VNF LCM API Emulator

User Manual

*Authors (ETSI STF 598 Experts):*

*Hammad Zafar, Rehan Abbasi, Muhammad Hamza (XFlow Research)*

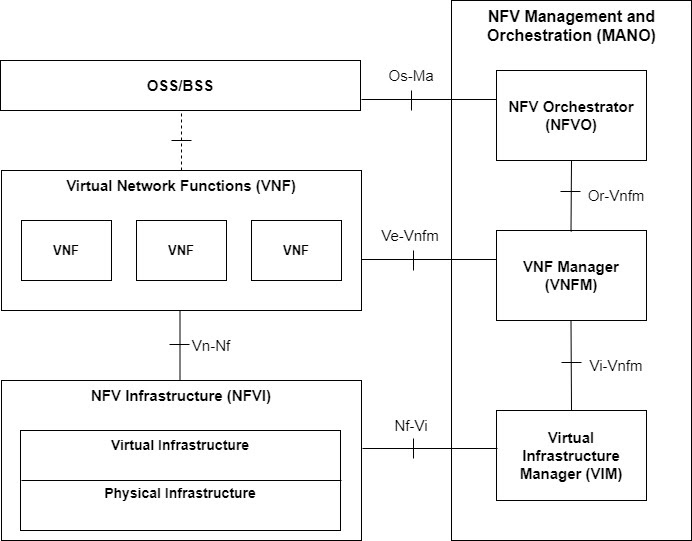
*Pietro Piscione, Giacomo Bernini (Nextworks)*

1 Introduction

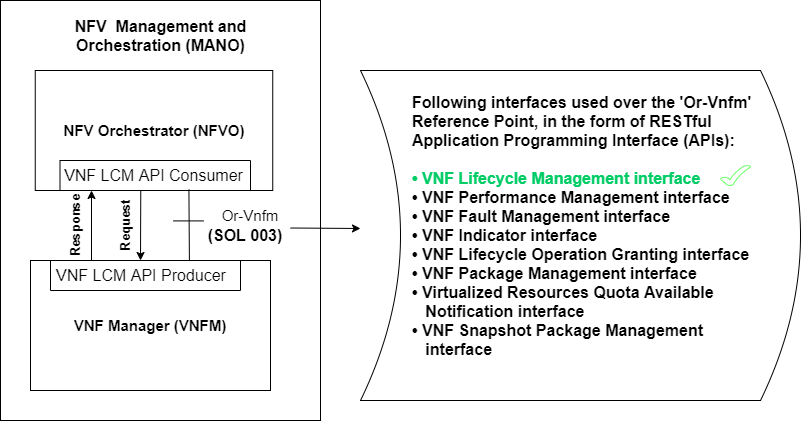
VNF LCM API Emulator is a web-based application which emulates the functionality of the VNF Lifecycle Management (LCM) API producer. The tool is developed with an aim to provide a try-and-learn approach to understand how requests are served and responses are generated between the NFV architectural blocks.

2 Emulator and NFV Architecture:

ETSI ISG NFV specifies a standard NFV Management and Orchestration (MANO) framework along with the reference points between the functional blocks. These reference points contain interfaces to support distinct functionalities such as Configuration, Lifecycle Management, Fault Management etc. A high-level architecture of ETSI NFV functional blocks and the various reference points between these blocks can be seen in Figure 1.

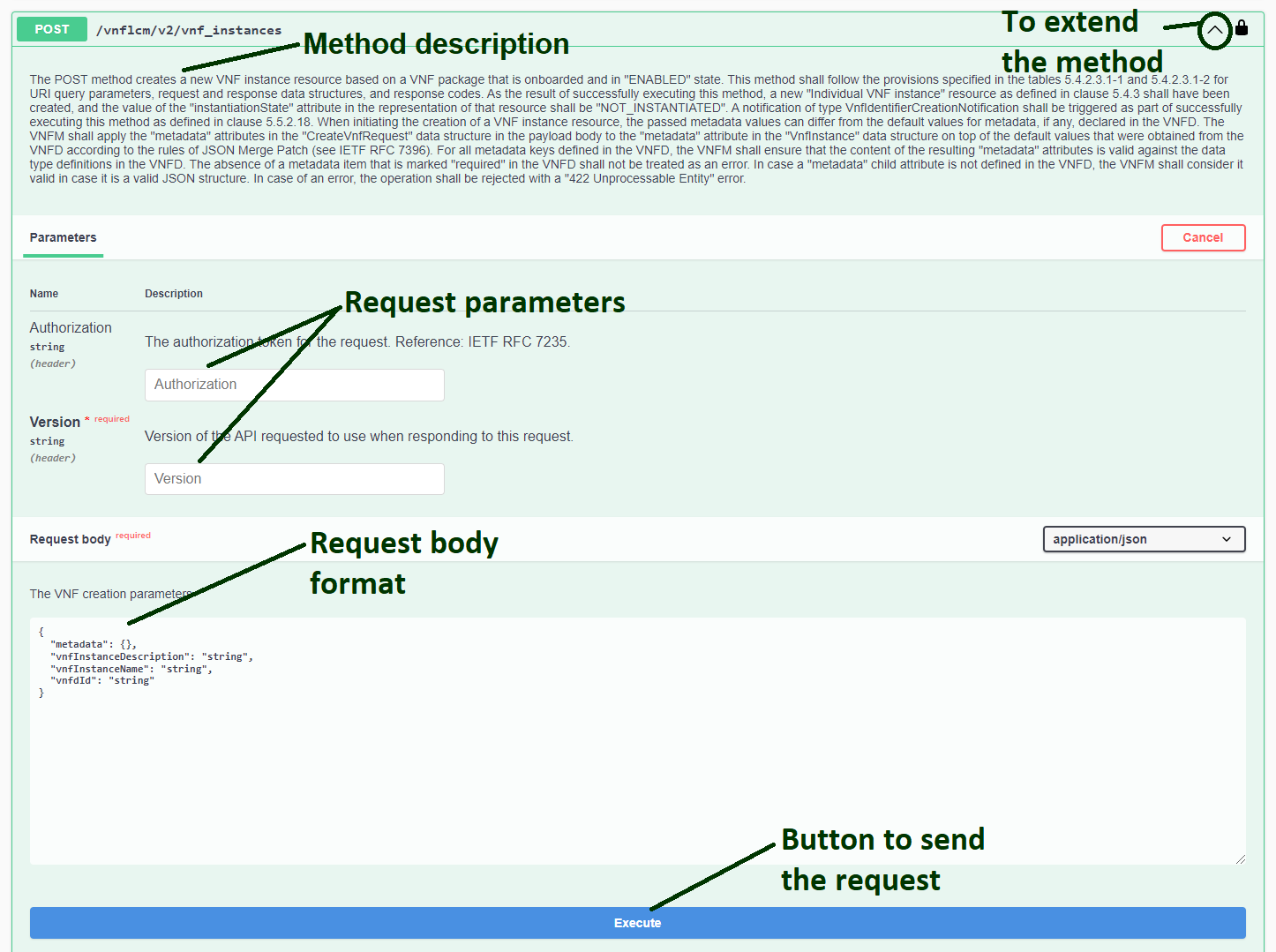
  
Figure 1

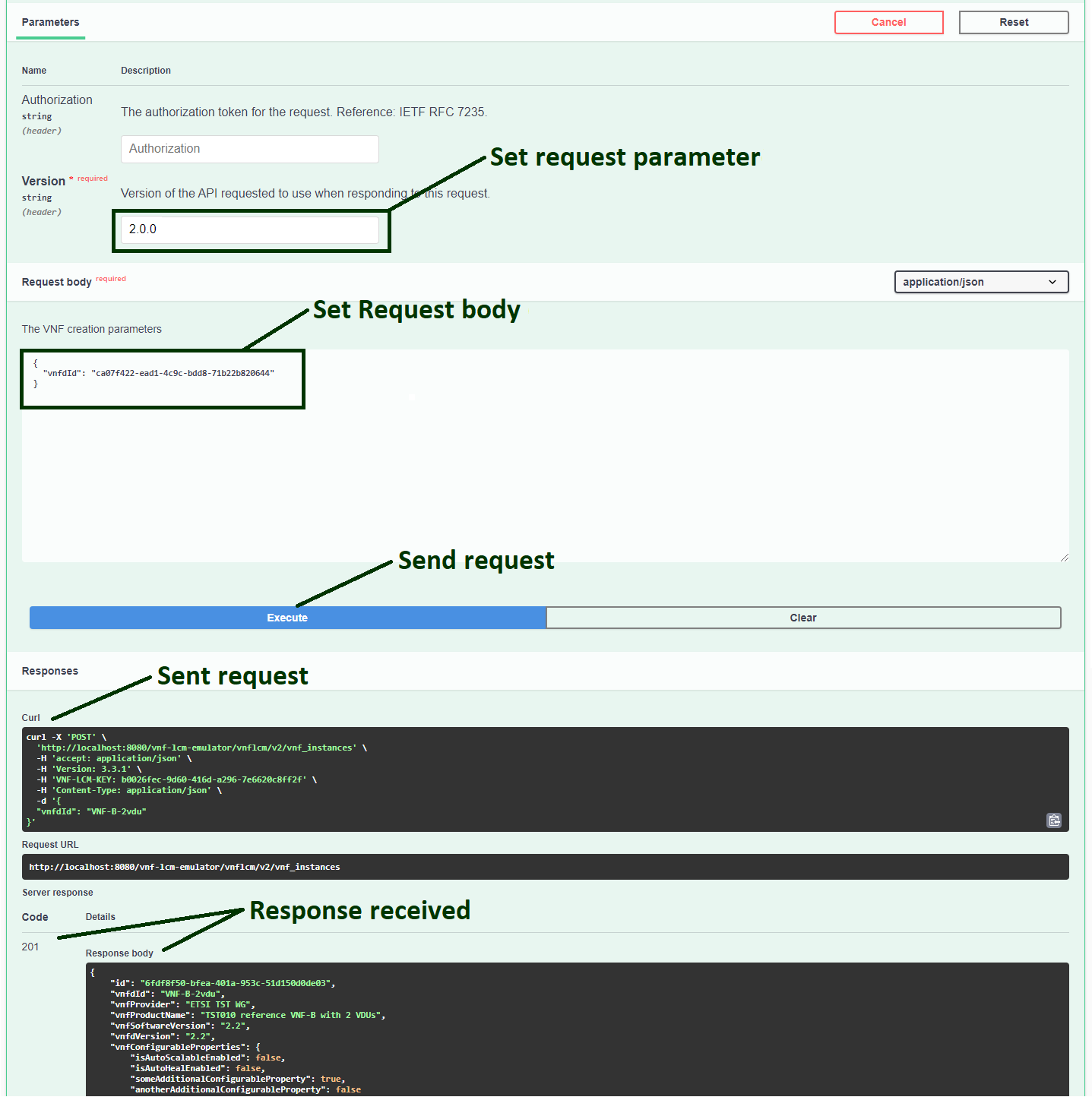
NFV Orchestrator (NFVO) is a central component in the NFV ecosystem, responsible for the orchestration of Network Services (NS) and the VNFs that make up those NS. On the other hand, the VNF Manager (VNFM) is mainly responsible for the configuration, fault management and lifecycle management of VNFs running on top of the NFVI. The VNF LCM Emulator provides a playground environment to interact with the SOL003 VNF LCM API, produced by the VNFM and consumed by the NFVO as specified in the [ETSI NFV SOL003 v3.3.1](https://www.etsi.org/deliver/etsi_gs/NFV-SOL/001_099/003/03.03.01_60/gs_NFV-SOL003v030301p.pdf). Figure 2 illustrates all the interfaces on the Or-Vnfm Reference Point including the VNF LCM API, which is being emulated in this tool.

