VNF LCM API Emulator

Software architecture and deployment documentation

*Authors (ETSI STF 598 Experts):*

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

*Pietro Piscione, Giacomo Bernini (Nextworks)*

1 Introduction

The purpose of this document is to describe the general software architecture and the deployment of the VNF LCM API Emulator designed, developed and tested within the STF-598 activity.

The VNF LCM API Emulator (for simplicity is called Emulator) is a software tool for emulating the lifecycle management of a Virtual Network Function. Details of the internal workflows, operations and data models related to the LCM of VNF Instances are available in the ETSI GS NFV-SOL003 V3.3.1 specification [1].

2 Software architecture

From a high-level architectural point of view, as depicted in Figure 1, the VNF LCM API Emulator is mainly composed of these software components, i.e. the Emulator API server, the VNF LCM API server, the Emulator Engine and the Databases (DB(s) in the image).

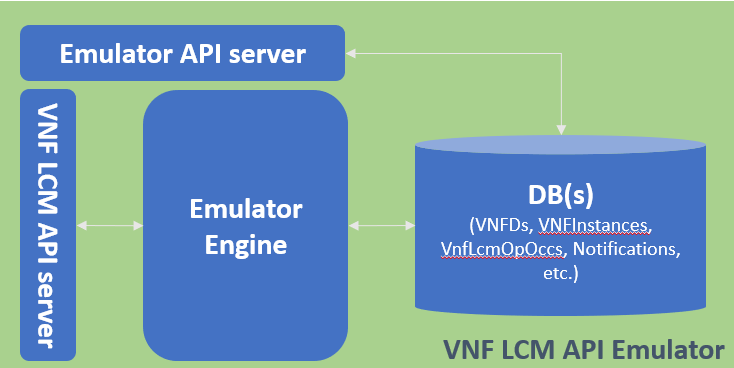


Figure 1 – High level architecture of VNF LCM API Emulator

The VNF LCM API Emulator exposes the whole set of functionalities through the Emulator API server and VNF LCM API server as REST endpoints.

The Emulator Engine implements the logic for realizing all the VNF LCM Operations, exposing most of the REST API in compliance with the SOL003 V3.3.1 specification. The DB(s) stores the related data model generated during the LCM of the VNFs and also information related to the user session management. Moreover, functionalities related to the authentication, management of the sessions and management of the VNF Descriptors are also part of the VNF LCM Emulator.

The Emulator Engine interacts with the Databases using a specific driver, capable of connecting and performing the classical CRUD (Create, Read, Update and Delete) operations. These operations are performed on the data models defined in the specification, i.e. VNF Instances, VNF LCM Occurrence Operations and so on.

Beyond the REST API in compliance with the SOL003 V3.3.1 specification, some custom REST APIs for the management of the VNF Descriptor and the user session management have been designed and implemented. The former are used for retrieving information about the available VNF Descriptors, while the latter for managing multiple sessions.

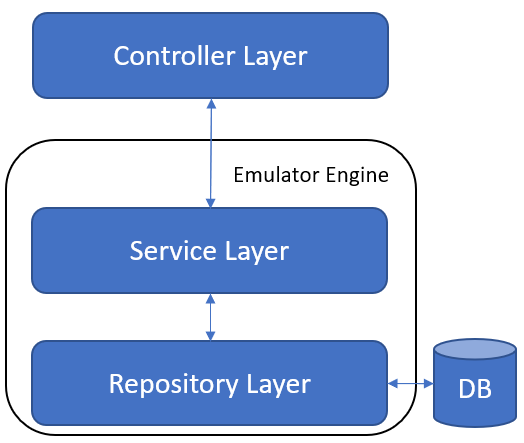


Figure 2 – Controller-Service-Repository pattern used for developed the VNF LCM API Emulator

As depicted in Figure 2, the pattern used for developing the VNF LCM API Emulator is composed of three layers:

* **Controller Layer**, the entry point of the requests where the body and header request, if any, are validated against the data model, e.g. checking if the mandatory attributes are available, checking if the data type is correct and so on.
* **Service Layer**, which realizes the whole engine logic. It takes the input from the Controller Layer, processes the request and the output is dispatched to the Repository Layer.
* **Repository Layer**, which is in charge of taking the input from Service Layer and then performing operations on the database, using the related driver.

