

VM Life Cycle Management

- Overview of VM Life Cycle Management, on page 1
- VM Image Packaging, on page 5
- Image Registration, on page 23
- VM Profiles or Flavors, on page 35
- Configure Internal Management Network, on page 36
- VM Deployment and Management, on page 36
- Access VNFs, on page 50
- Import and Export NFVIS VM, on page 51
- Secure Boot of VNFs, on page 53

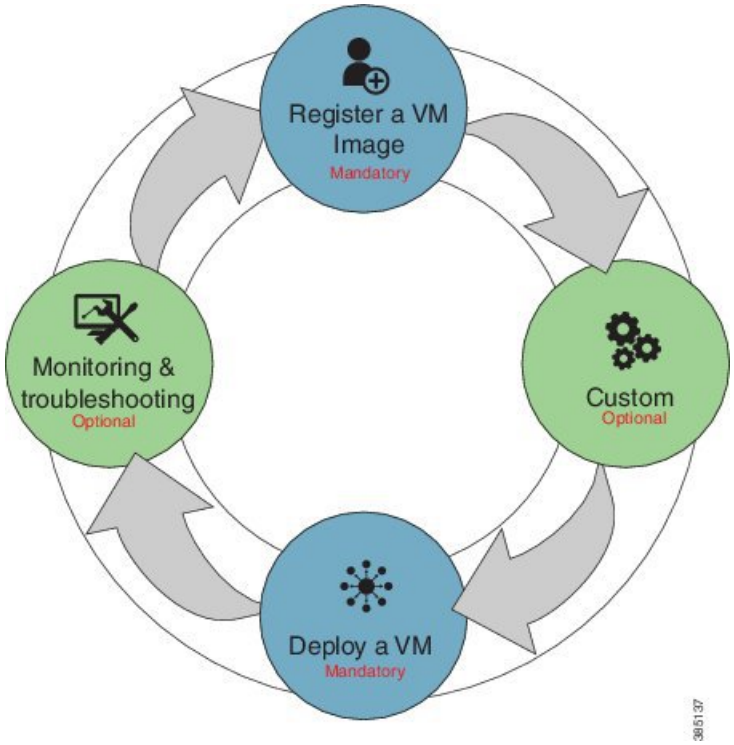
Overview of VM Life Cycle Management

VM life cycle management refers to the entire process of registering, deploying, updating, monitoring VMs, and getting them service chained as per your requirements. You can perform these tasks and more using a set of REST APIs or NETCONF commands or the Cisco Enterprise NFVIS portal.

Workflow of VM Life Cycle Management

The following diagram depicts the basic workflow of the VM life cycle management using REST APIs:

Figure 1: VM Life Cycle Management



1. **Register a VM Image—**To register a VM image, you must first copy or download the relevant VM image to the NFVIS server, or host the image on a http or https server. Once you have downloaded the file, you can register the image using the registration API. The registration API allows you to specify the file path to the location (on the http/https server) where the tar.gz file is hosted. Registering the image is a one-time activity. Once an image is registered on the http or https server, and is in active state, you can perform multiple VM deployments using the registered image.

2. **Customizing the VM—**After registering a VM image, you can optionally create a custom profile or flavor for the VM image if the profiles defined in the image file do not match your requirement. The flavor creation option lets you provide specific profiling details for a VM image, such as the virtual CPU on which the VM will run, and the amount of virtual memory the VM will consume.

   Depending on the topology requirement, you can create additional networks and bridges to attach the VM to during deployment.

3. **Deploy a VM—** A VM can be deployed using the deployment API. The deployment API allows you to provide values to the parameters that are passed to the system during deployment. Depending on the VM you are deploying, some parameters are mandatory and others optional.

4. **Manage and Monitor a VM—**You can monitor a VM using APIs and commands that enable you to get the VM status and debug logs. Using VM management APIs, you can start, stop, or reboot a VM, and view statistics for a VM such as CPU usage.

   A VM can also be managed by changing or updating its profile. You can change a VM's profile to one of the existing profiles in the image file; alternatively, you can create a new custom profile for the VM.

   The vNICs on a deployed VM can also be added or updated.

Note Before performing the VM life cycle management tasks, you will have to upload the VM images to the NFVIS server or http/s server.

For details on APIs, see the VM Lifecycle Management APIs chapter in the *API Reference for Cisco Enterprise Network Function Virtualization Infrastructure Software*.

Uploading VM Images to an NFVIS Server

You can upload VM images to an NFVIS server in the following ways. The files are copied to the default location (/data/intdatastore/uploads) on the host server.

- Copy the images from your local system to the NFVIS server—Use the **Image Upload** option from the Cisco Enterprise NFVIS portal.
- Copy the images using the USB drive—Ensure that you have plugged the USB drive that contains the required images into the server before mounting the USB drive.
- Copy using the **scp** command (scp username@external_server:/path/image.tar.gz intdatastore:image.tar.gz).

To copy an image using the USB device:

```
configure terminal
system usb-mount mount ACTIVE
system file-copy usb file name usb1/package/isrv-universalk9.16.03.01.tar.gz
commit
```

Note Use the **show system file-list disk usb** command in privileged EXEC mode to view a list of files available with the mounted USB drive. To save space, you can delete all unwanted text and TAR files from the default location using the **system file-delete** command in global configuration mode.

Verifying the Image Copied from the USB Drive

After copying the file from the USB drive to the host server, you can verify the file using the **show system file-list disk local** command:

```
nfvis# show system file-list disk local

SI NO     NAME     PATH                                                SIZE  TYPE    DATE MODIFIED
-------------------------------------------------------------------------------------------------------
1 lastlog-20170314.gz /data/intdatastore/logs/2017-03/14/10-00 337 Other 2017-03-14 21:55:42

2 escmanager-tagged-log.log-20170314.gz /data/intdatastore/logs/2017-03/14/10-00 167K Other
 2017-01-18 05:58:26
3 confd_audit.log-20170317.gz /data/intdatastore/logs/2017-03/17/09-30 4.6K Other 2017-03-17
 21:29:59
4 esc_postinit.log-20170317.gz /data/intdatastore/logs/2017-03/17/05-00 605K Other 2017-03-17
 16:40:19
5 error.log-20170317.gz /data/intdatastore/logs/2017-03/17/05-00 1.3K Other 2017-03-17
16:40:15
6 ovs-ctl.log-20170317.gz /data/intdatastore/logs/2017-03/17/12-00 20 Other 2017-03-16
```

```
00:00:01 4:01
!
!
!
62 ovs-ctl.log-20170323.gz /data/intdatastore/logs/2017-03/23/12-00 20 Other 2017-03-22
00:00:01
63 CentOS-7-x86_64-Everything-1511.ova /data/intdatastore/uploads 1.1G VM 2017-03-15 19:20:03
 Package
64 TinyLinux.tar.gz /data/intdatastore/uploads 17M VM 2017-03-15 18:25:00 Package
65 Cisco-KVM-vWAAS-1300-6.3.0-b98.tar.gz /data/intdatastore/uploads 979M VM 2017-03-15
19:19:11 Package
66 ubuntu_14.04.3-server-amd64-disk1.tar /data/intdatastore/uploads 527M VM 2017-03-15
19:20:17.gz Package
67 asav961.tar.gz /data/intdatastore/uploads 164M VM 2017-03-15 18:24:57 Package
68 isrv-universalk9.16.03.01.tar.gz /data/intdatastore/uploads 1.3G VM 2017-03-15 19:19:53
```

Related APIs and Commands

APIs	CLI Commands
• /api/operations/system/file-copy/usb/file • /api/config/system/usb-mount	• system file-copy usb file name • system usb-mount mount ACTIVE • system file-delete • show system file-list disk usb • show system file-list disk local

Performing Resource Verification

Given below are the APIs and commands to perform different types of resource verification:

Task	API	Command
To display CPU information for each CPU or the user specified CPU, and the VMs pinned to the CPU	• /api/operational/resources/cpu-info/cpus • /api/operational/resources/cpu-info/cpus/cpu • /api/operational/resources /cpu-info/cpus/cpu/<cpu-id>	show resources cpu-info cpus
To display information on the VMs running in all the physical CPUs or a specific physical CPU in the system	• /api/operational/resources/cpu-info/vnfs • /api/operational/resources/cpu-info/vnfs/vnf • /api/operational/resources/cpu-info/vnfs/vnf/ <deployment_name>.<vm_group_name>	show resources cpu-info vnfs

Task	API	Command
To get information on the number of CPUs allocated to VMs and the CPUs that are already used by the VMs	/api/operational/resources/cpu-info/allocation	show resources cpu-info allocation

Note To display information on all CPUs, VMs pinned to the CPUs, and VMs allocated to the CPUs, use the **show resources cpu-info** command.

CPU Over-Subscription

Cisco Enterprise NFVIS does not allow CPU over-subscription for low-latency network appliance VMs (for example, Cisco ISRv and Cisco ASAv). However, the CPU over-subscription is allowed for non low-latency VMs (for example, Linux Server VM and Windows Server VM).

Configuring Management IP Subnet

By default, all VMs with management interfaces will be provisioned with an IP address in the subnet of 10.20.0.1. To change the default subnet, the following commands needs to be executed in a sequence to first delete an exiting subnet and then add a new subnet in the network. For these commands to work, ensure there is no managed VNF's in the system before you change management network address.

To delete an existing subnet use **no vm_lifecycle networks network int-mgmt-net subnet int-mgmt-net-subnet** command.

To create a new subnet:

```
configure terminal
vm_lifecycle networks network int-mgmt-net subnet int-mgmt-net-subnet address 105.20.0.0
gateway 105.20.0.1 netmask 255.255.255.0 dhcp false
commit
```

VM Image Packaging

VM Image Packaging is a tool for converting qcow2 and img images into a tar.gz format with additional properties and profiles. VM image packaging can be done in two ways:

- VM Image Packaging Utility: This is an enhanced packaging process that allows the VM owner to run the **nfvpt.py** utility as a command with a combination of parameters to package the VM.
- Standard Image Packaging: This is a manual process in which a raw disk image (qcow2, img) is packaged along with the image properties file and bootstrap files (if needed) into a TAR archive file.

VM Image Packaging Utility

A VM image package is a TAR archive file with the root disk image and other descriptor files. This packaging method simplifies the process of a VM image registration and deployment. The attributes specified for the image enable resource requirement specification, creation of VM profiles, and a host of other properties for the VM.

The Cisco Enterprise NFVIS VM image packaging tool, nfvpt.py, helps VM owners package their VMs. The tool takes one or more qcow2 images (raw disk file) as the input file along with VM specific properties, bootstrap configuration files (if any), and generates a compressed TAR file.

Contents

The VM image packaging utility contains the following:

- nfvpt.py—It is a python based packaging tool that bundles the VM raw disk image/s along with VM specific properties.
- image_properties_template.xml—This is the template file for the VM image properties file, and has the parameters with default values. If the user provides new values to these parameters while creating the VM package, the default values get replaced with the user-defined values.
- nfvis_vm_packaging_utility_examples.txt—This file contains examples on how to use the image packaging utility to package a VM image.

