SEVENTH FRAMEWORK PROGRAMME

TNOVA

NETWORK FUNCTIONS AS-A-SERVICE OVER VIRTUALISED INFRASTRUCTURES

GRANT AGREEMENT NO. 619520

Deliverable D5.1

Network Function Store

Editor Nicolas Herbaut (Viotech)

Contributors: Aurora Ramos, Javier Melián (ATOS), Enzo Figini, Paolo Comi (Italtel), Yacine Rebahi (Fokus)

Version 1.0

Date 30/10/2015

Distribution PUBLIC (PU)

Executive Summary

In T-NOVA, the Function Store is the software component responsible for maintaining a repository of (1) the binaries composing each Virtual Network Function and (2) their metadata files.

It is not intended to have any user-facing interface, but it is designed to publish APIs that other T-NOVA components will use to perform CRUD (Create-Read-Update-Delete) operations on the resources it stores.

First of all, it allows Network Function Developers to securely upload their Network Functions along with other metadata files needed for the configuration through the T-NOVA Market Place Dashboard

Then it allows T-NOVA Marketplace Brokerage Module to retrieve Network Function metadata enabling service advertisement, request and brokerage/trading.

Finally it allows T-NOVA Orchestrator to retrieve VNF binaries and their metadata to onboard them onto the platform. It is also responsible to notify the Orchestrator in case a VNF is updated or deleted.

The NF Store has been developed in the WP5 “Network Functions” work-package of the project. It gets its inputs from WP3 “Orchestrator Platform” and from WP6 “T-NOVA Marketplace for which it exposes its API.

Table of Contents

Index of Figures

Index of Tables

1. INTRODUCTION

The NF Store is mainly a repository for the VNFs' software images and their metadata descriptions.

Figure 1 below shows a high level architectural description of the relationships of the NF Store and VNFs with the other elements of T-NOVA architecture, namely the Orchestrator and the Marketplace modules.

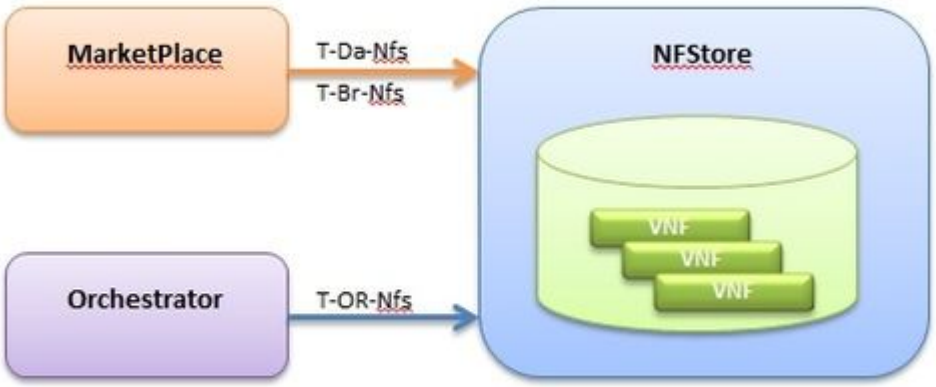

Figure 1. Interfaces with other T-NOVA components

Software developers can upload VNF images into the NF Store along with metadata descriptor containing both technical and business related information, such as business description of the cost of using such VNF.

The description of the VNFs in the NF Store is made available to the T-NOVA Marketplace. Whenever a VNF is chosen for being part of a service, the T-NOVA Orchestrator will get the VNF image for deploying and executing it over the virtualized execution environment.

2. NETWORK FUNCTION STORE

2.1. Requirements

The main requirements for the Network Function Store are summarized here.

2.1.1. General requirements

1. The NF Store is the repository for the VNF images and its metadata
2. A VNF can be composed of x VM images and one metadata descriptor
3. The VNF metadata descriptor and all the VM images composing a VNF shall be correlated, i.e. associated to a unique identifier (VNF-id)
4. The VNF-id is part of the VNF metadata descriptor.
5. A VNF can be versioned

2.1.2. Functional requirements

1. The NF Store informs the orchestrator whenever a VNF is added to or removed from the repository
2. The NF Store supports the following interfaces: T-Da-Nfs (Interface to the DashBoard, T-OR-Nfs (Interface to the Orchestrator)
3. T-DA-NFS shall allow to upload the VNF metadata descriptor and each VNF VM image
4. T-DA-NFS shall allow removing a VNF. Then, all VNF components are deleted from the NF Store
5. T-OR-NFS shall allow to download the VNF metadata descriptor and each VNF VM image
6. The operations over the supported interfaces shall be allowed upon authentication and authorization

2.1.3. Non-Functional Requirements

1. The NF Store shall provide storage capacity for a reasonable number of VNFs and VM images. These numbers belong to the set of NF Store configuration parameters that shall be provided at NF Store deployment time
2. The NF Store shall not introduce additional performance constraints beyond the available bandwidth and throughput

3. ARCHITECTURE

3.1. High Level Architecture

High level breakup of the NF Store includes:

- NFS repository : contains the VNF images
- NFS database : contains the VNF metadata and all data needed by the Web application
- NFS Web application: application logic governing the repository and the interactions over the exposed interfaces and implementing the Network Function Store functionality
- NFS interfaces: provides interfaces for interacting with the NF Store
- NFS Manager : application to manage the NF Store

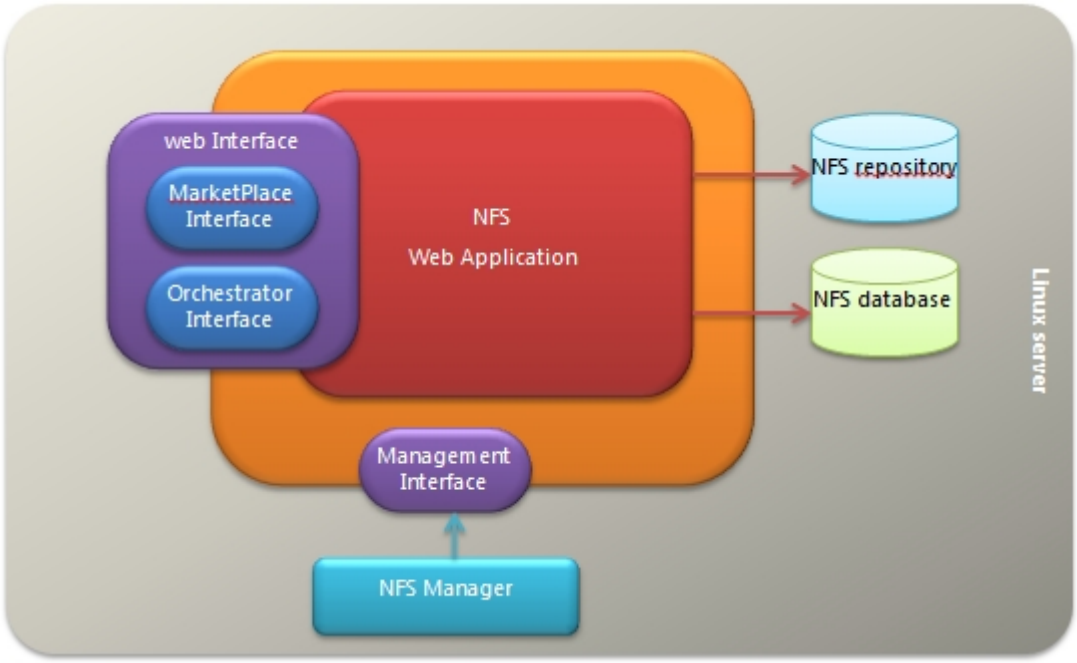

Figure 2. High Level Architecture

The NF Store provides REST interfaces to the orchestrator (T-OR-NFS) and marketplace (T-DA-NFS and T-BR-NFS); an additional shell interface is provided by NFS manager and is used to manage the NF Store service as standard linux service.
The Network Function Store function is developed like a web application running into an application server; the chosen server is Apache TomEE, an all-Apache Java EE 6 Web Profile certified stack built above the Apache Tomcat Servlet Container

The NF Store application implements the interfaces to repository, database and web interface; it is also responsible for concurrent operations that are performed (CRUD operations). In addition, it delegates some traits of its responsibility (like authentication and authorization access) to an external AA module. Security mechanisms can be adopted for data exchange over these interfaces.
The NFS database is used to archive information about VNF (metadata files); VNF image files are saved into a file system used as NFS repository.

