specification is used to refer to one or more NESTs and one or more RFS Specifications. When multiple RFS Specs are present, the selectionRule in the RFS Specification is evaluated to select the RFS Specification to be used. A RFS specification defines how a Network Slice is realized using the southbound resources. For example, the NSMF, NSSMF, or NFVO to be used, the Network Service descriptors to be used, etc.

One CFS can refer to multiple RFS Specifications. Depending on the characteristics in the NEST template, one RFS representing the end-to-end slice and its constituent slice

subnets is selected during instantiation. The RFS Specification also defines the NSDs that are used along with customization parameters such as a Datacenter or instantiation-variables.

When a CSC orders a Network Slice, the Service Order is received by RIFT.ware CSMF from the OSS. A Service Order is the process of ordering an Network Slice on a Customer Facing Service (TMF 641). When a service order is received, a Customer Service Instance is created using the appropriate CFS. The complete TM Forum Open API specification is TMF641 Service Ordering API User Guide v4.0.1.

CSMF APIs

The CSMF interfaces with OSS/BSS via TMForum APIs for slice management.

- CFS Specification (Create, Update, Delete, Get)
- RFS Specification (Create, Update, Delete, Get)
- Service Order (Create, Update, Delete)
- Notification for Service Order request completion

NSMF

Network Slice Management Function (NSMF) is responsible for the management of NSIs. It obtains network slice subnet related requirements from the network slice related requirements. The NSMF receives slice operation commands from the CSMF and communicates with the NSSMF for slice subnet operations. When a CFS Spec is processed by the RIFT CSMF, it is passed to the RIFT NSMF component which may then:

- Interface with the RIFT NSSMF component to create the slice subnet
- Interface with an external third-party NSSMF to create the slice subnet

NSSMF

Network Slice Subnet Management Function (NSSMF) is responsible for the management of NSSIs. The NSSMF receives slice subnet operation commands from the NSMF and interfaces with RIFT-NFVO or an external NFVO to instantiate ETSI NFV Network Services for the Network Slice.

UI Enhancements

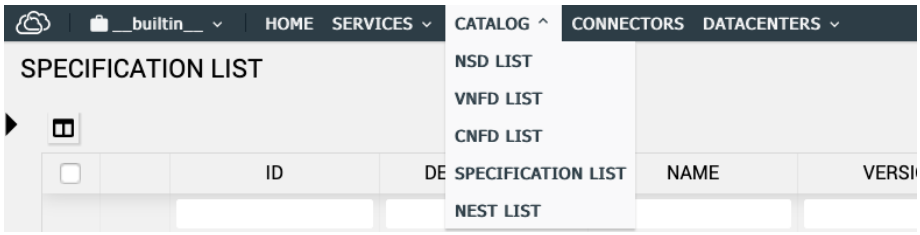
The RIFT.ware UI now contains a new 

project that all users have read-access to but only super-admins can modify. This project contains templates that are used to create the slice applications: CFS, NFS and RFS. They are stored in the **__builtin__** project. If a super-admin user wants to change the RIFT base NEST, RIFT base CFS or RIFT base RFS, then they can open a template and edit the different properties of a service specification.

The UI allows an operator to:

- Edit a Service Specification Template
- Create NEST
- Create a New Product
- Configure an OSS-BSS-Account
- Add a Secure API Endpoint Account
- Slice Management Configuration
- Place a Service Order
- View Slice Topology

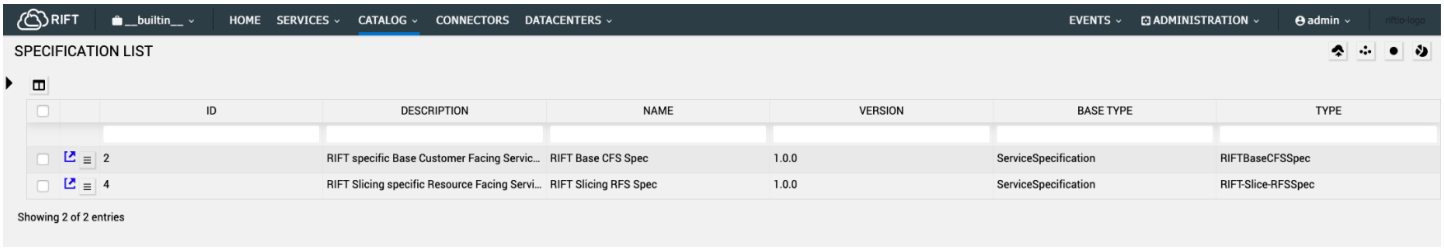
The new templates are located in the **Launchpad** > **CATALOG > SPECIFICATION LIST** menu in the UI.



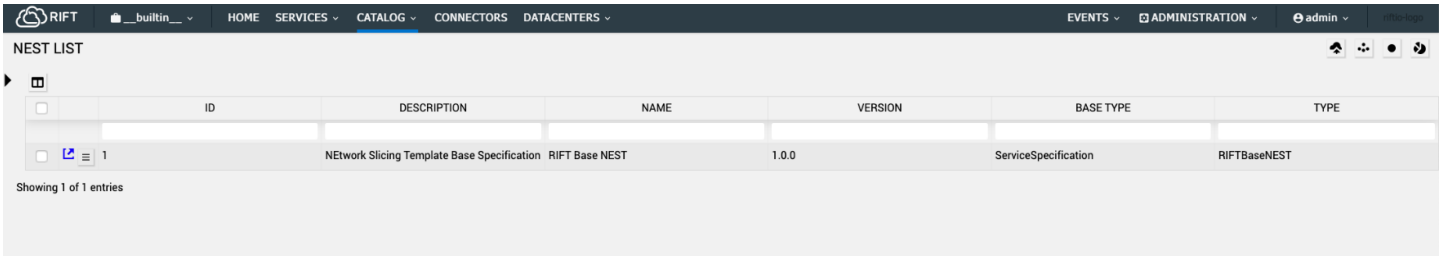
The templates are:

- RIFT specific Base Customer Facing Service Specification: The template is in the **CATALOG** > **SPECIFICATION LIST** menu.
- RIFT Slicing Specific Resource Facing Service Specification: The template is in the **CATALOG** > **SPECIFICATION LIST** menu.
- Network Slicing Template Base Specification: This template is not customer specific. The template is in the **CATALOG** > **NEST LIST** menu.

SPECIFICATION LIST

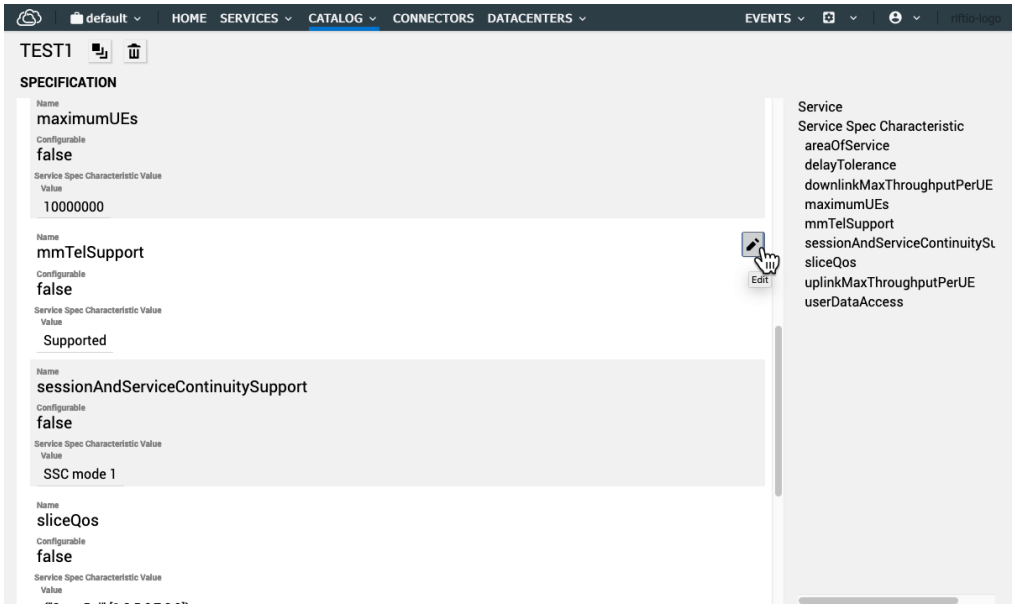


NEST LIST

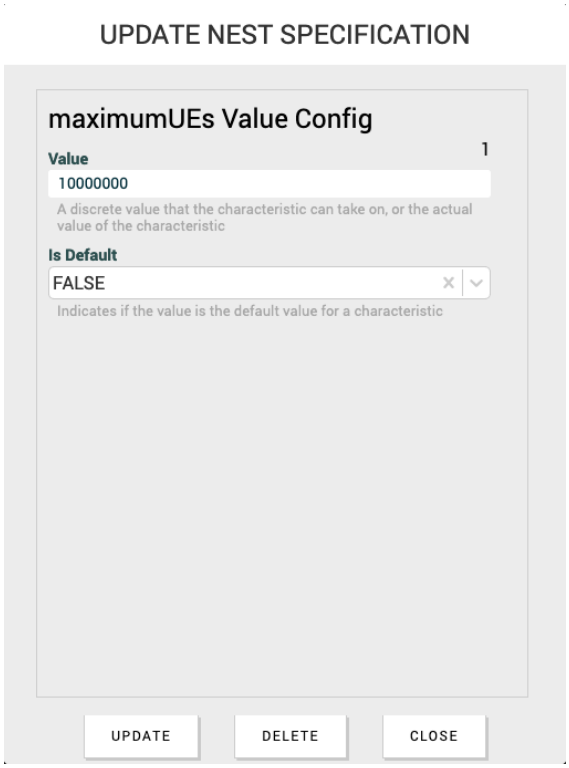


Edit a Service Specification Template

1. On the **Launchpad** menu, click **CATALOG > SPECIFICATION LIST** or **NEST LIST.**
2. Open a specification by clicking . This page displays a table of contents on the far-right of the screen listing all of the editable properties for a service.
3. Click to the right of a property to make changes to the template.