Usage

To get the list of parameters that can be included in the command, and to get an explanation of each of the parameters, run the **help** command for the tool.

```
nfvpt.py --help
optional arguments:
  -h, --help            show this help message and exit
  --json JSON           Provide JSON input for bootstrap variables; mutually
                        exclusive with custom and bootstrap configs
  --newjson NEWJSON     Provide JSON input for bootstrap variables; mutually
                        exclusive with custom and bootstrap configs
  --log_dir LOG_DIR     Log Directory to for logfiles
  --multi_use           Add options for use in multiple use-cases
  --console_type_serial {true,false}
                        Attach the console serial to the VM; default is false;
                        --console_type_serial=true/false;
  --root_file_disk_bus {virtio,ide}
                        root disk file type: --root_file_disk_bus=virtio/ide;
                        default is virtio
  --virtual_interface_model {rtl8139}
                        --virtual_interface_model=rtl8139; default is none
  --thick_disk_provisioning {true,false}
                        --thick_disk_provisioning=true; default is false
  --eager_zero {true,false}
                        --eager_zero=true; default is false
  --nocloud {true,false}
                        --nocloud=true/false; default is false
  --bootstrap_cloud_init_bus_type {ide,virtio}
                        --bootstrap_cloud_init_bus_type=virtio; default is ide
  --bootstrap_cloud_init_drive_type {cdrom,disk}
                        --bootstrap_cloud_init_drive_type=disk; default is
                        cdrom
```

```
  --bootstrap BOOTSTRAP
                        Every bootstrap file should be a different option Non
                        HA format: --bootstrap
                        <mountpoint>:<file1>,<mountpoint>:<file2>... See
                        usage.txt for more details HA format for SDWAN
                        NetworkHub: --bootstrap mount_point:<value>,file:<file
                        2mount>[,<attrib>:<value>] mount_point:<value> and
                        file:<file2mount> are mandatory followed by one or
                        more attributes in the format <attrib>:<value>
  --interface_hot_add {true,false}
                        VM supports interface add without power off. Default
                        is set to true; --interface_hot_add=true/false
  --interface_hot_delete {true,false}
                        VM supports interface delete without power off.
                        Default is set to false;
                        --interface_hot_delete=true/false
  -v, --verbose         verbose
  -q, --quiet           quiet
  --no_compress         creates tar file without compressing the input files
  --cleanup             deletes all the input and configuration files upon tar
                        file created
  --tablet {true,false}
                        : Add input device of type tablet --tablet=true/false;
  --ha_package          enable HA packaging
  --mgmt_vnic MGMT_VNIC
                        VM management interface identifier
  --pack_dir <DIR> PACK
                        package all files in directory

Required:
  -o PACKAGE_FILENAME, --package_filename PACKAGE_FILENAME
                        [REQUIRED] file name for the target VNF package name-
                        default is root disk image name with extension .tar.gz
  -i ROOT_DISK_IMAGE, --root_disk_image ROOT_DISK_IMAGE
                        [REQUIRED] List of root disk images to be bundled
                        example: --root_disk_image isrv.qcow2;
                        --root_disk_image isrv1.qcow2,isrv2.qcow2
  --prop_template PROP_TEMPLATE
                        image properties template file name including path
                        default path is the current dir of the tool and name
                        is image_properties_template.xml if the user doesn't
                        input this option example: --prop_template
                        /usr/bin/image_properties_template.xml
  -t VNF_TYPE, --vnf_type VNF_TYPE
                        [REQUIRED] VNF type, e.g. ROUTER, FIREWALL, vWAAS,
                        vWLC, and OTHER
  -n NAME, --vnf_name NAME
                        [REQUIRED] Name of the VNF image
  -r VNF_VERSION, --vnf_version VNF_VERSION
                        [REQUIRED] VNF version, e.g. --vnf_version 1.0 or
                        --vnf_version 0.9
  --app_vendor APP_VENDOR
                        Application Vendor e.g. Cisco, Juniper etc
  --monitored {true,false}
                        [REQUIRED] Monitored VNF: --monitored=true/false;
  --optimize {true,false}
                        [REQUIRED] optimized VM: --optimize=true/false;

HA options:
  --ha_capable
  --ha_vnic HA_VNIC     VM HA vnic
  --ha_vnic_count HA_VNIC_COUNT
                        Number of ha_vnics
```

```
Resources:
  Resources: min and max - vCPU, memory and disk

  --min_vcpu VCPU_MIN   min #vCPU : min number of vCPU supported by VM
                        example:--min_vcpu 2
  --max_vcpu VCPU_MAX   max #vCPU : max number if vCPU required for VM
                        example:--max_vcpu 4
  --min_mem MEMORY_MB_MIN
                        min mem : min mem in MB required for VM
                        example:--min_mem 1024
  --max_mem MEMORY_MB_MAX
                        max mem : max mem in MB required for VM
                        example:--max_mem 4196
  --min_disk ROOT_DISK_GB_MIN
                        min disk : min disk in GB required for VM
                        example:--min_disk 8
  --max_disk ROOT_DISK_GB_MAX
                        max disk : max disk in GB required for VM
                        example:--max_disk 8
  --vnic_max VNIC_MAX   max number of Vnics allowed for VM example:--vnic_max
                        8
  --vnic_names VNIC_NAMES
                        list of vnic number to name mapping in format
                        number:name example --vnic_names
                        1:GigabitEthernet2,2:GigabitEthernet4

Profile Options:
  --profile PROFILE     enter the profile name, profile description, no of
                        vCPU required, min memory required in MB, min disk
                        space required in MB, example: --profile
                        profile1,"This is profile 1",2,2048,4096 --profile
                        profile2,"This is profile 2",4,4096,4096
  --default_profile DEFAULT_PROFILE
                        default profile

Driver Support Options:
  --sriov {true,false}  Enable/Disable SRIOV support: --sriov=true/false;
                        default is false
  --sriov_list SRIOV_DRIVER_LIST
                        list of SRIOV drivers example: --sriov_list
                        igb,igbvf,i40evf
  --pcie {true,false}   Not supported
  --pcie_list PCIE_DRIVER_LIST
                        Not supported

Privilege/Priority Options:
  --privileged {true,false}
                        Not supported

Custom Properties:
  --custom CUSTOM       custom properties format: --custom ["propattr_"<attr>:
                        <value>],key:<value>,[keyattr_<attr>:<value>],type:<va
                        lue>,val<N>:<value>,[val<N>attr_<attr>:<value>] Allows
                        specification of custom properties: 0 or more
                        propattr_<attr>:<value> pairs - 'propattr' is a
                        keyword and used to specify property attributes
                        key:<value> pairs 0 or more keyattr_<attr>:value pairs
                        - 'keyattr' is a keyword and is used to specify key
                        attributes type:<value> pair - type of value
                        valN:<value> pair - val1:value,val2:value etc 0 or
                        more valNattr_<attr>:<value> pairs - 'val<N>attr' is
                        an attribute for val<N> See usage_examples.txt
```

The table lists the parameters that can be passed to the **nfvpt.py** command.

Parameter	Mandatory/Optional	Description
version	Not applicable	Show program's version number and exit.
help	Not applicable	Show this help message and exit.
package_file_name	Mandatory	File name for the target VNF package. The default is the root disk image name with extension *.tar.gz*.
disk_img_names	Mandatory	List of root disk images to be bundled. Only the qcow2 images are supported.
img_name	Mandatory	Name of the VNF image.
vnf_type	Mandatory	VNF type Supported types are: ROUTER, FIREWALL, vWAAS, vWLC, and OTHER.
vnf_version	Mandatory	VNF version
monitored	Mandatory	VM health monitoring for those VMs that can be bootstrapped Options are: true/false Monitoring timeout period for a monitored VM is 600 seconds by default
optimize	Mandatory	Optimized VM Options are: true/false
virtual_interface_model	Optional	Default is none.
thick_disk_provisioning	Optional	Default is false.
eager_zero	Optional	Default is false.
bootstrap_cloud_init_bus_type	Optional	Default is IDE.
bootstrap_cloud_init_drive_type	Optional	Mounts the day0 configuration file as disk Default is CD-ROM.
bootstrap	Optional	Bootstrap files for VNF. Two parameters are required in the format of dst:src; dst filename including path has to match exactly to what the VM expects; up to 20 bootstrap files are accepted. For example: --bootstrap ovf-env.xml for ISRv and --bootstrap day0-config for ASAv.

Parameter	Mandatory/Optional	Description
min_vcpu	Optional	Minimum number of vCPUs supported by the VM. The default is 1.
max_vcpu	Optional	Maximum number of vCPUs required for the VM. The default is 8.
min_mem	Optional	Minimum memory in MB required for the VM. The default is 4 GB.
max_mem	Optional	Maximum memory in MB required for the VM. Physical memory: 2 GB The default is 8 GB.
min_disk	Optional	Minimum disk in GB required for the VM. The default is 8 GB.
max_disk	Optional	Maximum disk in GB required for the VM. Available disks are SSD and HDD: 15 GB The default is 16 GB
vnic_max	Optional	Maximum number of VNICs allowed for the VM. The default is 8.
profile	Optional	The profile name, profile description, number of vCPUs required, minimum memory required in MB and minimum disk space required in MB.
default_profile	Optional	The default profile.
sriov	Optional	Enable or disable SRIOV support. The default is false.
sriov_list	Optional	List of SRIOV drivers.
pcie	Optional	Not supported.
pcie_list	Optional	Not supported.
privileged	Optional	Not supported.

Parameter	Mandatory/Optional	Description
custom	Optional	Custom properties to be supported and/or passed to the bootstrap configuration with tokenized variables. This is only used for the local portal to display options for the user to choose while deploying.
pack_dir	Optional	package all files in directory

NFVIS Specific Enhancements

Note Use pack_dir option if the *.tar.gz already exists and you want to modify the bootstrap configuration file or image_properties.xml manually.

The following parameters are added as part of the NFVIS specific enhancements:

```
 --pack_dir <DIR> PACK

                       package all files in directory

Resources:

  --vnic_names VNIC_NAMES

                       list of vnic number to name mapping in format

                       number:name example --vnic_names

                       1:GigabitEthernet2,2:GigabitEthernet4
```

Usage

Follow the steps to change a single line in day-0 configuration file or add a single option in image_properties.xml:

1. Get the working VM packaging image - isrv*.tar.gz.
2. Extract the contents - tar -xvf isrv*.tar.gz.
3. Modify the file contents as required.
4. nfvpt.py --pack_dir current-working-dir-with-files -i isrv.qcow2 -o isrv.tar.gz

VM Packaging Utility Usage Examples

Given below are the contents of the file *nfvis_vm_packaging_utility_examples.txt*:

Example 1: Usage for TinyLinux

```
nfvpt.py -o TinyLinux -i TinyLinux.qcow2 -n TinyLinux -t linux -r 1.0 --monitored false
--min_vcpu 1 --max_vcpu 2 --min_mem 1024 --max_mem 1024 --min_disk 1 --max_disk 2
--vnic_max 1 --optimize false
```

Example 2: Usage for ASAv



Note The bootstrap filename has to be *day0-config*. This cannot be modified as ASAv looks for the exact filename.

```
nfvpt.py -o asav961-201 -i asav961-201.qcow2 -n ASAv -t firewall -r 961-201 --monitored
true --bootstrap day0-config:filename1
--min_vcpu 1 --max_vcpu 4 --min_mem 1024 --max_mem 8192 --min_disk 8 --max_disk 16 --vnic_max
 8 --optimize true
--profile ASAv5,"ASAv5 profile",1,1024,8192 --profile ASAv10,"ASAv10 profile",1,4096,8192
--profile ASAv30,"ASAv30 profile",4,8192,16384
--default_profile ASAv5
```

Example 3: Usage for ISRv



Note The bootstrap filename has to be *ovf-env.xml* . This cannot be modified as ISRv looks for the exact filename.

```
nfvpt.py -o isrv.16.03.01 -i isrv-universalk9.16.03.01.qcow2 -n ISRv.16.03.01 -t ROUTER -r
 16.03.01 --monitored true --privileged true
--bootstrap ovf-env.xml:file1,ios-xe.txt:file2 --min_vcpu 2 --max_vcpu 8 --min_mem 4096
--max_mem 8192 --min_disk 8 --max_disk 8
--vnic_max 8 --optimize true --profile ISRv-small,"ISRv small profile",2,4096,8192 --profile
 ISRv-medium,"ISRv medium profile",4,4096,8192
--default_profile ISRv-small --sriov_list igb,igbvf,i40evf --custom tech_package,ax
```

Example 4: Usage for a third party VM with config drive (ISO) mounted at specific path on the VM:

```
nfvpt.py -o test.1.0 -i test-1.0.qcow2 -n TEST -t OTHER -r 1.0 --monitored true --privileged
 true
--bootstrap /:bootstrap.xml,/license/lic.txt:license.txt --min_vcpu 2 --max_vcpu 8 --min_mem
 4096 --max_mem 8192
--min_disk 8 --max_disk 8 --vnic_max 8 --optimize true --profile small,"small
profile",2,4096,8192
--profile medium,"medium profile",4,4096,8192 --default_profile small
```

In this case, *test.1.0.pkg : bootstrap.xml* gets mounted as *bootstrap.xml* at the root, and the *license.txt* gets mounted as */license/lic.txt*.