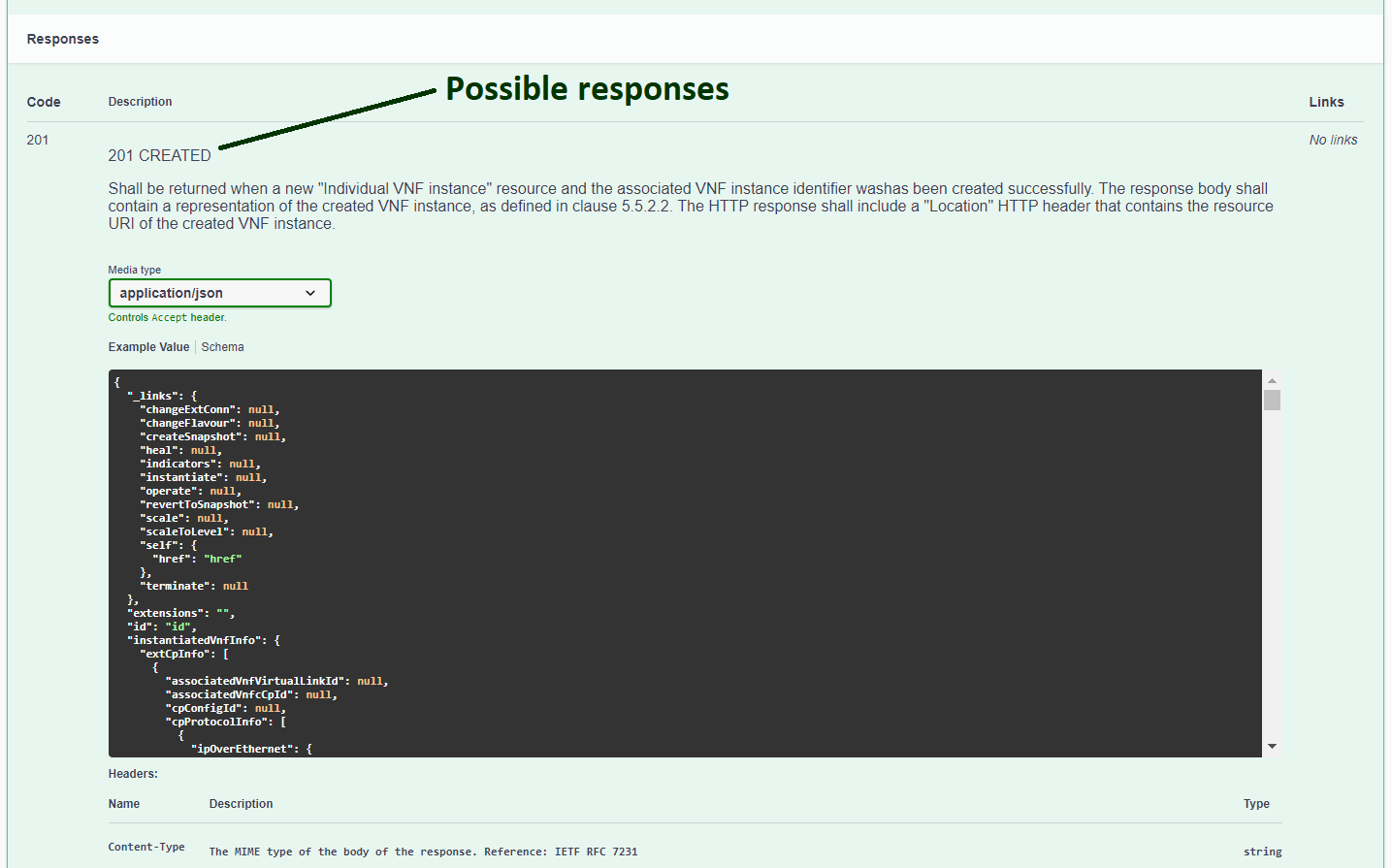
  
Figure 2

3 User Interface:

Swagger UI is used as a user interface for the VNF LCM Emulator. The UI contains the supported endpoints of the VNF LCM API and their associated HTTP methods as specified in SOL003 v3.3.1. All the endpoints can be expanded, upon expansion every method shows a brief description about its functionality, parameters’ input fields, and request body format (as shown in Figure 3). To send a request, users are required to enter valid data in the input fields for setting the request parameters. And for the POST methods, an additional input box is there for populating the body of the HTTP request (as shown in Figure 4). Furthermore, every method also contains all the possible response codes along with their description, and response bodies (as shows in Figure 5).

Figure 3

Figure 4

Figure 5

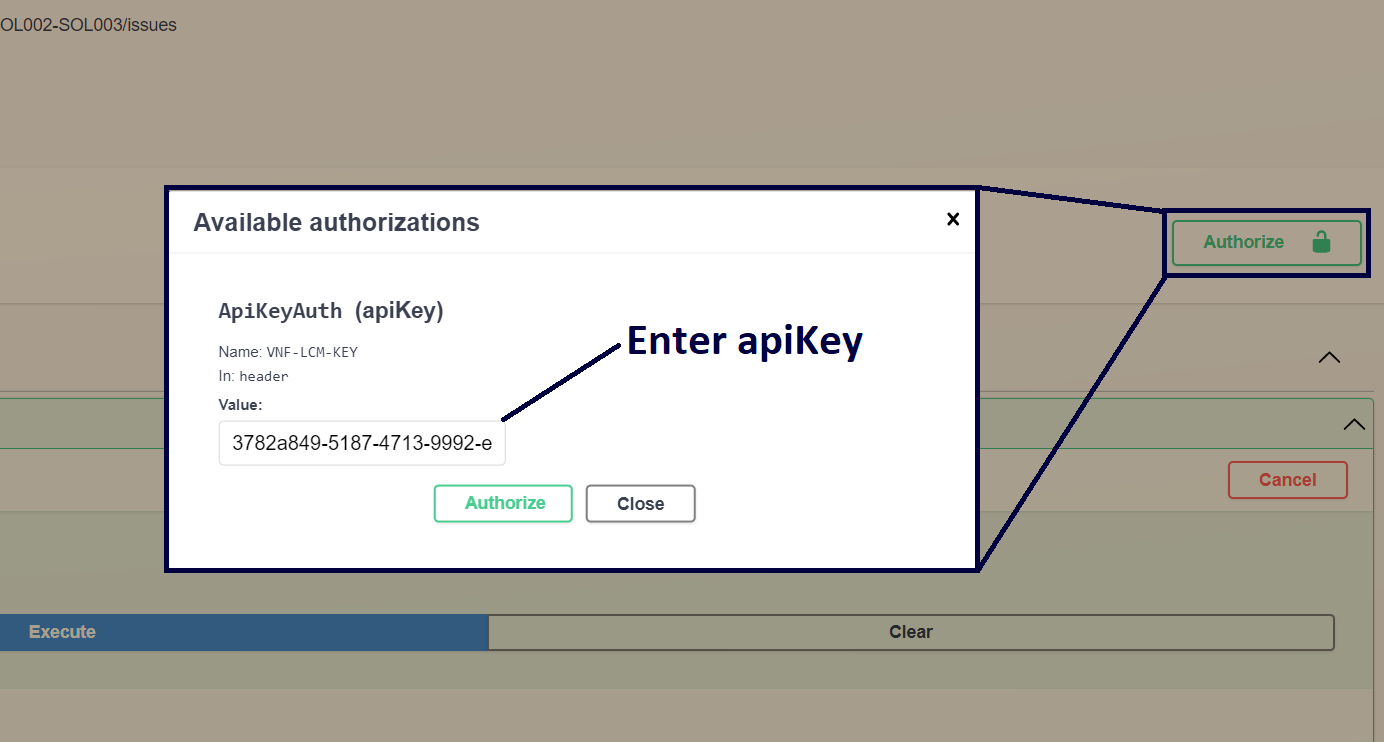
4 Getting Started:

To interact with the tool, session authorization has been made mandatory. Users are first required to get authorized using API key to start interacting with the emulator.

4.1 Api Key Management:

4.1.1 POST /api\_key:

For authorization, it is necessary to send the POST request on */api\_key* endpoint which will generate a unique API key for the user. Expand the *POST /api\_key* method, and click *Execute* button to send a request. A successful response will be received with the status code 200 containing the API key in the response body. Copy this API key without quotation marks, click the *Authorize* button on the top right corner, a popup will appear. In the popup, paste the copied API Key in the input field of the *ApiKeyAuth* section and click the *Authorize* button to get authorized, as shown in Figure 6.

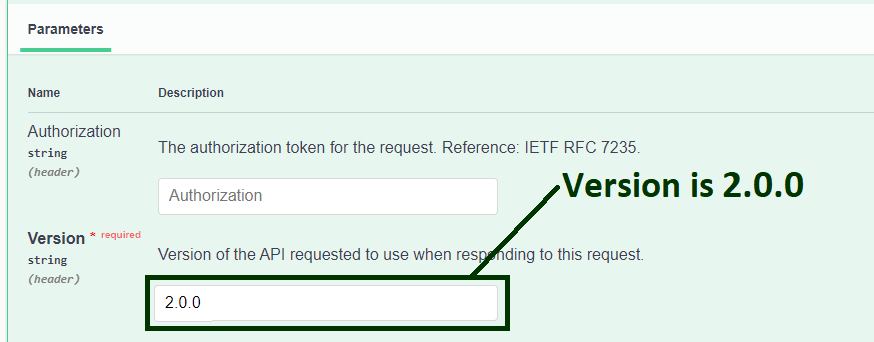
  
Figure 6

5 Endpoints and Methods:

This section will give a brief overview of all of the featured endpoints and their supported methods that allow the NFVO to invoke VNF lifecycle management operations of VNF instances towards the VNFM.

In addition to standard endpoints for VNF LCM API specified in SOL003, we have introduced a few custom endpoints to support tool-specific functionality. One such endpoint is the POST /api\_key endpoint that is used for user authorization. Furthermore, there are some “Management” endpoints to fetch available VNF descriptors (VNFDs) and to retrieve generated notifications against their associated subscriptions during LCM operations.

Current release of the emulator is limited to version 3.3.1 of the VNF LCM API​ as specified in SOL003 (v3.3.1). In all the standard endpoints, the user is required to enter “2.0.0” in the “Version” request parameter as shown in Figure 7. Non-standard, custom endpoints (such as /api\_key and Management endpoints) do not require the *Version* parameter.

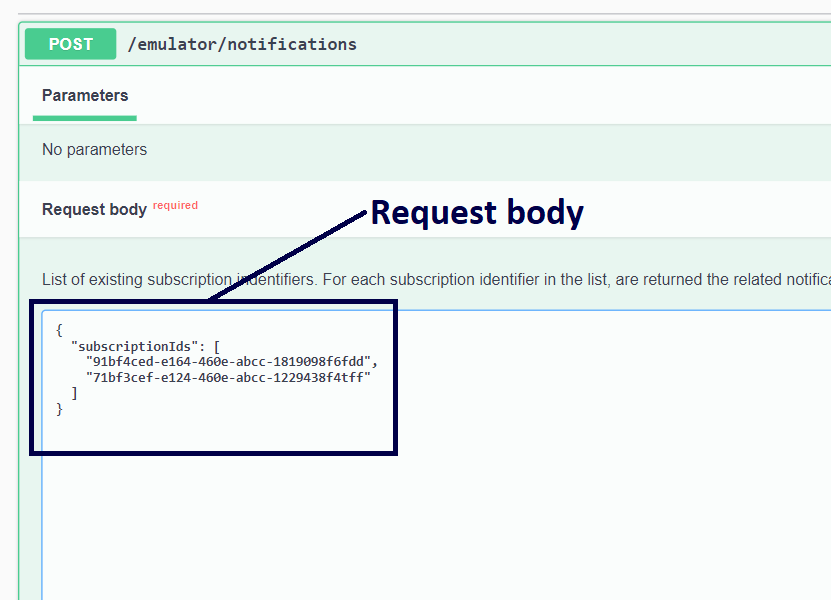
  
Figure 7

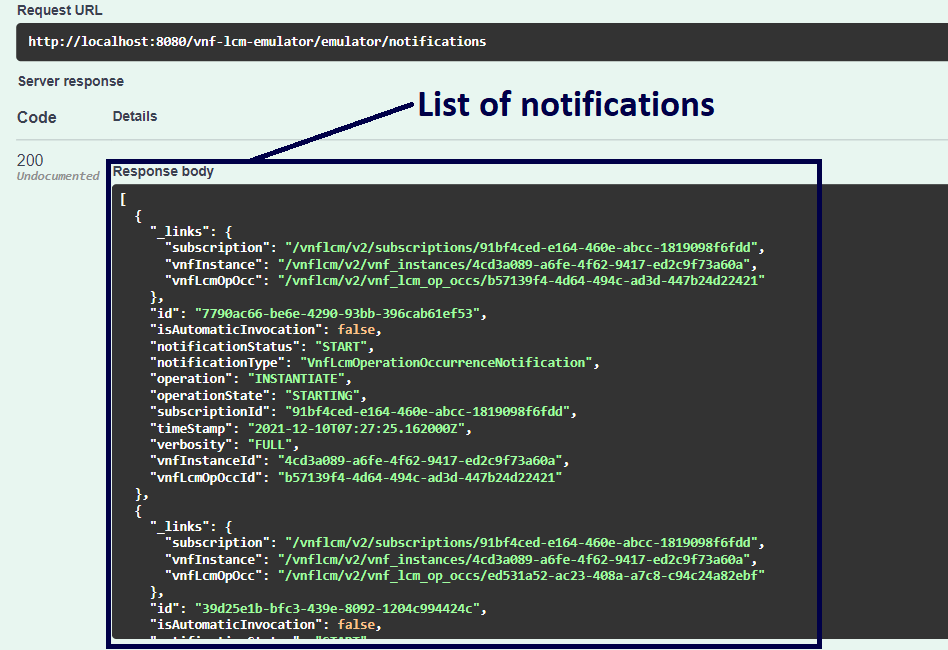
5.1 Management:

This section describes the non-standard, custom endpoints introduced to support the retrieval of VNFDs and notifications.

