Understanding Virtual Network Function Descriptors

Virtual Network Function Descriptor Overview

ESC supports a TOSCA-based Virtual Network Function Descriptor (VNFD) to describe the VNF characteristics. The VNFD conforms to the *GS NFV-SOL 001 v.2.7.1* specifications and standard specified by ETSI (which in turn implements *TOSCA Simple Profile in YAML Version 1.2*).

The VNFD file describes the instantiation parameters and operational behaviors of the VNFs, their internal topology as well as their external connectivity. It also contains KPIs and other key requirements that can be used in the process of onboarding and managing the lifecycle of a VNF.

For VNF Lifecycle operations, see VNF Lifecycle Operations.

Defining Extensions to the Virtual Network Function Descriptor

The VNFM implements extensions to the VNFD to expose the more advanced concepts supported by ESC that are not explicitly defined in the ETSI standards. These extensions have been implemented in an ETSI-compliant way to ensure maximum compatibility with other ETSI NFV MANO components.

If there is a requirement to control these properties on a per-deployment basis, then replace the hard-coded values with inputs in the VNFD that can be supplied as *additionalParams* in the incoming request.

VNFCs (tosca.nodes.nfv.Vdu.Compute)

The Compute node allows for many of the ESC features to be exposed via the extended `tosca.datatypes.nfv.VnfcAdditionalConfigurableProperties`. This includes the following:

- Overriding the automatically generated name for a VNFC on the VIM.
- VIM flavor (overriding the ETSI capabilities specified for a VNFC).
- Supplying ESC with an expected bootup time to prevent further actions being taken until this timer has expired.

- Providing Day-0 configuration blocks to execute/store on the VNFC once deployed.
- Specifying KPI parameters and associated rules to configure the monitoring agent.
- Intra-VM Group placement rules.

Here is the data type extension definition:

```
data_types:
  ...
  cisco.datatypes.nfv.VnfcAdditionalConfigurableProperties:
    derived_from: tosca.datatypes.nfv.VnfcAdditionalConfigurableProperties
    properties:
      vim_flavor:
        type: string
        required: true
      bootup_time:
        type: integer
        required: true
      vm_name_override:
        type: string
        required: false
      recovery_action:
        type: string
        required: true
      recovery_wait_time:
        type: integer
        required: true
      monitor_on_error:
        type: boolean
        description: Continue monitoring of VNFC on error state.
        required: false
      max_retries:
        type: integer
        description: The number of recovery attempts
        required: false
      kpi_data:
        type: map # key: event_name
        description: The different KPIs applicable to this VDU
        required: false
        entry_schema:
          type: cisco.datatypes.nfv.data.Kpi
          description: A single KPI
      admin_rules:
        type: map # key: event_name
        description: Actions for events
        required: false
        entry_schema:
          type: cisco.datatypes.nfv.data.Admin_rules
          description: Define actions for events
      name_override:
        type: string
        description: An optional custom name that can be configured on the VIM
        required: false
      vendor_section:
        type: cisco.datatypes.nfv.VendorExtension
        required: false

  cisco.datatypes.nfv.VnfcConfigurableProperties:
    derived_from: tosca.datatypes.nfv.VnfcConfigurableProperties
    properties:
      additional_vnfc_configurable_properties:
        type: cisco.datatypes.nfv.VnfcAdditionalConfigurableProperties
        required: false
```

```
node_types:
  cisco.nodes.nfv.Vdu.Compute:
    derived_from: tosca.nodes.nfv.Vdu.Compute
    properties:
      configurable_properties:
        type: cisco.datatypes.nfv.VnfcConfigurableProperties
        description: Describes the configurable properties of all VNFC instances based on
this VDU
        required: false
```