3.2. Low Level Architecture

3.2.1. Web Application

The Web application architecture is based on standard JSR-318 Enterprise Java Bean (EJB)

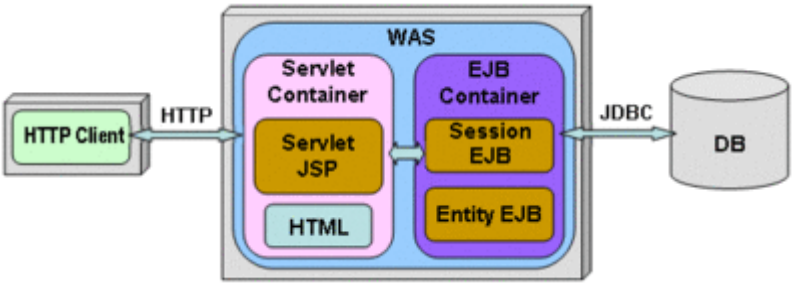

Figure 3 Web application Architecture

Stateless EJB are used to realize the NF Store front-ends giving to users the possibility to use the interfaces also when VNF image operations (upload, download, modify) are active; this type of operations can require a lots of time depending from the length of the image file so this feature prevents to have the interface locked and unusable for many time.

Singleton EJB is used for the NF Store Manager Service, realizing the synchronization center of the web application. When an operation requires executing an operation in exclusive way this should be developed inside this EJB.

3.2.2. Database

The database layers is provided by H2, an open source Java SQL database with small footprint used in embedded mode because no other actors needs to use it; the Database is saved as a standard file into the Linux file system.

An H2 console application is available to operate on DB using a web interface (that needs to be configured on TomEE) mainly for maintenance purposes.

The database model is made using JPA and the connection to DB use standard JDBC interface (statically configured on TomEE) giving us the possibility to change the DB without changing the web application code.

The main entities of the database model are:

- **VNFDescriptor** - used to save information of the VNF metadata. The VNF descriptor is saved in String form inside this entity
- **VNFFile** - used to model a VNF image file (that is saved on file system) giving us the possibility to describe a one-to-many relation from VNF images and VNF.

The other entities in the database are used only for Brokerage interface to improve search performances instead of looking inside all saved VNF descriptors.

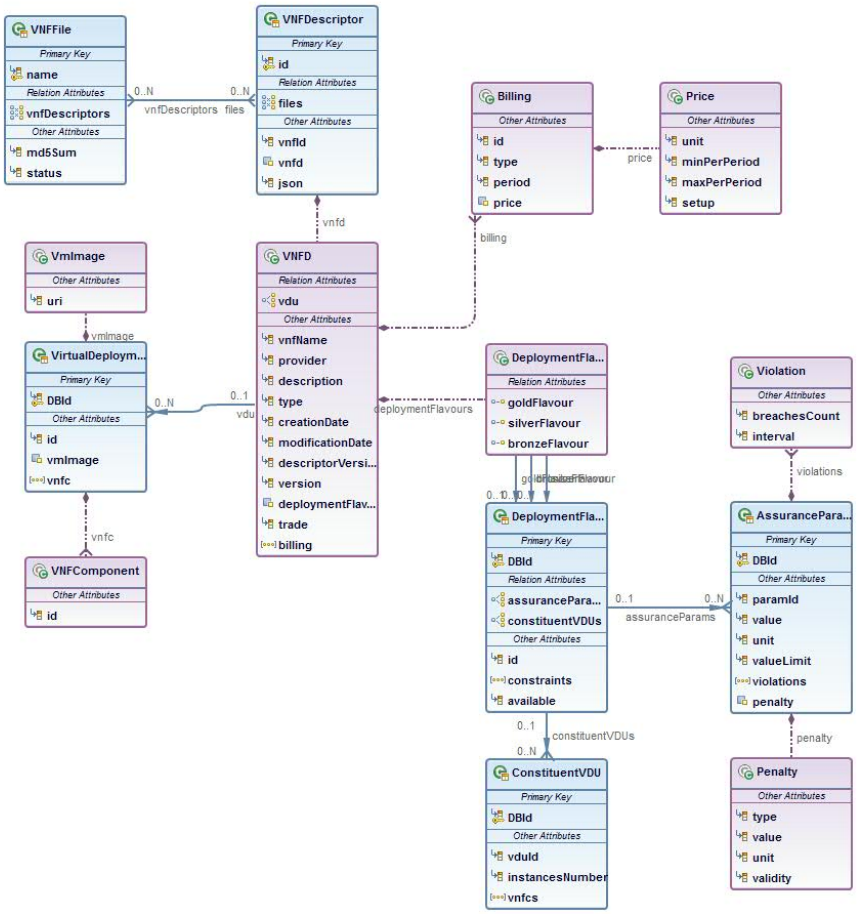

Figure 4. Relational Schemas

3.2.3. NFS service

3.2.3.1. Configuration file

In order to manage easily the NF Store, some bash shell script have been developed.

A configuration file *nfs.conf* is available to override the default values of the following configuration parameters.

Table 1 Configuration options of nfs.conf

Variable	Default Value
Monitor log level	Notice
store path	/usr/local/store
NFS protocol	https
NFS address	any address
NFS port	8443
orchestrator protocol	path
orchestrator host	api.t-nova.eu

orchestrator port	443
orchestrator path	/orchestrator

3.2.3.2. nfsMonitor script

This bash script is responsible to manage the Apache server TomEE.

Possible Operations are:

- **Start**: this operation is used to start the server with the nfs application inside. At the beginning, a check is done to verify that server is not running, then the server is started after building the server configuration file with the interface to be used (protocol/address/port) and setting the environment variables. Server and application are periodically monitored to check that all the operations are working well; in case of failures, the server is stopped and restarted. This operation need to be run in background mode since the script doesn't exit.

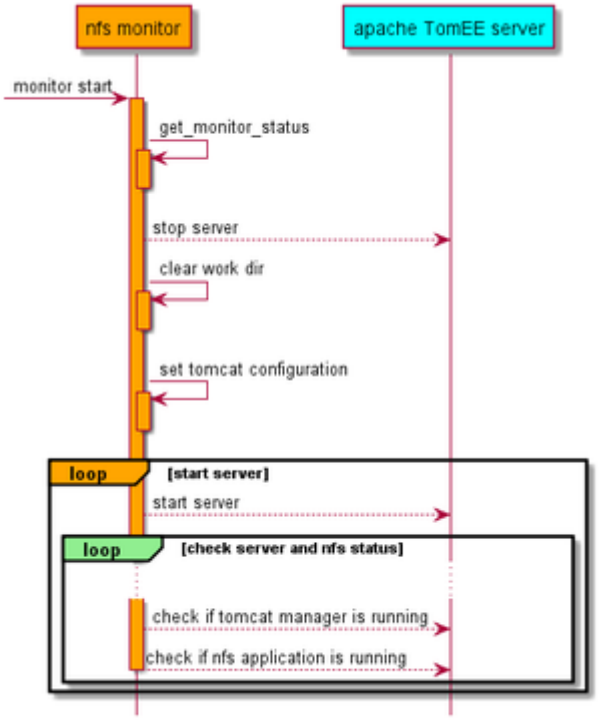

Figure 5. Starting the NF Store

- **Stop**: this operation is used to stop the monitor and the server. A check is done to see if monitor is active and server is running, then they are stopped verifying if all is done. In case of failure a try using SIGKILL is done.