4. After clicking the edit icon, an **UPDATE SPECIFICATION** window appears.



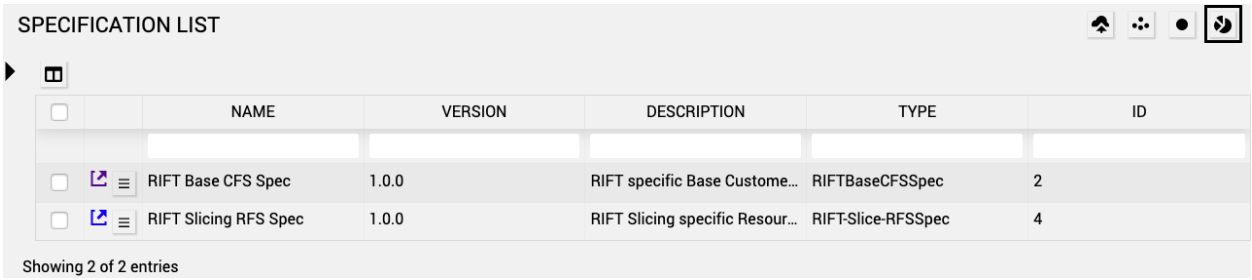
5. Click **UPDATE** to make changes to the specification template.

> **Note:** The __builtin__ project should ONLY be used for templates. Move to another project to create a nest, create a new product, place an order or any other work besides editing specification templates.

Create NEST

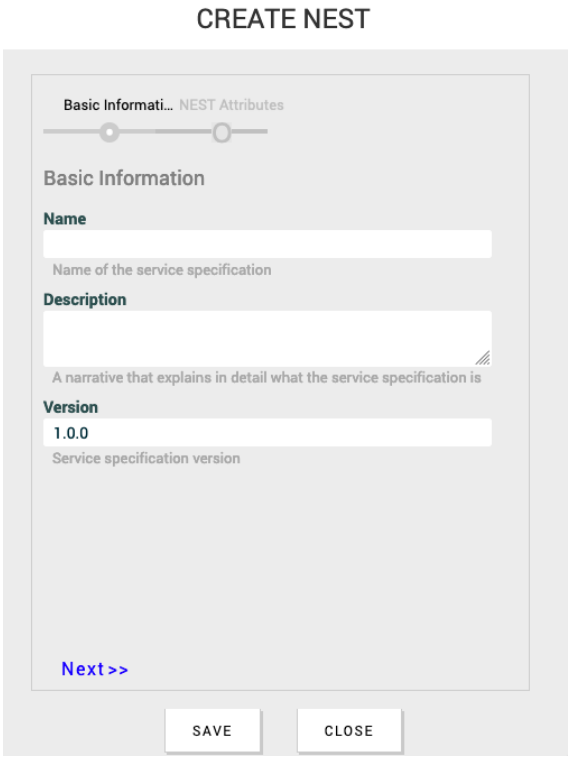
In this release, a user can create a new NEST. Once created, this new NEST is in the **SERVICES** > **SERVICE ORDER LIST**. The attributes in this step are all defined by standards. There are no RIFT specific attributes.

1. On the **Launchpad** menu, click **CATALOG**.
2. Click  to create a NEST.



A **CREATE NEST** wizard appears.

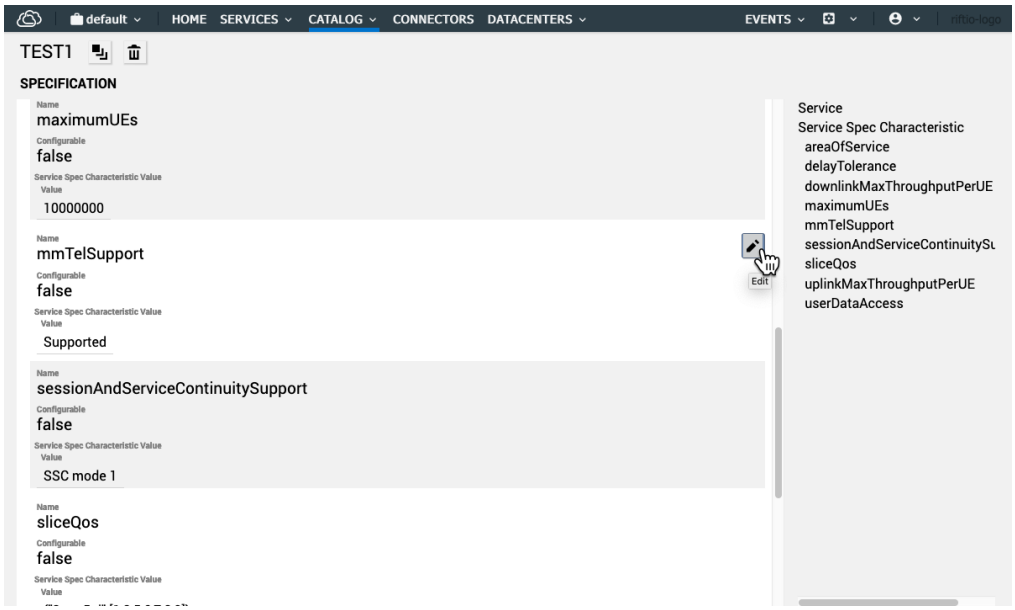
3. Provide the following slice application **Basic Information**:
   - **Name**: Name of the service specification.
   - **Description**: A narrative that explains this service application in detail.
   - **Version**: Service specification version.



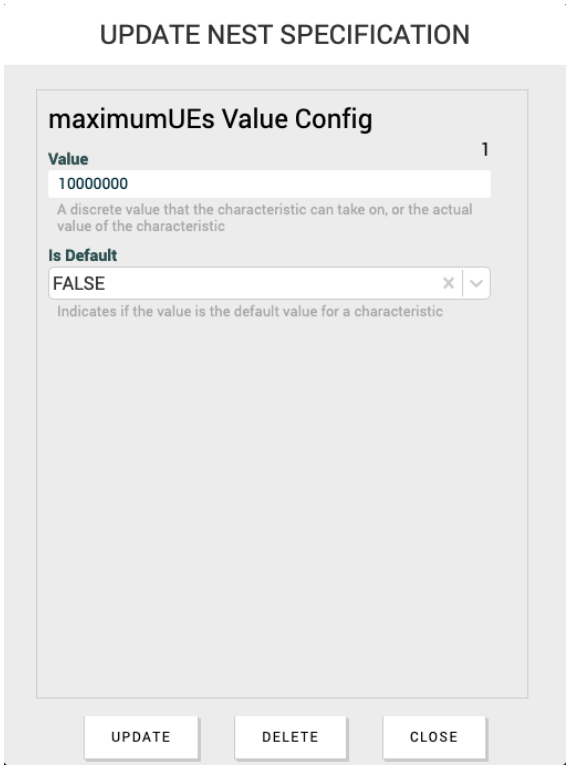
4. Click **Next >>** and provide the appropriate **NEST Attributes** details. Refer to the NG.116-Generic Network Slice Template for additional information on NEST attributes.
5. Click **Next >>** and provide the following **RFS Create** details:
   - **Name**: Name of the service specification.
   - **Description**: A narrative that explains this service application in detail.
   - **Version**: Service specification version.

     Selection Rule
   - **Selection Rule**: match-all or match-one
     - After selecting match-all, add **Match All Expressions**: List of expressions. Operators - eq, ne, gt, gte, lt, lte, has. Enter each expression on a new line.
     - After selecting match-one, add **Match One Expressions**: List of expressions. Operators - eq, ne, gt, gte, lt, lte, has. Enter each expression on a new line.
   - **Priority**: Selection priority of this RFS amongst other RFS

6. Click **Next >>** and provide the following **NEST Support** details:
   - **Create New Nest**: Support new NEST.
   - **Support Rfs**: Supporting RFS specifications.
   - **Supporting Nest**: Supporting NEST Specifications.
7. Click **Next >>** and provide the appropriate details for your slice.
8. Click **SAVE**. The new slice will automatically appear in the **CATALOG** > **SERVICE ORDER LIST**. The list includes 3 specifications: the RFS Specification, the CFS Specification and the generated NEST specification.
9. After creating the slice applications, you can edit any of the attributes by opening the specification and clicking . This page displays a table of contents on the far-right of the screen listing all of the editable properties for a service.
10. Click to the right of a property to make changes to the NEST.



