TNOVA

NETWORK FUNCTIONS AS-A-SERVICE
OVER VIRTUALISED INFRASTRUCTURES

GRANT AGREEMENT NO. 619520

Deliverable D5.2

Function Deployment, Configuration and Management

Editor B. Parreira, J. Bonnet (PTIN)

Contributors G. Dimosthenous (PrimeTel), N. Herbaut (VIO), M. Arnaboldi (Italtel)

Version 1.0

Date March 31th, 2016

Distribution PUBLIC (PU)

Executive Summary

This report constitutes the final report on the work carried out to achieve the initial objectives in Task 5.2 of the EU-FP7 Project T-NOVA "Functions as-a-Service over Virtualised Infrastructures". The initial goal for this task, as defined in the DoW, is the implementation of a middleware component that enables an API for the control and management of network functions. This report will build on deliverable 5.01 and on WP2 deliverables for requirements and specifications, which are used as a basis for the implementation work within the task.

The middleware API component thoroughly presented in this report realizes the middleware framework mentioned in the previous paragraph. This component is located between the VNF Manager, which is part of the T-NOVA Orchestrator, and the VNFs instantiated on-demand by the end-clients. It realizes the Ve-vnfm-vnf interface specified by ETSI and uses the configuration procedures defined in ETSI GS-NFV-MAN 001 for the IMS MRF instance. These procedures are relatively simple without hindering functionality, the VNF Manager uses the middleware API to upload a configuration file to the VNF and afterwards sends the lifecycle event command to trigger the configuration operation.

There are various advantages of using the middleware API in the communication between the VNF Manager and the VNFs. From the VNF Manager perspective it will only see one generic southbound interface that enables the VNF lifecycle management. This way the T-NOVA Orchestrator does not need to adapt to new and specific interfaces which brings added stability to the platform. From the VNFs integration in T-NOVA point-of-view, VNF developers see various available plugins that enable the management communication protocol of their choice. This way VNF developers will not need to create new interfaces to integrate their VNFs in the T-NOVA platform. Finally, from a general T-NOVA perspective, the platform is able to integrate and support new protocols without jeopardizing the system stability due to the self-contained nature of the middleware API. Currently, two plugins are available, HTTP and SSH, which can be used to interact with VNFs. In terms of security, at the north of the middleware API and for consistency with the remaining components of T-NOVA, the Gatekeeper Authentication mechanism is used. Additionally, for stand-alone tests the "basic auth" mechanism is also supported. This mechanism consists of username and password and can be configured at will. At the south of the middleware API, each plugin supports its own authentication mechanism, for SSH a private key is used while the HTTP uses "basic auth".

The current report provides all the necessary information regarding the middleware API and should be used by VNF developers to integrate their VNFs with the T-NOVA platform. Moreover, a thorough description of all the implementation work in middleware API is provided to better understand this component internal mechanisms.

Table of Contents

1. INTRODUCTION

This document presents the implementation of the middleware API (mAPI) component that is part of the T-NOVA platform. With that in mind, the work presented here builds on the initial specification of this component discussed in [1]. Some information already submitted can be duplicated here with the intention of making this deliverable as auto-contained as possible and ease the VNF developer's access to information. As such, this document presents the architecture for this component and provides a high-level description of the elements that constitute the middleware API. Moreover the northbound interface (towards the T-NOVA Orchestration framework) and the southbound interface (towards the VNFs) are specified in section 2.

Section 3 focuses on the description of VNF Descriptor (VNFD) parameters related with lifecycle description of VNFs, including the specification of the management interface. A function which enables VNF developers to define deployment specific parameters and is used in the VNFD, is also presented.

In section 4 the candidate technologies for the implementation of the middleware APIs are described.

Section 5 describes thoroughly the implementation of the middleware API, which is of a particular interest for this deliverable, including the technologies and workflows that compose this component.

Finally the last section presents the conclusions and future work for the middleware API component.