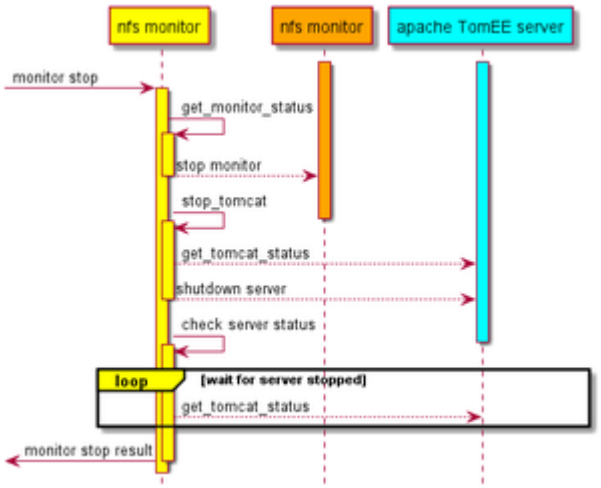

Figure 6. Stopping the NF Store

- **Status** : using this operation makes it possible to check the running status of the server and its deployed application. A report is send to standard output reporting the status of monitor, server, manager application and NFS application.

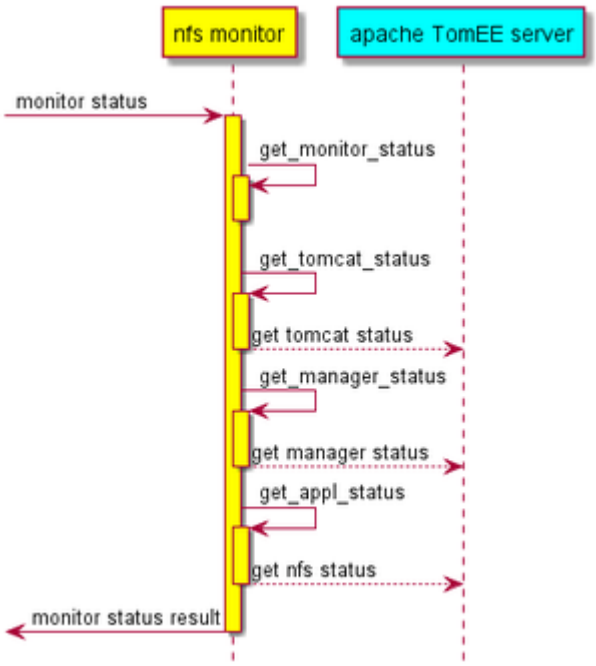

Figure 7. Retrieving the status of the NF Store

3.2.3.3. nfs script

This bash script gives the possibility to manage the NF Store as a standard linux SysVinit service.

Available operations are:

start : start of nfs service checking if it is started using the nfsMonitor

stop : stop of nfs service using the nfsMonitor

restart : restart of nfs service using the nfsMonitor

- status : retrive of nfs service using the nfsMonitor

3.3. Functional description

The Dashboard front-end provides the T-DA-NFS interface, used by software developers to publish their VNFs.

The supported operations are:

- Publication
- Modification
- Removal
- List

The Orchestrator front-end provides the T-OR-NFS interface, used mainly to recover the available VNFs.

The supported operations are:

- Modification
- Retrieve
- List

Each time an HTTP request is received from the TomEE interface, the server makes a check to verify that the URL is mapped to a deployed Web application.

In case of success, the request is sent to the correct application where the interface is realized with an EJB.

In order to synchronize some parts of the operations and in case of concurrent access from users, an internal Singleton EJB is used; this is needed in particular when a long running operation on an image file is occurring.

3.3.1. Publication

The VNF is composed by a VNF descriptor and several VM images. The VNF descriptor contains the VNF metadata information's and then the list of VM images composing the VNF.

For each VM image, a file containing the MD5 image checksum should be uploaded by the Network Function developer; the name of the file should be the same of the image with added the suffix md5.

Table 2 An image file deployed with its md5 hash

file type	Name
image file	vSBC.img
md5 sum file	vSBC.img.md5

Different interfaces are available to publish VNF descriptor and image files.

Each time a VM file or a descriptor is uploaded, the NFStore checks if the VNF is complete, namely that (1) the VNF descriptor is available, (2) all images have been uploaded with their md5 sum files, (3) and the MD5 values computed by the NF Store match the MD5 values uploaded by the VNF developer. In case these conditions are overcome, a communication to Orchestrator using the T-OR-NFS interface is sent to advice that a new VNF is available into

NFStore. If more VNFs share the same image file, the orchestrator receives a notification for each published VNF.