5.1.1 POST /emulator/notifications:

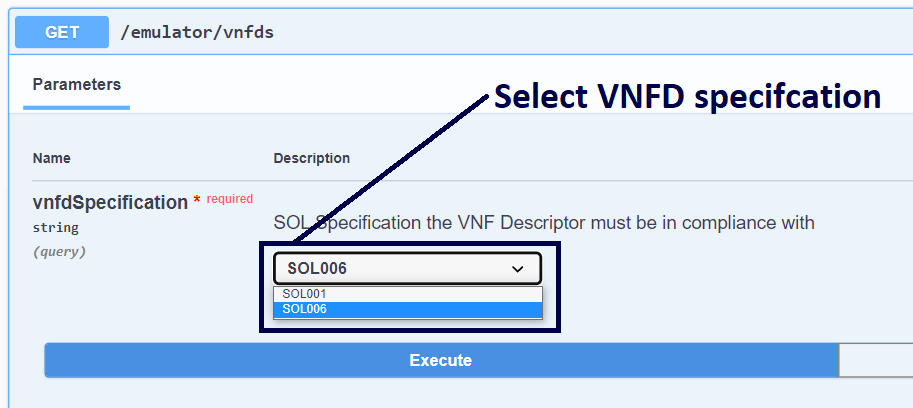
This POST method is used to retrieve generated notifications for that particular subscription. To execute this method, multiple *subscriptionIds* can be provided in the request body, as shown in Figure 8. A successful response will be received with the status code 200 along with the list of notifications for all requested subscriptions, as shown in Figure 9.

  
Figure 8

  
Figure 9

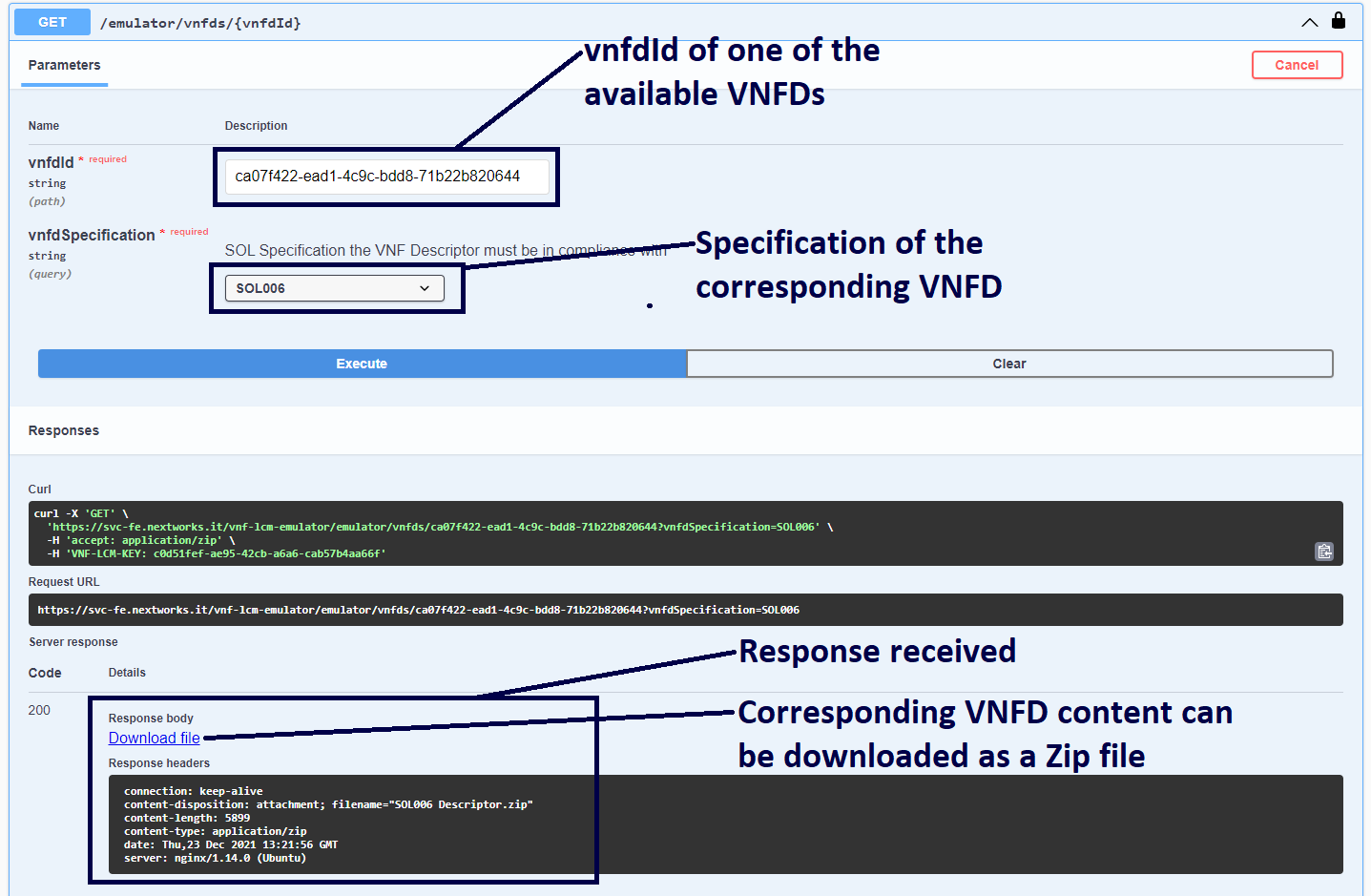
5.1.2 GET /emulator/vnfds:

This method is used to get the available Virtual Network Function Descriptors (VNFDs). A VNFD is a deployment template which describes a VNF in terms of deployment and operational behaviour requirements. The information within a VNFD is structured according to one or more VNF deployment flavours (VnfDf) that specify different deployment configurations for a VNF, in terms of its internal topology and resource needs. As a request parameter, select the *vnfdSpecification* from the dropdown and execute the method to retrieve the VNFDs in desired specification, as shown in Figure 10. A successful 200 response will return all on-boarded VNFDs in a zip format.

  
Figure 10

5.1.3 GET /emulator​/vnfds​/{vnfdId}:

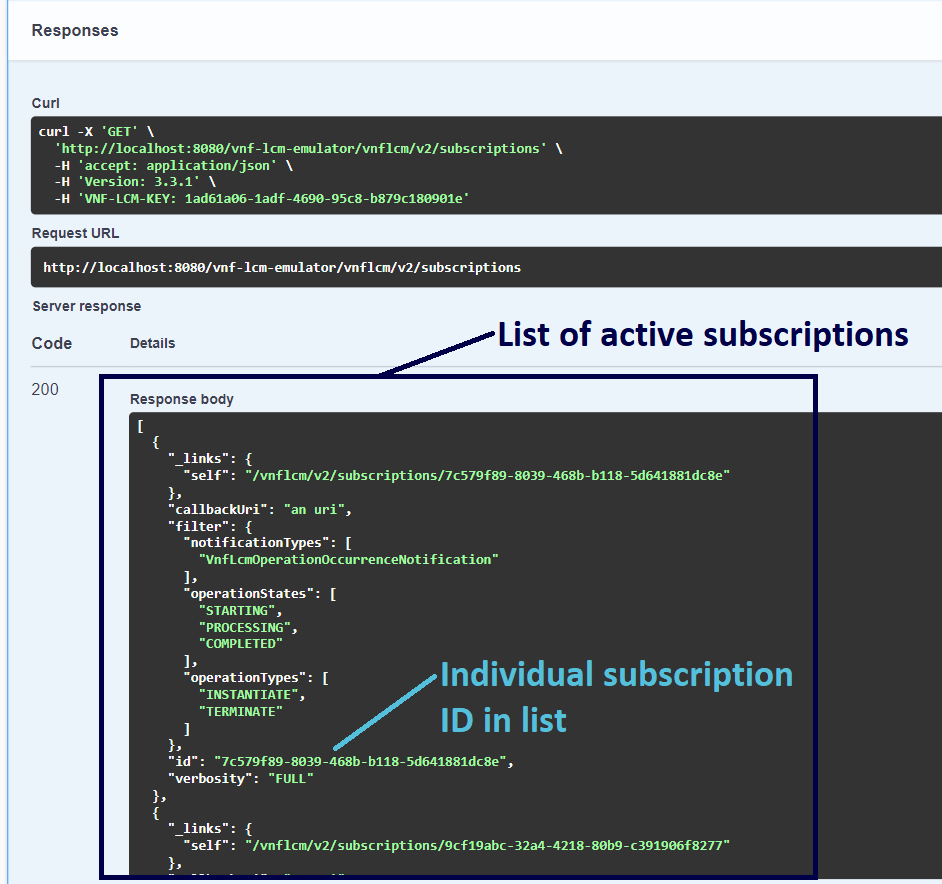
The GET method to this endpoint returns a specific VNFD when the corresponding *vnfdId* is provided as request parameter. Provide a valid vnfdId of one of the available VNFDs, select *vnfdSpecification* for that particular VNFD, and click on the Execute button. A successful response will be received with the status code 200 along with a link to download corresponding VNFD content as a zip file, as shown in Figure 11. Flavour IDs, scaling aspects and other information for the corresponding VNFD can be seen in the extracted file.

  
Figure 11

5.2 Subscriptions:

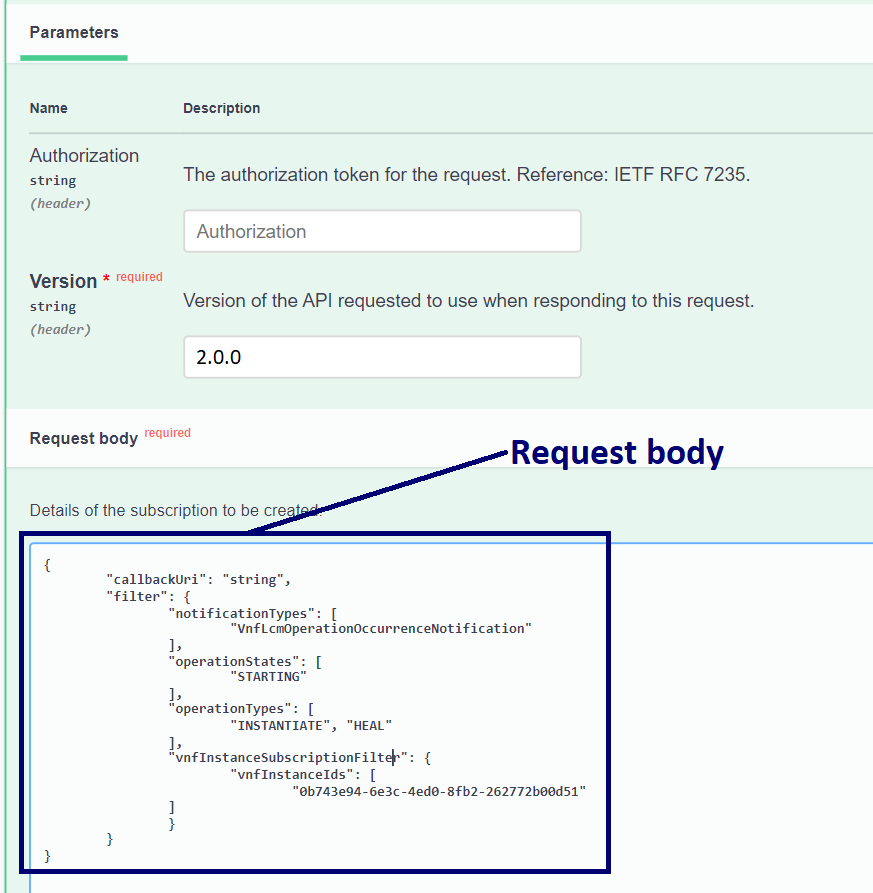
5.2.1 GET /vnflcm/v2/subscriptions:

The GET method is used to query the list of all the existing subscriptions. Upon execution, a successful response will be received containing an array representation of all the created subscriptions as shown in Figure 12.