This pattern allows having a clear distinction about the different operations and responsibilities of each layer performed in the VNF LCM API Engine. Moreover, a Presentation Layer is exploited for providing to the final user a graphical representation of the functionalities of the VNF LCM API Emulator. The principal responsibility of the Presentation Layer is to abstract and simplify the usage of the VNF LCM API Emulator.

From a workflow point of view, the user sends through the Presentation Layer a request to the Controller Layer, which validates the request. Then, the request is forwarded to the Service Layer, which processes the request. The repository Layer performs some operations on the database. Then, after the request is processed and the related information is stored, a response is returned to the requestor eventually. Figure 3 summarizes the described workflow.

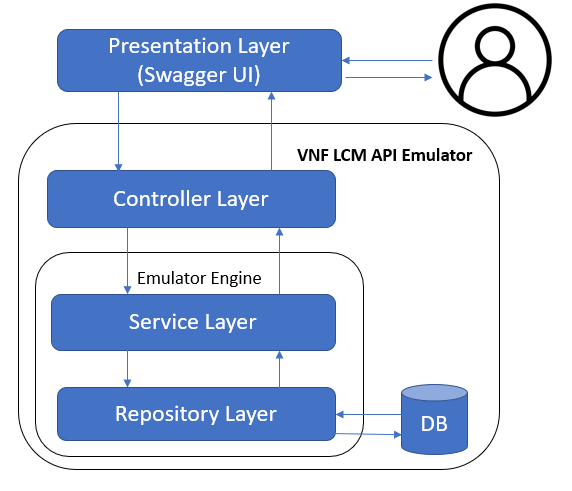


Figure 3 – High-level workflow of a request send to the VNF LCM API Emulator

The specific internal logic executed by the Service Layer strictly depends on the type of request and parameters. At the current state, the Presentation Layer is realized by a Swagger UI, which consists of a graphical representation of the list of implemented endpoints. The Swagger UI allows the user to select a specific endpoint, add the body and header request, send the request to the VNF LCM API Emulator for triggering the corresponding workflow. A generic HTTP REST client can interact with the VNF LCM API Emulator providing the request with the correct format.

2.1 Source code

The source code of the VNF LCM API Emulator is available in the corresponding repository [2]. The language and framework used in its development is Flask[3], a Python web Server Framework.

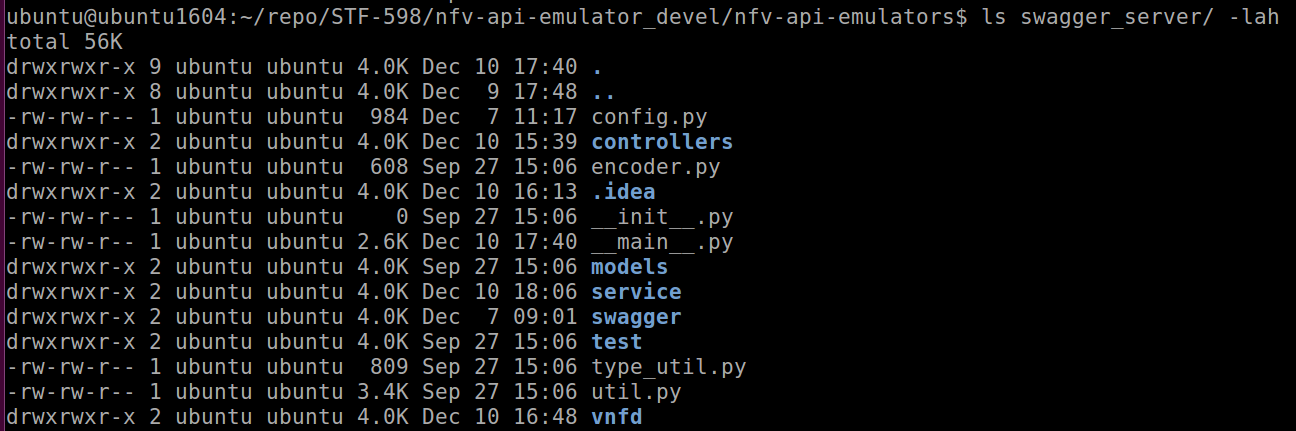


Figure 4– List of directories and files containing the source code of the Emulator

As depicted in Figure 4, the source code of the Emulator is available in the *swagger\_server* directory where the following subdirectories and files are available:

* *Controllers* directory, which contains the source code of the controller layer.
* *models* directory, which contains the python classes of the data models defined in the SOL003 specification.
* *service* directory, which contains the Service and Repository Layers, which implement the logic of the VNF LCM API Emulator and the interaction with the database, respectively.
* *swagger* directory, which contains the YAML file for the Swagger UI generation.
* *test* directory, which contains the tests. Not used for now.
* *vnfd* directory, which contains all the VNF Descriptor that are on boarded at the start-up of the VNF LCM Emulator.
* *Config.py,* configuration file.
* \_\_main.py\_\_ and *init.py,* used for starting the web server.
* The other directories and files are for minor purposes.