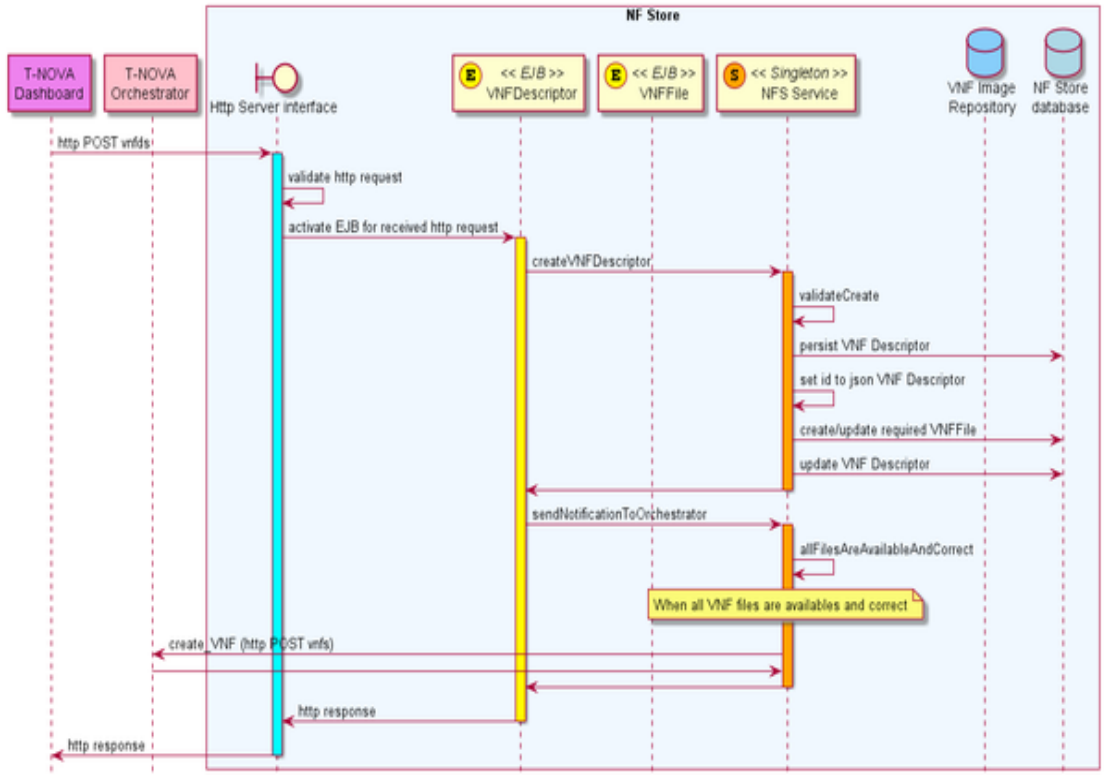

Figure 8. Upload VNF descriptor

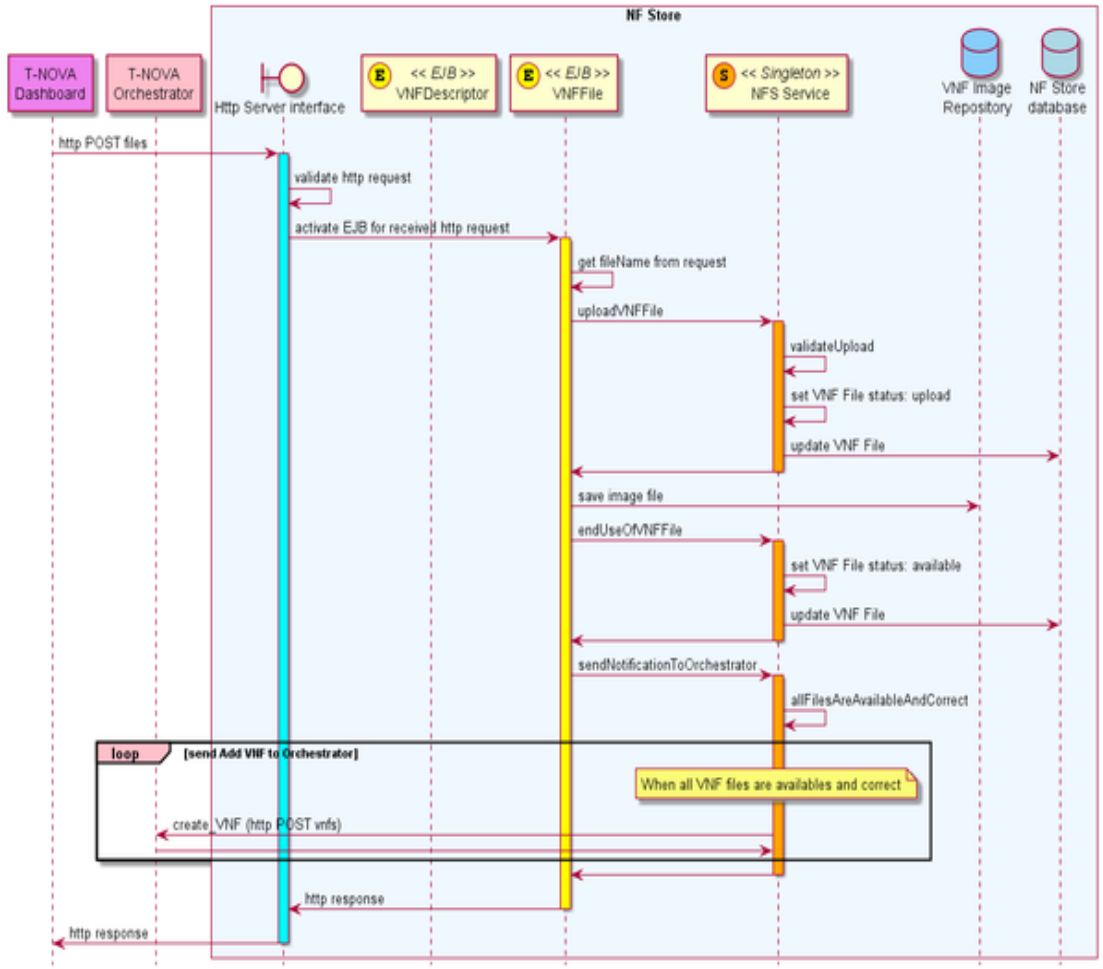

Figure 9. Upload VNF file

3.3.2. Modification

Changes can be done on VNF descriptor and/or VNF images. Like in publish operation the order is not important and when the upload of a VNF is complete, a notification to Orchestrator using the T-OR-NFS interface is sent informing that the VNF was changed.

If the image is shared by several VNF, several notifications are sent to the Orchestrator, one for each VNF.

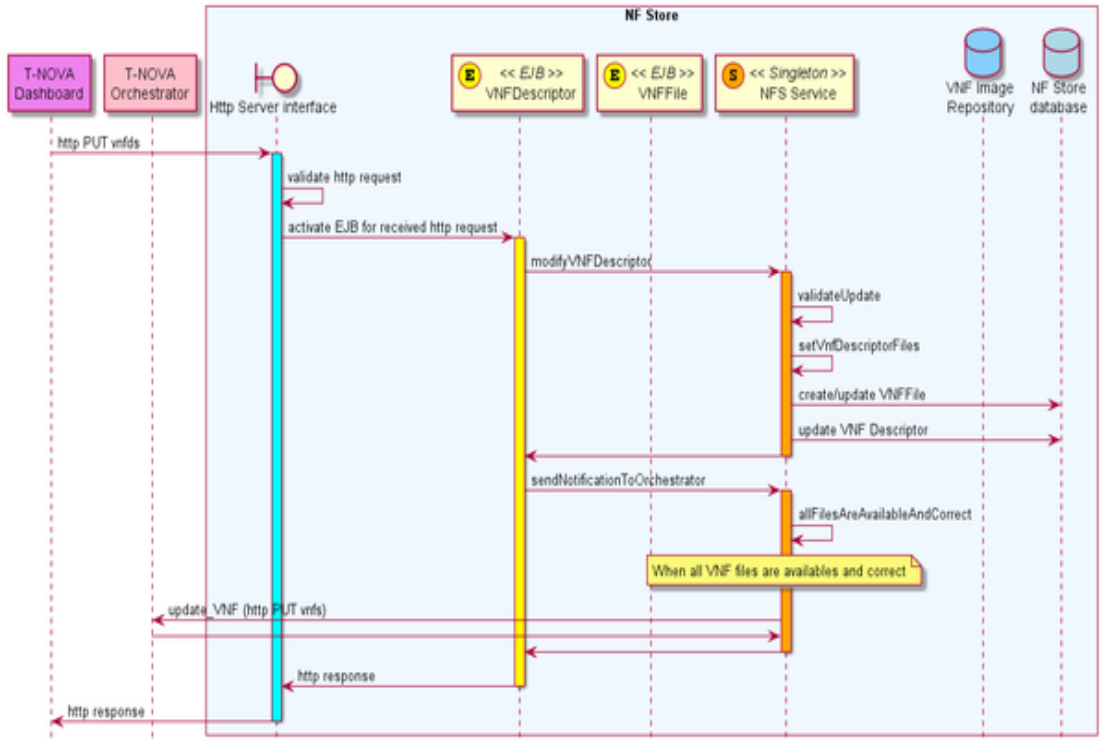

Figure 10. Update VNF descriptor

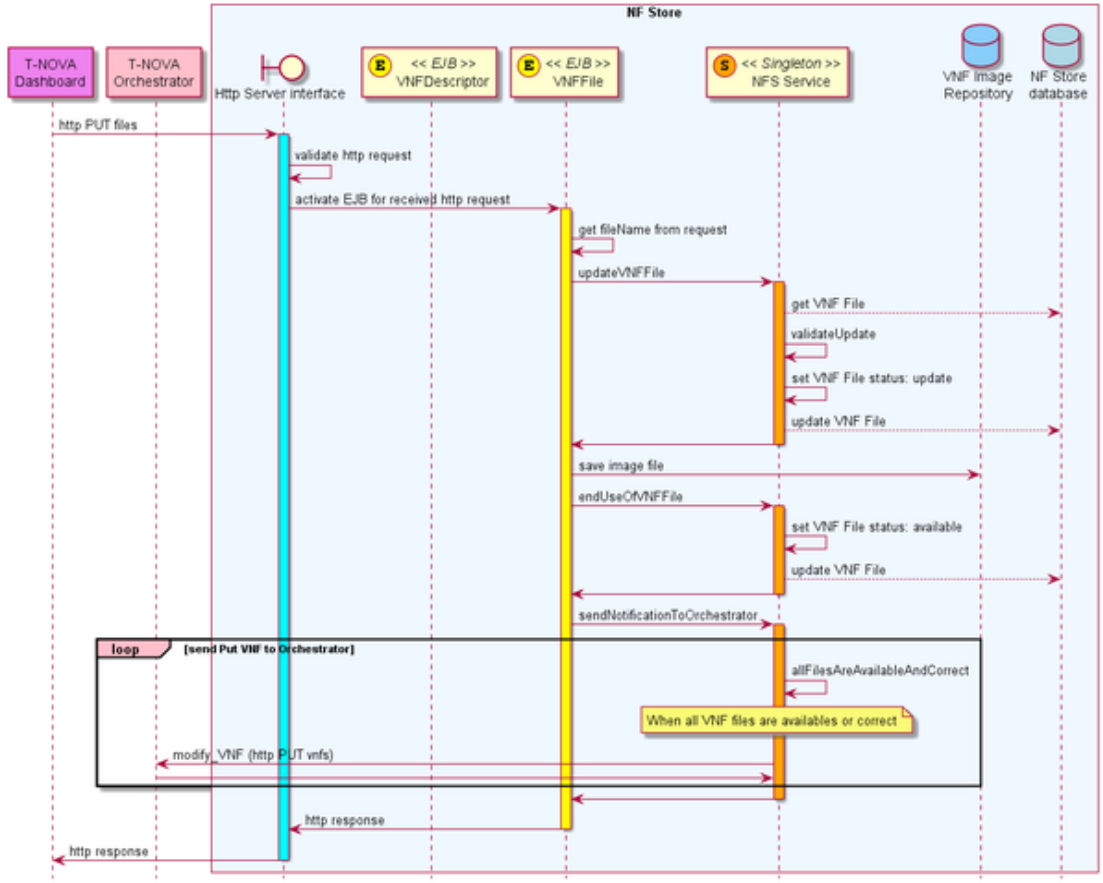

Figure 11. Update VNF file

3.3.3. Removal

Each time a file used by a VNF is removed, the VNF is not usable anymore. In this case the NF Store send a notification to the Orchestrator though the T-OR-NFS interface to inform that a VNF is no more available.