Example 5: Usage for Palo Alto Firewall

```
nfvpt.py -o PA_L3_HA -i PA-VM-KVM-8.0.5.qcow2 --json d.json -t firewall -n "PA FIREWALL"
-r 8.0.5 --app_vendor PA --monitor true --ha_package
```

Example 6: Usage for Asav

```
nfvpt.py -i foo.qcow2 -o asav.tar.gz --json pa1.json --app_vendor cisco -t firewall -r 10
--optimize true -n asav --monitored true --ha_package —ha_capable
```

Example 7: Usage for csr

```
nfvpt.py --ha_package --pack_dir /data/intdatastore -i csr1000v-universalk9.16.09.01.qcow2
  -o csr1000v-universalk9.16.09.01-ha.tar.gz
```

Standard VM Image Packaging

The standard VM packaging is based on the Open Virtualization Format (OVF) packaging standard, in which a single file is distributed in open virtualization appliance (OVA) format. The VM image is shared using a TAR archive file with the root disk image and descriptor files.



Note Cisco Enterprise NFVIS supports VM packaging in *.tar.gz* (compressed form of OVA) format. Ensure that all supported third party VM images are available in the supported format.

Generating a VM Package

Package files are provided for Cisco ISRv, Cisco ASAv, and tiny Linux and Windows server 2000. Vendors are responsible for packaging all third party VMs in the supported format.

1. Create a VM qcow2 image.
2. Create an *image_properties.xml* file with the VM properties. Ensure that you add all mandatory fields. Include the profiles supported for the VM in this file, and select one default profile. If you do not want to monitor the VM bootup, make the bootup time as -1.
3. Create *bootstrap-config* or *day0-config*, if any bootstrap configuration is required for the VM. If the bootstrap configuration requires inputs from the user, use the tokens in the xml or text file. These tokens are populated during the VM deployment with the provided data.



Note A VM deployment may fail, if there are tokens in the configuration, and the user does not provide the token values in the deployment payload.

4. Create a *package.mf* file, which lists all the files to be bundled into the *.tar.gz* file along with checksums.
5. Generate the packaging file using "tar -cvzf ova_file_name list_of_files_to_be_bundled".

   For example, *tar -cvzf isrv.tar.gz isrv-universalk9.03.16.02.S.155-3.S1a-ext-serial.qcow2 image_properties.xml isr_ovf_env.xml package.mf*.

Appendix

VM Image Package Files

The table lists the contents of the VM package that are generated using the packaging tool:

Table 1: VM Image Package Files

File	Description	Mandatory/Optional
Package Manifest (package.mf)	Lists the files in the package and the expected checksum for the files.	Mandatory

VM image properties (vmname_properties.xml)	XML file with resources and features supported by the VM	Mandatory
VM image (vmname.qcow2)	Image file of the VM. Multiple images are supported. One root_disk image file is mandatory.	Mandatory
Bootstrap (bootstrap_file)	Optional	Bootstrap files for VNF. Two parameters are required in the format of dst:src; dst filename including path has to match exactly to what the VM expects; up to 20 bootstrap files are accepted. For example: --bootstrap ovf-env.xml for ISRv and --bootstrap day0-config for ASAv.

Package Manifest File

The package manifest XML file provides a list of the files in the package with their names and their expected checksum. SHA1 algorithm (sha1sum) is used to calculate the checksum. This is a mandatory file to be bundled in the VM package. The manifest file must be named as *package.mf*.

Table 2: Package Manifest File Details

Property Name	Description	Property Tag	Mandatory/Optional
File information	XML tree with details of file name, file type, and expected checksum. The root_image and image_properties files are required.	<file_info>	Mandatory
File name	Name of the file	<name>	Mandatory
File type	Describes the file type. Supported types: • root_image • image_properties • bootstrap_config_file • ephemeral_disk1_image • ephemeral_disk2_image	<type>	Mandatory
Expected checksum	The calculated SHA1 checksum to be validated.	<sha1_checksum>	Mandatory

Bootstrap Configuration File

The bootstrap configuration file is an XML or a text file, and contains properties specific to a VM and the environment. Properties can have tokens, which can be populated during deployment time from the deployment payload.

VM Image Properties File

This XML file provides information about the resources supported or required for the VM operation. All mandatory parameters have to be defined. It also supports custom attributes. This is a mandatory file to be bundled in the VM package. The VM package supports up to 10 disks to be bundled into the package.

Table 3: VM Image Properties File Details

Property Name	Description	Property Tag	Possible Values	Mandatory/Optional
VNF Type	VM functionality provided. Router and firewall are predefined types.	<vnf_type>	Router, firewall, Windows, Linux, and custom_type	Mandatory
Name	Name associated with the VM packaging. This name is referenced for VM deployment.	<name>	Any	Mandatory
Version	Version of the package	<version>	Any	Mandatory
Boot-up time	Boot-up time (in seconds) of the VNF before it can be reachable via ping.	<bootup_time>	Any in seconds, (-1) to not monitor boot-up	Mandatory
Root Disk Image Bus	Root image disk bus	<root_file_disk_bus>	virtio, scsi, and ide	Mandatory
Disk-1 bus type	Additional disk1 image disk bus	<disk_1_file_disk_bus>	virtio, scsi, and ide	Optional
Disk-2 bus type	Disk2 image disk bus	<disk_2_file_disk_bus>	virtio, scsi, and ide	Optional
Disk-10 bus type	Disk10 image disk bus	<disk_10_file_disk_bus>	virtio, scsi, and ide	Optional
Root Disk Image format	Root image disk format	<root_image_disk_format>	qcow2 and raw	Mandatory
Disk-1 Image format	Additional disk 1 image format	<disk_1_image_format>	qcow2 and raw	Optional

Disk-2 Image format	Disk 2 image format	<disk_2_image_format>	qcow2 and raw	Optional
Disk-10 Image format	Disk 10 image format	<disk_10_image_format>	qcow2 and raw	Optional
Serial Console	Serial console supported	<console_type_serial>	true, false	Optional
Minimum vCPU	Minimum vCPUs required for a VM operation	<vcpu_min>		Mandatory
Maximum vCPU	Maximum vCPUs supported by a VM	<vcpu_max>		Mandatory
Minimum memory	Minimum memory in MB required for VM operation	<memory_mb_min>		Mandatory
Maximum memory	Maximum memory in MB supported by a VM	<memory_mb_max>		Mandatory
Minimum root disk size	Minimum disk size in GB required for VM operation	<root_disk_gb_min>		Optional
Maximum root disk size	Maximum disk size in GB supported by a VM	<root_disk_gb_max>		Optional
Maximum vNICs	Maximum number of vNICs supported by a VM	<vnic_max>		Mandatory
SRIOV support	SRIOV supported by VM interfaces. This should have a list of supported NIC device drivers.	<sriov_supported>	true, false	Optional
SRIOV driver list	List of drivers to enable SRIOV support	< sriov_driver_list>		Optional

PCI passthru support	PCI passthru support by VM interfaces	<pcie_supported>	true, false	Optional
PCIE driver list	List of VNICS to enable PCI passthru support	< pcie _driver_list>		Optional
bootstrap_cloud_init_drive_type	Mounts day0 config file as disk (default is CD-ROM)	<bootstrap_cloud_init_drive_type>	disk, cdrom	Optional
bootstrap_cloud_init_bus_type	Default is IDE	<bootstrap_cloud_init_bus_type>	virtio, ide	Optional
BOOTSTRAP	Bootstrap files for the VNF. Two parameters are required in the format of dst:src; dst filename including path has to match exactly to what the VM expects; up to 20 bootstrap files are accepted. For example: --bootstrap ovf-env.xml for ISRv and --bootstrap day0-config for ASAv	< bootstrap_file>	File name of the bootstrap file	Optional

Custom properties	List of properties can be defined within the custom_property tree. (Example: For ISRv, the technology packages are listed in this block.) If the Cisco Enterprise NFV portal is used to deploy the VM, the portal prompts you for inputs for custom properties fields, and can pass the values to the bootstrap configuration.	<custom_property>		Optional
Profiles for VM deployment	List of VM deployment profiles. Minimum one profile is required	<profiles>		Optional
Default profile	The default profile is used when no profile is specified during deployment.	<default_profile>		Optional
Monitoring Support	A VM supports monitoring to detect failures.	<monitoring_supported>	true, false	Mandatory
Monitoring Method	A method to monitor a VM. Currently, only ICMP ping is supported.	<monitoring_methods>	ICMPPing	Mandatory if monitoring is true
Low latency	If a VM's low latency (for example, router and firewall) gets dedicated resource (CPU) allocation. Otherwise, shared resources are used.	<low_latency>	true, false	Mandatory

Privileged-VM	Allows special features like promiscuous mode and snooping . By default, it is false.	<privileged_vm>	true, false	Optional
Virtual interface model		<virtual_interface_model>		Optional
Thick disk provisioning	By default, it is false.	<thick_disk_provisioning>	true, false	Optional
Profile for VM deployment	A profile defines the resources required for VM deployment. This profile is referenced during VM deployment.	<profile>		Optional
Name	Profile name	<name>	Any	Mandatory
Description	Description of the profile	<description>	Any	Mandatory
vCPU	vCPU number in a profile	<vcpus>		Mandatory
Memory	Memory - MB in profile	<memory_mb>		Mandatory
Root Disk Size	Disk size - MB in profile .	<root_disk_mb>		Mandatory
VNIC Offload	List of properties that can be set for vnic offload	<vnic_offload>		Optional
Generic Segmentation Offload	Turn generic segmentation offload on or off	<generic_segmentation_offload> (parent: <vnic_offload>)	on, off	Optional
Generic Receive Offload	Turn generic receive offload on or off	<generic_receive_offload> (parent: <vnic_offload>)	on, off	Optional
RX Checksumming	Turn RX checksumming on or off	<rx_checksumming> (parent: <vnic_offload>)	on, off	Optional
TX Checksumming	Turn TX checksumming on or off	<tx_checksumming> (parent: <vnic_offload>)	on, off	Optional

TCP Segmentation Offload	Turn TCP segmentation offload on or off	<tcp_segmentation_offload> (parent: <vnic_offload>)	on, off	Optional



Note A virtual console is supported by default. Specify the root disk size as zero for multiple disks (for example, vWaas deployment) as the system does not support populating multiple disk sizes. Actual disk sizes are calculated from the root_disk files.