3 Deployment

In this section, the steps for configuring, deploying and checking the status of the VNF LCM API Emulator are described.

The VNF LCM API Emulator can be deployed using Docker containers. In particular, a docker-compose file is available for deploying the Emulator Engine with the REST APIs and the Database in two separated containers using the docker-compose tool [4].

Figure 5 depicts how the software components of the VNF LCM API Emulator are mapped to the two containers. The set of APIs externally exposed and the Emulator Engine are in the web server container, while the databases, using an instance of MongoDB, are in the DB container. Moreover, on the web server container, the generation of the Swagger UI can be enabled. At the current status, it is enabled.

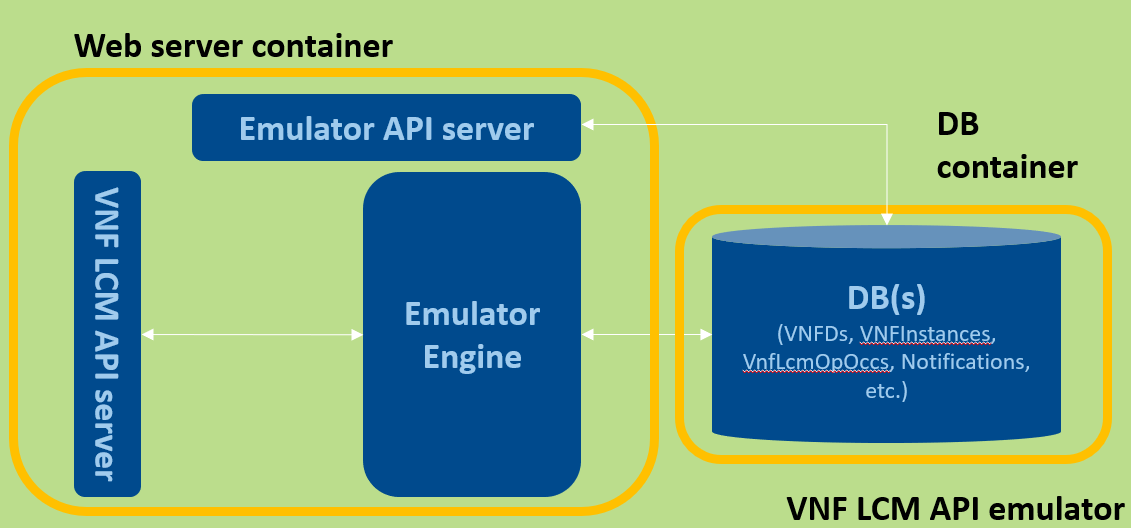


Figure 5 – Docker containers of VNF LCM API Emulator

To make the VNF LCM API Emulator up, running and usable a set of minimum hardware and software requirements must be satisfied.

3.1 Minimum hardware requirements:

* 1GB RAM
* 2CPU
* 20GB HD

Such minimum hardware requirements make the VNF LCM API Emulator properly work with a few users interacting with it. If the number of users starts to consistently increase, the it may affect the performance of the VNF LCM API Emulator itself. Increasing the hardware requirement should be taken into account if a consistent number of users is predicted.