The same thing happens when a VNF descriptor is removed. If more than one VNFs share the same image file, when the file is removed, an Orchestrator notification is sent for every VNF using that file.

When a file is removed, but it is required by a single VNF, the binary file is deleted from the repository but his description into DB is only updated with the file status to *unavailable*.

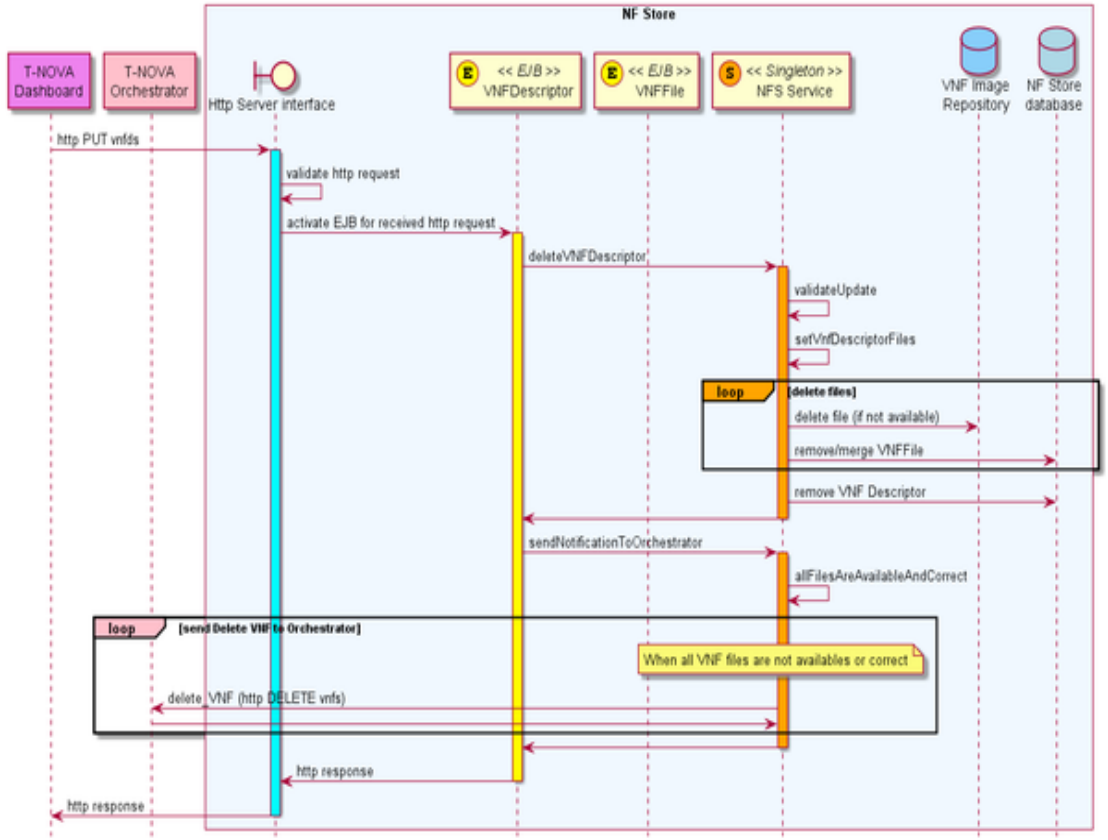

Figure 12. Remove VNF descriptor

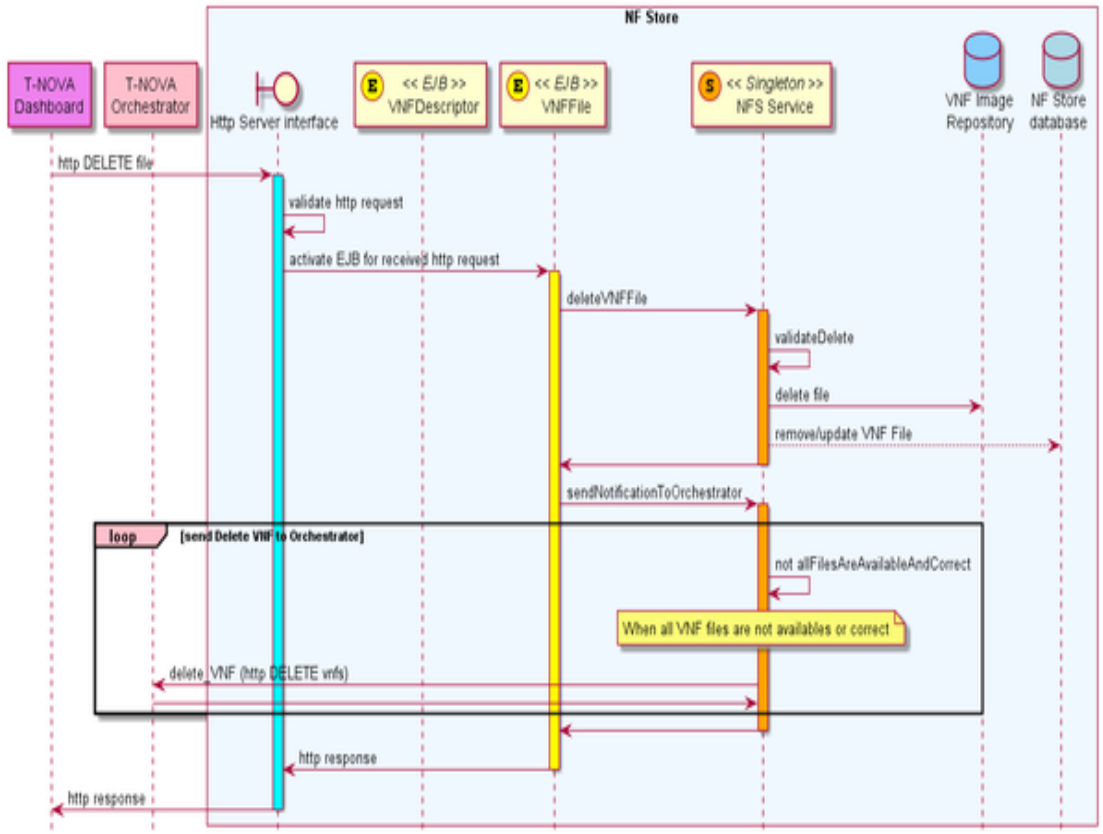

Figure 13. Remove VNF file

3.3.4. Retrieval

Retrieving an image files is the main operation required by the Orchestrator and is needed to start a VNF.

The retrieval of a VNF descriptor is mainly needed by the Marketplace dashboard.