11. After clicking the edit icon, an **UPDATE SPECIFICATION** appears.

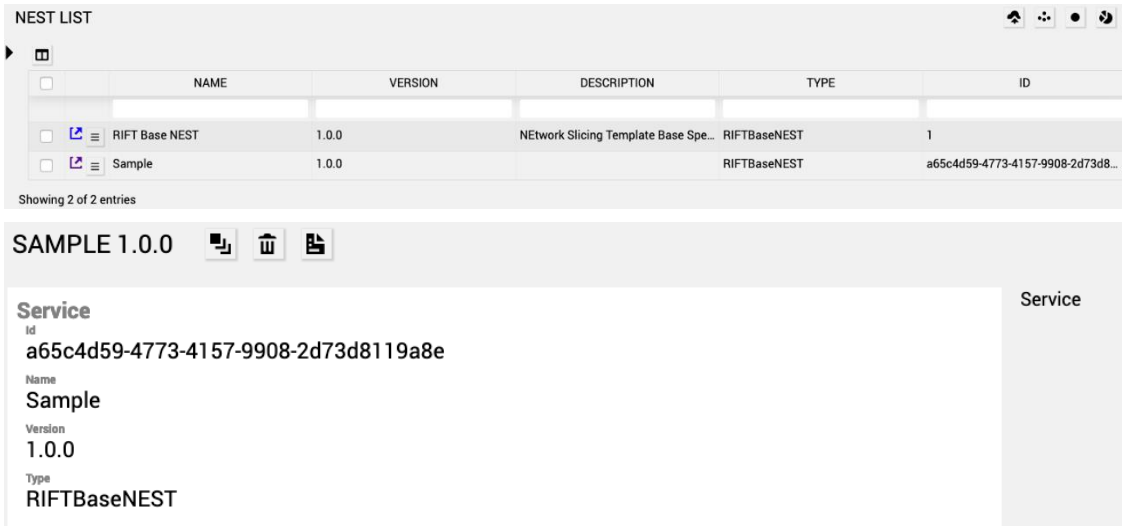


12. Click **UPDATE** to make changes to the specification template.

Create a New Product

Once you create a NEST, you can create a Product.

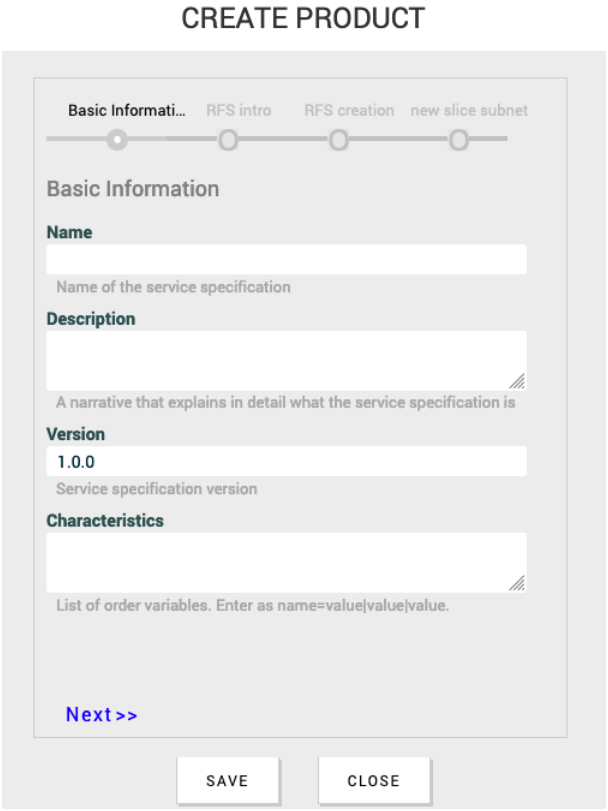
1. On the **Launchpad** menu, click **CATALOG > NEST LIST.**
2. Click to open the NEST and then select to create a Product.



A **CREATE PRODUCT** wizard appears.

3. Provide the following slice application **Basic Information**:
   - **Name**: Name of the service specification.

- **Description**: A narrative that explains this service application in detail.
- **Version**: Service specification version.
- **Characteristics**: List of order variables. Enter as name=value|value|value.



4. Click **Next >>** and provide the following **RFS intro** details:
   - **Name**: Name of the service specification.
   - **Description**: A description of this service specification.
   - **Version**: Service specification version

   Selection Rule

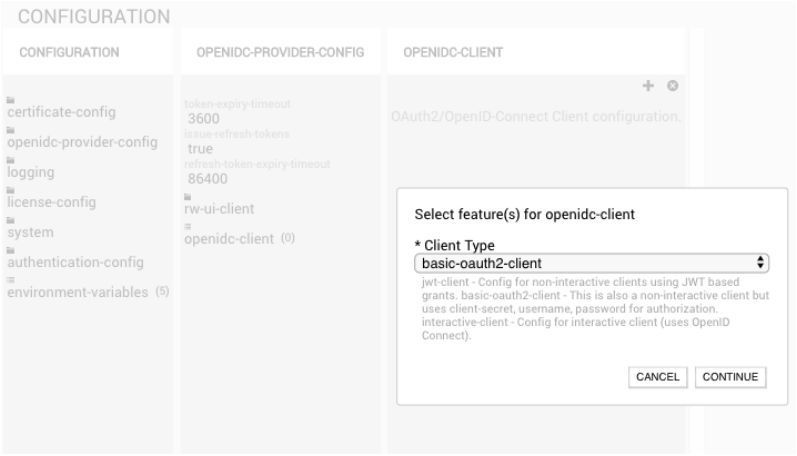
   - **Priority**: Selection priority of this RFS among other RFS
   - **Selection Rule**: Selection to evaluate the list of test expressions based on the CFS/NEST attributes. Choose match-one or match-all.
   - **Match All Expressions**: A list of test expressions based on the CFS/NEST attributes. Enter each expression on a new line.
5. Click **Next >>** and provide the following **RFS creation** details:
   - **On Capability Exceeded**: Action to perform when slice capacity is exceeded. Select new, fail or scale.
   - **Use Transport Nssmf:** The way an NS Resource is defined. Select enabled or disabled.

6. Click **Next >>** and provide the **new slice subnet** details:
   - **Name**: Name of the slice subnet descriptor
   - **South Bound Type**: The type of southbound resource.
     - rift-nfvo: External version of RIFT
     - internal-nssmf
     - ext-nssmf
   - **Resource Selector**: The way an NS Resource will be defined.
     - rule
     - query
     - dbLookup
7. If you choose query in the Resource Selector drop-down menu, then include the:
   - **Query Url**: Query URL to be used for an external resource lookup
   - **Query Params**: List of parameters to be sent in the HTTP query. The value for each parameter is obtained from CFS, NEST, sliceProfile.
8. If you choose dbLookup in the Resource Selector drop-down menu, then include the **Db Lookup Keys**. These keys are used for lookup from the RIFT.ware config DB.
9. Carefully check your entries and click **SAVE**. The Product order appears in the Service Specification List.

Configure an OSS-BSS-Account

OAuth2 authentication is supported for the REST APIs. The marketplace is a RIFT.CSMF API client. To receive notifications, oss-bss-account must be configured.

1. On the Launchpad Dashboard menu, click **ADMINISTRATION** > **CONFIGURATION**.
2. On the **CONFIGURATION** page, click **openidc-provider-config**.
3. In the **OPENIDC-PROVIDER-CONFIG** section, click **openidc-client**.
4. In the **OPENIDC-CLIENT** section, click the **+** icon. A ***Client Type** screen appears.
5. Choose the **basic-oauth2-client** type and click **Continue**. An edit box appears.



6. Fill in the following fields for RIFT.CSMF API client:

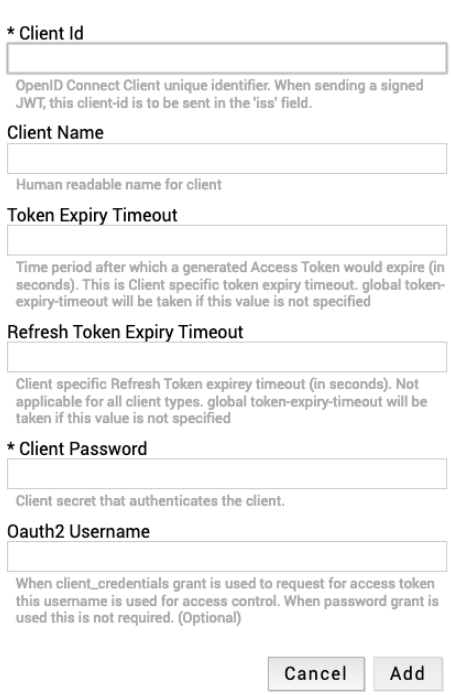
   - ***Client ID**: OpenID Connect Client unique identifier.
   - **Client Name**: Name for the client
   - **Token Expiry Timeout**: Time period after which a generated Access Token expires. Default: 1 hour.

     Note: The token expiry should be a least 1 day (in seconds).

   - **Refresh Token Expiry Timeout**: Client specific Refresh Timeout expiry timeout.

     Note: The token expiry should be a least 1 day (in seconds).