For example:

```
vdu1:
      type: tosca.nodes.nfv.Vdu.Compute
      properties:
        name: Example VDU1
        description: Example VDU
        boot_order:
          - boot1-volume
        configurable_properties:
          additional_vnfc_configurable_properties:
            vim_flavor: Automation-Cirros-Flavor
            bootup_time: 1800
            vm_name_override: my-vdu-1
            recovery_action: REBOOT_THEN_REDEPLOY
            recovery_wait_time: 100
            monitor_on_error: false
            max_retries: 2
            kpi_data:
              VM_ALIVE-1:
                event_name: 'VM_ALIVE-1'
                  metric_value: 1
                  metric_cond: 'GT'
                  metric_type: 'UINT32'
                  metric_occurrences_true: 1
                  metric_occurrences_false: 30
                  metric_collector:
                    type: 'ICMPPing'
                    nicid: 1
                    poll_frequency: 10
                    polling_unit: 'seconds'
                    continuous_alarm: false
            admin_rules:
              VM_ALIVE-1:
                event_name: 'VM_ALIVE-1'
                action:
                  - 'ALWAYS log'
                  - 'FALSE recover autohealing'
                  - 'TRUE esc_vm_alive_notification'
            placement_type: zone
            placement_target: nova
            placement_enforcement: strict
            vendor_section:
              cisco_esc:
                config_data:
                  example.txt:
                    file: ../Files/Scripts/example.txt
                      variables:
                        DOMAIN_NAME: { get_input: DOMAIN_NAME }
                        NAME_SERVER: { get_input: NAME_SERVER }
                        VIP_ADDR: { get_input: VIP_ADDR }
                        VIP_PREFIX: { get_input: VIP_PREFIX }
        vdu_profile:
          min_number_of_instances: 1
```

```
        max_number_of_instances: 1
             capabilities:
      virtual_compute:
        properties:
          virtual_cpu:
            num_virtual_cpu: 8
          virtual_memory:
            virtual_mem_size: 16
    requirements:
      - virtual_storage:  cdr1-volume
      - virtual_storage:  boot1-volume
```

If *vm_name_override* is not specified, ESC will auto-generate the VM names.

ESC stores the VNFC specific value in *VnfInstance.instantiatedVnfInfo.vnfcResourceInfo.metadata.vim_vm_name* for the VNFC identified by the vduId, which matches the label given to the Compute node representing the VNFC.

Note You can supply a high number of input parameters, allowing the use of a single template for multiple deployments.

Connection Points (tosca.nodes.nfv.VduCp)

The Cisco extensions to the VduCp node type mainly allows for defining the interface requirements map. The features added to the connection point are as follows:

- Overriding the automatically generated name for a port on the VIM
- Identification of whether the port is a management port (i.e. used for monitoring)
- Allowed Address Pairs
- Support for specific network card types and interface types, e.g. SR-IOV
- Support for port binding profiles
- Whether port security is enabled

For example:

```
vdu1_nic0:
      type: tosca.nodes.nfv.VduCp
      properties:
        layer_protocols: [ ipv6 ]
        protocol:
          - associated_layer_protocol: ipv6
        trunk_mode: false
        order: 0
        virtual_network_interface_requirements:
          - support_mandatory: true
            network_interface_requirements:
                    allowed_address_pairs: { get_input: VDU1_NIC0_AADR_PAIRS }
                    nw_card_model: virtio
                    iface_type: direct
                    management: true
                    name_override: my-vdu1-nic0
                    port_security_enabled: false
                    binding_profile:
                      trusted: true
```

```
        requirements:
         - virtual_binding: vdu1
```

ESC supports SR-IOV properties using the network interface requirements. You can configure the interface to associate the VNFC with an SR-IOV pass-through adapter by specifying the type as direct, as per the previous example.

If there is a requirement to control these properties on a per-deployment basis, then replace the hard-coded values with inputs in the VNFD that can be supplied as *additionalParams* in the incoming request, as per the allowed address pairs above.

Note The port binding profile is available for Pike and later versions of OpenStack.