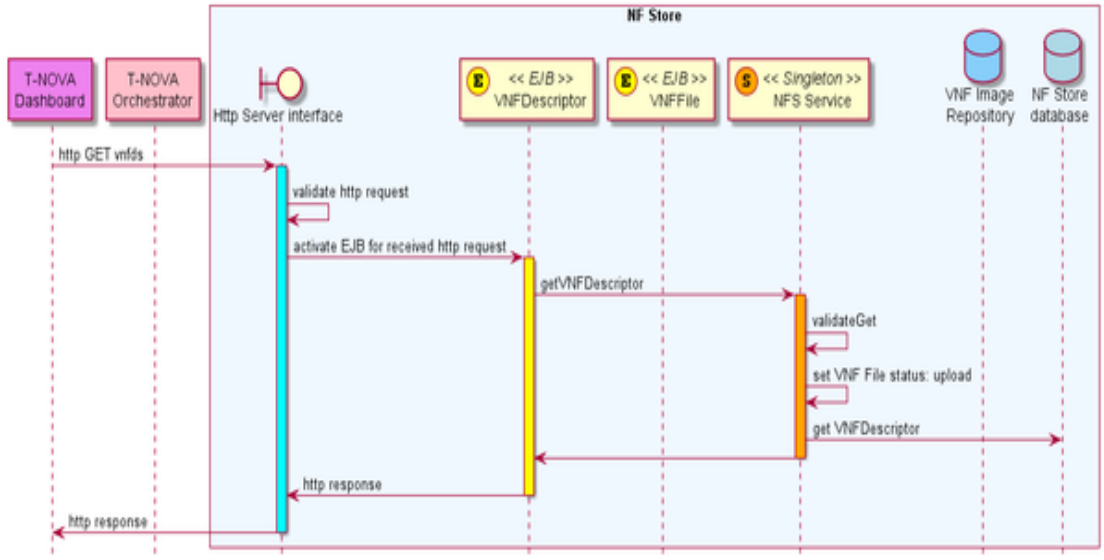

Figure 14. Get VNF descriptor

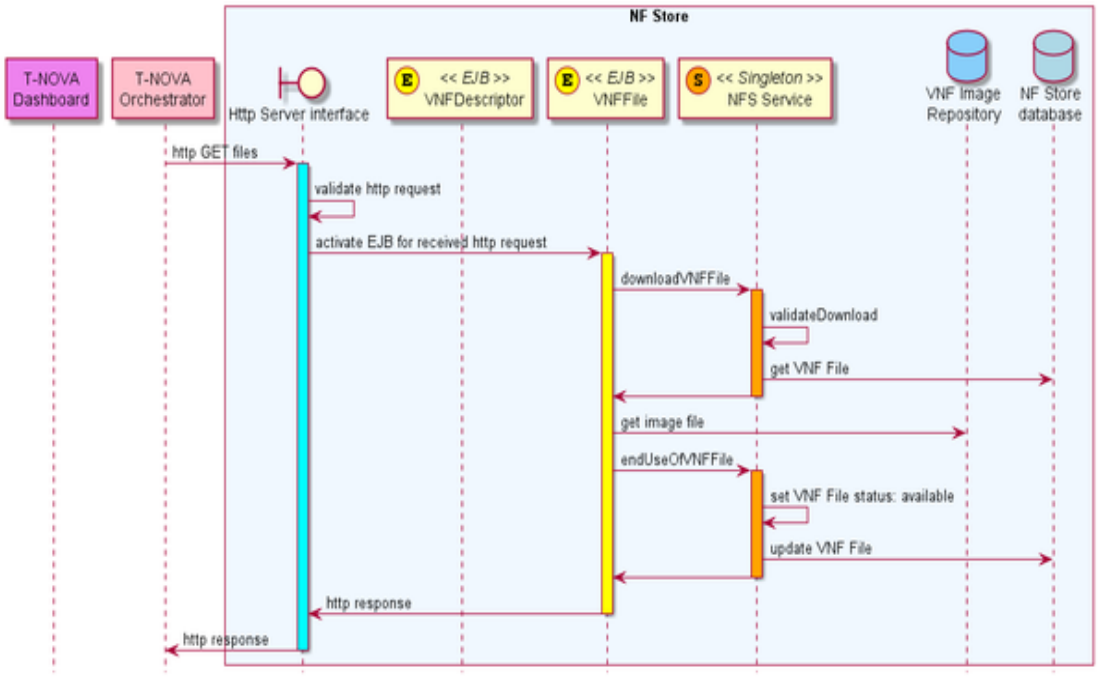

Figure 15. Get VNF file

3.3.5. List

This operation gives to the Dashboard and the Orchestrator the possibility to know the VNF descriptors and files that are available on the NF Store.

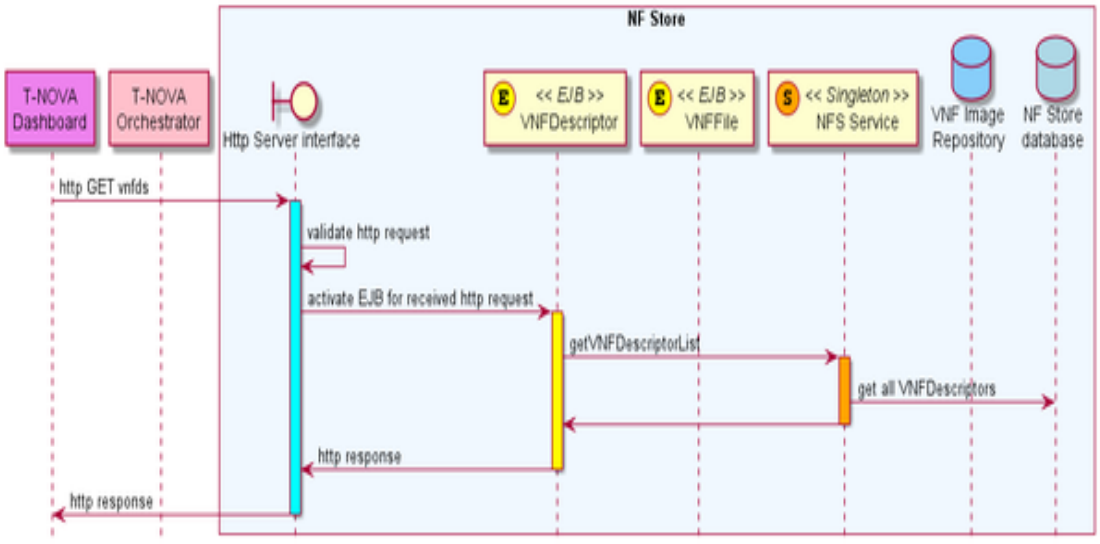

Figure 16. List VNF descriptors

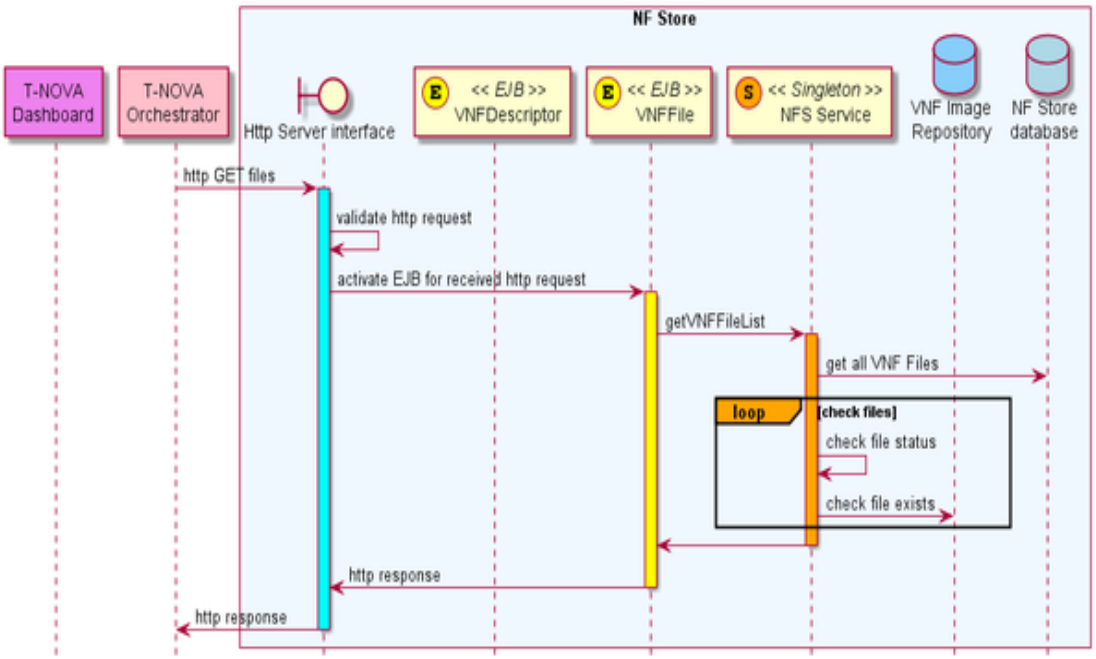

Figure 17. List VNF files

4. INTERFACES

The NF Store provides APIs for the different T-NOVA modules to interact with it. Three interfaces have been defined:

- T-DA-NFS - Dashboard Interface: The interface between the dashboard and the NF Store has the single functionality of providing the API for the Function Providers (FP) to insert their VNFs offerings and associated metadata in the Function Store. Each FP introduces their Functions using the Function Definition feature in the Dashboard. From that, a VNF Descriptor is created and the Function images are uploaded onto the Function Store by means of the previously mentioned API.
- T-OR-NFS - Orchestrator Interface: This interface gives to Orchestrator, when the Function is loaded in the NF Store, the possibility to perform VNFs checks and label the Function as "Available". A second use of this interface is to extract the images and the metadata of the selected VNFs once they have been purchased by a Service Provider and are ready to be deployed.
- T-BR-NFS - Brokerage Interface: The interface between the brokerage and the NF Store has the functionality of providing the API for crawls the list of VNFs and performs trading mechanisms to be able to offer to SP the best possible set of options given the following parameters:
  1. availability=true
  2. trade=true
  3. Type, billing model and price

  For a deeper brokering, we should also look inside the deployment flavor:

  1. Assurance parameters. (Those include the monitoring parameters, thresholds, violations and penalties. For more details on those parameters, please check deliverable D6.2)
  2. Constituent VDU.

The expected response from the VNFs matching the search would include:

- VNF ID
- Provider name
- Description
- Billing model
- Key-flavor/Assurance parameters
- Constituent VDU

The interfaces exposed by the NFS web application deployed into Apache TomEE server give the possibility to operate on VNF image files and VNF metadata descriptor.

4.1. Files Interface

4.1.1. Upload file to NFStore

This method allows uploading a file to NF Store.

The file should be inserted into HTML body as parts of a **multipart/form-data** Content. The **Content-Type** field of the part should be configured to *application/octet-stream*.

The **Content-Disposition** field of the part should be configured setting **name** to *file* and **filename** with the name of file to upload.

The response header **Location** field reports the URL to be used for the other request about this object.

The response body returns the name of the file and the Id list of VNF Descriptors that use the files.

Method	POST		
Endpoint	/NFS/files		
Parameters	**Name**	**Type**	**Description**
Sample Request	**URL**	https://83.212.108.105:8443/NFS/files	
	Body	.... Content-Type: multipart/form-data; boundary=--uuid:2e67c5cf-6c59-4011-ace3-a9d420592392 --uuid:2e67c5cf-6c59-4011-ace3-a9d420592392 Content-Type: application/octet-stream Content-Transfer-Encoding: binary Content-Disposition: form-data; name="file"; filename="file1.img" 1 22 333 4444 55555 666666 7777777 88888888 999999999 0000000000 --uuid:2e67c5cf-6c59-4011-ace3-a9d420592392--	
Sample response	**Status**	**Response body**	
	201 CREATED	{ "name": "file1.img", "vnfd_id": [104,368]}	
Possible errors	**Code**	**Description**	
	400	Bad request	
	500	Internal Server Error	

4.1.2. Download file from NFStore

This method allows getting the file specified into request from NF Store.

The file is returned in the HTML body as parts of a multipart/form-data Content.

The **Content-Type** field is set configured to application/octet-stream.

The **Content-Disposition** field of the part is configured setting name to file and filename with the name of file.

Response 503 (Service Unavailable) is returned when a modify operation on the same file is already in progress.

Method	GET
Endpoint	/NFS/files/<fileName>

Parameters	**Name**	**Type**	**Description**
	fileName	string	Name of the file to download.

Sample Request	**URL**	**Body**
	https://83.212.108.105:8443/NFS/files/file1.img	

Sample response	Status	Response body
	200 OK	--uuid:2e67c5cf-6c59-4011-ace3-a9d420592392 Content-Type: application/octet-stream Content-Transfer-Encoding: binary Content-Disposition: form-data; name="file"; filename="file1.img" 1 22 333 4444 55555 666666 7777777 88888888 999999999 0000000000 --uuid:2e67c5cf-6c59-4011-ace3-a9d420592392--

Possible errors	Code	Description
	400	Bad Request
	404	Not Found
	500	Internal Server Error
	503	Service Unavailable

4.1.3. Update NFStore file

This method allows updating a file already uploaded to NF Store.

The file should be inserted into HTML body as parts of a multipart/form-data Content.

The **Content-Type** field of the part should be configured to application/octet-stream.

The **Content-Disposition** field of the part should be configured setting name to file.

The response returns the name of the file and the Id list of VNF Descriptors that use the files.

Response 503 (Service Unavailable) is returned when another operation on same file is already in progress.

Method	PUT		
Endpoint	/NFS/files/<fileName>		
Parameters	Name	Type	Description
	fileName	string	Name of the file to update.

Sample Request	URL	https://83.212.108.105:8443/NFS/files/file1.img
	Body	.... Content-Type: multipart/form-data; boundary=--uuid:2e67c5cf-6c59-4011-ace3-a9d420592392 --uuid:2e67c5cf-6c59-4011-ace3-a9d420592392 Content-Type: application/octet-stream Content-Transfer-Encoding: binary Content-Disposition: form-data; name="file";

		filename="file1.img" 1 22 333 4444 55555 666666 7777777 88888888 999999999 0000000000 --uuid:2e67c5cf-6c59-4011-ace3-a9d420592392--
Sample response	Status	Response body

	200 OK	{
		"name": "file1.img",
		"vnfd_id": [104,368]
		}

Possible errors	Code	Description
	400	Bad Request
	404	Not Found
	500	Internal Server Error
	503	Service Unavailable

4.1.4. Delete NFStore files

4.1.4.1. Delete all NFStore files

This method allows deleting all files available into the NF Store.

Method	DELETE		
Endpoint	/NFS/files		
Parameters	Name	Type	Description
Sample Request	URL	https://83.212.108.105:8443/NFS/files	
	Body		
Sample response	Status	Response body	
	204 No Content		
Possible errors	Code	Description	
	500	Internal Server Error	

4.1.4.2. Delete NFStore file

This method allows deleting from NF Store the file specified into request.

Response 503 (Service Unavailable) is returned when another operation on the same file is already in progress.

Method	DELETE		
Endpoint	/NFS/files/<fileName>		
Parameters	Name	Type	Description
	fileName	string	Name of the file to delete.
Sample	URL	https://83.212.108.105:8443/NFS/files/file1.img	

Request	Body	
Sample response	Status	Response body
	204 No Content	
Possible errors	Code	Description
	400	Bad Request
	404	Not Found
	500	Internal Server Error

4.1.5. Get NFStore files list

This method allows retrieving the list of all files available into NF Store.

The response report for each file:

- file name
- VNF Descriptor Identifiers of VNFs that use the file

Method	GET		
Endpoint	/NFS/files		
Parameters	Name	Type	Description
	fileName	string	Name of the file to download.
Sample Request	URL		Body
	https://83.212.108.105:8443/NFS/files		
Sample response	Status	Response body	
	200 OK	{ "files": [{ "name": "file1.img", "vnfd_Id": [104] }, { "name": "file2.img", "vnfd_Id": [104,368] }] }	
Possible errors	Code	Description	
	500	Internal Server Error	

4.2. VNF Descriptors interface

4.2.1. Add VNF Descriptor to NFStore

This method allows adding one VNF descriptor to NF Store.

The descriptor should be inserted in json format into the request.

The Content-Type field of the part should be configured to application/json.