   - **Client Password**: Password for the Oauth2 client to the RIFT server.
   - **Oauth2 Username**: If an operator uses client_credentials to request an access token, then the username is used for access control.



Add a Secure API Endpoint Account

Add a secure endpoint account for the South Bound type that you selected in step 6 of the Create a New Product section.

1. On the Launchpad menu, click **CONNECTORS**.
2. Under **ACCOUNT TYPE**, choose **Secure API Endpoint** and click **CREATE ACCOUNT**.
3. Add the unique **Secure API Endpoint** account **Name*** for the South Bound type that you added in the Create a New Product section.
4. Under Account Details, provide the following information (fields with * are required):
   - **CLIENT ID***: This client ID must match the client ID used in the **openidc-client that is created in the** Configure an OSS-BSS-Account section.
   - **CLIENT SECRET**
   - **TOKEN URI***
   - **USERNAME**: URL to the UI
   - **PASSWORD**: URL for the REST API endpoint
   - **API SERVER HOST***: URL to the Prometheus endpoint
   - **API SERVER PORT**
   - **REVOKE URI**
5. Carefully check your entries and click **SAVE**.

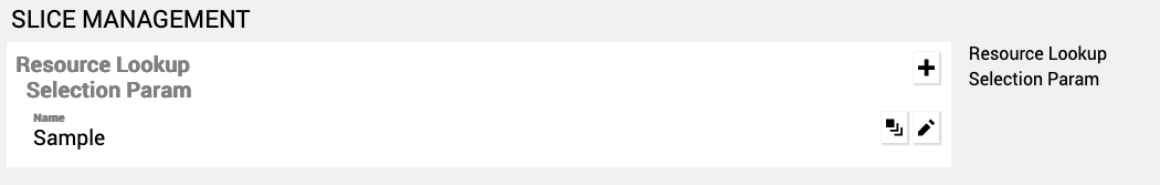
Slice Management Configuration

An operator can configure the dbLookup Resource Selector type.

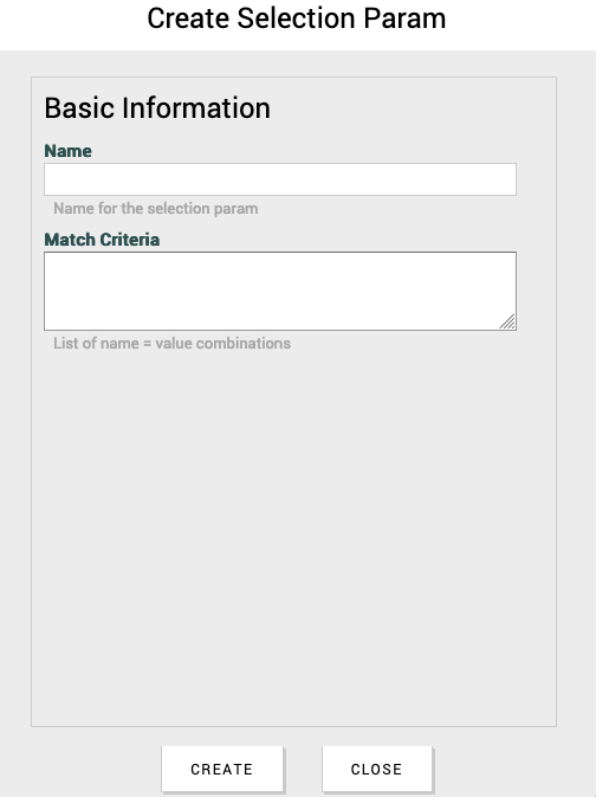
1. On the Launchpad menu, click **ADMINISTRATION** > **SLICE MANAGEMENT**.

2. On the **SLICE MANAGEMENT** page, you will see all of the parameters configured for your service order. The Lookup key has a name and value. RIFT uses the dbLookup from the RFS to identify from which parameter we need to fetch from to instantiate. This application is fetched from the service order NEST.

   It is possible to have multiple selection params.



3. Click + to create a new selection parameter. Provide the following **Create Selection Param** details:



- **Name**: Name for the selection param
- **Match Criteria**: List of name = value combinations.
- **Resource Name**: Name of the resource definition
- **NFVO**: Orchestrator account to be used for the Network Service LCM. This is the account that you added in the Add a Secure API Endpoint Account.
- **Project:** Project/Tenant to use in the Orchestrator
- **NSD:** NSD to instantiate

Param Files

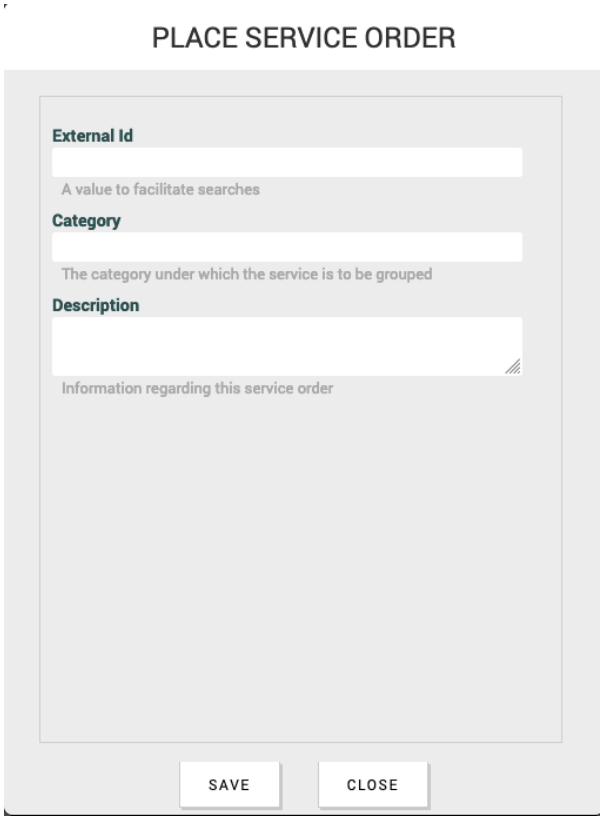
- **URL List:** Orchestrator account to be used for Network Service LCM

4. Continue adding specific details for instantiation.
5. Carefully check your entries and click **Create**.

Place a Service Order

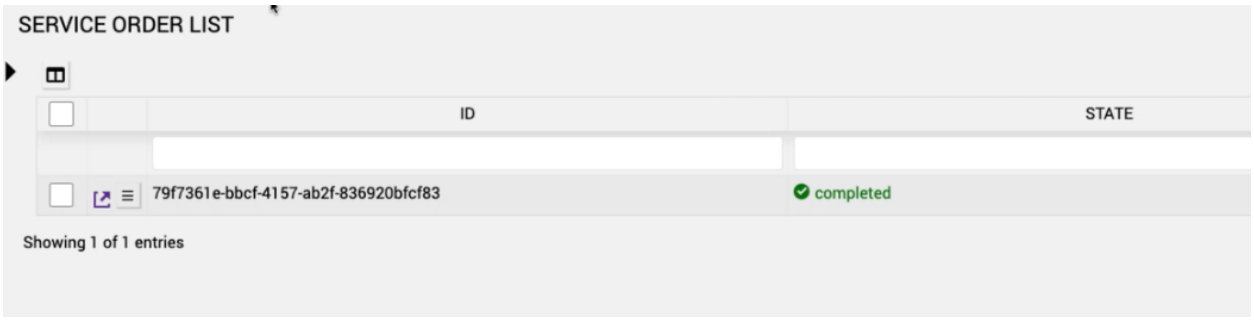
You are now ready to place a service order.

1. On the **Launchpad** menu, click **CATALOG > SPECIFICATION LIST.**
2. Click to open the new specification and then select to place a Service Order. Provide the following **Place Service Order** details:



- **External Id**: A value to facilitate searches
- **Category**: The category under which the service is to be grouped
- **Description**: Information regarding this service

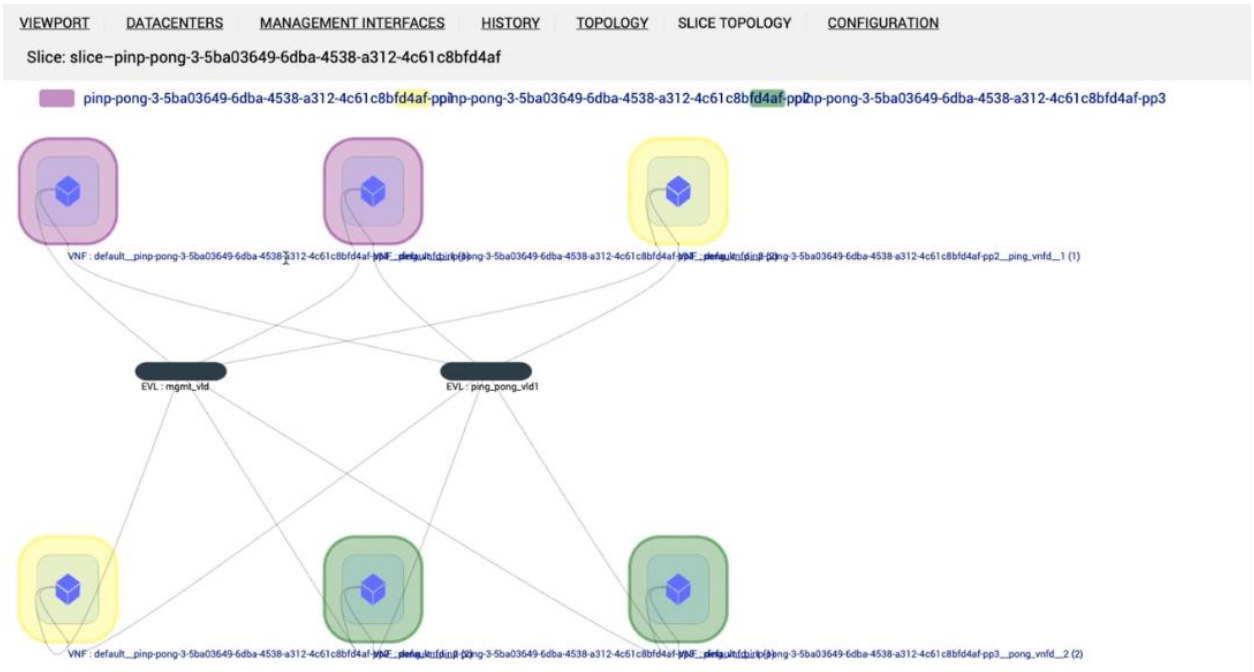
3. Carefully check your entries and click **SAVE**.