  
Figure 12

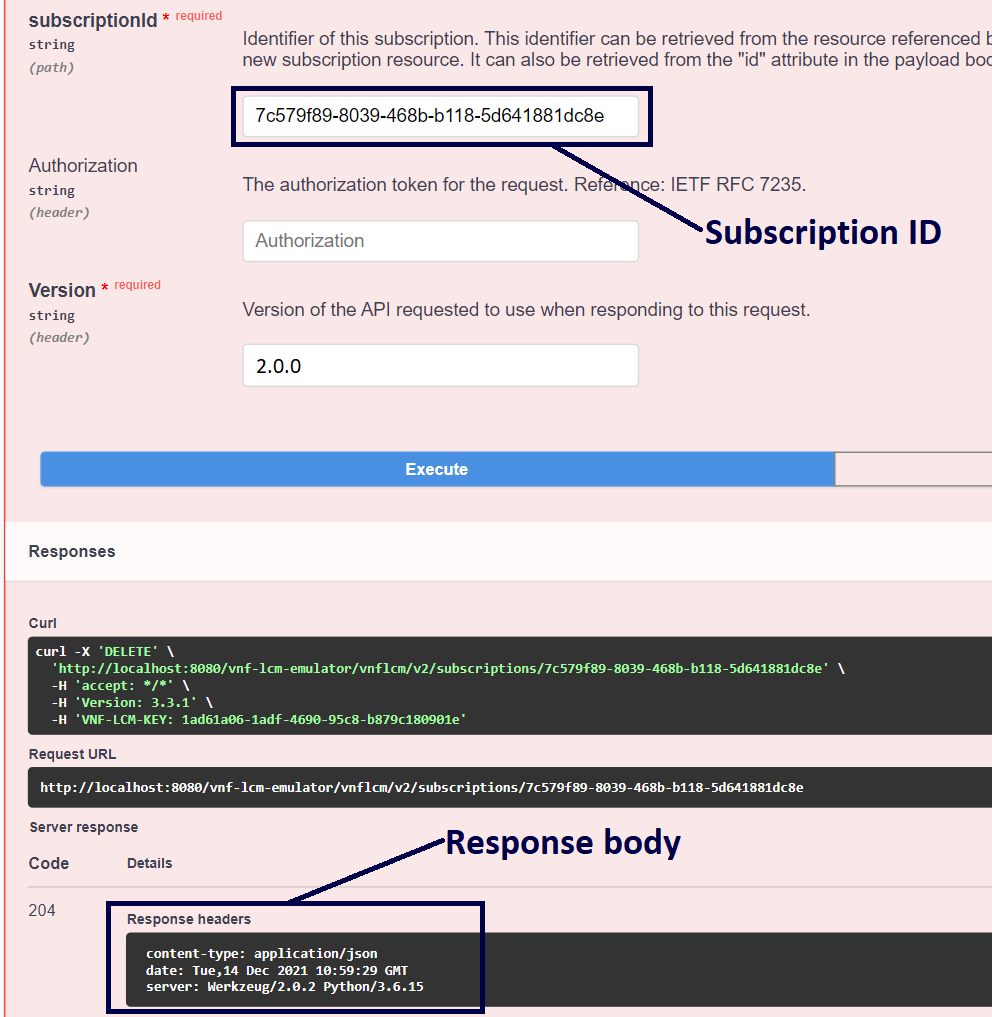
5.2.2 POST /vnflcm/v2/subscriptions:

This method is used to create a new subscription resource. To retrieve the notifications about the operations being performed on VNF instance(s), it is mandatory to subscribe first. To execute this method, provide the mandatory attribute *callbackUri* along with the optional attribute *filter*. Multiple attributes within *filter* can be provided in the request body to get notifications about desired operations, as shown in Figure 13. Note that in request body, *operationTypes* and *operationStates* attributes can only be provided if *notficationTypes* attribute is present.

  
Figure 13

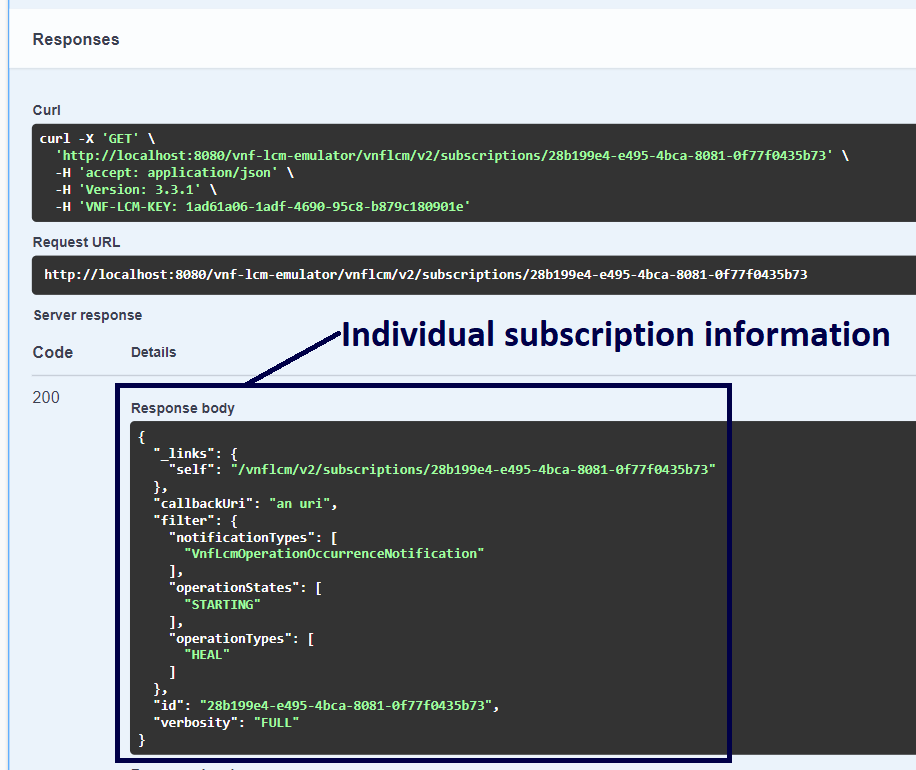
5.2.3 DELETE /vnflcm/v2/subscriptions/{subscriptionId}:

The DELETE method is used to terminate an individual subscription resource. To execute this method, provide *subscriptionId*of one of the created subscriptions as a request parameter, as shown in Figure 14. Upon execution of this method, a successful response will be received with the status code 204 indicating that the corresponding subscription resource has deleted successfully.

  
Figure 14

5.2.4 GET /vnflcm/v2/subscriptions/{subscriptionId}:

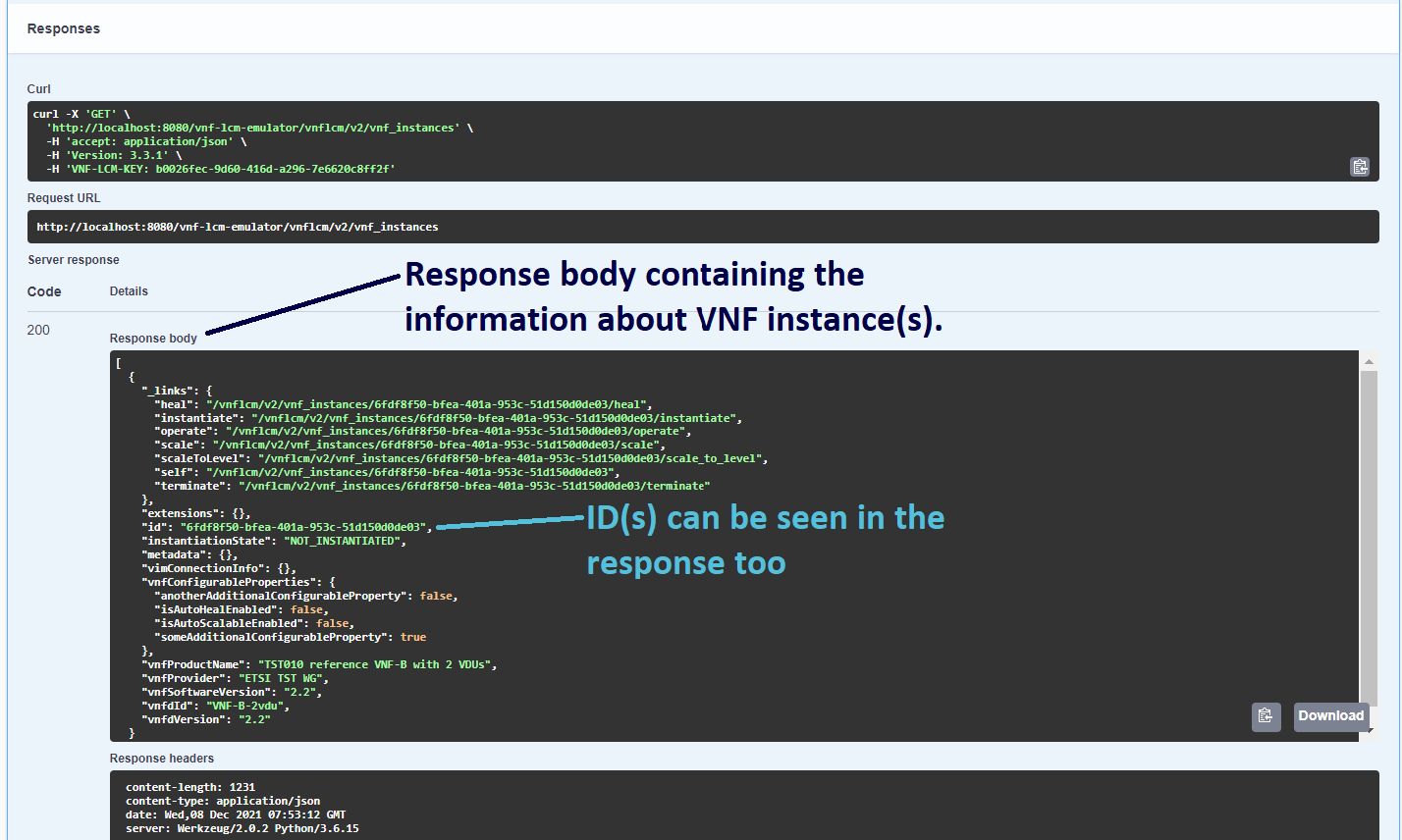
This method is used to retrieve information about an individual subscription. Provide *subscriptionId*of one of the created subscriptions as a request parameter to GET representation of the individual subscription resource, as shown in Figure 15.