Example: Package.mf

```
** sha1sum - for calculating checksum
<PackageContents>
  <File_Info>
    <name>ISRv_serial_3.16.02.qcow2</name>
    <type>root_image</type>
    <sha1_checksum>93de73ee3531f74fddf99377972357a8a0eac7b</sha1_checksum>
  </File_Info>
  <File_Info>
    <name>image_properties.xml</name>
    <type>image_properties</type>
    <sha1_checksum>c5bb6a9c5e8455b8698f49a489af3082c1d9e0a9</sha1_checksum>
</File_Info>
  <File_Info>
    <name>ISRv_ovf_env.xml</name>
    <type> bootstrap_file_1</type>
    <sha1_checksum>c5bb6a9c5e8455b8698f49a489af3082c1d9e0a9</sha1_checksum>
   </File_Info>
   <File_Info>
     <name>ISRv_disk1_image.qcow2</name>
     <type>ephemeral_disk1_image</type>
     <sha1_checksum>aac24513098ec6c2f0be5d595cd585f6a3bd9868</sha1_checksum>
   </File_Info>
</PackageContents>
```

Example: Image Properties

```
<?xml version="1.0" encoding="UTF-8"?>
<image_properties>
    <vnf_type>ROUTER</vnf_type>
    <name>isrv-universalk9</name>
    <version>03.16.02</version>
    <bootup_time>600</ bootup_time >
    <root_file_disk_bus>virtio</root_file_disk_bus>
    <root_image_disk_format>qcow2</root_image_disk_format>
    <vcpu_min>1</vcpu_min>
    <vcpu_max>8</vcpu_max>
    <memory_mb_min>4096</memory_mb_min>
    <memory_mb_max>8192</memory_mb_max>
    <vnic_max>8</vnic_max>
    <root_disk_gb_min>8</root_disk_gb_min>
    <root_disk_gb_max>8</root_disk_gb_max>
    <console_type_serial>true</console_type_serial>
    <sriov_supported>true</sriov_supported>
    <sriov_driver_list>igb</sriov_driver_list>
    <sriov_driver_list>igbvf</sriov_driver_list>
```

```
    <sriov_driver_list>i40evf</sriov_driver_list>
    <pcie_supported>true</pcie_supported>
    <pcie _driver_list> igb </pcie_driver_list>
    <pcie _driver_list> igbvf</pcie_driver_list>
    <pcie _driver_list> i40evf</pcie_driver_list>
    <bootstrap_file_1> ovf-env.xml </bootstrap_file_1>
    <monitoring_supported>true</monitoring_supported>
    <monitoring_methods>ICMPPing</monitoring_methods>
    <low_latency>true</low_latency>
    <privileged_vm>true</privileged_vm>
    <cdrom>true</cdrom>
    <custom_property>
        <tech_package>ax</tech_package>
        <tech_package>sec</tech_package>
        <tech_package>ipbase</tech_package>
        <tech_package>appx</tech_package>
    </custom_property>
    <profiles>
         <profile>
            <name>ISRv1kv-small</name>
            <description>ISRv upto 50MBPS performance</description>
            <vcpus>1</vcpus>
            <memory_mb>4096</memory_mb>
            <root_disk_mb>8</root_disk_mb>
         </profile>
         <profile>
            <name>ISRv1kv-medium</name>
            <description>ISRv upto 250MBPS performance</description>
            <vcpus>2</vcpus>
            <memory_mb>4096</memory_mb>
            <root_disk_mb>8</root_disk_mb>
         </profile>
    </profiles>
    <default_profile>small</default_profile>
 </image_properties>
```

Example: Bootstrap Configuration File

```
<?xml version="1.0" encoding="UTF-8"?>
<Environment
xmlns:oe="http://schemas.dmtf.org/ovf/environment/1">
  <PropertySection>
    <Property oe:key="com.cisco.ISRv.config-version.1" oe:value="1.0"/>
    <Property oe:key="com.cisco.isrv.enable-ssh-server.1" oe:value="True"/>
    <Property oe:key="com.cisco.isrv.login-password.1" oe:value="admin"/>
    <Property oe:key="com.cisco.isrv.login-username.1" oe:value="lab"/>
    <Property oe:key="com.cisco.isrv.mgmt-interface.1" oe:value="GigabitEthernet1"/>
   <Property oe:key="com.cisco.isrv.mgmt-ipv4-addr.1" oe:value="${NICID_0_IP_ADDRESS}/24"/>

    <Property oe:key="com.cisco.isrv.mgmt-ipv4-network.1" oe:value=""/>
    <Property oe:key="com.cisco.isrv.license.1" oe:value="${TECH_PACKAGE}"/>
   <Property oe:key="com.cisco.isrv.ios-config-0001" oe:value="vrf definition Mgmt-intf"/>

    <Property oe:key="com.cisco.isrv.ios-config-0002" oe:value="address-family ipv4"/>
    <Property oe:key="com.cisco.isrv.ios-config-0003" oe:value="exit-address-family"/>
    <Property oe:key="com.cisco.isrv.ios-config-0004" oe:value="address-family ipv6"/>
    <Property oe:key="com.cisco.isrv.ios-config-0005" oe:value="exit-address-family"/>
    <Property oe:key="com.cisco.isrv.ios-config-0006" oe:value="exit"/>
   <Property oe:key="com.cisco.isrv.ios-config-0007" oe:value="interface GigabitEthernet1"/>

   <Property oe:key="com.cisco.isrv.ios-config-0008" oe:value="vrf forwarding Mgmt-intf"/>
```

```
    <Property oe:key="com.cisco.isrv.ios-config-0009" oe:value="ip address
${NICID_0_IP_ADDRESS} ${NICID_0_NETMASK}"/>
    <Property oe:key="com.cisco.isrv.ios-config-0010" oe:value="no shut"/>
    <Property oe:key="com.cisco.isrv.ios-config-0011" oe:value="exit"/>
    <Property oe:key="com.cisco.isrv.ios-config-0012" oe:value="ip route vrf Mgmt-intf
0.0.0.0 0.0.0.0 ${NICID_0_GATEWAY}"/>
  </PropertySection>
</Environment>
```

Image Properties Template File

The parameters that go into the image properties file are listed in the code extract below.

```
<?xml version="1.0" encoding="UTF-8"?>
<image_properties>
    <vnf_type>ROUTER</vnf_type>
    <name>TEMPLATE</name>
    <version>1.0</version>
    <bootup_time>600</bootup_time>
    <root_file_disk_bus>virtio</root_file_disk_bus>
    <root_image_disk_format>qcow2</root_image_disk_format>
    <vcpu_min>1</vcpu_min>
    <vcpu_max>8</vcpu_max>
    <memory_mb_min>4096</memory_mb_min>
    <memory_mb_max>8192</memory_mb_max>
    <vnic_max>8</vnic_max>
    <root_disk_gb_min>8</root_disk_gb_min>
    <root_disk_gb_max>16</root_disk_gb_max>
    <console_type_serial>false</console_type_serial>
    <sriov_supported>true</sriov_supported>
    <sriov_driver_list>s1</sriov_driver_list>
    <sriov_driver_list>s2</sriov_driver_list>
    <sriov_driver_list>s3</sriov_driver_list>
    <pcie_supported>false</pcie_supported>
    <monitoring_supported>true</monitoring_supported>
    <monitoring_methods>ICMPPing</monitoring_methods>
    <low_latency>true</low_latency>
    <privileged_vm>false</privileged_vm>
    <cdrom>true</cdrom>
    <bootstrap_file_1>b1.xml</bootstrap_file_1>
    <bootstrap_file_2>b2.txt</bootstrap_file_2>
    <custom_property>
        <key>val</key>
    </custom_property>
    <profiles>
        <profile>
            <name>small</name>
            <description>small</description>
            <vcpus>1</vcpus>
            <memory_mb>1024</memory_mb>
            <root_disk_mb>4096</root_disk_mb>
        </profile>
        <profile>
            <name>medium</name>
            <description>medium</description>
            <vcpus>2</vcpus>
            <memory_mb>4096</memory_mb>
            <root_disk_mb>8192</root_disk_mb>
        </profile>
```

```
    </profiles>
    <default_profile>small</default_profile>
</image_properties>
```

Image Registration

To register a VM image, you must first copy or download the relevant VM image to the NFVIS server, or host the image on a http or https server. Once you have downloaded the file, you can register the image using the registration API. The registration API allows you to specify the file path to the location (on the http/https server) where the tar.gz file is hosted. Registering the image is a one-time activity. Once an image is registered on the http or https server, and is in active state, you can perform multiple VM deployments using the registered image. All VM images are available in VM packaging and VM package content. For more information see, VM Image Packaging Utility, on page 6

Register VM Packages Using REST API

This example shows the sequence of registering a tar.gz package on Cisco Enterprise NFVIS using REST API.

Post Image Registration

```
curl -v -u -k admin:admin -H Accept:application/vnd.yang.data+xml -H
Content-Type:application/vnd.yang.data+xml
-X POST https://209.165.201.1 /api/config/vm_lifecycle/images -d
'<image><name>WinServer2012R2.iso</name><src>file:///data/intdatastore/uploads/WinServer2012R2.iso/WinServer2012R2.iso</src></image>'

HTTP/1.1 201 Created
```

Get Image Status

```
curl -k -v -u admin:admin -H Accept:application/vnd.yang.data+xml -H
Content-Type:application/vnd.yang.data+xml -X
GET https://209.165.201.1/api/operational/vm_lifecycle/opdata/images/image/isrv-03.16.02?deep

HTTP/1.1 200 OK
<image xmlns="http://www.cisco.com/nfvis/vm_lifecycle" xmlns:y="http://tail-f.com/ns/rest"
xmlns:esc="http://www.cisco.com/nfvis/vm_lifecycle">
<name>isrv.03.16.02</name>
<image_id>585a1792-145c-4946-9929-e040d3002a59</image_id>
<public>true</public>
<state>IMAGE_ACTIVE_STATE</state></image>
```

Get Registered Image Status

```
Accept:application/vnd.yang.data+xml -H
Content-Type:application/vnd.yang.data+xml -X
GET https://209.165.201.1/api/config/vm_lifecycle/images?deep

HTTP/1.1 200 OK
<images xmlns="[http://www.cisco.com/esc/esc|http://www.cisco.com/nfvis/vm_lifecycle]"
xmlns:y="[http://tail-f.com/ns/rest|http://tail-f.com/ns/rest]" 
xmlns:esc="[http://www.cisco.com/nfvis/vm_lifecycle|http://www.cisco.com/nfvis/vm_lifecycle]">
<image>
<name>isrv-9.16.03.01</name>
<src>http://data/nfvos-pkg/isr/isrv-universalk9.16.03.01.tar.gz</src>
```

```
</image>
</images>
```

Delete Registered Image

```
curl -k -v -u admin:admin -H Accept:application/vnd.yang.data+xml -H
Content-Type:application/vnd.yang.data+xml -X
DELETE https://209.165.201.1/api/config/vm_lifecycle/images/image/isrv-3.16.0.1a

HTTP/1.1 204 No Content
```

For more information on REST APIs related to image registration, see API Reference for Cisco Enterprise NFVIS.

Register VM Image with Multiple Root Disks

If any image requires multiple root disks, you can specify it in the image properties file.

This example shows how to specify multiple root disks in image properties.

```
<image_properties>
 …
 <root_file_disk_bus>virtio</root_file_disk_bus>
 <root_image_disk_format>qcow2</root_image_disk_format>
 <disk_1_file_disk_bus>virtio</ disk_1_file_disk_bus>
 <disk_1_image_format>qcow2</ disk_1_image_format>
 <disk_2_file_disk_bus>virtio</ disk_2_file_disk_bus>
 <disk_2_image_format>qcow2</ disk_2_image_format>
 …
</image_properties>
```

Register a VM Image through a Root Disk

A VM can also be registered using just a disk image (qcow2 or iso) without packaging into a tar.gz. As there will be no image properties in this scenario, the default image properties are used.

Default Properties for Root Disk Registration

The following table lists the default properties that are provisioned if you register a VM image by uploading the root disk for the image.

Property Name	Property Tag	Default Value
Version	<version>	NA
VNF Type	<vnf_typ>	OTHER
VCPU Min	<vcpu_min>	1
VCPU Max	<vcpu_max>	64
Memory Min (MB)	<memory_mb_min>	256
Memory Max (MB)	<memory_mb_max>	1048576
Root Disk Min Size (GB)	<root_disk_gb_min>	1

Property Name	Property Tag	Default Value
Root Disk Max Size (GB)	<root_disk_gb_max>	10240
VNIC Max	<vnic_max>	8
Bootup Time	<bootup_time>	-1
Interface Hot Add	<interface_hot_add>	true
Interface Hot Delete	<interface_hot_delete>	false

Register a Remote VM Image

Cisco Enterprise NFVIS allows you to register a VM image that is stored at a remote location or a web server. by specifying the URL to the image location in the source field.

If the web server supports HTTPS, the you can choose to enable Certificate Validation to validate its authenticity. Certificate Validation requires the server's public key to be specified, either in string or file format, in the image registration payload. This allows NFVIS to perform asymmetric encryption and download/register the image file securely over HTTPS.