Monitor Slice Topology

After creating a Service order, it is possible to see the slice topology for that order.

1. On the **Launchpad** menu, click **CATALOG > SERVICE ORDER LIST.**
2. Open a service order by clicking .
3. On the Services tab, select a service and click **SLICE TOPOLOGY** to open.



Communication Services Management Function TM Forum APIs

- TM Forum API Resources
- Service Specification APIs
  - NEST APIs
    - Get the list of attributes supported for NEST
    - Schema referred by Base NEST
    - Create a NEST
    - GET NEST
    - Delete NEST
  - RFS Specification APIs
    - RFS Specification Model

TM Forum API Resources

Communication Services Management Function (CSMF) is a 3GPP defined node responsible for translating Communication Services requirements to Network Slice related requirements. It interfaces with OSS/BSS and in the southbound interfaces with NFV-O or NSMF. It exposes TMForum APIs towards OSS/BSS. A Network Slice blueprint is defined using the following specification APIs.

Service Specification APIs

NEST APIs

- **Get the list of attributes supported for NEST**

```
/tmf-api/serviceCatalogManagement/v3/__builtin__
/serviceSpecification/1
```

- **Schema referred by Base NEST**

  Some of the characteristics defined in the above base NEST have their own schema (json-schema). The schema for such characteristics are provided by @valueSchemaLocation attribute. To get the json schema, perform a GET query on the path provided in @valueSchemaLocation. For example if the valueSchemaLocation is "/tmf-api/schema/v3/AreaOfService.schema.json", then perform a GET request using this path to get the json schema for this attribute.

```
/tmf-api/schema/v3/{schema.json}
```

- **Create a NEST**

```
/tmf-api/ serviceCatalogManagement/v3/
```

```
{project_name}/serviceSpecification
```

- **Get NEST**

```
/tmf-api/ serviceCatalogManagement/v3/
{project_name}/serviceSpecification/{nest-id}
```

- **Delete NEST**

```
/tmf-api/serviceCatalogManagement/v3/
{project_name}/service Specification/{nest-id}
```

RFS Specification APIs

- **RFS Specification Model**

  To get the RFS specification model (the characteristics defined for RFS) use the following query.

```
/tmf-api/serviceCatalogManagement/v3/__builtin__/
serviceSpecification/3
```

- **Create RFS Specification**

  The URL and the method are the same as the RFS Specification API. However in the payload, the @type attribute points to "RIFT-NFVO-RFSpec", which means that a RFS Specification is to be created.

```
/tmf-
api/serviceCatalogManagement/v3/{project_name}/serviceSpecificati
on
```

- **Get RFS Spec**

```
/tmf-
api/serviceCatalogManagement/v3/{project_name}/serviceSpecification/{rfsspec-
id}
```

- **Delete RFS Spec**

```
/tmf-
api/serviceCatalogManagement/v3/{project_name}/serviceSpecification/{rfsspec-
id}
```

CFS Specification APIs

- **CFS Specification Model**

  To get the CFS specification model (the characteristics defined for CFS) use the following query.

```
/tmf-api/serviceCatalogManagement/v3/__builtin__/
serviceSpecification/2
```

- **Create CFS Specification**

```
/tmf-
api/serviceCatalogManagement/v3/{project_name}/serviceSpecificati
on
```

- **Get CFS Spec**

```
/tmf-api/serviceCatalogManagement/v3/{project_name}/
serviceSpecification/{cfsspec-id}
```

- **Delete CFS Spec**

```
/tmf-api/serviceCatalogManagement/v3/{project_name}/
serviceSpecification/{cfsspec-id}
```

Service Order APIs

Create Service Order

Note: Only one orderItem is currently supported.

```
/tmf-api/serviceOrderManagement/v3/{project_name}/serviceOrder
```

Get Service Order

```
/tmf-api/serviceOrderManagement/v3/{project_name}/serviceOrder/
{service-order-id}
```

Delete Service Order

```
/tmf-
api/serviceOrderManagement/v3/{project_name}/serviceOrder/{service-
order-id}
```

Service Get APIs

Get CFS

The CFS service-id can be obtained from the service-order. The CFS service-id can be obtained from serviceOrder["orderItem"][0]["service"]["id"]. The supportingService attribute of the CFS will point to the constituent RFS.

```
/tmf-api/serviceInventory/v3/{project_name}/service/{service-id}
```

Get RFS

The RFS ID can be obtained from the CFS service details. The supportingService attributes points to the NetworkServices that constitutes the Network Slice.

```
/tmf-api/serviceInventory/v3/{project_name}/service/
{rfs-service-id}
```

Get Network Service

Network Service details are available with the RFS service. The network service ID is the same as the one created by the NFVO.

```
/tmf-api/serviceInventory/v3/{project_name}/service/{ns-service-id}
```

Notification APIs

Register for Notifications

Notifications can be registered for serviceCatalogManagement, serviceInventory and serviceOrderManagement. The URL mentioned in the callback parameter will be invoked. The notification will be sent to "callback/listener/<eventType>". A notification registration can be removed using the DELETE method on the hub/id.

```
/tmf-api/serviceCatalogManagement/v3/{project_name}/hub
/tmf-api/serviceInventory/v3/{project_name}/hub
/tmf-api/serviceOrderManagement/v3/{project_name}/hub
```

Delete Registration

```
/tmf-api/serviceCatalogManagement/v3/{project_name}/hub/{reg-id}
/tmf-api/serviceInventory/v3/{project_name}/hub/{reg-id}
/tmf-api/serviceOrderManagement/v3/{project_name}/hub/{reg-id}
```

Sample Notification

The notification is sent from RIFT.csmf to the API client. The generic notification url is /listener/{eventType}. The eventTypes are defined in TMForum OpenAPI documents (TMF 633, TMF 638 and TMF 641).

```
/listener/serviceOrderStateChangeNotification
```

REST API Authentication

OAuth2 authentication is supported for the REST APIs. The marketplace is a RIFT.CSMF API client. To receive notifications, oss-bss-account must be configured. See Configure an OSS-BSS-Account.

RIFT.ware Disaster Recovery Support

RIFT.ware now supports disaster recovery. This process restores a fresh new RIFT.ware instance to exactly same working state as the former RIFT.ware instance that went faulty. The new functional instance is populated with all the configurations and operational data of the former instance. This procedure provides a seamless last resort option for the operator when all the other existing resiliency options such as Geographic Redundancy are no longer viable.

A user with a platform role such as platform-admin, platform-oper or super-admin can save and backup Launchpad configuration and operational data at any point of time. Then, the operator imports the backed-up data into the new Launchpad instance.

- Export a File
  - Set up a Cron Job to Export a File
- Recover Launchpad
- UI Enhancements
  - Export File
  - Import File
- Model Enhancements
  - export-launchpad API
  - export-launchpad RPC
  - delete-export-job API
  - delete-export-job RPC
  - launchpad-export-state API (Operational Data)
  - launchpad-export-state (Operational Data)
  - import-launchpad-prepare API
  - import-launchpad-prepare RPC
  - import-launchpad-start API
  - import-launchpad-start RPC
  - delete-import-job API
  - delete-import-job RPC
  - launchpad-import-state API (Operational Data)
  - launchpad-import-state (Operational Data)

Export a File

Launchpad exposes an RPC that enables easy exporting of Launchpad data to an archive file. Once the archive file is created, the user can download that file and store it anywhere. The archive file includes the key dataset required to restore a Launchpad instance to a new Launchpad instance as well as the key configuration and the operational state data. The data is converted to an XML format using the schema and saved to multiple XML files (configuration, operational data).

The exported file includes the following data items as well as complete Launchpad configuration information:

- Launchpad artifacts (descriptors)
- Relevant log files
- Selected environment data
- Version file
- Project specific configurations like catalogs, accounts, datacenters, etc
- Launchpad artefacts used by a specific instance
- Platform configuration excluding launchpad instance specific configurations
- Federation ID
- Operational data and configuration entries from Redis

The following data is not exported:

- System configuration
- Launchpad instance specific configurations like system ID
- License configuration
- Certificates
- HA-GR configuration

Note: The system might be slow during an export job.