  
Figure 15

5.3 [VNF Instances](http://localhost:8080/ui/#/VNF%20Instances):

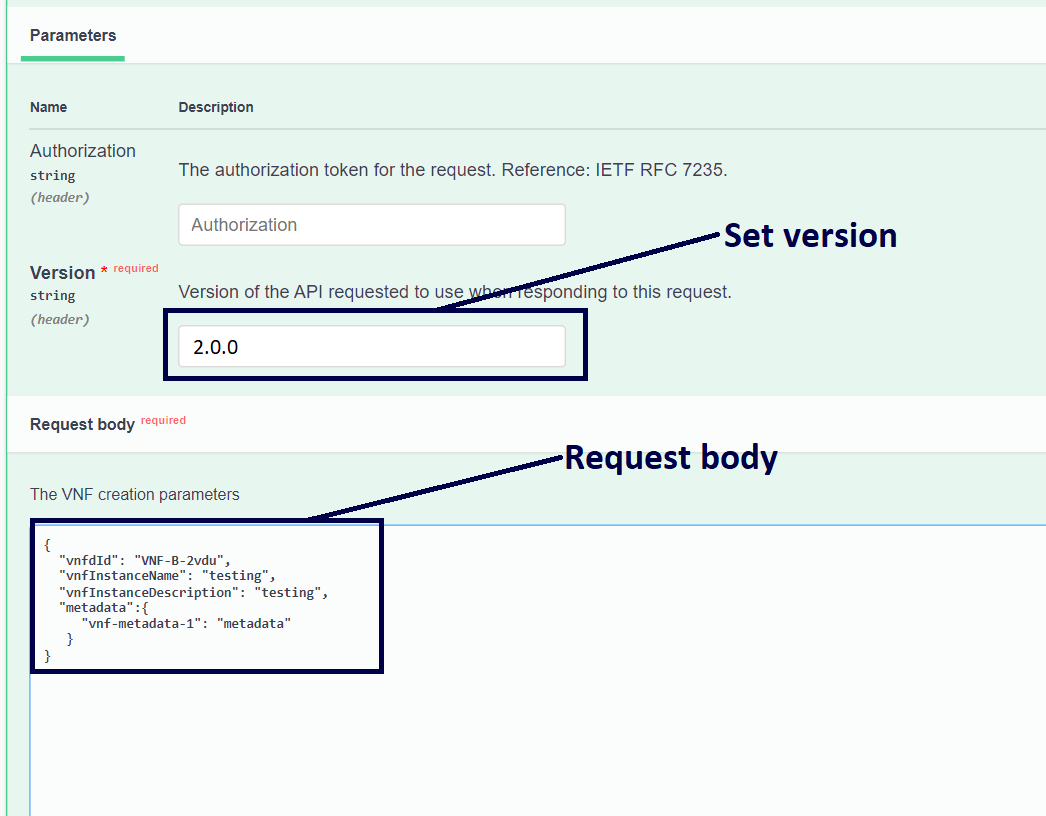
5.3.1 GET /vnflcm/v2/vnf\_instances:

The GET method is used to query information about all existing VNF instances. Upon execution, a successful response will be received containing the information about VNF instance(s) as shown in Figure 16.

  
Figure 16

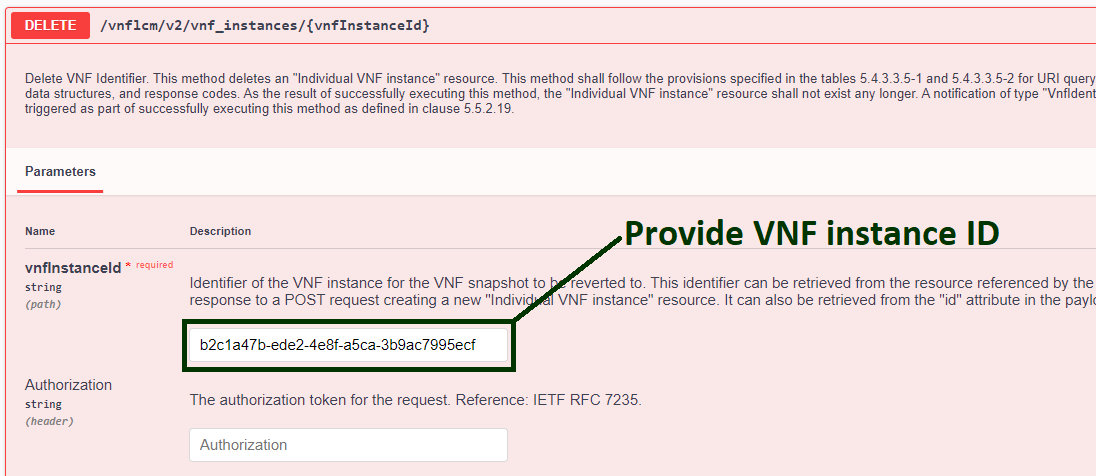
5.3.2 POST /vnflcm/v2/vnf\_instances:

This method is used to create a new VNF instance resource based on one of the VNFD templates. As shown in Figure 17, fill out the mandatory attribute *vnfdId* (ID of the desired VNFD template) in request body and *Execute* the method to create a VNF instance. A successful response will be received with the status code 201 containing the information about the newly created VNF instance.

  
Figure 17

5.3.3 DELETE /vnflcm/v2/vnf\_instances/{vnfInstanceId}:

The DELETE method is used to delete an individual VNF instance resource. As shown in Figure 18, provide *vnfInstanceId* of one of the created VNF instances as a request parameter. The VNF instance should not be in “INSTANTIATED” state.

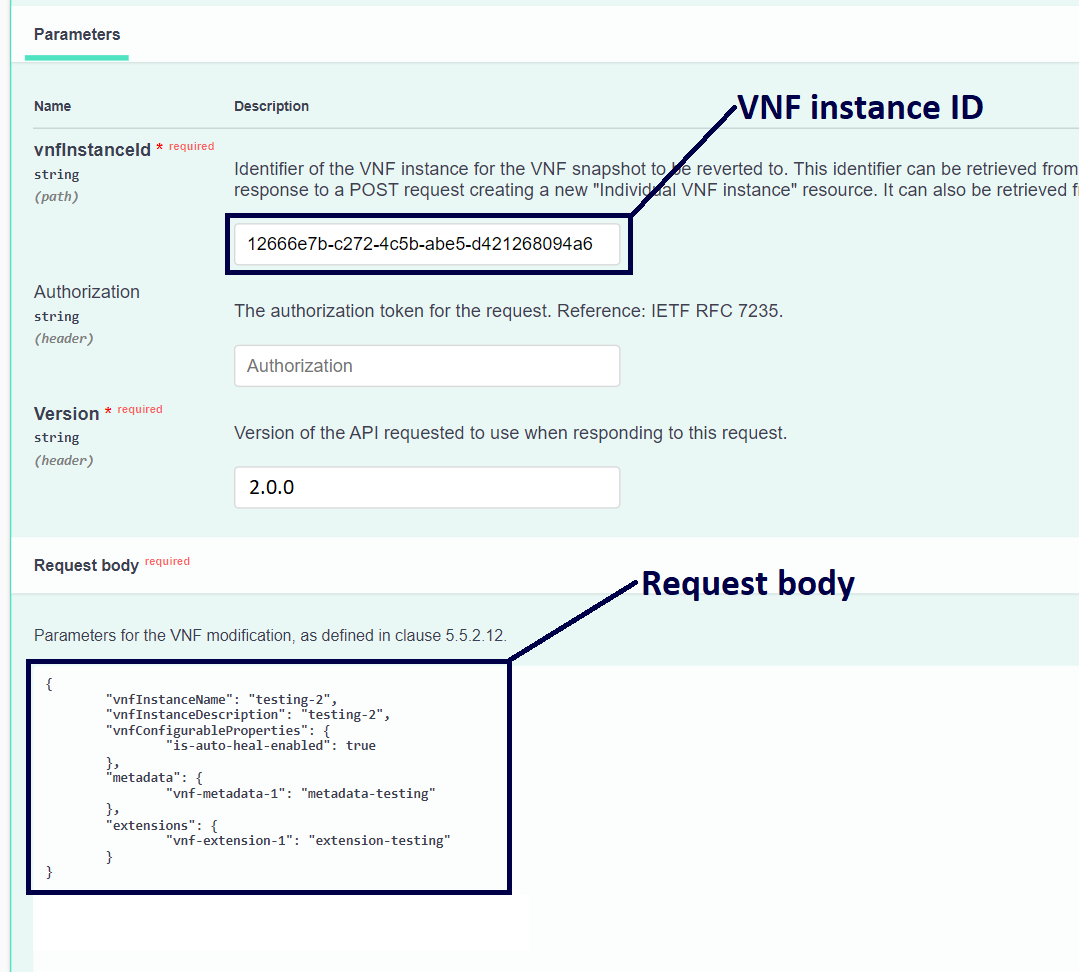
  
Figure 18

5.3.4 GET /vnflcm/v2/vnf\_instances/{vnfInstanceId}:

This method is used to retrieve information about an individual VNF instance resource. Provide *vnfInstanceId* of one of the created VNF instances as a request parameter to GET an individual VNF instance resource.

5.3.5 PATCH /vnflcm/v2/vnf\_instances/{vnfInstanceId}:

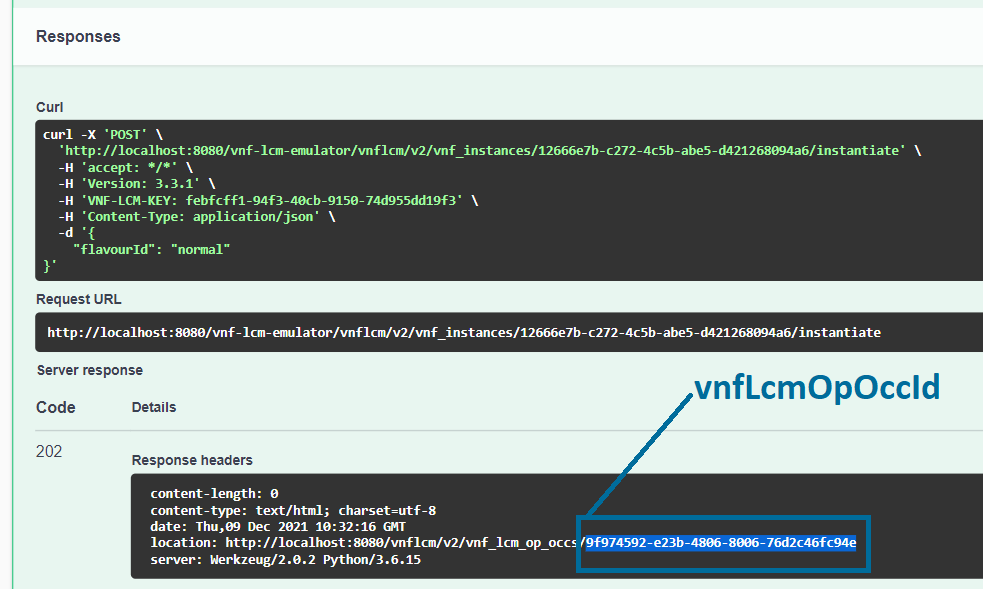
This method is used to modify individual VNF instance resource. Provide *vnfInstanceId* of one of the created VNF instances as a request parameter, and in request body provide the desired attributes for the VNF modification, as shown in Figure 19.

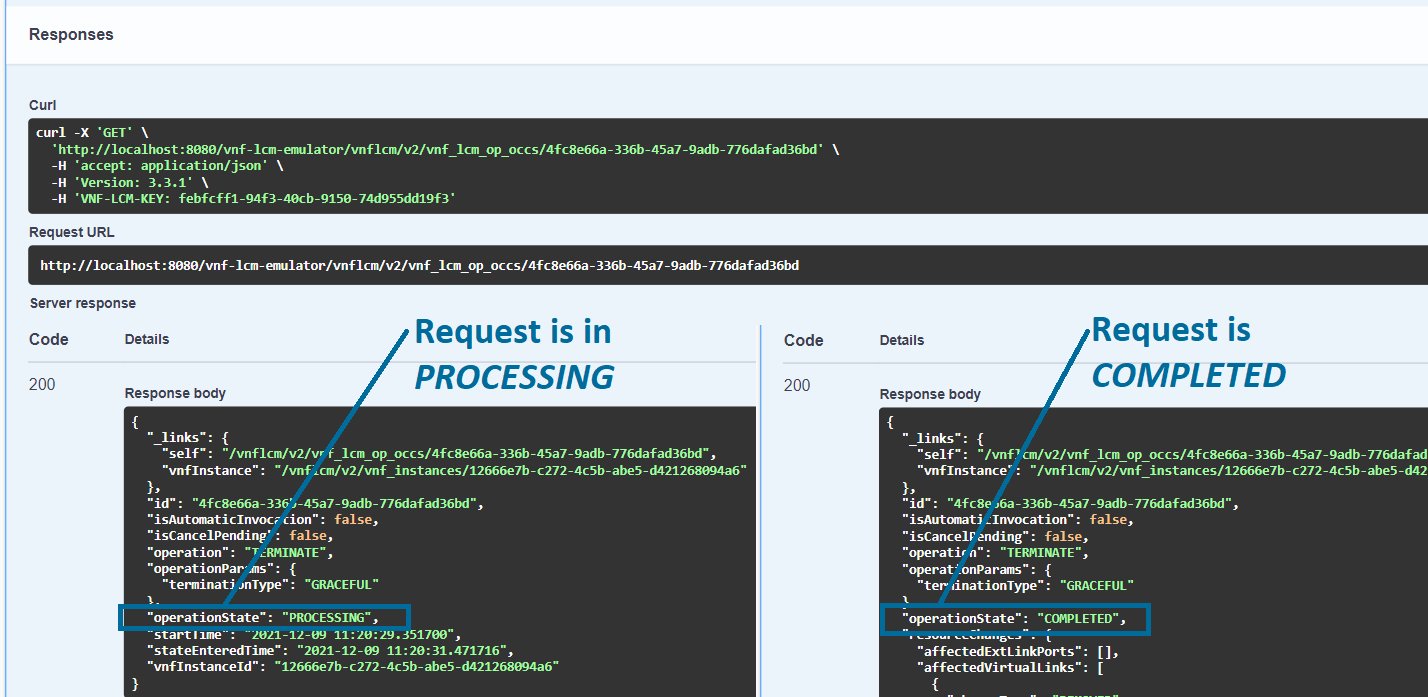
  
Figure 19

### 5.4 VNF LCM Operations:

This section describes the standard endpoints which supports the VNF Lifecycle Management operations, these operations can influence the allocation of virtualized resources to a VNF instance, and/or modify the state of the VNF instance.