3.2 Software requirements:

* Docker version 20.10.7, build f0df350
* docker-compose version 1.29.2, build 5becea4c

3.3 Web server and containers configurations

Once all the aforementioned hardware and software requirements have been satisfied, the source code of the VNF LCM API Emulator must be cloned from the corresponding repository.

git clone https://forge.etsi.org/rep/nfv/nfv-api-emulators/

As an optional step, the VNF LCM API Emulator can be configured. Table 1 and Table 2 describe all the configurations that can be set in the *config.py* and the docker-*compose.yaml* files, respectively.

|  |  |  |
| --- | --- | --- |
| **Configuration** | **Description** | **Example** |
| PORT | Port the web server is listening to | 8080 |
| SPEC\_DIR | Directory of the Open API Specification for the Swagger UI generation | ./swagger |
| SPEC\_FILENAME | Open API Specification file name for the Swagger UI generation | swagger.yaml |
| MAX\_COUNT\_API\_KEY | Maximum number of API KEY. It indicates how many parallel sessions can be simultaneously managed. | 100 |
| EXPIRATION\_TIME\_API\_KEY\_MIN | Expiration time (in minutes) of the API KEY since last usage. | 1440 |
| VNF\_LCM\_EMULATOR\_DB\_NAME | The of the database of VNF LCM Emulator | vnf\_lcm\_emulator\_db |
| FLASK\_ENV | Environment of Flask server. | development |
| DEBUG | Boolean value for setting the web server in bug mode or not | True |
| MONGO\_DB\_URL | URL of database | mongodb |
| MONGO\_DB\_PORT | Port of the database | 27017 |
| MONGO\_DB\_USER | Username to access the database | user |
| MONGO\_DB\_PWD | Password to access the database | password |

Table 1 – Web server configuration

|  |  |  |
| --- | --- | --- |
| **Configuration** | **Description** | **Example** |
| MONGO\_INITDB\_ROOT\_USERNAME | Username of MongoDB | user |
| MONGO\_INITDB\_ROOT\_PASSWORD | Password of MongoDB | password |
| ports | Port mapping of the MongoDB and the web server | 27017:27017  8080:8080 |
| max-file | Maximum number of log files of each container | 5 |
| Max-size | Maximum size of a single log file | 20m |

Table 2 – Containers configuration

3.4 Deployment through docker-compose

Before building and running the Docker containers, please make sure that the ports where the web server and the MongoDB instance are running are not used by other either processes or services.

To build and run in the background the VNF LCM API Emulator containers:

sudo docker-compose build; sudo docker-compose up –d;

The final output is depicted in Figure 6:

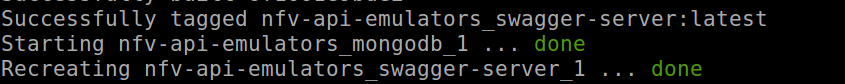
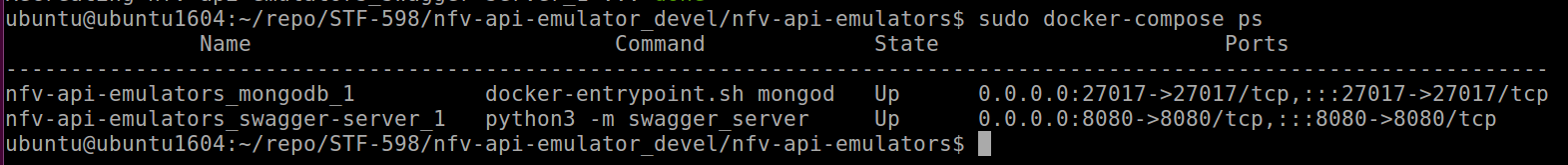


Figure 6 – Output of sudo docker-compose up –d command

As soon as the VNF LCM API Emulator starts, it stores the VNF Descriptor available into vnfd directory into the corresponding database. At this point, the VNF API LCM Emulator is up and running. Its status can be checked through the following command:

sudo docker-compose ps

The output of such command is depicted in Figure 7.

Figure 7 – Output of sudo docker-compose ps command

At this point is possible to use the VNF LCM API Emulator as described in the User Guide documentation.

3.5 Logs

During the usage of VNF LCM API Emulator, both web server and MongoDB containers produce logs about their activities. The logs of the VNF LCM API Emulator containers can be obtained in two ways:

1. Using the following command: sudo docker-compose logs -f

The -f option can added for seeing the logs while using the Emulator.

1. Looking at the file located at /var/lib/docker/containers/<container\_id>/<container\_id>-json.log

3.6 VNF Descriptor on boarding procedure

The VNF Descriptors of the Emulator are available into the *vnfd* directory. At the current state, both Descriptors in compliance with the SOL001 and SOL006 specification can be retrieved from the Emulator through the Swagger UI.