Set up a Cron Job to Export a File

Configure a cron job to take a timely snapshot of the system state. Store the archive backup file locally. Creating a backup archive file is a critical step to restore your system.

Recover Launchpad

Note: If the system is in a Geographic Redundancy (GR) state, then the operator must restore a single instance of Launchpad on an existing ACTIVE Launchpad in a GR Paired node state. It is necessary to keep the STANDBY instance down so that it will not takeover during the import process of the ACTIVE Launchpad. Once the new Launchpad comes back up as ACTIVE, then the operator can start the STANDBY node.

Launchpad allows an operator to restore a failed Launchpad to a new instance of Launchpad. There are two RPCs to restore Launchpad using backed up archive files. There is one RPC to upload and validate a previously saved archive file and another RPC to execute the recovery process using that file. The recovery process involves restarting Launchpad.

UI Enhancements

Disaster recovery operations now allow users to manually export a backup file to restore the RIFT.ware Launchpad on a new instance of Launchpad. After exporting a file, a user can then import that file to bring Launchpad back to a previously saved running state.

- Export File
- Import File

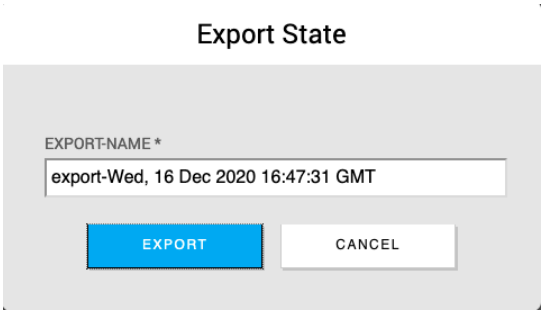
Export File

This release introduces a new **Export Launchpad State** button in the RIFT.ware Launchpad UI. Users should periodically export a zip file to have a current backup copy.

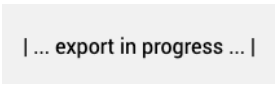
1. On the **Launchpad** menu, click **ADMINISTRATION** > **SYSTEM**.
2. On the **SYSTEM** page, click EXPORT LAUNCHPAD STATE to export an archive file that can be used to recover from a system down condition.

> **Note:** An operator must be a super user with super-admin roles to open this URL.

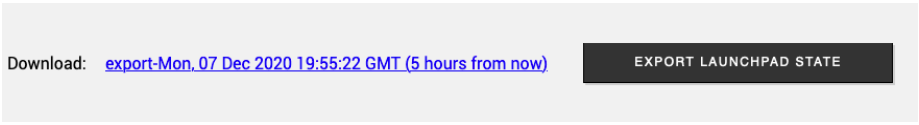
The Export State screen appears. The most recent export file appears with a timestamp. The timestamp is expressed as export-Day, Date Month Year Time.



3. Click EXPORT. The **SYSTEM** page displays that the export is in progress.



4. When the export is complete, the **SYSTEM** page includes a link to the export file.



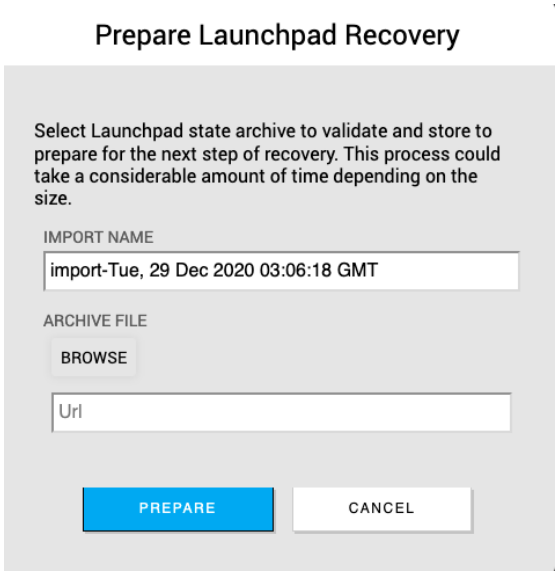
Click the link to download the file. Only one export request can be processed at a time. If an export is in progress, then RIFT.ware rejects any additional export requests.

> **Note:** The last 10 archive files are available to download.

Import File

This release introduces a new **Prepare Launchpad Recovery** button in the RIFT.ware Launchpad UI.

1. On the **SYSTEM** page, click PREPARE LAUNCHPAD RECOVERY to import a backup file.



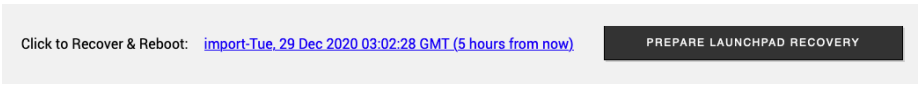
The Prepare Launchpad Recovery screen appears. The timestamp is a default string (export-Day, Date Month Year Time GMT). The **IMPORT NAME** is customizable. This name appears in the link after the imported file is uploaded and validated. Browse to your import file.

2. Click PREPARE. The RIFT.ware UI verifies that the file is generated by another RIFT.ware Launchpad. This process could take a considerable amount of time depending on the size.

   In order to complete the Prepare Launchpad Recovery process your file and Launchpad must adhere to the following requirements.

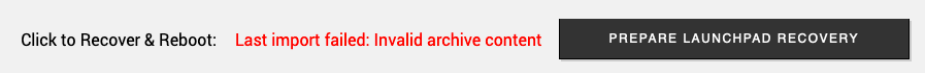
   - The maximum size of the importing archive is restricted to 1GB.
   - The importing Launchpad release version must match with the archive release version (For example: 8.3.0.x.x).
   - The importing Launchpad must have matching or better system configurations to the exporting system at archive creation such as lcm worker count and lcm config worker count.
   - The importing Launchpad must have a valid Launchpad license.

3. When the import is complete, the SYSTEM page includes a link to complete the recovery.



Click the link to finish the Launchpad recovery process. If the system is successfully restored, then the state of the new RIFT.ware Launchpad instance is exactly the same as the failed Launchpad.

If the import fails, then the UI displays an error message.



Model Changes

The following APIs and RPCs can be used to export or import a launchpad instance state.

Export APIs and RPC and Operational Data Model

- export-launchpad API
- export-launchpad RPC
- delete-export-job API
- delete-export-job RPC
- launchpad-export-state API (Operational Data)
- launchpad-export-state (Operational Data)

Import APIs RPC and Operational Data Model

- import-launchpad-prepare API
- import-launchpad-prepare RPC
- import-launchpad-start API
- import-launchpad-start RPC
- delete-import-job API
- delete-import-job RPC
- launchpad-import-state API (Operational Data)
- launchpad-import-state (Operational Data)

export-launchpad API

An operator can save or backup a Launchpad instance state using the below API.

```
URL : https://<launchpad-ip>/api/operations/export-launchpad
Method: POST
Body:
{
    "input": {
        "export-name" : "text-export",
        "export-metadata" : "test export description"
    }
}
Response:
{
    "output": {
        "status": "success",
        "export-job-id": "6f565b37-1a0c-49d5-ab92-c76faf7fa5cb"
    }
}
```

export-launchpad RPC

An operator can save or backup a Launchpad instance state using the below RPC.

```
  rpc export-launchpad-state {
    description "RPC to export current launchpad state";
    input {
      leaf export-name {
        description "Name of launchpad export job";
        type string {
          length "1..255";
        }
      }
      leaf export-metadata {
        description "The metadata added by user to identify or define
a particular export job";
        type string;
      }
    }

    output {
      uses rw-nats-msg:nats-rpc-output;
      leaf export-job-id {
        description "Unique identifier for the export job";
        type yang:uuid;
      }
    }
  }
```

delete-export-job API

An operator can clean up any backed up Launchpad instance data using the below API.

```
URL : https://<launchpad-ip>/api/operations/delete-export-job
Method: POST
Body:
{
    "input": {
        "export-job-id" : "6f565b37-1a0c-49d5-ab92-c76faf7fa5cb"
    }
}
Response:
{
    "output": {
        "status": "success"
    }
}
```

delete-export-job RPC

An operator can clean up any backed up Launchpad instance data using the below RPC.

```
  rpc delete-export-job {
    description "RPC to delete any particular launchpad export jobs";
    input {
      leaf export-job-id {
        description "Export job identifier";
        type yang:uuid;
```

```
      }
    }
    output {
      uses rw-nats-msg:nats-rpc-output;
    }
  }
```

launchpad-export-state API (Operational Data)

An operator can export the archive file using the below API. In the below example, 6f565b37-1a0c-49d5-ab92-c76faf7fa5cb is the export job ID in the response for the export-launchpad API.

```
URL : https://<launchpad-ip>/api/operational/launchpad-export-
state/6f565b37-1a0c-49d5-ab92-c76faf7fa5cb
Method: GET
Response:
{
    "rw-export-import:launchpad-export-state": {
        "export-job-id": "6f565b37-1a0c-49d5-ab92-c76faf7fa5cb",
        "export-job-name": "text-export",
        "export-metadata": "test export description",
        "export-state": "EXPORT-COMPLETED",
        "start-time": "2020-12-20 12:57:29.358166",
        "end-time": "2020-12-20 12:57:48.847177",
        "exported-launchpad-version": "8.3.0.0.118062",
        "current-state-detail": "test export description",
        "archive-download":
"https://10.64.223.221:443/exportarchive/6f565b37-1a0c-49d5-ab92-
c76faf7fa5cb/export_launchpad_archive.tar.gz"
    }
}
```

launchpad-export-state (Operational Data)