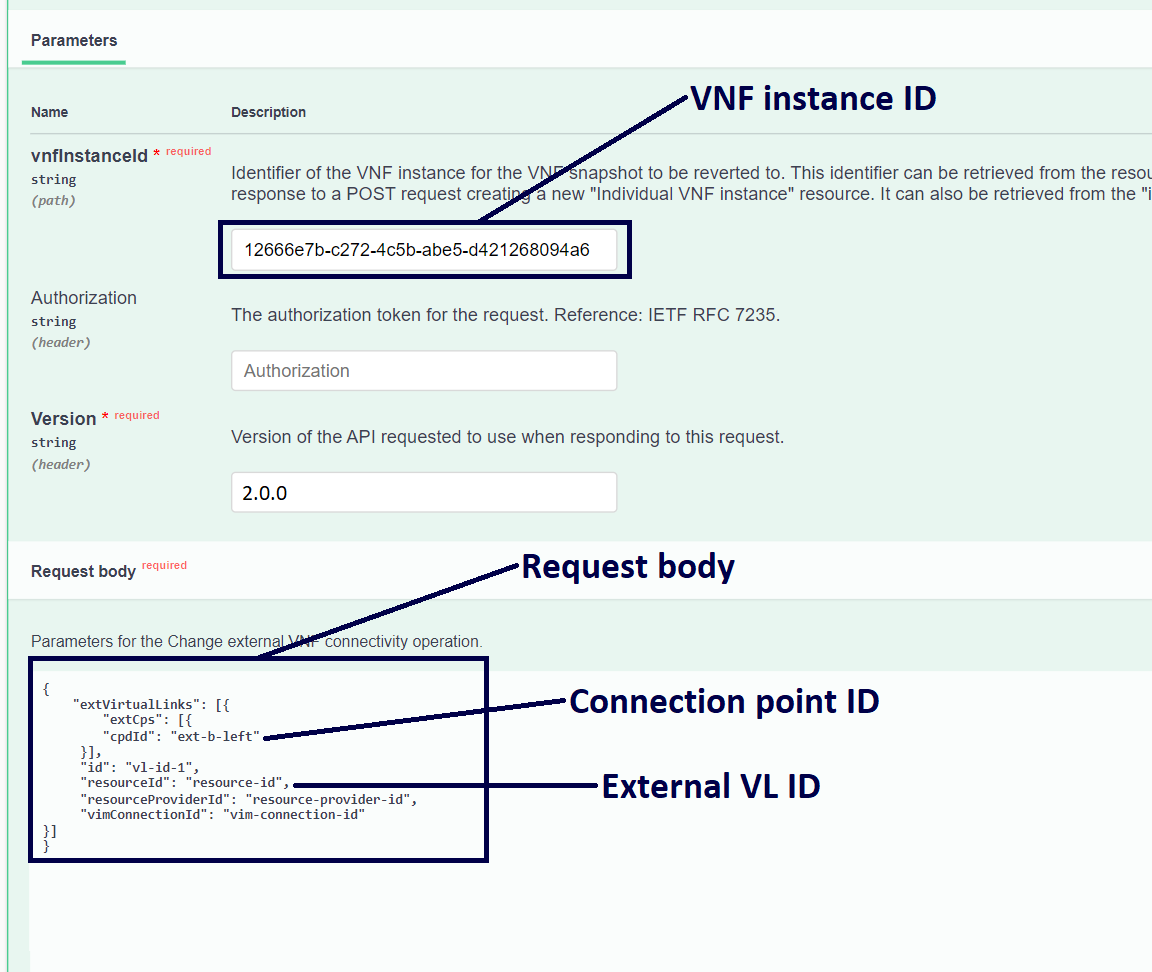
For every successful request to the VNF LCM endpoints, a 202 response status code will be received along with the response headers indicating that the request has been accepted. To check the status of ongoing VNF LCM operation, copy *vnfLcmOpOccId* (as shown in Figure 20) and paste it in the [GET /vnflcm/v2/vnf\_lcm\_op\_occs/{vnfLcmOpOccId}](#_5.5.2__) method as a request parameter and click the *Execute* button. The *operationState* in the response body can be *PROCESSING* or *COMPLETED*. *PROCESSING* means that operation is under processing and has not completed yet and *COMPLETED* means that the corresponding VNF LCM operation has completed successfully, as shown in the Figure 21. Once the operation is completed, the modified resource representation of the corresponding VNF instance can be verified using the GET method for the Individual VNF Instance resource.

  
Figure 20

  
Figure 21

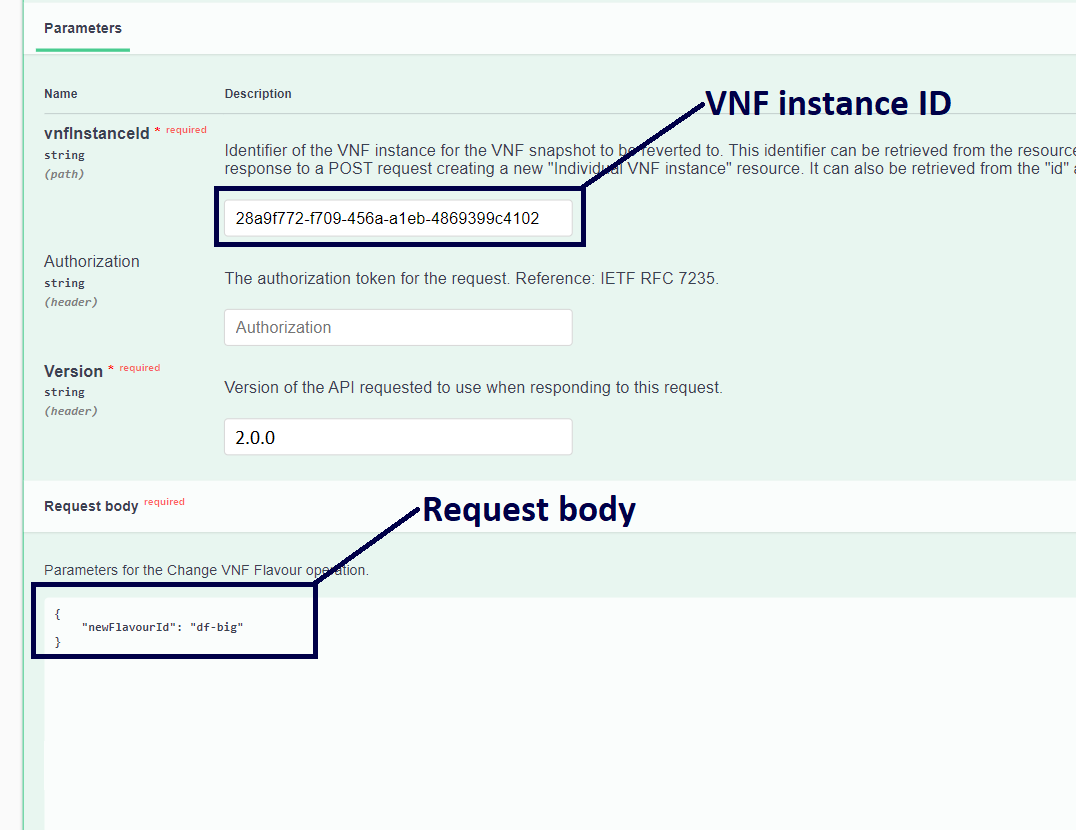
#### 5.4.1 POST /vnflcm/v2/vnf\_instances/{vnfInstanceId}/change\_ext\_conn:

This POST method is used to change the external connectivity of a VNF instance resource. Changing external connectivity means disconnecting a VNF's external Connection Points (CPs) from one external Virtual Link (VL) and connecting them to another external VL. Provide vnfInstanceId of one of the created VNF instances as a request parameter, and in request body provide the mandatory attributes *cpdId* (Identifier of the CP) and *id* (Identifier of new VL), as shown in Figure 22. A successful response will be received with the status code 202 indicating the request to change the VNF external connectivity has accepted.

  
Figure 22

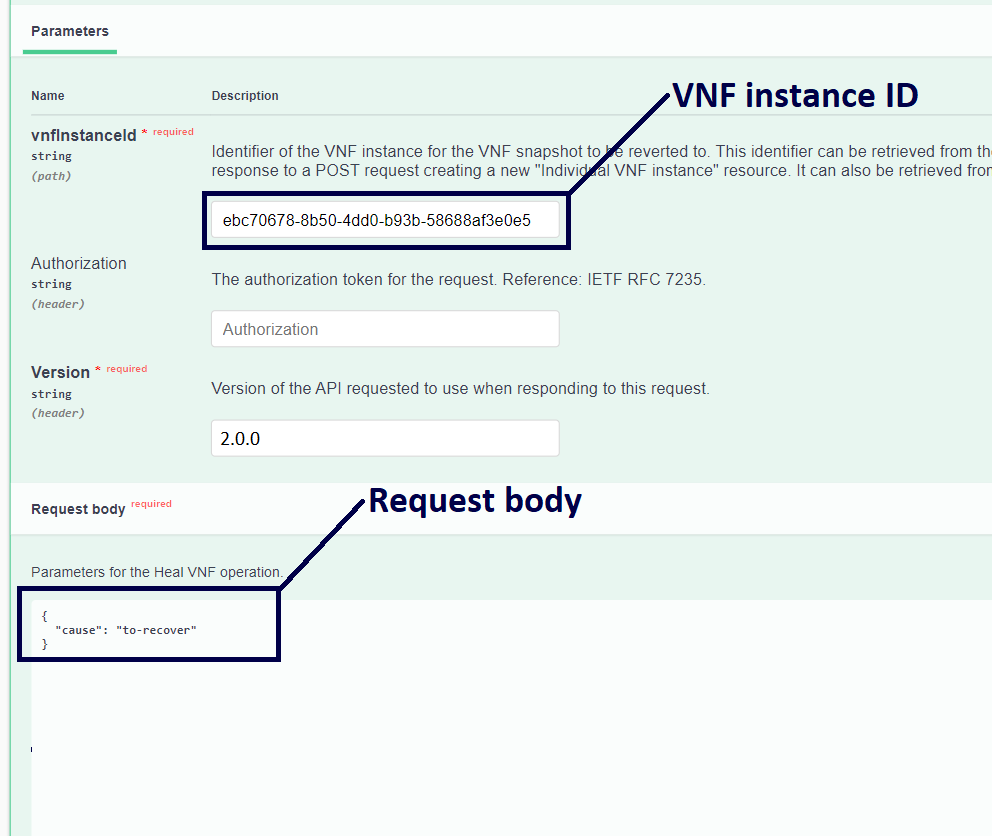
#### 5.4.2 POST /vnflcm/v2/vnf\_instances/{vnfInstanceId}/change\_flavour:

This POST method is used to change the deployment flavour for a VNF instance. VNF deployment flavour defines a specific deployment configuration for a VNF, and are described in the VNFDs. Provide vnfInstanceId of one of the “*INSTANTIATED*” VNF instances as a parameter and in request body provide the mandatoryattribute *newFlavourId (*ID of the desired flavour) and execute the method, as shown in Figure 23. A successful response will be received with the status code 202 indicating the request to change the VNF flavour has been accepted.

  
Figure 23

#### 5.4.3 POST /vnflcm/v2/vnf\_instances/{vnfInstanceId}/heal:

This POST method is used to heal a VNF instance resource in case of any failure. As Figure 24 illustrates, provide *vnfInstanceId* of one of the “*INSTANTIATED*” VNF instances as a request parameter and in request body provide *cause* attribute*, which* Indicates the reason why a healing procedure is required.