Example: POST Remote Image Registration from Webserver over HTTPS

```
curl -k -v -u admin:Esc123# -H
Accept:application/vnd.yang.data+xml -H
Accept:application/vnd.yang.data+xml -H Content-
Type:application/vnd.yang.data+xml -X POST
https://172.29.91.28/api/config/vm_lifecycle/images/ -d '
<image>
 <name>ASAV</name>
 <src>https://172.20.117.124/nfvis/asav982.tar.gz</src>
</image>'

HTTP/1.1 201 Created
```

Example: POST Remote Image Registration from Webserver over HTTPS

```
curl -k -v -u admin:Esc123# -H
Accept:application/vnd.yang.data+xml -H
Accept:application/vnd.yang.data+xml -H Content-
Type:application/vnd.yang.data+xml -X POST
https://172.29.91.28/api/config/vm_lifecycle/images/ -d '
<image>
 <name>ASAV</name>
 <src>https://172.20.117.124/nfvis/asav982.tar.gz</src>
 <certificate_validation>true</certificate_validation>
 <certificate_file>/data/intdatastore/uploads/pub_key.cert</certificate_file>
</image>'

HTTP/1.1 201 Created
```

Specify Storage Location for a VM Image

Cisco Enterprise NFVIS allows users to specify the location where the register image should be stored, using the *placement* property tag.

The following table lists the placement values supported and their respective mappings.

storage name	directory map
datastore1	/data
datastore2	/mnt/extdatastore1
datastore3	/mnt/extdatastore2
nfs:nfs_mount_name	/data/mount/nfs/nfs_mount_name
nfs or nfs:nfs	/data/mount/nfs/
nfs:nfs_storage or nfs_storage	/data/mount/nfs_storage

Note If your preferred storage location is **nfs**, you must have it configured to be mounted on NFVIS using appropriate CLIs before registering the image on it.

Example: VM Image Storage Placement

```
curl -k -v -u admin:Esc123# -H
Accept:application/vnd.yang.data+xml -H
Accept:application/vnd.yang.data+xml -H Content-
Type:application/vnd.yang.data+xml -X POST
https://172.29.91.28/api/config/vm_lifecycle/images/ -d '
<image>
 <name>ASAV</name>
 <src>https://172.20.117.124/nfvis/asav982.tar.gz</src>
 <properties>
  <property>
   <name>placement</name>
   <value>nfs:my_nfs_mount</value>
  </property>
 </properties>
</image>'

HTTP/1.1 201 Created
```

Update VM Image

You can only update the following image properties after a VM image has been registered.

- interface_hot_add
- interface_hot_delete

Any requests to modify other image properties are rejected.

Note When using the REST API, the previously set value of the property must be deleted before updating it with the new value.

Example: Delete Value and Add New Value

1. Delete the previously set property value as shown below.

```
curl -k -v -u admin:Esc123# -H
Accept:application/vnd.yang.data+xml -H
Accept:application/vnd.yang.collection+xml -X DELETE
https://172.29.91.28/api/config/vm_lifecycle/images/image/ISR_IMAGE/properties/property/interface_hot_add/value

HTTP/1.1 204 No Content
```

2. Add (PUT) the new property value to replace the one you deleted in the previous step.

```
curl -k -v -u admin:Esc123# -H
Accept:application/vnd.yang.data+xml -H
Accept:application/vnd.yang.data+xml -H
Content-Type:application/vnd.yang.data+xml -X
PUT
https://172.29.91.28/api/config/vm_lifecycle/images/image/ISR_IMAGE/properties/property/interface_hot_add
 --data '<value>true</value>'

HTTP/1.1 201 Created
```

Example: Get All Properties and Values

```
curl -k -v -u admin:Esc123# -H
Accept:application/vnd.yang.data+xml -H
Accept:application/vnd.yang.data+xml -H Content-
Type:application/vnd.yang.data+xml -X GET
https://172.29.91.28/api/config/vm_lifecycle/images/image/ISR_IMAGE/properties?deep

HTTP/1.1 200 OK
<properties xmlns="http://www.cisco.com/nfvis/vm_lifecycle"
xmlns:y="http://tail-f.com/ns/rest"  xmlns:vmlc="http://www.cisco.com/nfvis/vm_lifecycle">
  <property>
    <name>interface_hot_add</name>
    <value>true</value>
  </property>
  <property>
    <name>interface_hot_delete</name>
    <value>false</value>
  </property>
</properties>
```

Image Properties

The image_properties.xml file in the tar.gz package contains the property configuration data for a particular image. Figure 3 depicts an example image_properties.xml file.

Some of the properties are mandatory and must be specified to register an image. If any of the mandatory properties are omitted, the image registration fails.

Optional properties can be specified on an image-by-image basis and are not required.

The following table lists all the image properties that are supported in Cisco Enterprise NFVIS.

Property Name	Description	Property Tag	Possible Values	Mandatory/ Optional
VNF Type	VM functionality provided. Router and firewall are predefined types.	<vnf_type>	Router, firewall, Windows, Linux, and custom_type	Mandatory
Name	Name associated with the VM packaging. This name is referenced for VM deployment.	<name>	Any	Mandatory
Version	Version of the package	<version>	Any	Mandatory
Boot-up time	Boot-up time (in seconds) of the VNF before it can be reachable via ping.	<bootup_time>	Any in seconds, (-1) to not monitor boot-up	Optional, default -1
Root Disk Image Bus	Root image disk bus	<root_file_disk_bus>	virtio, scsi, and ide	Mandatory
Boot Mode	Specifies the mode in which the VNF will boot. Used for Secure Boot feature	<boot_mode>	efi-secure-boot, bios	Optional, default bios
Shim Signature	If using efi-secure-boot boot mode, a shim signature must be provided	<shim_signature>	microsoft, N/A	Required if Boot Mode is specified
Disk-1 bus type	Additional disk1 image disk bus	<disk_1_file_disk_bus>	virtio, scsi, and ide	Optional
Disk-2 bus type	Disk2 image disk bus	<disk_2_file_disk_bus>	virtio, scsi, and ide	Optional
Disk-10 bus type	Disk10 image disk bus	<disk_10_file_disk_bus>	virtio, scsi, and ide	Optional

Property Name	Description	Property Tag	Possible Values	Mandatory/ Optional
Root Disk Image format	Root image disk format	<root_image_disk_format>	qcow2 and raw	Mandatory
Disk-1 Image format	Additional disk 1 image format	<disk_1_image_format>	qcow2 and raw	Optional
Disk-2 Image format	Disk 2 image format	<disk_2_image_format>	qcow2 and raw	Optional
Disk-10 Image format	Disk 10 image format	<disk_10_image_format>	qcow2 and raw	Optional
Serial Console	Serial console supported	<console_type_serial>	true, false	Optional
Minimum vCPU	Minimum vCPUs required for a VM operation	<vcpu_min>		Mandatory
Maximum vCPU	Maximum vCPUs supported by a VM	<vcpu_max>		Mandatory
Minimum memory	Minimum memory in MB required for VM operation	<memory_mb_min>		Mandatory
Maximum memory	Maximum memory in MB supported by a VM	<memory_mb_max>		Mandatory
Minimum root disk size	Minimum disk size in GB required for VM operation	<root_disk_gb_min>		Optional
Maximum root disk size	Maximum disk size in GB supported by a VM	<root_disk_gb_max>		Optional
Maximum vNICs	Maximum number of vNICs supported by a VM	<vnic_max>		Mandatory

Property Name	Description	Property Tag	Possible Values	Mandatory/ Optional
SRIOV support	SRIOV supported by VM interfaces. This should have a list of supported NIC device drivers.	<sriov_supported>	true, false	Optional
SRIOV driver list	List of drivers to enable SRIOV support	< sriov_driver_list>		Optional
PCI passthru support	PCI passthru support by VM interfaces	<pcie_supported>	true, false	Optional
PCIE driver list	List of VNICS to enable PCI passthru support	< pcie _driver_list>		Optional
bootstrap_cloud_init_drive_type	Mounts day0 config file as disk (default is CD-ROM)	<bootstrap_cloud_init_drive_type>	disk, cdrom	Optional
bootstrap_cloud_init_bus_type	Default is IDE	<bootstrap_cloud_init_bus_type>	virtio, ide	Optional
BOOTSTRAP	Bootstrap files for the VNF. Two parameters are required in the format of dst:src; dst filename including path has to match exactly to what the VM expects; up to 20 bootstrap files are accepted. For example: --bootstrap ovf-env.xml for ISRv and --bootstrap day0-config for ASAv	< bootstrap_file>	File name of the bootstrap file	Optional

Property Name	Description	Property Tag	Possible Values	Mandatory/ Optional
Custom properties	List of properties can be defined within the custom_property tree. (Example: For ISRv, the technology packages are listed in this block.) If the Cisco Enterprise NFV portal is used to deploy the VM, the portal prompts you for inputs for custom properties fields and can pass the values to the bootstrap configuration.	<custom_property>		Optional
Profiles for VM deployment	List of VM deployment profiles. Minimum one profile is required	<profiles>		Optional
Default profile	The default profile is used when no profile is specified during deployment.	<default_profile>		Optional
Monitoring Support	A VM supports monitoring to detect failures.	<monitoring_supported>	true, false	Mandatory
Monitoring Method	A method to monitor a VM. Currently, only ICMP ping is supported.	<monitoring_methods>	ICMPPing	Mandatory if monitoring is true

Property Name	Description	Property Tag	Possible Values	Mandatory/ Optional
Low latency	If a VM's low latency (for example, router and firewall) gets dedicated resource (CPU) allocation. Otherwise, shared resources are used.	<low_latency>	true, false	Mandatory
Privileged-VM	Allows special features like promiscuous mode and snooping . By default, it is false.	<privileged_vm>	true, false	Optional
Disable Spoof Check	Used to disable spoof check for Privledged VMs	<disable_spoof_check>	true, false	Optional
Virtual interface model		<virtual_interface_model>		Optional
Interface Hot Add	If true, an active VNF’s virtual interface can be added/updated without shutting down the VNF.	<interface_hot_add>	true, false	Optional, default true
Interface Hot Delete	If true, an active VNF’s virtual interface can be deleted without shutting down the VNF.	<interface_hot_delete>	true, false	Optional, default false
Thick disk provisioning	During deployment, VM will be a fully allocated root disk with size specified by flavor.	<thick_disk_provisioning>	true, false	Optional, default false

Property Name	Description	Property Tag	Possible Values	Mandatory/ Optional
Eager Zero	Used in conjunction with Thick disk provisioning. During deployment, root disk is zeroed out to improve I/O operations	<eager_zero>	true, false	Optional, only valid if Thick disk provisioning is enabled. Default false
Profile for VM deployment	A profile defines the resources required for VM deployment. This profile is referenced during VM deployment.	<profile>		Optional
Name	Profile name	<name>	Any	Mandatory
Description	Description of the profile	<description>	Any	Mandatory
vCPU	vCPU number in a profile	<vcpus>		Mandatory
Memory	Memory - MB in profile	<memory_mb>		Mandatory
Root Disk Size	Disk size - MB in profile .	<root_disk_mb>		Mandatory
VNIC Offload	List of properties that can be set for vnic offload	<vnic_offload>		Optional
Generic Segmentation Offload	Turn generic segmentation offload on or off	<generic_segmentation_offload> (parent: <vnic_offload>)	on, off	Optional
Generic Receive Offload	Turn generic receive offload on or off	<generic_receive_offload> (parent: <vnic_offload>)	on, off	Optional
RX Checksumming	Turn RX checksumming on or off	<rx_checksumming> (parent: <vnic_offload>)	on, off	Optional

Property Name	Description	Property Tag	Possible Values	Mandatory/ Optional
TX Checksumming	Turn TX checksumming on or off	<tx_checksumming> (parent: <vnic_offload>)	on, off	Optional
TCP Segmentation Offload	Turn TCP segmentation offload on or off	<tcp_segmentation_offload> (parent: <vnic_offload>)	on, off	Optional