1.1. Deliverable interdependencies

This deliverable builds on other deliverables published under the T-NOVA project. Although this document tries to be as self-contained as possible, some information not considered essential for this deliverable is further explained in other deliverables. The following table shows this deliverable interdependencies:

Table 1. Deliverable interdependencies

Deliverable(s)	Related Information
D2.21 and D2.22	T-NOVA overall architecture and interface definition and initial specification
D2.41 and D2.42	VNF lifecycle and VNF API descriptions
D3.1	T-NOVA Orchestrator platform APIs which includes the interface with mAPI
D3.41	Integration between the T-NOVA Orchestrator and mAPI
D5.01	Initial middleware API specification
D5.31	VNF implementation including lifecycle management

2. MIDDLEWARE API ARCHITECTURE

The T-NOVA platform enables VNF developers to program the configuration during the entire lifecycle of VNFs. In the VNFD, VNF developers can define the templates (e.g. JSON files) and commands to be used for each lifecycle event. Moreover, during a lifecycle event the Virtual Network Function Manager (VNFM) will fetch the necessary information, defined in the VNFD and send it to VNFs. Therefore, for each lifecycle event VNF developers can define in the VNFD the configuration files and commands to trigger the VNF state change.

Currently there is no standard interface between the VNFM and the VNFs (Ve-vnfm-vnf) and VNF developers shouldn't be limited to a technology that might not suit their needs. On the other hand, having the VNFM support multiple southbound interfaces would need a strong effort from T-NOVA. With this in mind, the middleware API was designed to support two distinct requirements:

- A single and unified interface towards VNFs from the VNFM point-of-view
- The most suitable management technology to VNFs

Other requirements for the middleware API are discussed in section 2.3.

The internal architecture of the Middleware API is shown below:

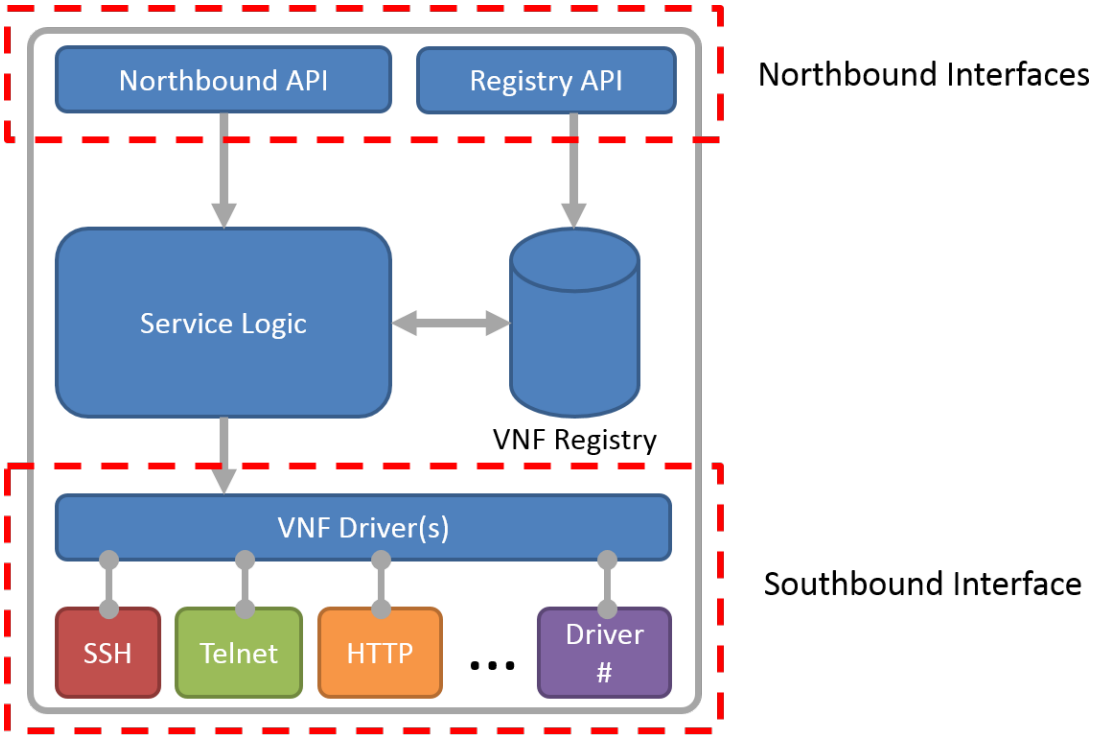

Figure 1. Middleware API Architecture

When a new Network Service (NS) is requested to the Orchestrator, the latter will onboard the constituent VNFs in middleware API using the Registry API. In this process, the service logic will use the VNFD to retrieve the lifecycle definition of the VNFs and persistently store it in the VNF registry. Afterwards, whenever the Orchestrator needs to make a change in the configuration of the VNFs, it will use the VNF generic