The descriptors are added at the start-up of the VNF LCM API Emulator containers. In particular, the SOL001 and SOL006 subdirectories contain the Descriptors in compliance with SOL001 and SOL006 specification, respectively.

However, there are two different procedures for adding a new VNF Descriptor:

1. Putting one or more descriptors in the correct *vnfd sub*directory.

In the SOL006 case, the VNFD is contained in a single file, hence it can be placed within SOL006 subdirectory.

In the SOL001 case, a directory which name is the identifier of the VNF Descriptor must be created. In such directory, all the files composing the VNF Descriptor must be placed.

After to have copied the VNF Descriptor file(s) in the correct subdirectory, both containers must be restarted and all descriptors are on boarded.

1. Adding manually in the database. This procedure works only for SOL006 Descriptors. The procedure is the following:
   1. Access the mongoDB container through the following command:

docker exec -it <mongodb\_container\_id> mongo admin -u admin -p password

admin\admin credentials are for example purpose only.

* 1. Access the VNF Descriptor database

use vnf\_lcm\_emulator\_db;

* 1. Insert the VNF Descriptor through the following command:

db.vnfd\_coll.insert({<vnf\_descriptor>})

The output is something like in the image depicted in Figure 8.

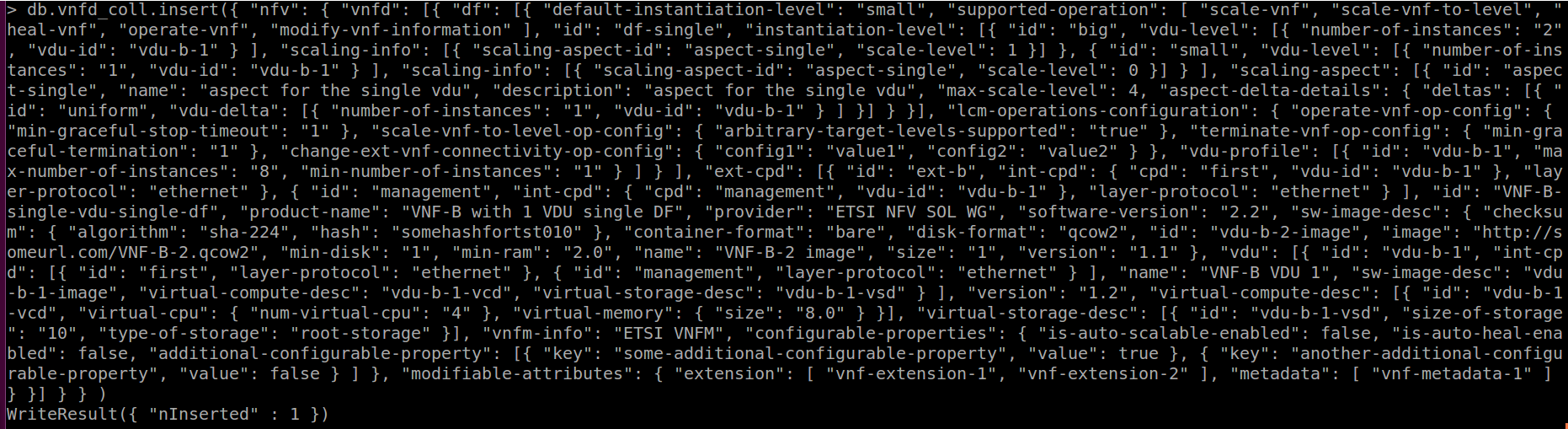


Figure 8 – Example of SOL006 VNF Descriptor inserting using the MongoDB command line

Please make sure that the specification key at the root of the JSON has the SOL006 value. Moreover, please note that following this procedure, makes the VNF Descriptor available into the database only. Restarting both container might delete the VNF Descriptor available into database only. It is advised to add the Descriptor both on the database and as a file(s) as well.

**References**

[1] **ESTI GS NFV-SOL003 v3.3.1** <https://docbox.etsi.org/ISG/NFV/open/Publications_pdf/Specs-Reports/NFV-SOL%20003v3.3.1%20-%20GS%20-%20Or-Vnfm%20RESTful%20%20protocols%20spec.pdf>

[2] **NFV API Emulator repository** https://forge.etsi.org/rep/nfv/nfv-api-emulators/

[3] **Python Flask framework** https://flask.palletsprojects.com/en/2.0.x/

[4] **Docker-compose** https://docs.docker.com/compose/