  
Figure 24

#### 5.4.4 POST /vnflcm/v2/vnf\_instances/{vnfInstanceId}/instantiate:

This POST method is used to instantiate a VNF instance. Instantiation is the process of deploying a VNF instance on the NFVI. VNFs will be instantiated with the Virtual Deployment Units (VDUs) as specified in the VNFDs. Provide *vnfInstanceId* of the “*NOT-INSTANTIATED”* VNF instance resource as a request parameter. In request body, provide the mandatory attribute *flavourId* (as defined in the VNFD) which describes a specific deployment flavour for the VNF, as illustrated in Figure 25.

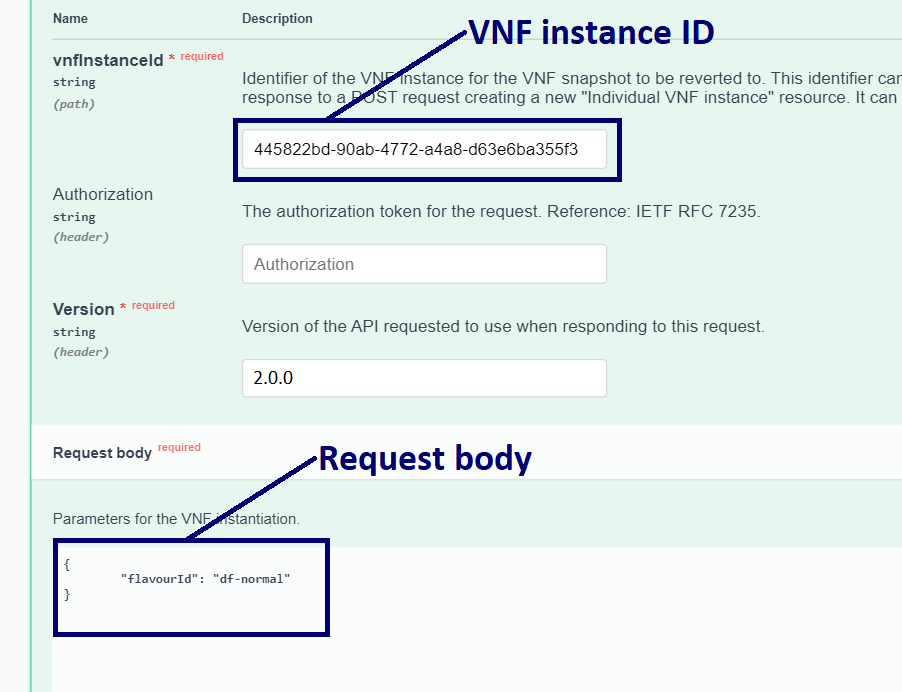
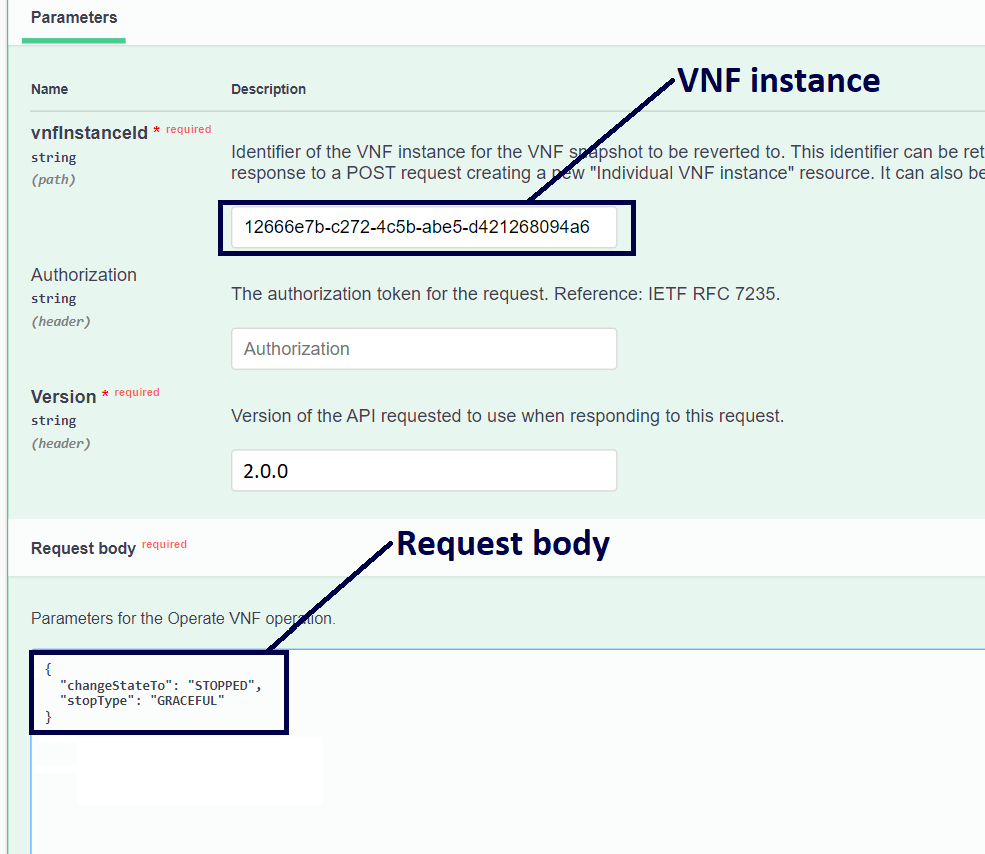


Figure 25

#### 5.4.5 POST /vnflcm/v2/vnf\_instances/{vnfInstanceId}/operate:

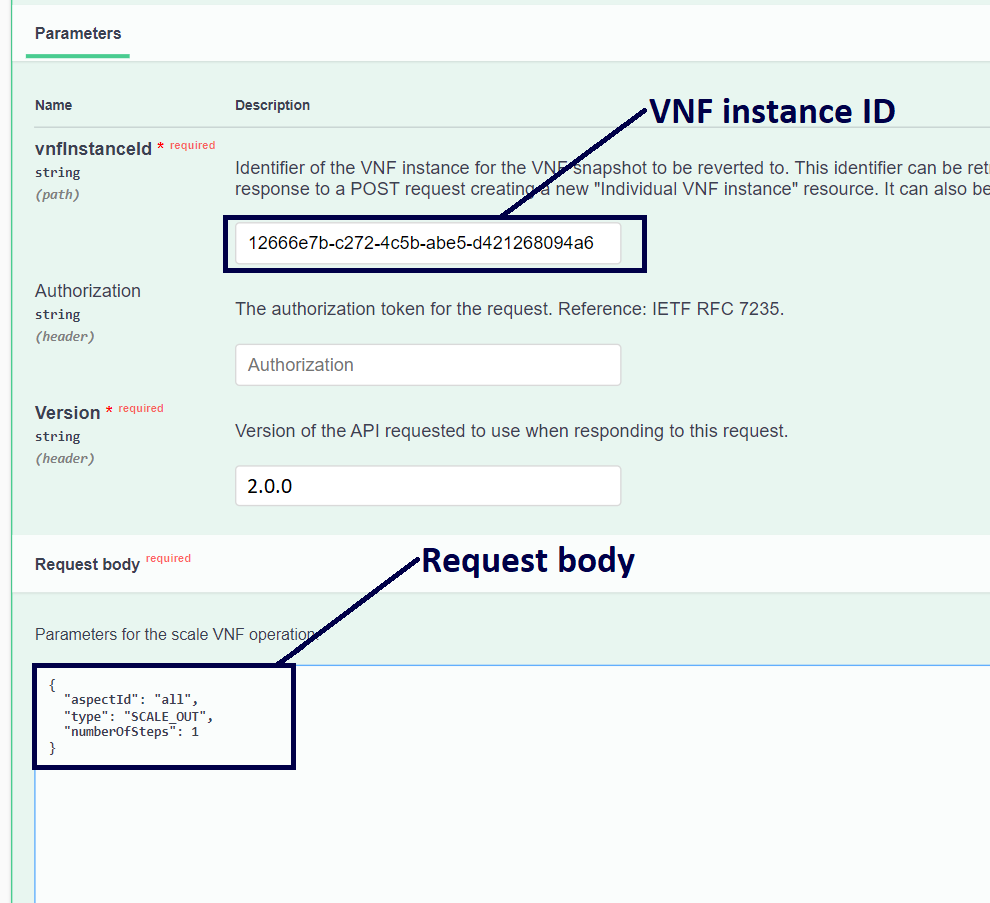
This POST method is used to change the operational state of a VNF instance resource. VNF instances can be in the STARTED (a VNF instance is up and running) or STOPPED (a VNF instance has been shut down) states. Operation to stop a VNF instance can be performed forcefully or gracefully, *GRACEFUL* means a VNF instance would be stopped after some specified time duration, and *FORCEFUL* means a VNF instance would be stopped instantly. For now, in the Emulator both attributes *GRACEFUL* and *FORCEFUL* stops a VNF instance gracefully. To execute this method, provide *vnfInstanceId* of one of the “*INSTANTIATED”* VNF instance resources as a request parameter and in request body provide the mandatory attribute *changeStateTo* to change the operational state of a VNF instance, as shown in Figure 26.

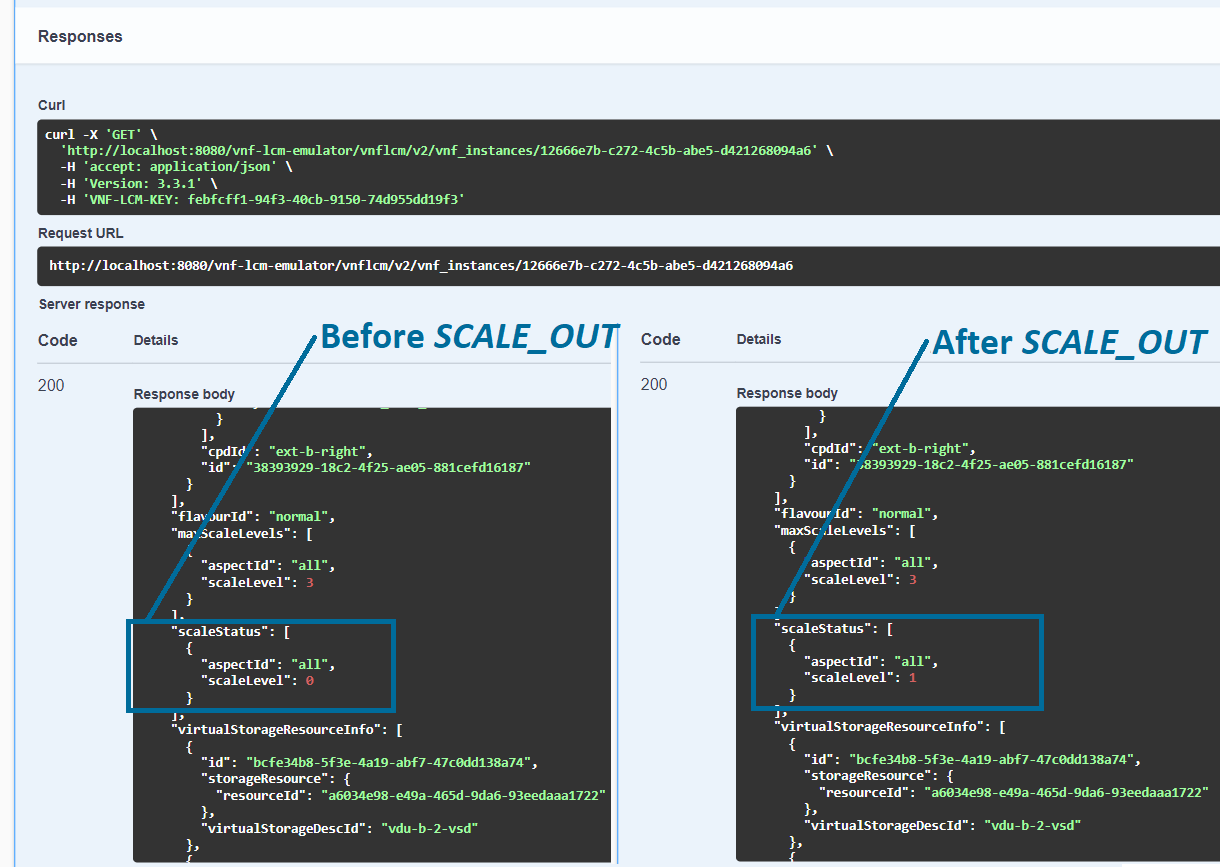
  
Figure 26

#### 5.4.6 POST /vnflcm/v2/vnf\_instances/{vnfInstanceId}/scale:

This POST method is used to scale a VNF instance resource incrementally. Scaling a VNF instance means reducing or increasing a service capacity by deleting or adding VNF Components (VNFCs). As a request parameter, provide *vnfInstanceId* of one of the “*INSTANTIATED”* VNF instance resources and in request body provide the mandatory attributes *aspectId*, *type*,and *numberOfSteps*. *aspectId* denotes ID of the target scaling aspect and is defined in the VNFD. The *type* attribute can be *SCALE\_OUT* or *SCALE\_IN. SCALE\_OUT* means adding additional VNFCs for that VNF instance while *SCALE\_IN* removes the VNFCs to reduce the VNF size. Scaling operations can only SCALE\_IN or SCALE\_OUT a VNF instance within the minimum and maximum scale levels for that VNF, which are defined in the VNFD.