northbound API and supply the deployment specific parameters. The service logic will process the request with the included parameters and create the configuration files, if applicable, based on the templates received during the onboarding process. After the service logic aggregates all the information and files it will use one of the available drivers to interact with the VNF and trigger a state change.

2.1. Components Functional Description

Herein is presented the various elements that compose the middleware API.

Northbound API - The northbound API realizes the VNF configuration and management API and it is through this interface that the VNFM can manage the VNFs lifecycle. The specification for this interface is presented in section 2.2.2.

Registry API - The registry API constitutes the interface through which the VNFM can onboard new VNFs in the Middleware API. The onboarding of VNFs in mAPI involves storing data in the database (DB), creation and storage of the template files and creation of the VNF management API. The specification of this interface is presented in section 2.2.1.

Service Logic - The Service Logic component is the component within the Middleware API that holds the necessary intelligence to orchestrate all the operations exposed by the mAPI northbound interface. Typically, this involves processing the parameters sent by the VNFM and creating the VNF configuration file. Afterwards, this component will use the VNF Drivers to trigger the reconfiguration of the VNF.

VNF Registry - This element constitutes the database of the Middleware API. In this database the information regarding the deployed VNFs and lifecycle events is persistently stored.

VNF Drivers - The drivers are realized through plugins that enable the use of different protocols to interface the VNFs.

2.2. Northbound Interfaces

In this section the middleware API northbound interfaces are specified. This includes the Register interface and the VNF Management interface. Each operation in its respective interface is presented using the following model:

- **Endpoint**: Partial URI for the operation, the complete URI is <mAPI IP address>:<mAPI Port>/<Endpoint>
- **Method**: The HTTP Verb used in the operation
- **Parameters**: When applicable a table of parameters in the HTTP request body is presented
- **Possible Errors**: The possible error responses that can be presented
- **Sample Request**: A cURL command used to make a sample request to mAPI

Each row in the 'Parameters' table is presented with the following information:

- **Name**: the name of the parameter

- **Type**: type can be 'Leaf' for single information elements; 'RefElement' when the value contains a reference to another element; and 'SubElement' when the value is another element by itself
- **Description**: short description explaining the information contained in the value

For 'GET' and 'DELETE' HTTP Verbs the body of the request contains no information, therefore, no parameters table will be present in the specification.

2.2.1. Register Interface

This interface enables the registration of a new VNF in the middleware API with the upload of the VNFD. Only sections which are related to the configuration and management of the VNF will be parsed. The following information is contained:

- VNFR Id
- Driver to be used to connect to the VNF
- Authentication information
- Description of the lifecycle events
- VNF container
- Templates to build the configuration files (optional)

The mentioned information is further described and explained in section 3.1.

After successfully registering the VNF (this operation is further described in section 5.2.1), the middleware API provides the corresponding VNF API ID, which is used to access the specific VNF northbound interface. The specification of this interface is presented below:

Endpoint: /vnf_api/

Method: POST

Parameters:

Table 2. Register Operation Parameters

Name	Type	Description
VNFR ID	Ref Element	Id of the VNF Inventory registry to which the API will be made available
VNFD	Sub Element	JSON file containing the complete description of the VNF

Possible Errors:

Table 3. Register Operation Errors

Code	Description
400	Bad Request
401	Unauthorized

405	Method Not Allowed
500	Internal Server Error

Sample Request:

To upload the VNF "vTC" in the Middleware API the following cURL command can be used:

```
curl -X POST 0.0.0.0:1234/vnf_api/ -u admin:changeme -d
@~/mAPI/vTC_mAPI_request.json -v
```

In this example the file vTC_mAPI_request.json can be built with the following structure:

```
{
  "id":"<vTC_VNFR_ID>",
  "vnfd": "<vTC_VNFD>"
}
```

The items present in the VNFD relevant for the Middleware API are presented in section 3.1.