Example: Contents of an image_properties.xml File

```
<?xml version="1.0" encoding="UTF-8"?>
<image_properties>
    <vnf_type>ROUTER</vnf_type>
    <name>ISRV</name>
    <version>16.06.05</version>
    <bootup_time>600</bootup_time>
    <root_file_disk_bus>virtio</root_file_disk_bus>
    <root_image_disk_format>qcow2</root_image_disk_format>
    <vcpu_min>1</vcpu_min>
    <vcpu_max>8</vcpu_max>
    <memory_mb_min>4096</memory_mb_min>
    <memory_mb_max>8192</memory_mb_max>
    <vnic_max>8</vnic_max>
    <vnic_names>vnics:1:GigabitEthernet2</vnic_names>
    <vnic_names>vnics:2:GigabitEthernet3</vnic_names>
    <vnic_names>vnics:3:GigabitEthernet4</vnic_names>
    <vnic_names>vnics:4:GigabitEthernet5</vnic_names>
    <vnic_names>vnics:5:GigabitEthernet6</vnic_names>
    <vnic_names>vnics:6:GigabitEthernet7</vnic_names>
    <vnic_names>vnics:7:GigabitEthernet8</vnic_names>
    <root_disk_gb_min>8</root_disk_gb_min>
    <root_disk_gb_max>8</root_disk_gb_max>
    <console_type_serial>true</console_type_serial>
    <sriov_supported>true</sriov_supported>
    <sriov_driver_list>igb</sriov_driver_list>
    <sriov_driver_list>igbvf</sriov_driver_list>
    <sriov_driver_list>i40evf</sriov_driver_list>
    <pcie_supported>true</pcie_supported>
    <pcie_driver_list>igb</pcie_driver_list>
    <pcie_driver_list>igbvf</pcie_driver_list>
    <pcie_driver_list>i40evf</pcie_driver_list>
    <monitoring_supported>true</monitoring_supported>
    <monitoring_methods>ICMPPing</monitoring_methods>
    <low_latency>true</low_latency>
    <privileged_vm>true</privileged_vm>
    <cdrom>true</cdrom>
    <bootstrap_file_1>ovf-env.xml</bootstrap_file_1>
    <bootstrap_file_2>iosxe_config.txt</bootstrap_file_2>
    <custom_property>
        <tech_package>ax</tech_package>
        <tech_package>security</tech_package>
        <tech_package>ipbase</tech_package>
        <tech_package>appx</tech_package>
    </custom_property>
    <custom_property>
        <ngio>enable</ngio>
    </custom_property>
    <custom_property>
```

```
        <SSH_USERNAME> </SSH_USERNAME>
    </custom_property>
    <custom_property>
        <SSH_PASSWORD> </SSH_PASSWORD>
    </custom_property>
    <profiles>
     <profile>
     <name>ISRv-mini</name>
            <description>ISRv-mini</description>
     <vcpus>1</vcpus>
            <memory_mb>4096</memory_mb>
            <root_disk_mb>8192</root_disk_mb>
 </profile>
     <profile>
     <name>ISRv-small</name>
            <description>ISRv-small</description>
     <vcpus>2</vcpus>
            <memory_mb>4096</memory_mb>
            <root_disk_mb>8192</root_disk_mb>
 </profile>
     <profile>
     <name>ISRv-medium</name>
            <description>ISRv-medium</description>
     <vcpus>4</vcpus>
            <memory_mb>4096</memory_mb>
            <root_disk_mb>8192</root_disk_mb>
 </profile>
    </profiles>
    <default_profile>ISRv-small</default_profile>
</image_properties>
```

VM Profiles or Flavors

Flavors or profiles define VMs in terms of number of parameters for how to run the VM. Some of the parameters that you can define in a VM profile or flavor are: number of vCPUs, RAM , disk size and so on.

Flavors are created as part of image registration if you use the tar.gz image packages for registering a VM. However, for other image packages such as .qcow2, iso, and raw, you must define custom flavors based on your requirements.



Note Unless specified otherwise in the deployment payload, the value assigned to the custom image property *default_profile* is used at the time of deploying the VM. Only applicable to *tar.gz* image packages.

Example: Create VM Profile Using Rest API

The following example shows how to create a VM profile or flavor using REST API.

```
curl -k -v -u admin:admin -H Accept:application/vnd.yang.data+xml -H
Content-Type:application/vnd.yang.data+xml -X POST
https://209.165.201.1/api/config/vm_lifecycle/flavors -d '<flavor>
    <name>windows</name>
    <ephemeral_disk_mb>0</ephemeral_disk_mb>
    <memory_mb>4096</memory_mb>
    <root_disk_mb>12288</root_disk_mb>
    <swap_disk_mb>0</swap_disk_mb>
```

```
    <vcpus>2</vcpus>
</flavor>'
```

For more information on REST APIs related to creating VM flavors, see API Reference for Cisco Enterprise NFVIS.

Configure Internal Management Network

Monitored VNFs must have one internal management network specified to the NIC ID “0” in the deployment payload. If the IP address is configured, it needs to belong to the same network as the internal subnet. By default, the internal management network has the range *10.20.0.0*. If the IP address is not specified in the VM deployment payload, the system assigns an IP address for internal management network using IP address pool.

Here an example of configuring internal management network.

```
<interface>
   <nicid>0</nicid>
   <network>int-mgmt-net</network>
   <ip_address>10.20.0.21</ip_address>
</interface>
```

VM Deployment and Management

VM Deployment

A VM can be deployed through API or CLI. Ensure that you have registered a VM image before attempting to deploy it. For more details, see Image Registration, on page 23.

Note The VM name must meet the following requirements:

- Must contain an uppercase character and a lowercase character.
- Must contain a digit.
- Must contain one of the following special characters: dot (.), underscore (_) and hyphen (-).
- Must not have more than 256 characters.

The deployment API allows you to provide values to the parameters that are passed to the system during deployment. Depending on the VM you are deploying, some parameters are mandatory and others optional.

Example: Deploy VMs Using REST API

Use the following API to deploy a VM.

Method	URL
POST	/api/config/vm_lifecycle/tenants/tenant/admin/deployment

Here's a sample payload of deploying a VM.

```
curl -k -v -u admin:admin -H Accept:application/vnd.yang.data+xml -H
Content-Type:application/vnd.yang.data+xml -X POST
https://209.165.201.1/api/config/vm_lifecycle/tenants /tenant/admin/deployments --data '
<deployment>
  <name>ISRdep</name>
  <vm_group>
    <name>ISRvmgrp</name>
    <image>ISR_IMAGE</image>
    <bootup_time>500</bootup_time>
    <recovery_wait_time>0</recovery_wait_time>
    <interfaces>
      <interface>
        <nicid>0</nicid>
        <network>int-mgmt-net</network>
        <ip_address>10.20.0.21</ip_address>
        <port_forwarding>
          <port>
            <type>ssh</type>
            <protocol>tcp</protocol>
            <vnf_port>22</vnf_port>
            <external_port_range>
              <start>20022</start>
              <end>20022</end>
            </external_port_range>
          </port>
        </port_forwarding>
      </interface>
    </interfaces>
    <kpi_data>
      <kpi>
        <event_name>VM_ALIVE</event_name>
        <metric_value>1</metric_value>
        <metric_cond>GT</metric_cond>
        <metric_type>UINT32</metric_type>
        <metric_collector>
          <type>ICMPPing</type>
          <nicid>0</nicid>
          <poll_frequency>3</poll_frequency>
          <polling_unit>seconds</polling_unit>
          <continuous_alarm>false</continuous_alarm>
        </metric_collector>
      </kpi>
    </kpi_data>
    <rules>
      <admin_rules>
        <rule>
          <event_name>VM_ALIVE</event_name>
          <action>ALWAYS log</action>
          <action>TRUE servicebooted.sh</action>
          <action>FALSE recover autohealing</action>
        </rule>
      </admin_rules>
    </rules>
    <config_data>
      <configuration>
        <dst>bootstrap_config</dst>
        <variable>
          <name>TECH_PACKAGE</name>
          <val>ax</val>
        </variable>
      </configuration>
    </config_data>
```

```
  </vm_group>
</deployment>'
```

Verify VM Deployment

The following example shows how to get the operational data for a VM deployment using the command

```
show vm_lifecycle opdata tenants tenant admin deployments
<deployment_name>/<deployment_id>/<vmgroup_name>
```

```
nfvis# show vm_lifecycle opdata tenants tenant admin deployments ROUTER
deployments ROUTER
 deployment_id SystemAdminTenantIdROUTER
 vm_group ROUTER
  bootup_time 600
  vm_instance d1c462e9-2706-4868-befd-d8f7806b9444
   name     ROUTER
   host_id  NFVIS
   hostname nfvis
   interfaces interface 0
    model       virtio
    type        virtual
    port_id     vnic0
    network     int-mgmt-net
    subnet      N/A
    ip_address  10.20.0.2
    mac_address 52:54:00:fe:34:53
    netmask     255.255.255.0
    gateway     10.20.0.1
   interfaces interface 1
    type        virtual
    port_id     vnic1
    network     GE0-1-SRIOV-1
    subnet      N/A
    mac_address 52:54:00:e4:13:67
 state_machine state SERVICE_ACTIVE_STATE
VM
NAME    STATE
------------------------
ROUTER  VM_ALIVE_STATE
```

VM Deployment Parameters

VNFs can be deployed using multiple mandatory and option parameters. The following table lists some of the parameters.

Parameter	Notes	Example
`image`	Mandatory. The image needs to be registered and active when it is being referred in deployment.	`<image>ISR_IMAGE</image>`

`bootup_time`	This parameter is no longer mandatory from 3.12 and later, provided that it is specified in image properties. **Accepted Values:** • For unmonitored VMs: -1 • For monitored VMs: Number of seconds	`<bootup_time>500</bootup_time>`
`vim_vm_name`	Optional. If a custom VM name is provided at the time of deployment, it may be used for all commands that accept VM name as an input.	
`kpi_data`	Mandatory for monitored VMs.	
Rules	Mandatory for monitored VMs.	
`config_data`	Mandatory if the day-0 configuration has variables that have tokens assigned to them.	
`Encrypted config variable`	Optional. Only one value for a variable is allowed to be encrypted.	<variable> <name>TEST_VARIABLE</name> <encrypted_val>test_value</encrypted_val> </variable>

`placement`	Optional.
	The ***placement*** tag under vm group points to the location where the VNF would be deployed. This parameter supports deploying a VNF in a local data store (default-if not specified), external data store (datastore2), or NFS.
		`<placement>`
		`<type>zone_host</type>`
		`<host>nfs:nfs_storage</host>`
		`</placement>`

`volumes`	Optional. Up to 2 volumes could be added to a deployment. Location of the volumes can be local or NFS (needs NFS mount name to be specified in case of NFS)	

`port_forwarding`	Optional. If port forwarding is included, all elements under it are mandatory.	`<port_forwarding>`
		`<port>`
		`<type>ssh</type>`
		`<protocol>tcp</protocol>`
		`<vnf_port>22</vnf_port>`
		`<external_port_range>`
		`<start>20022</start>`
		`<end>20022</end>`
		`</external_port_range>`
		`</port>`
		`</port_forwarding>`

`Ngio interface`	Optional. Used in config_data. To enable NIM support on a Cisco ISRv running on Cisco ENCS, you must use the variables in the ISRv deployment payload.	`<variable> <name>ngio</name>` `<val>enable</val> </variable>`
Interface model	Optional. If the model is not specified for an interface, the default model is used. For Windows, the default model is rtl8139.	<interface> <nicid>3</nicid> <network>wan-net</network> <model>virtio</model> </interface>
VNC password	Optional. If the VNC password is not specified, there is no default password.	

VM Bootstrap Configuration Options with a VM Deployment

You can include the bootstrap configuration (day zero configuration) of a VM in the VM deployment payload in the following ways:

- Bundle bootstrap configuration files into the VM package: In this method, the bootstrap configuration variables can be assigned tokens. Token names must be in bold text. For each variable with a token assigned, key-value pairs must be provided during deployment in the deployment payload.
- Bootstrap configuration as part of the deployment payload: The entire bootstrap configuration is copied to the payload without tokens.
- Bootstrap configuration file in the NFVIS server: In this method, the configuration file is copied or downloaded to the NFVIS server, and referenced from the deployment payload with the filename, which includes the full path.

For examples on how to use bootstrap configuration options in the deployment payload, see the API Reference for Cisco Enterprise Network Function Virtualization Infrastructure Software.