The following new model can be used to check the status of the export job.

```
list launchpad-export-state {
    description "Launchpad export job details and status";
    config false;
    key "export-job-id";
    leaf export-job-id {
      description "Unique Id created for the export job";
      type yang:uuid;
    }
    leaf export-job-name {
      description "The export job name";
      type string;
    }
    leaf export-metadata {
      description "The metadata added by user to identify or define a
particular export job";
      type string;
    }
    leaf export-state {
      description "Provides the current state of export job";
      type enumeration {
```

```
        enum NONE;
        enum EXPORT-IN-PROGRESS;
        enum EXPORT-COMPLETED;
        enum EXPORT-CANCELLED;
        enum EXPORT-ERROR;
      }
    }
    leaf start-time {
      description "Start time of launchpad export job";
      type rwt:date-and-time;
    }
    leaf end-time {
      description "Finish time of launchpad export job";
      type rwt:date-and-time;
    }
    leaf exported-launchpad-version {
      description "Launchpad version and build details during export";
      type string;
    }
    leaf current-state-detail {
      description "Detailed error description or current job state
detail.";
      type string;
    }
    leaf archive-download {
      description "Fully qualified filesystem path of the archive";
      type string;
    }
    list project-export-stat {
      description "Project level statistics during export";
      key project-name;
      leaf project-name {
        description "Name of the project";
        type string {
          length "1..255";
        }
      }
      leaf nsd-count {
        description "Number of nsd's configured in the project";
        type uint32;
      }
      leaf nfd-count {
        description "Number of nfd's configured in the project";
        type uint32;
      }
      leaf datacenter-count {
        description "Number of datacenters configured in the project";
        type uint32;
      }
      leaf network-service-count {
        description "Number of network services running in the
project";
        type uint32;
      }
    }
 }
```

import-launchpad-prepare API

An operator can restore Launchpad from the backed-up Launchpad archive using the below API. This RPC validates the archive and updates the status in launchpad-import-state model. In the below example, the archive path is from the response in the launchpad-export-state API (Operational Data).

```
URL : https://<launchpad-ip>/api/operations/import-launchpad-prepare
Method: POST
Body:
{
    "input": {
        "import-name" : "test-import",
        "import-metadata" : "Test import data",
        "archive-path" :
"https://localhost:4568/api/download/ZGVmYXVsdA==/dcb07ea7-04fc-4e74-
a53d-d84f0c83211f/test-1.zip"
    }
}
Response:
{
    "output": {
        "status": "success",
        "import-job-id": "4e9cf6e3-99fc-4984-a3f4-0ef6084d071a"
    }
}
```

import-launchpad-prepare RPC

An operator can restore Launchpad from the backed-up Launchpad archive using the below RPC. This RPC validates the archive and updates the status in launchpad-import-state model.

```
   rpc import-launchpad-prepare {
     description "RPC to import a particular launchpad state. The user
provided archive is downloaded, validated and respond with appropriate
status";
    input {
      leaf import-name {
        description "Launchpad Import job name";
        type string {
          length "1..255";
        }
      }
      leaf import-metadata {
        description "The metadata added by user to identify or define
a particular import job";
        type string;
      }
      // ATTN: Should support both FS path and http link for download.
      leaf archive-path {
        description "Path of the archive to be imported";
        type string;
      }
    }
    output {
```

```
      uses rw-nats-msg:nats-rpc-output;
      leaf import-job-id {
        description "A unique identifier to track launchpad import job
status";
        type yang:uuid;
      }
    }
  }
```

import-launchpad-start API

An operator can start the Launchpad import using the below API. This API triggers the Launchpad restart.

```
URL : https://<launchpad-ip>/api/operations/import-launchpad-start
Method: POST
Body:
{
    "input": {
        "import-job-id" : "4e9cf6e3-99fc-4984-a3f4-0ef6084d071a"
    }
}
Response:
{
    "output": {
        "status": "success"
    }
}
```

import-launchpad-start RPC

An operator can start the Launchpad import using the below RPC. This RPC triggers the Launchpad restart.

```
 rpc import-launchpad-start {
    description "RPC to apply imported launchpad state data, this will
trigger launchpad restart.";
    input {
      leaf import-job-id {
        description "A unique identifier to track import job status";
        type yang:uuid;
      }
    }
    output {
      uses rw-nats-msg:nats-rpc-output;
    }
  }
```

delete-import-job API

An operator can cleanup any import related data from the Launchpad instance using the below API.

```
URL: https://<launchpad-ip>/api/operations/delete-import-job
Method: POST
Body:
{
    "input": {
```

```
        "import-job-id" : "4e9cf6e3-99fc-4984-a3f4-0ef6084d071a"
    }
}
Response:
{
    "output": {
        "status": "success"
    }
}
```

delete-import-job RPC

An operator can cleanup any import related data from the Launchpad instance using the below RPC.

```
  rpc delete-import-job {
    description "Cleanup any import related data saved in launchpad
specific to a job id";
    input {
      leaf import-job-id {
        description "A unique import job identifier";
        type yang:uuid;
      }
    }
    output {
      uses rw-nats-msg:nats-rpc-output;
    }
  }
```

launchpad-import-state API (Operational Data)

An operator can import the archive file using the below API.

```
URL : https://<launchpad-ip>/api/operational/launchpad-import-
state/4e9cf6e3-99fc-4984-a3f4-0ef6084d071a
Method: GET
Response:
{
    "rw-export-import:launchpad-import-state": {
        "import-job-id": "4e9cf6e3-99fc-4984-a3f4-0ef6084d071a",
        "import-job-name": "test-import",
        "import-metadata": "Test import data",
        "import-prepare-time": "2020-12-20 13:25:51.542744",
        "import-state": "IMPORT-PREPARE-DONE"
    }
}
```

launchpad-import-state (Operational Data)

The following new model can be used to check the status of the import job.

```
list launchpad-import-state {
    description "Launchpad import job details and status";
    config false;
    key import-job-id;
    leaf import-job-id {
      description "Unique identifier of import job";
      type yang:uuid;
```

```
    }
    leaf import-job-name {
      description "The import job name";
      type string {
        length "1..255";
      }
    }
    leaf import-metadata {
      description "The metadata added by user to identify or define a
particular import job";
      type string;
    }
    leaf exported-launchpad-version {
      description "Launchpad version and build details from where the
state was exported from";
      type string;
    }
    leaf import-prepare-time {
      description "Time when import job preparation was initiated";
      type rwt:date-and-time;
    }
    leaf import-start-time {
      description "Time when import job was started";
      type rwt:date-and-time;
    }
    leaf import-end-time {
      description "Time when import job was completed";
      type rwt:date-and-time;
    }
    leaf import-state {
      description "Current state of import job";
      type enumeration {
        enum NONE;
        enum IMPORT-PREPARE-DONE;
        enum IMPORT-INITIATED;
        enum IMPORT-IN-PROGRESS;
        enum IMPORT-COMPLETED;
        enum IMPORT-ERROR;
      }
    }
    leaf current-state-detail {
      description "Import job state description or Error description";
      type string;
    }
  }
```

Fixed Issues in RIFT.ware 8.3.0

Support Tickets	Title	Description
RIFT-24992	Running AWS NS fails after failover to a new LP.	A user created an NS on Launchpad 1. The service was in the running state for tiny descriptors with an AWS VM account. The user set up an HA pairing with another Launchpad of a different version. Failover occurred to Launchpad 2. The NS was that running in the Launchpad 1 failed in instantiate in Launchpad 2. This issue is closed.
RIFT-25610	AWS - NS stuck indefinitely in VNF-INIT phase when using a scaling-group and constituent VNF's start-by-default = False.	A user tried to instantiate a simple NS with one VNF, a single VDU (for an Ubuntu VM), and a single VLD. The NSD had a scaling group and the constituent VNFD's start-by-default value was False. When the NS was instantiated, it is stuck forever in the VNF-INIT phase. There were no errors seen in the Event Logs or in rift.log. This issue is resolved.
RIFT-28142	Create service progress did not complete but service is created. Cannot instantiate.	An operator attempted to instantiate a Network Service and it failed to initialize. However, the Services tab on the UI listed the service as Created. This issue only occurred while creating Projects and adding datacenter accounts via the API. The problem occurred when a Datacenter account was created before project creation was complete. This issue is resolved.
RIFT-30839	Rollback is not supported if segment profile is not created in datacenter while using import-instantiation-variables RPCs.	An operator added a datacenter and then uploaded ping pong descriptors. In the Create NS wizard, a user attached the instantiation variable file and clicked YES on the instantiate radio button. This caused the NS to be stuck in Failed state. This issue is resolved.
RIFT-31075	Common LP Discovery status is in the Discovered state even after the	After configuring a valid Kubernetes account from the datacenter tab, the account was successful and discovery was validated.