Remember that the id field of VNF descriptor should not be present because the value should be inserted by NFStore.

The response return the id of VNF descriptor inserted.

Method	POST
Endpoint	/NFS/vnfds

Parameters

Name	Type	Description

Sample Request

URL	Body
https://83.212.108.105:8443/NFS/vnfds	

Sample response

Status	Response body
201 CREATED	{ "vnfd_id":3901}

Possible errors

Code	Description
400	Bad request
500	Internal Server Error

4.2.2. Get NFStore VNF Descriptor

This method allows getting from NFStore the VNF descriptor with id specified into request.

The descriptor is returned in json form into HTML body.

The **Content-Type** field is set to *application/json*.

Method	GET
Endpoint	/NFS/vnfds/<vnfd_id>

Parameters

Name	Type	Description
vnfd_id	integer	Identifier of VNF descriptor.

Sample Request

URL	Body
https://83.212.108.105:8443/NFS/vnfds/3901	

Sample response

Status	Response body
200 OK	{ "id": 3901 "descriptor_version": "1.0", "version": "1.0", "Vdu": [{ "id": "VDU-1", "vm_image": "file1.img" }, { "id": "VDU-2", "vm_image": "file2.img" }], ... }

Possible errors

Code	Description

400	Bad Request
404	Not Found
500	Internal Server Error

4.2.3. Modify NFStore VNF Descriptor

This method allows modifying one VNF descriptor already available into NF Store.

The new descriptor should be inserted in json format into request.

The **Content-Type** field of the part should be configured to *application/json*.

Remember that the id field of VNF descriptor should be present and should be the same specified into request URL.

The response return the id of VNF descriptor modified.

Method	PUT		
Endpoint	/NFS/vnfds		
Parameters	Name	Type	Description
Sample Request	URL		Body
	https://83.212.108.105:8443/NFS/files/3901		
Sample response	Status	Response body	
	200 OK	{ "vnfd_id":3901 }	
Possible errors	Code	Description	
	400	Bad Request	
	404	Not Found	
	500	Internal Server Error	

4.2.4. Delete NFStore VNF Descriptor

4.2.4.1. Delete all VNF Descriptors

This method allows deleting all VNF Descriptors from NF Store with all files specified into Vdus fields.

Method	DELETE		
Endpoint	/NFS/vnfds		
Parameters	Name	Type	Description
Sample Request	URL	https://83.212.108.105:8443/NFS/files	
	Body		
Sample response	Status	Response body	
	204 No Content		

Possible errors	Code	Description
	500	Internal Server Error

4.2.4.2. Delete NFStore VNF Descriptor

This method allows deleting from the NF Store the VNF Descriptor specified into the request and all files specified into Vdus fields.

The file used by other VNF descriptors will not be removed.

Method	DELETE		
Endpoint	/NFS/vnfds/<vnfd_id>		
Parameters	Name	Type	Description
	vnfd_id	integer	Identifier of VNF descriptor.
Sample Request	URL	https://83.212.108.105:8443/NFS/files/1309	
	Body		
Sample response	Status	Response body	
	204 No Content		
Possible errors	Code	Description	
	400	Bad Request	
	404	Not Found	
	500	Internal Server Error	

4.2.5. Get NFStore VNF Descriptor list

This method allows retrieving the list of all VNF descriptors available into the NF Store.

Method	GET		
Endpoint	/NFS/vnfds		
Parameters	Name	Type	Description
Sample Request	URL		Body
	https://83.212.108.105:8443/NFS/vnfds		
Sample response	Status	Response body	
	200 OK	{ "vnfd_id": [3901, 1309] }	
Possible errors	Code	Description	
	500	Internal Server Error	

4.3. Management Interface

This interface is used to manage the NF Store.

The deployment of the NF Store installs a standard linux SysVinit service called *nfs* that enables users to perform the following operations:

- Start NF Store

Shell command: *service nfs start*

- Stop NF Store

Shell command*: service nfs stop*

- Restart NF Store

Shell command: *service nfs restart*

- Get NF Store status

Shell command: *service nfs status*

5. TECHNOLOGIES

The NF Store implementation is made with a java web service running on Tomcat application server.

Apache Tomcat is an open source software implementation of the Java Servlet and Java Server Pages technologies; for our implementation, we use TomEE, which bundles Tomcat version 7 with along with enterprise dependencies like CDI, EJB, JPA, JTA, Bean Validation.

In our case, we use the TomEE version + that contains also the JAX-RS packages needed for implementing the required REST interfaces.

Database data is described using JPA (Java Persistence API) and the interconnection with DB using JDBC API, giving us the possibility to change it without requiring changes in the source code.

The used database is H2, a pure Java SQL database with small footprint that matches very well with java applications, used in this project in embedded mode as the application is the only resource that needs to access to DB.

Data exchange on interfaces T-DA-NFS and T-OR-NFS are implemented by REST primitives using the standard JSON (JavaScript Object Notation) format. JSON is a lightweight data-interchange format easy for humans to read and write and also easy for machines to parse and generate a text format that is completely language independent and based on a subset of the JavaScript Programming Language.

The Java Virtual Machine is the 1.8 version and the development is made using Eclipse platform (Luna version).

The build procedure use Apache ANT, a Java library and command-line tool for drive processes described in build files.

The final production is a RPM file generated using redline, a pure Java library for manipulating RPM Package Manager packages.

The deployment of RPM on target installs a standard service called nfs that can also be activated at server startup adding it to appropriate run level.

6. DIMENSIONING AND PERFORMANCE

Dimensioning and performances tasks will be carried out when integrating with Marketplace and Orchestrator, please refer to D6.3 for more details.

7. CONCLUSIONS AND FUTURE WORK

7.1. Conclusions

The Network Function store plays a crucial role in delivering VNFs in a timely fashion, VNF images to the Orchestrator, and in providing an API for network developers to publish their work and also an API for Brokerage functions.

By selecting a standard JEE implementation to support a simple but robust REST API, we mitigated the far too common integration issues we may face when interfacing with the rest of the system.

7.2. Future work

In this deliverable, we reported on the current implementation of the network function store. As integration tasks will be carried out when other modules are ready, we decided to postpone the performance measurements until the interfaces are mature enough.

8. LIST OF ACRONYMS

Acronym	Explanation
[API]	Application Program Interface
[CDI]	Contexts and Dependency Injection
[EJB]	Enterprise JavaBeans
[FIQL]	Feed Item Query Language
[IDE]	Integrated Development Environment
[JDBC]	Java Database Connectivity
[JPA]	Java Persistence API
[JSON]	JavaScript Object Notation

9. REFERENCES

[ANT]	Java library to drive processes	http://ant.apache.org/
[CDI]	Contexts and Dependency Injection for Java EE	http://www.cdi-spec.org
[CXF]	Apache CXF services framework	http://cxf.apache.org
[D2.21]	T-NOVA: Overall System Architecture and Interfaces	
[D2.41]	Specification of the Network Function framework and T-NOVA Marketplace	
[Eclipse]	Java IDE	https://eclipse.org/
[H2]	H2 database	http://www.h2database.com/html/main.html
[EJB]	Enterprise JavaBeans	http://www.oracle.com/technetwork/java/javaee/ejb/index.html
[JAX-RS]	Java API for RESTful Services	https://jax-rs-spec.java.net
[JAVA EE]	Java Enterprise Edition	http://www.oracle.com/technetwork/java/javaee/overview/index.html
[JDBC]	Java database connectivity technology	http://www.oracle.com/technetwork/java/javase/jDBc/index.htm
[JPA]	Java Persistence API	http://www.oracle.com/technetwork/java/javaee/tech/persistence-jsp-140049.html
[OpenJPA]	Apache openJPA	http://openjpa.apache.org/
[Redline]	Java RPM Package Manager packages	http://redline-rpm.org/index.html
[Servlet]	Java Servlet	http://www.oracle.com/technetwork/java/index-jsp-135475.html
[Tomcat]	Tomcat application server	http://tomcat.apache.org/
[TomEE]	Tomcat Enterprise Edition	http://tomee.apache.org/apache-tomee.html