VM Monitoring

After VMs are deployed, they monitored periodically based on the metrics defined in the KPI section of deployment data model. Monitoring can be enabled or disabled by modifying the <actionType> tag. See the VM Actions section for details on the allowed values for the action Type tag and what they mean.

The following example shows how to disable monitoring for a VM.

```
curl -k -v -u "admin:password" -H
"Accept:application/vnd.yang.data+xml" -H
"Content-Type:application/vnd.yang.data+xml" -X POST
https://<NFVIS_IP>/api/operations/vmAction --data '<vmAction>
<actionType>DISABLE_MONITOR</actionType><vmName><vm-instance name></vmName></vmAction>'
```

- If the bootup_time is set at -1, it signifies that VM monitoring is disabled.
- You are not required to set a boot up time during image registration. However, you must set it during VM deployment.
- If a qcow2 image is used during registration, the bootup_time defaults to -1.

VNF Deployment Placement

Cisco NFVIS allows you to specify where a VNF should be deployed using the placement tag for parameter deployment. See VM Deployment Parameters for more information on supported placement parameters and their accepted values.



Note If you are placing the VNF deployment on **nfs**, ensure that you have configured this storage option to be mounted on NFVIS using appropriate CLIs before deploying the VNF.

Example: VM Deployment Using Placement

```
curl -k -v -u admin:admin -H Accept:application/vnd.yang.data+xml -H
Content-Type:application/vnd.yang.data+xml -X POST
https://209.165.201.1/api/config/vm_lifecycle/tenants
/tenant/admin/deployments --data
'<deployment>
<name>WINIsodep</name>
<vm_group>
  <name>WINIsovmgrp</name>
  <image>WinServer2012R2.iso</image>
  <flavor>windows</flavor>
  <bootup_time>-1</bootup_time>
  <recovery_wait_time>0</recovery_wait_time>
  <kpi_data>
   <enabled>true</enabled>
  </kpi_data>
   <scaling>
   <min_active>1</min_active>
   <max_active>1</max_active>
   <elastic>true</elastic>
   </scaling>
   <placement>
   <type>zone_host</type>
```

```
   <enforcement>strict</enforcement>
   <host>datastore1</host>
   </placement>
   <recovery_policy>
   <recovery_type>AUTO</recovery_type>
   <action_on_recovery>REBOOT_ONLY</action_on_recovery>
   </recovery_policy>
</vm_group>
</deployment>'
```

VNF Volumes

A VNF can be created and deployed with multiple volumes. Currently, NFVIS supports a maximum of two volumes per VNF.

Restrictions for VNF Volumes

- Volumes cannot be updated for a VNF after a VNF has already been deployed.
- NFVIS currently supports only two volumes per VNF.

Example: Payload for Creating Volumes

```
…
<volumes>
  <volume>
    <name>Volume1</name>
    <volid>1</volid>
    <bus>ide</bus>
    <size>1</size>
    <sizeunit>GiB</sizeunit>
    <format>qcow2</format>
    <device_type>disk</device_type>
    <storage_location>local</storage_location>
  </volume>
</volumes>
…
```

Volume Storage Locations

The following are the accepted values for **storage_location** tags:

- <**storage_location**>*local*</**storage_location**>
- <**storage_location**>*nfs:NFS_MOUNT_NAME*</**storage_location**>

Note The storage locations *datastore1*, *datastore2* and *datastore3* are not supported for NFVIS 3.12.3 and later releases.

Volume Deployment Parameters

Cisco NFVIS supports the following fields for volume deployment.

Property	Allowed Values	Description

name	string, { length 1..255 }	Name of the volume
volid	uint16	Volumes will be presented to the VM sorted by volume ID.
bus	ide, scsi, virtio	The bus type
size	unit 16	Size of the Volume
sizeunit	MiB,GiB,TiB,PiB,EiB	Size units. MiB/GiB/TiB/PiB/EiB
format	qcow2, raw	Format of the disk to be created.
device_type	disk, CD ROM	Type of the device being attached to the VM.
storage_location	local, nfs:NFS_MOUNT_NAME	Storage location name

Port Forwarding

By default, the wan bridge interface (**wan-br**) is used to redirect incoming traffic from WAN to access the internal management network (**int-mgmt-net**) of the VM.

The bridge interface that is used to redirect traffic coming from the WAN side can be modified using the **source_bridge** tag in the deployment payload as shown below.

```
<port_forwarding>
  <port>
  <type>ssh</type>
  <protocol>tcp</protocol>
  <vnf_port>22</vnf_port>
  <source_bridge>MGMT</source_bridge>
  <external_port_range>
    <start>20122</start>
    <end>20122</end>
  </external_port_range>
  </port>
</port_forwarding>
```

With the payload above, the traffic coming from the WAN side is redirected through the management interface **(MGMT)** instead of the default WAN bridge **(wan-br)** interface.

NGIO

Next Generation Input/Output (NGIO) is supported on Cisco ENCS platforms from NFVIS release 3.11 and later. Using the NGIO flag, the VM image informs its capability. Between ISRv and NFVIS image, NGIO is used to decide the NIM enablement capability available for the VM. This is only for Cisco ISRv with Cisco IOS XE image or Cisco IOS XE with SD-WAN image.

To enable NGIO on a VNF. a config data variable is added to the deployment payload as shown below.

```
<config_data>
  <configuration>
    <dst>bootstrap_config</dst>
    <variable>
      <name>TECH_PACKAGE</name>
      <val>ax</val>
    </variable>
    <variable>
      <name>ngio</name>
      <val>enable</val>
```

```
    </variable>
  </configuration>
```

The following is an example of the interfaces added to a VNF, when the NGIO variable is detected and set to **enable**.

```
<interface type='bridge'>
  <source bridge='csxbr'/>
  <model type='e1000'/>
  <address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x0' multifunction='on'
 />
</interface>
<interface type='hostdev' managed='yes'>
  <driver name='vfio'/>
  <source>
    <address type='pci' domain='0x0000' bus='0x0e' slot='0x10' function='0x1'/>
  </source>
  <address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x1'
multifunction='on'/>
</interface>
<interface type='hostdev' managed='yes'>
  <driver name='vfio'/>
  <source>
    <address type='pci' domain='0x0000' bus='0x0e' slot='0x10' function='0x3'/>
  </source>
  <address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x2' multifunction='on'
 />
</interface>
```

Requirements for NGIO Validation

When NGO deployment request is initiated, it is validated against the specified image and the platform itself. To be accepted, the platform must be supported and the registered image must have the following properties.

```
<image_properties>
    ...
    <custom_property>
        <ngio>enable</ngio>
    </custom_property>
    ...
</image_properties>
```

VM States

The following table describes various VM states.

VM States	Description
VM_UNDEF_STATE	The initial state of a VM or VNF before deployment of this VM.
VM_DEPLOYING_STATE	The VM or VNF is being deployed on to the NFVIS.
VM_MONITOR_UNSET_STATE	The VM or VNF is deployed in NFVIS but the monitoring rules are not applied.
VM_MONITOR_DISABLED_STATE	Due to a VM action request or recovery workflow, the monitoring or KPI rules applied to the VM were not enabled.
VM_STOPPING_STATE	VM or VNF is being stopped.
VM_SHUTOFF_STATE	VM or VNF is in stopped or shutoff state.

VM States	Description
VM_STARTING_STATE	VM or VNF is being started.
VM_REBOOTING_STAT	VM or VNF is being rebooted.
VM_INERT_STATE	VM or VNF is deployed but not alive. The KPI monitor is applied and waiting for the VM to become alive.
VM_ALIVE_STATE	VM or VNF is deployed and successfully booted up or alive as shown in the KPI metric.
VM_UNDEPLOYING_STATE	The deployment of a VM or VNF is being terminated.
VM_ERROR_STATE	The VM or VNF is in an error state because the deployment or some other operation has failed.

VNF Deployment Update

After you have deployed a VNF, you can update it in terms of its flavor, CPU topology, or interfaces.

Update VNF Flavor

You can update a VNF deployment to have a different flavor from the one you deployed it with. The flavor can also be custom-defined.



Note Before updating a VNF with another flavor, we recommend that you check whether CPUs are avaialble for the required update.

Updating a VNF flavor only supports CPU and Memory changes and does not support disk size change.

The following is a typical workflow for updating a VNF flavor.

Get Available Flavors

Use the following command to get a list of available flavors.

```
curl -k -v -u admin:admin -X GET
https://209.165.201.1/api/operational/vm_lifecycle/flavors?deep
```

Check System CPU Usage

Use the following command to check the CPU usage of the system.

```
curl --tlsv1.2 -k -i -u admin:Esc123# -H
Accept:application/vnd.yang.data+json -H content-
type:application/vnd.yang.data+json -X GET
https://<nfvis_ip>/api/operational/resources/cpu-info/allocation
```

Update VNF Flavor

Use the following command to update the flavor of a VNF.

```
curl -k -v -u admin:admin -H Accept:application/vnd.yang.data+xml -H
Content-Type:application/vnd.yang.data+xml
-X PUT
```

```
https://<nfvis_ip>/api/config/vm_lifecycle/tenants/tenant/admin/deployments/deployment/<deploymentID>
/vm_group/<VMGroupName>/flavor
--data
'<flavor><FlavorName></flavor>
```

Example: Changing the Flavor of a VNF from Flavor from ASAv5 to ASAv10

```
curl -k -v -u admin:Esc123# -H
Accept:application/vnd.yang.data+xml -H Content-Type:application/vnd.yang.data+xml -X PUT
https://172.29.91.32/api/config/vm_lifecycle/tenants/tenant/admin/deployments/deployment/ASAdep/vm_group/ASAvmgrp/
flavor --data '<flavor>ASAv10</flavor>
```

Update CPU Topology

Updating the CPU topology of a VNF involves updating a VM to a custom-defined topology that you would have created at the time of creating a VM flavor. For more details, see VM Profiles or Flavors, on page 35.

The process to update the CPU topology is similar to updating a VNF flavor—by replacing the name of the CPU topology. The following command is used to update a CPU topology.

```
curl -k -v -u admin:admin -H
Accept:application/vnd.yang.data+xml -H
Content-Type:application/vnd.yang.data+xml
-X PUT
https://<nfvis_ip>/api/config/vm_lifecycle/tenants/tenant/admin/deployments/deployment/<deploymentID>/vm_group/
<VMGroupName>/flavor
--data
'<flavor><FlavorName_withCPUTopology></flavor>'
```

Example: Updating ISRv to Include a CPU Topology

```
curl -k -v -u admin:Esc123# -H
Accept:application/vnd.yang.data+xml -H Content-
Type:application/vnd.yang.data+xml -X PUT
https://172.29.91.32/api/config/vm_lifecycle/tenants/tenant/admin/deployments/deployment/ROUTER/vm_group/ROUTER/flavor

--data '<flavor>Isrv_CPUToplogy</flavor>'
```

Verify Changed Configuration

Use the `support virsh dumpxml <uid>` command to verify whether the changed configuration was applied to the VNF.

The following is an example of the output you would see.

```
<vcpu placement='static'>3</vcpu>
  <cputune>
    <vcpupin vcpu='0' cpuset='11'/>
    <vcpupin vcpu='1' cpuset='10'/>
    <vcpupin vcpu='2' cpuset='9'/>
    <emulatorpin cpuset='21-23'/>
  </cputune>
  . . .
  <cpu mode='host-passthrough' check='none'>
    <topology sockets='1' cores='3' threads='1'/>
  </cpu>
```

About Updating VNF Interfaces

NFVIS supports updating VNF interfaces at the time of updating a VM deployment. VM interface updates include adding an interface, deleting an interface, and moving a VNIC of a VM from one interface to another.

Hot and Cold Updates

All the options related to updating interfaces mentioned above can be done in two ways: hot or cold.

Hot Update: Hot update refers to an update operation that runs when a VM is in ACTIVE state. In such cases, the VM does not reboot during the update.

Cold Update: Cold update refers to an update operation that runs after a VM is put in a VM_SHUTOFF_STATE. In such cases, the VM reboots during the update.



Note Whether you would need to do a hot or cold update depends on the custom image properties set during image registration. Refer to the table below for various custom image properties related to hot and cold updates.