Support Tickets	Title	Description
	kubeconfig is modified incorrectly.	Next, an operator updated a kubeconfig by making a slight change. The discovery state did not change to “Undiscovered” but the account status changed to failed. The discovery state was not updated when connectivity to the K8s cluster was lost. This issue is resolved.
RIFT-31595	Default port for Remote LP creation should be 443, instead of 8008	While creating Remote LP in Connectors, the default port that was used was 8008. The port needs to be 443. This issue is resolved.

Known Issues in RIFT.ware 8.3.0

Support Tickets	Title	Description	Impact	Workaround
RIFT-13715	Confd configuration transaction abort results in inconsistent state.	Confd can abort configuration change transactions due to its own internal reasons. RIFT.ware cannot undo the changes (on an ABORT form Confd) because it is already internally committed.	This could result in a data mismatch between that is there in the configuration database and what is known to the RIFT.ware backend.	This issue mostly occurs when a user makes successive config changes without any idle time in between the modifications. Any test or automation script using RIFT REST APIs to complete configuration changes must have a delay between two successive config change operations.
RIFT-21978	Password is displayed in plain text in event logs.	A password is displayed in plain text in the events log.	A password is displayed in plain text in the events log.	N/A
RIFT-21923	TC_CONCURRENT_NS_TERMINATE: PackageDeleteError	When multiple descriptor (NSD/VNFD) packages are deleted concurrently using an API, the config data in the ConfD might be missing for a brief period for a few packages.	The config data in the ConfD might be missing for a brief period for a few packages.	When using an API to delete multiple package entries, do not send multiple DELETE requests concurrently. Wait for a request to complete before firing another request.
RIFT-23621	Introduce a new field in VNFM account page for RIFT.ware<-->VNFM handshake URL.	This is a request to add a new field in the VNFM accounts page that the Operator so the operator can populate the URL for handshaking. If the URL field is empty,	RIFT.ware uses a GET call to /vnf_instances to validate VNFM Account. If a SVNFM account does not support GET on	N/A

Support Tickets	Title	Description	Impact	Workaround
		handshake is not required.	/vnf_instances, then the VNFM Account validation may Fail. The UI shows the account status as failed. If the SVNFM account supports notification, then RIFT.ware sends a subscription request next as part of the validation. This is successfully and the VNFM account status is successful. Therefore, a SVNFM account which doesn't support GET call on /vnf_instances, then the VNFM Account status will appear to fail. This will not block any further operations such as instantiation and termination.	
RIFT-23760	Terminating one service with the same name on any Launchpad interrupts existing services on all Launchpads.	This issue occurs because all services on all Launchpads have the same UniqueID which is the InstanceID on ES2. When a user terminates a service, then terminates Instance on ES2 which causes issues for all existing services	If a user terminates any of the services on either of the network-services, this will make any other services orphan.	Use a unique NSR name if an organization is using multiple Launchpads. Ensure that an OpenStack tenant is controlled by a single Launchpad. **Note**: There are no plans to change this behavior.

Support Tickets	Title	Description	Impact	Workaround
		because they have the same UniqueID.		
RIFT-23916	Master: HA- Re-Paring of active-node after failover leaves the nodes in split brain condition.	HA pairing with a node can only be executed once. Upon re-pairing an already paired node, then the system might go into an unstable state.	If an operator attempts to re-pair an already pair node, then the system might go into an unstable state.	Be careful while performing HA pairing. Always wait for a few seconds (30 - 60 seconds) and watch the redundancy state of the peers before performing any commanded action.
RIFT-24872	Updating ipv4 to ipv6 fails in the case of FIXED IP ADDRESS and vice versa.	A user modified the 'FIXED IP ADDRESS' field and then added an IPv4 address in the descriptor details pane for a VDU. Next the user updated the configuration to IPV6 and clicked the 'Update' button. An error message appeared on the UI.	This issue only occurs when there are two IP addresses in interface configurations (IPv4 or IPv6) and the user tries to change one address to one with a different version.	Delete the existing address and add one new address with different a version.
RIFT-25902	Discovery gets triggered even on failed account due to non-reachable network status.	A user created a new VIM account to OpenStack. If the network is down and the cloud setup is not reachable, then when a user attempts discovery it fails but the process is still triggered.	Misleading status of VIM account on UI.	N/A
RIFT-26409	AWS alarm was failed to create after one of the instance (running) failure in AWS	RIFT supports AWS alarms (SNS notifications). These alarms help in monitoring the state of the VM (VDU). Upon failure of the	After the instance failure from the AWS console, the NS should fail in Launchpad. This issue is not occurring correctly.	A user must depend on the monitoring params to check the state of VMs.

Support Tickets	Title	Description	Impact	Workaround
		instance in the AWS, an SNS notification appears.		
RIFT-26498	Creation of custom "User" in the VDUs via RIFT.ware isn't working.	Launchpad provides the option to create custom user accounts in VMs from RIFT.ware using cloud-init. This is not working correctly.	It is not possible to create custom user accounts in VMs with RIFT.ware. User accounts must be created directly in the VM after logging into Launchpad.	Once the VM is running, user accounts can be created by logging into the VM and using the operator interface.
RIFT-26567	Placement groups: transaction failure prints 5 traps riftLPUserSessionEnd	An operator loaded ping pong scaling descriptors and then deleted placement groups from the NSD an VNFDs. Next, the user created and instantiated the NS. An error message and 5 traps appear.	Duplicate traps are generated.	N/A
RIFT-26409	AWS alarm was failed to create after one of the instance (running) failure in AWS	RIFT supports AWS alarms (SNS notifications). These alarms help in monitoring the state of the VM (VDU). Upon failure of the instance in the AWS, an SNS notification appears.	After the instance failure from the AWS console, the NS should fail in Launchpad. This is not occurring correctly.	A user must depend on the monitoring params to check the state of VMs.
RIFT-27186	Service instantiation fails with incorrect timeout error	RIFT.ware was not capturing the real reason why pods failed. Pods only retry in the case of failure, so it was not possible to say that pod creation was in a Failure state.	Instantiation fails in this scenario.	View the cluster event logs in Launchpad.

Support Tickets	Title	Description	Impact	Workaround
RIFT-27241	Instantiating ping-pong fails after ping VNF descriptor is modified and it is impossible to tell why	If a user choses the management interface type in a VNFD as VDU or Connection Point and the corresponding VDU/CP is not selected in the next input, then no error appears until the user attempts to instantiate. The configuration fails in the instantiation attempt.	If a user chooses the mgmt-interface type but no ID is selected, then RIFT.ware considers both the type and the ID as empty.	If a user configures the NS properly, then an issue will not occur.
RIFT-28834	Overwrite existing package option is not available	An operator onboarded a package in the UI. An operator might want to onboard the same package again and overwrite the existing package. The option to overwrite the existing package is missing from the UI.	It is not possible to overwrite an existing package in the UI.	If a package is not in use, then it can be deleted from the catalog and then onboarded again. To onboard an updated version of a package, ensure that it has a new unique id. Note: When using Launchpad to copy a package, the new package is automatically given a new ID. The package with the new ID can be onboarded and it will co-exist with the current package.
RIFT-29543	CNFD service instantiation failing with error release monitoring failed: 'str' object	CNFD service instantiation fails when a helm-chart-based Launchpad is setup in a non-default namespace and uses	NS instantiations fail with the following error message: Release monitoring failed.	Use kube-config with a explicitly defined namespace.

Support Tickets	Title	Description	Impact	Workaround
	has no attribute 'type_yang'	the chart to launch CNFs in other namespaces. The issue is not seen with standalone Launchpads.		
RIFT-31050	Changing user login password throws "Server Error - 503" message	Error 503 returned when updating a user password.	The user password is still updated and this error can be safely ignored.	N/A
RIFT-31602	cnfd:chart-info:interface:connection-point-ref doesn't seem to do anything	Creating a CNFD interface to an external CP ref via cnfd:chart-info:interface:connection-point-ref is not working correctly. An operator has one CNFD with none of the CPs marked as external and another with several CPs marked as external. Next, the operator dragged these into an NSD and it does not show any differences. Also, it is possible to set the same CP to be used in multiple interfaces.	N/A	N/A
RIFT-31732	Discovered VDU instances are marked as init under VDU operational-status.	A Network Service which is created using Sol003 based VNF Brownfield Discovery shows VNFR:VDUR status as ‘init’. The status should be	N/A	N/A

Support Tickets	Title	Description	Impact	Workaround
		'running' instead of 'init'.		
RIFT-31744	Service primitive trigger from UI is failing in cnfd based service.	On a K8s based Launchpad, an operator created a scaling CNFD based service on bm3 cluster and instantiate. After triggering 'ping scale' service primitive under NSR → 'Service Primitive', the UI section displayed that the service primitive trigger was failing.	N/A	N/A
RIFT-31872	Retry fails for cnfd service even if physical network is corrected to proper value in BM datacenter	A network attachment is created in a cluster with the wrong physical network during the VL init stage. The cluster does not validate it and as a result the cluster returns a successful response and the VL creation is successful.	The NF init stage fails while trying to attach a pod with an invalid network.	Delete the NS and instantiate again.