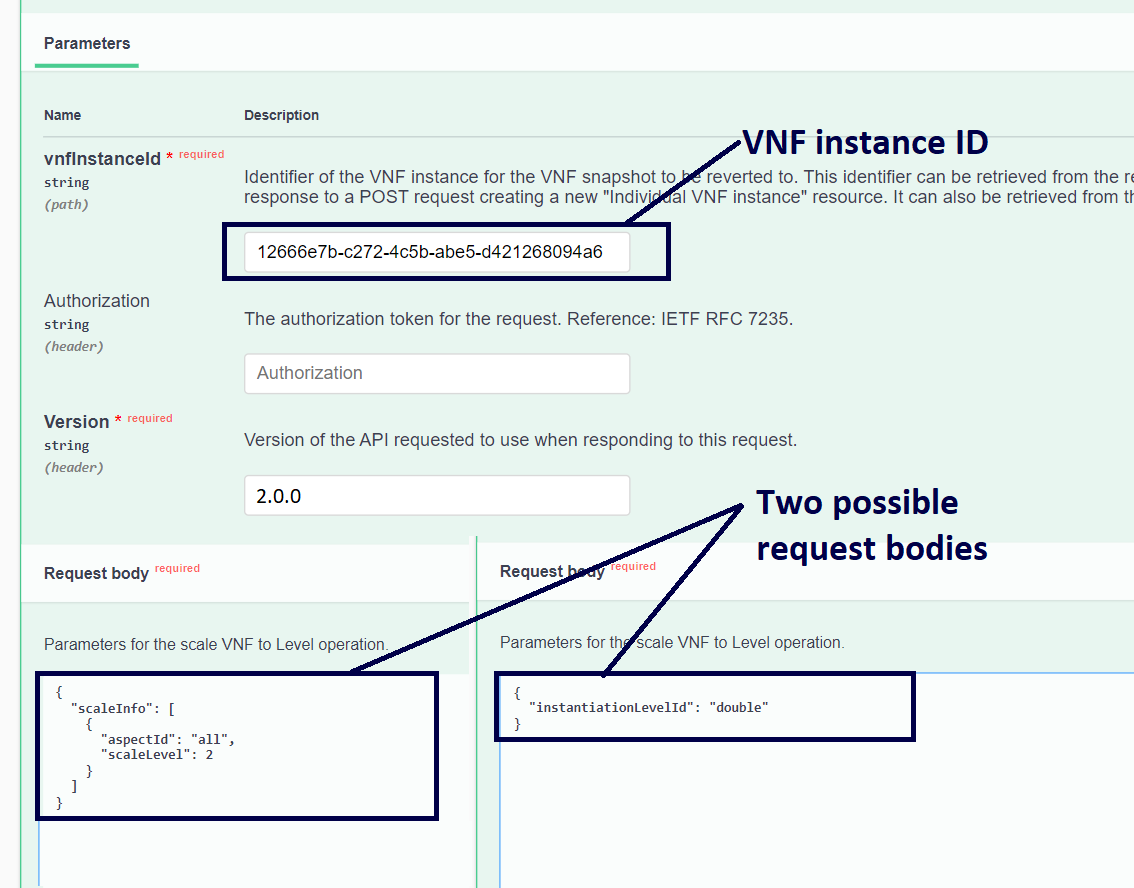
As an example, for a VNF created using the *vnfdId* equals to “ca07f422-ead1-4c9c-bdd8-71b22b820644”, a successful *SCALE\_OUT* operation with *numberOfSteps* set to 1 will create two additional VNFCs for that particular VNF instance. For each scaling aspect the information about the number of VNFC instances that need to be added or removed in a scaling step is defined in the VNFD. Figure 28 illustrates the successful SCALE\_OUT operation that was performed by the request shown in Figure 27. In this example, the VNF instance was instantiated with two VNFCs (scaleLevel: 0). The SCALE\_OUT operation adds two additional VNFCs and the scale level goes up one step (scaleLevel: 1).

  
Figure 27

Figure 28

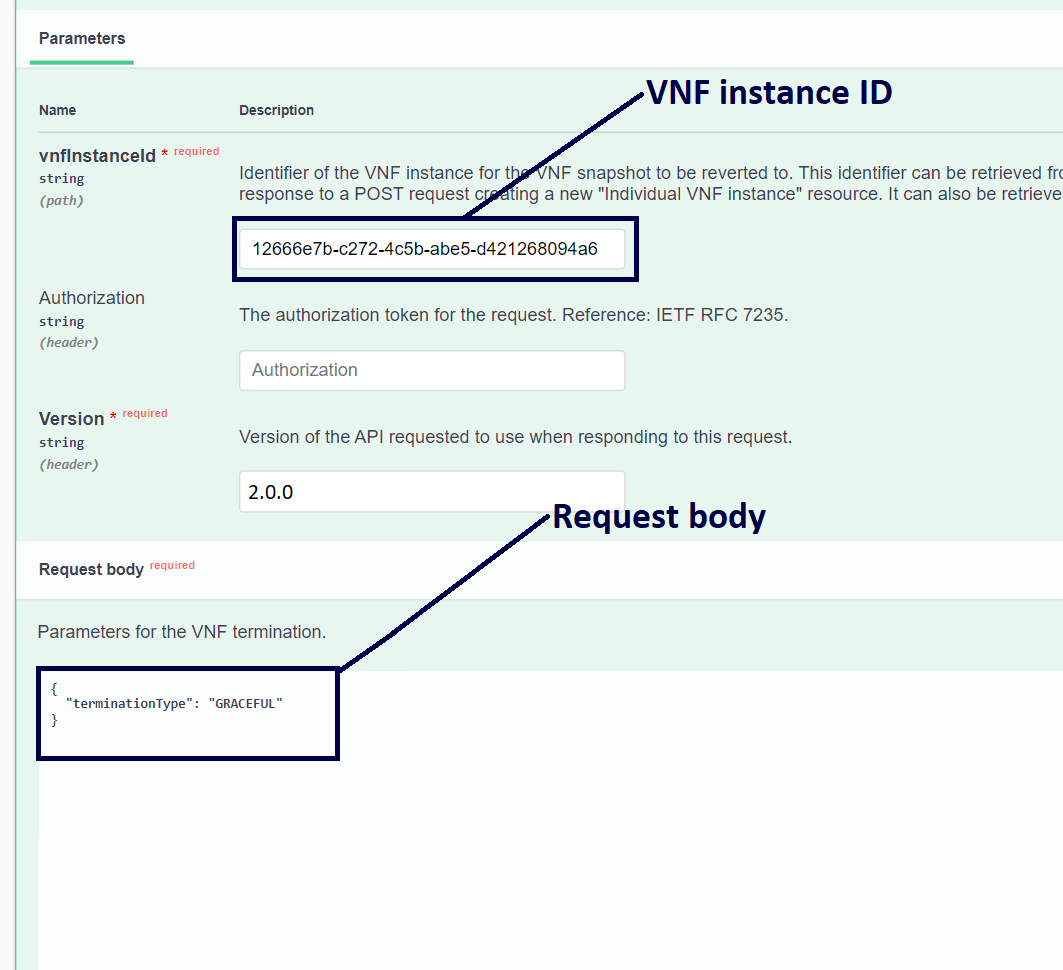
#### 5.4.7 POST /vnflcm/v2/vnf\_instances/{vnfInstanceId}/scale\_to\_level:

This POST method is used to scale a VNF instance resource to a target level. Provide *vnfInstanceId* of one of the “*INSTANTIATED”* VNF instance resources as a request parameter and for request body two approaches can be followed as shown in Figure 29. Attributes *scaleInfo* or *instantiationLevelId* can be provided in request body, *scaleInfo* indicates the target scale level to which the VNF is to be scaled for a given scaling aspect of the current deployment flavour whereas *instantiationLevelId* indicates the target instantiation level within the same deployment flavour to which the VNF is requested to be scaled.

  
Figure 29

#### 5.4.8 POST /vnflcm/v2/vnf\_instances/{vnfInstanceId}/terminate:

This POST method is used to terminate a VNF instance and release its virtualized resources. To execute this method, provide *vnfInstanceId* of one of the “*INSTANTIATED”* VNF instance resources as a request parameter and in request body provide *terminationType,* as shown in Figure 30. Termination of a VNF instance can be *FORCEFUL* or *GRACEFUL*. In *FORCEFUL* termination, the VNF instance resource will be shut down immediately and its resource will be released, and in *GRACEFUL* termination VNF instance would be terminated after some specific time. For now, in the Emulator both attributes *GRACEFUL* and *FORCEFUL* terminates a VNF instance gracefully.

  
Figure 30

### 5.5 VNF LCM Operations Occurrences:

#### 5.5.1 GET /vnflcm/v2/vnf\_lcm\_op\_occs:

This GET method is used to query status information about multiple VNF lifecycle management operation occurrences. Provide *Version* as a request parameter and execute the method to GET all of the VNF LCM operation occurrences.

#### 5.5.2 GET /vnflcm/v2/vnf\_lcm\_op\_occs/{vnfLcmOpOccId}:

This GET method is used to query the resource representation of an individual VNF LCM operation occurrence. Provide vnfLcmOpOccId as a request parameter and the response body will contain the resource representation. The *operationState* attribute of the individual VNF LCM operation occurrence conveys the information about the status of the associated LCM operation. This method can also be used to probe the operation state of an ongoing LCM operation by providing the *vnfLcmOpOccId* received in the “Location” header of the 202 response for that LCM operation.

6 Examples:

### 6.1 Instantiate a VNF instance:

**Step 1:** Authorize the session via the API key using the *POST /api\_key* method.

**Step 2:** Execute the *GET /emulator/vnfds* method to fetch all the available descriptors and note the ID of one of the available VNFDs.

As an optional step, you can also download a specific VNFD by providing its ID as a request parameter in *GET /emulator/vnfds/{vnfdId}* method. You can see the Flavour IDs of the supported deployment flavors and further information associated with these flavours such as instantiation levels, scaling aspects, etc., in the extracted file.

**Step 3:** In the request body of the *Post /vnflcm/v2/vnf\_instances* method, paste the VNFD ID from Step 2 in the *vnfdId* attribute and execute the method to create a new VNF instance resource based on a VNF package. Note the *ID* of the created VNF instance from the response body of the 201 Response.

**Step 4:** Now go to the *Post /vnflcm/v2/vnf\_instances/{vnfInstanceId}/Instantiate* method provide *vnfInstanceId* (from Step 3) and *version* (“2.0.0") in the request headers, and *flavourId* (from Step 2) in the request body to instantiate a VNF instance.

Note the *vnfLcmOpOccId* from the location header of the 202 Response.

**Step 5:** Use the *vnfLcmOpOccId* from previous step and pass it as a request parameteralong with the version in the *GET /vnflcm/v2/vnf\_lcm\_op\_occs/{vnfLcmOpOccId}* method to query status information about the Instantiate operation. Keep executing the request till the *opeartionState* attribute of the *VnfLcmOpOcc* object changes to COMPLETED.

**Step 6:** Once the operation is complete, provide the *vnfInstanceId* (from Step 3) as a request parameter in the GET /vnflcm/v2/vnf\_instances/{vnfInstanceId} method to verify successful instantiation. *instantiationState* and *instantiatedVnfInfo* attributes of the *VnfInstance* data model should be updated after the VNF goes into the INSTANTIATED state.