Custom Image Property	Value	Interface Update Description	Default Value
interface_hot_add	true	Hot add an interface to a VM	true
interface_hot_add	false	Cold add an interface to VM	
interface_hot_delete	true	Hot delete a VM interface	false
interface_hot_delete	false	Cold delete a VM interface	

Starting from ISRv 17.1, interface_hot_add and interface_hot_delete are set to true by default.

Prerequisites for Updating VNF Interfaces

- The VNF should support hot add or hot delete operations for the interface.
- The custom image properties for interface update should be set during image registration to allow values other than the default.
- The VM to be updated needs to be in one of the following states: VM_ALIVE_STATE, VM_ERROR_STATE or VM_SHUTOFF_STATE.

Supported Interface Update Operations by Interface Types

The following table shows which hot interface update operations are supported for various interface types.

Interface Type	Hot Add	Hot Delete	Hot Update for Moving VNIC
VIRTIO	Yes	Yes	Yes
SRIOV	Yes	Yes	Yes
DPDK	Yes	Yes	Yes

Note NFVIS also supports moving VNICs from one interface to another. For example, you can move a VNIC from a VIRTIO interface to SRIOV, or from SRIOV to DPDK, and so on.

If the VNIC is updated to a different interface type like SRIOV or DPDK, the configuration of the vnic will not be preserved.

Syslog is not generated in ISRv when an interface is updated from a DPDK enabled network to another DPDK enabled network.

Update Interfaces

This topic walks you through how to perform various tasks related to updating interfaces such as adding an interface, deleting an interface, and so on.

Add Interfaces

Single Interface: The following example shows how to add a single interface to a VM deployment.

```
curl -k -v -u admin:Esc123# -H
Accept:application/vnd.yang.data+xml -H Content-Type:application/vnd.yang.data+xml -X PUT
https://<NfvisIpAddress>/api/config/vm_lifecycle/tenants/tenant/admin/deployments/deployment/ASAdep/vm_group/ASAvmgrp/interfaces
 --data '
<interfaces>
<interface>
        <nicid>0</nicid>
        <network>int-mgmt-net</network>
    </interface>
    <interface>
        <nicid>newNIC</nicid>
        <network>networkName</network>
    </interface>
    </interfaces>'
```

Multiple Interfaces: The following example shows how to add multiple interfaces (in this case, two) to a VM deployment.

```
curl -k -v -u admin:Esc123# -H
Accept:application/vnd.yang.data+xml -H
Content-Type:application/vnd.yang.data+xml -X PUT
https://172.29.91.32/api/config/vm_lifecycle/tenants/tenant/admin/deployments/deployment/ASAdep/vm_group/ASAvmgrp/interfaces
 --data '
<interfaces>
<interface>
          <nicid>0</nicid>
          <network>int-mgmt-net</network>
      </interface>
      <interface>
          <nicid>1</nicid>
          <network>wan-net</network>
      </interface>
      <interface>
      <nicid>2</nicid>
      <network>lan-net</network>
      </interface>
      </interfaces>'

HTTP/1.1 204 No Content
```

Delete Interfaces

The following REST API is used to delete an interface from a VM deployment.

```
curl -k -v -u admin:<password> -H
Accept:application/vnd.yang.data+xml -H
Content-Type:application/vnd.yang.data+xml -X PUT
https://<NfvisIpAddress>/api/config/vm_lifecycle/tenants/tenant/admin/deployments/deployment/ASAdep/vm_group/ASAvmgrp/interfaces
 --data '
<interfaces>
<interface>
        <nicid>0</nicid>
        <network>int-mgmt-net</network>
    </interface>
**** Note: Remove the required nicID along with content between <interface> and </interface>
 ******
    </interfaces>'
```

Example: The following example shows how to delete an interface called NIC ID2, which was added in the example of adding multiple interfaces in the Add Interfaces section above.

```
curl -k -v -u admin:Esc123# -H
Accept:application/vnd.yang.data+xml -H
Content-Type:application/vnd.yang.data+xml -X PUT
https://172.29.91.32/api/config/vm_lifecycle/tenants/tenant/admin/deployments/deployment/ASAdep/vm_group/ASAvmgrp/interfaces
 --data '
    <interfaces>
      <interface>
              <nicid>0</nicid>
              <network>int-mgmt-net</network>
          </interface>
          <interface>
              <nicid>1</nicid>
              <network>wan-net</network>
       </interface>
     </interfaces>'

HTTP/1.1 204 No Content
```

Notice that in the example above, NIC ID 2 has been excluded from the REST API for it to be deleted from the deployment.

Move VNICs from One Network to Another

```
curl -k -v -u admin:<password> -H
Accept:application/vnd.yang.data+xml -H
Content-Type:application/vnd.yang.data+xml -X PUT
https://<NfvisIpAddress>/api/config/vm_lifecycle/tenants/tenant/admin/deployments/deployment/ASAdep/vm_group/ASAvmgrp/interfaces
 --data '
<interfaces>
    <interface>
        <nicid>0</nicid>
        <network>int-mgmt-net</network>
    </interface>
    <interface>
        <nicid>selectedNicId</nicid>
         <network>NewNetworkSelected</network>
    </interface>
</interfaces>'
```

Example: The following example shows how to move nicid 1 from wan-net to wan2-net .

```
curl -k -v -u admin:Esc123# -H
Accept:application/vnd.yang.data+xml -H Content-Type:application/vnd.yang.data+xml -X PUT
https://172.29.91.32/api/config/vm_lifecycle/tenants/tenant/admin/deployments/deployment/ASAdep/vm_group/ASAvmgrp/interfaces
```

```
 --data '
<interfaces>
         <interface>
                 <nicid>0</nicid>
                 <network>int-mgmt-net</network>
             </interface>
             <interface>
                 <nicid>1</nicid>
                 <network>wan2-net</network>
          </interface>
</interfaces>'

HTTP/1.1 204 No Content
```

Access VNFs

In Cisco NFVIS, you can access VNFs in two ways after they have been deployed: through the VNC Console and through the Serial Console.

Access VNFs Using VNC Console

A VNC console allows you to access VNFs through a VNC client. This method enables you to manipulate the VNF through the NFVIS portal.

If you are using the NFVIS portal, follow these steps to access the VNF using the VNC console.

1. On the NFVIS portal, click the **VM Life Cycle** tab.
2. Next, click the black console icon next to the VNF you want to access (see screenshot below).



Secure Access to the VNC Console

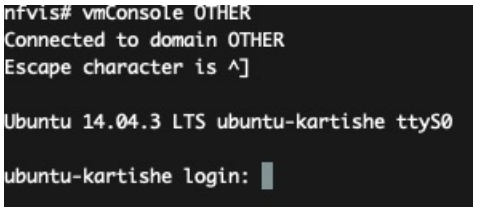
For added security, access to the VNC console can be restricted through a passphrase. To enable VNC passphrase, include the contents as shown in the example below.

```
<vm_lifecycle>
 <tenants>
  <name>admin</name>
   <deployments>
    <deployment>
     …
     <vnc>
      <password>PASSWORD</password>
     </vnc>
    </deployment>
   </deployments>
  </tenant>
 </tenants>
</vm_lifecycle>
```

Access VMs Using Serial Console

The serial console allows you to access the VM using the serial interface provided by the VM itself. This method is applicable only if the VM supports serial interfaces in both, its image and its image properties.

To access the VNF through the serial console, use the command `vmConsole` followed by the name of the VNF as shown in the CLI example below.



Access a VM Using Port Forwarding

To access a VM using port forwarding:

1. Deploy a VNF using the deployment payload with **port_forwarding** configuration:

```
<port_forwarding>
  <port>
    <type>ssh</type>
    <protocol>tcp</protocol>
    <vnf_port>22</vnf_port>
    <external_port_range>
      <start>20122</start>
      <end>20122</end>
    </external_port_range>
  </port>
  <port>
    <type>telnet</type>
    <protocol>tcp</protocol>
    <vnf_port>23</vnf_port>
    <external_port_range>
      <start>20123</start>
      <end>20123</end>
    </external_port_range>
  </port>
</port_forwarding>
```

2. Log into VNF using SSH and port number given in the example payload (20122):

```
USER-M-G2PT:~ user$ ssh cisco@172.29.91.28 -p 20122
Password:

isrv-encs#
```

Import and Export NFVIS VM

Starting from NFVIS 3.10.1 release, you can backup or export (vmExportAction) and restore or import (vmImportAction) VMs. To backup or restore the whole NFVIS system, refer Backup and Restore NFVIS and VM Configurations.

VM Export and Import Limitations

- The imported VM cannot change datastore.
- The original registered image must exist.
- The OVS network name must be identical to the one used by original deployment.
- NFVIS does not check the disk space before exporting or importing a VM.

To export a VM ensure that:

- VM is in powered off state.
- Backup file must be saved to NFVIS datastore or USB.
- Provide a backup name for NFVIS to append .vmbkp extension to the backup name.

You can only create and save a VM backup to datastores. The backup file has .vmbkp extension. To verify the backup:

```
nfvis# show system file-list disk local | display xpath | include backup

/system/file-list/disk/local[si-no='84']/name tiny_backup.vmbkp
nfvis# show system file-list disk local 84
SI NO  NAME               PATH                SIZE  TYPE               DATE MODIFIED
-------------------------------------------------------------------------------------------
84     tiny_backup.vmbkp  /mnt/extdatastore1  17M   VM Backup Package  2019-01-31 19:31:32
```

To import a VM ensure that:

- The Backup file is placed under NFVIS datastores or USB.
- The registered image used by the original deployed VM is in the same datastore, with same properties.
- The exported VM does not exist on the system.
- OVS network used by the original deployment should exist.
- Restored VM is created with the same datastore with same deployment properties.
- The full path name to backup file is used (for example, /mnt/extdatastore1/backup.vmbkp, not extdatastore1:backup)

```
nfvis# vmImportAction importPath /mnt/extdatastore1/tiny_backup.vmbkp
System message at 2019-01-31 19:53:32...
Commit performed by admin via ssh using maapi.
```

The following examples show export failures:

- Original deployment is not deleted

```
nfvis# vmImportAction importPath /mnt/extdatastore1/tiny_backup.vmbkp
Error: Exception from action callback: Deployment Configuration :
'SystemAdminTenantIdtiny' already exists , can not be imported/restored due to conflict!
```

- 2. OVS network used by original deployment is deleted.

```
nfvis# vmImportAction importPath /mnt/extdatastore1/tiny_backup.vmbkp
Error: Exception from action callback: Restoration Request rejected, see logs for root
 cause
```

Secure Boot of VNFs

A key aspect of a secure compute system is to ensure that the system is running the intended software without malware or tampered software. This protection must begin as soon as the system is powered-on. The UEFI (Unified Extensible Firmware Interface) specification defines a secure boot methodology that prevents loading software which is not signed with an acceptable digital signature.

NFVIS already supports UEFI Secure Boot of Host. Secure boot ensures that the booted NFVIS software is genuine. Starting from NFVIS 3.11.1 release, the UEFI secure boot feature has been extended to VNFs being used in the Cisco NFV solution.

Booting a VNF securely requires the environment to support UEFI secure boot and requires modification to VNFs to support secure boot. NFVIS 3.11.1 and later releases have the infrastructure to support UEFI secure boot for VNFs that are capable of secure boot.



Note ISRv versions 16.12 and later support secure boot.

The firmware modes in which VNFs can boot are:

- BIOS - for VNFs which are not capable of secure boot
- UEFI secure - for VNFs capable of secure boot

VNFs can indicate secure boot capability using properties in the image_properties.xml file in the tar.gz package for the VNF.

You must set the following properties to enable secure boot of VNFs:

Name of the Image Property	Value for non-secure boot VNF	Value for secure boot VNF
boot_mode	bios	efi-secure-boot
shim_signature	N/A	microsoft **Note** For secure-boot capable VNFs, the VNF shim is typically signed by Microsoft.

Any combinations not matching the above default to the following:

- boot_mode: bios

- shim_signature: N/A



Note On the NFVIS portal, the **Image Repository** page shows if the image is capable of secure boot.