Volumes (tosca.nodes.nfv.Vdu.VirtualBlockStorage)

ESC supports out-of-band volumes as a Cisco extension. This allows the specification of the persistent volume UUID as the `resourceId` property against the `VirtualBlockStorage` node to be used in place of the ephemeral volume defined in the VNFD. ESC allows the request to override the volume specified in the VNFD and supplies its own persistent (deployed out-of-band) storage by identifying it with a UUID from the VIM.

For example:

```
boot1-volume:
      type: tosca.nodes.nfv.Vdu.VirtualBlockStorage
      properties:
        virtual_block_storage_data:
          size_of_storage: 4GB
          vdu_storage_requirements:
            resource_id: { get_input: VDU1_BOOT_VOL_UUID }
            vol_id: 1
            bus: ide
            type: LUKS
        sw_image_data:
          name: 'Automation_Cirros'
          version: '1.0'
          checksum: 9af30fce37a4c5c831e095745744d6d2
          container_format: bare
          disk_format: qcow2
          min_disk: 2 GB
          size: 2 GB
      artifacts:
        sw_image:
          type: tosca.artifacts.nfv.SwImage
          file: ../Files/Images/Automation-Cirros.qcow2
```

Note The VNFD accepts the volume or software image size in mebibyte-based units such as MiB, GiB or TiB equivalent. If the volume or software image size is in megabyte-based units such as MB, GB or TB, ESC converts the size to mebibyte-based equivalent and adjusts to the nearest value. Ensure you use mebibyte-based units for volume or software image size for clarity.

Security Group Rule (tosca.nodes.nfv.VduCp)

As per the handling a persistent of the volume above, ESC provides the ability to specify an out-of-band security group instead of configuring one in the VNFD. This is because the verbs used to describe the security

group in the standards documentation are too simplistic for a very complicated configuration. Since the security group is being specified for use on a connection point, this is where it is defined in the VNFD.

For example:

```
c1_nic0:
  type: tosca.nodes.nfv.VduCp
  properties:
    order: 0
    layer_protocols: [ ipv6 ]
    protocol:
      - associated_layer_protocol: ipv6
    trunk_mode: false
    virtual_network_interface_requirements:
      - support_mandatory: true
        network_interface_requirements:
          management: "false"
          iface_type: "virtual"
  metadata:
    security_groups: { get_input: VIM_NETWORK_SEC_GRP_0 }
  requirements:
    - virtual_binding: c1
```

Virtual Links (tosca.nodes.nfv.VnfVirtualLink)

The virtual links defined in the VNFD can be used to define those physical provider networks.

For example:

```
 vpc-di-internal1:
  type: tosca.nodes.nfv.VnfVirtualLink
  properties:
    connectivity_type:
      layer_protocols: [ ipv4 ]
    description: DI Internal 1 Network VL
    vl_profile:
      max_bitrate_requirements:
        root: 100000
      min_bitrate_requirements:
        root: 0
      virtual_link_protocol_data:
        - associated_layer_protocol: ethernet
          l2_protocol_data:
            network_type: vlan
            segmentation_id: { get_input: VL1_SEG_ID }
            physical_network: vlan_network
```

They can also be used to specify the IP subnets that an internal connection point may use when using DHCP to assign an address to them.

For example:

```
vpc-di-internal2:
  type: tosca.nodes.nfv.VnfVirtualLink
  properties:
    connectivity_type:
      layer_protocols: [ ipv4 ]
    description: DI Internal 1 Network VL
    vl_profile:
      max_bitrate_requirements:
        root: 100000
      min_bitrate_requirements:
        root: 0
      virtual_link_protocol_data:
```

```
- associated_layer_protocol: ipv4
  l3_protocol_data:
    ip_version: ipv4
    cidr: 1.180.10.0/29
    dhcp_enabled: true
```

For information on lifecycle management operations, see Managing the VNF Lifecycle.

Note The previous versions of ESC supported Cisco-only extensions to support the above functionality. These extensions were outside of the specification and although now these extensions are largely conformant with the SOL001 standard, the previous definitions are still supported by ESC for backward compatibility. For more information, see the Cisco Elastic Services Controller 5.5 documentation.