2.2.2. Generic API for Network Function Lifecycle Management

In this section all the operations exposed to manage the VNF lifecycle are specified.

2.2.2.1. Get Configuration

This method allows VNFM to retrieve the last configuration operation from a specific VNF. At least the initial configuration must be accomplished.

Endpoint: /vnf_api/<vnfr_id>/config/

Method: GET

Parameters: This operation does not involve sending information in the body.

Possible Errors:

Table 4. Get Configuration Errors

Code	Description
400	Bad Request
401	Unauthorized
404	Not Found
500	Internal Server Error

Sample Request

To request the last successful VNF configuration in "vTC" using the Middleware API the following cURL command can be used:

```
curl -X GET 0.0.0.0:1234/vnf_api/<vnfr_id>/config/ -u admin:changeme -v
```

2.2.2.2. Post Configuration

This method allows VNFM to set the initial configuration to a specific VNF.

Endpoint: /vnf_api/<vnfr_id>/config/

Method: POST

Parameters:

Table 5. Set Configuration Parameters

Name	Type	Description
Event	Leaf	The event mapped in this request. In this specific case it is always 'start'
VNF Controller	Leaf	The IP address of the VNFC that is capable of configuring all the VNFCs that constitute the VNF
Parameters	Sub Element	Object that contains all the instantiation parameters known at deployment. (Optional)

Possible Errors:

Table 6. Set Configuration Errors

Code	Description
400	Bad Request
401	Unauthorized
404	Not Found
500	Internal Server Error

Sample Request

To set the initial configuration for the VNF "vTC" through the Middleware API the following cURL command can be used:

```
curl -X POST 0.0.0.0:1234/vnf_api/<vTC_VNFR_ID>/config/ -u
admin:changeme -d @~/mAPI/vTC_start_request.json -v
```

In this example the file vTC_start_request.json can be built with the following structure:

```
{
  "event": "start",
  "vnf_controller": ["x.x.x.x"],
  "parameters": {
    "some_parameter":["parameter_value"],
    "another_parameter": ["another_param_value"]
  }
}
```

2.2.2.3. Update Configuration

This method allows the VNFM to update a VNF configuration. To use this operation, at least the initial configuration must be set.

Endpoint: /vnf_api/<vnf_id>/config/

Method: PUT

Parameters:

Table 7. Update Configuration Parameters

Name	Type	Description
Event	Leaf	The event mapped in this request. With the exception of the 'start' and 'stop' events, all other events are mapped in this operation
VNF Controller	Leaf	The IP address of the VNFC that is capable of configuring all the VNFCs that constitute the VNF
Parameters	Sub Element	Object that contains all the instantiation parameters known at deployment. (Optional)

Possible Errors:

Table 8. Update Configuration Errors

Code	Description
400	Bad Request
401	Unauthorized
404	Not Found
405	Method Not Allowed
500	Internal Server Error

Sample Request:

In this example the cURL command to request the VNF "vTC" configuration for the scale up event is showcase:

```
curl -X PUT 0.0.0.0:1234/vnf_api/<vTC_VNFR_ID>/config/ -u admin:changeme
-d @~/mAPI/vTC_scale_up_request.json -v
```

In this example the file vTC_scale_up_request.json can be built with the following structure:

```
{
  "event": "scale_up",
  "vnf_controller": "x.x.x.x",
  "parameters": {
    "some_parameter":"parameter_value",
    "another_parameter": "another_param_value"
  }
}
```

The items present in the VNFD relevant for the Middleware API are presented in section 3.1.

2.2.2.4. Delete Configuration

This method enables VNFM to trigger the stop operation on a running VNF before being destroyed. Moreover, with this operation the VNF API resource in mAPI will be destroyed, which means all the processes that constitute the VNF onboarding will be reverted.

Endpoint: /vnf_api/<vnf_id>/

Method: DELETE

Parameters: This operation does not involve sending information in the body.

Possible Errors:

Table 9. Delete VNF API Errors

Code	Description
400	Bad Request
401	Unauthorized
404	Not Found
405	Method Not Allowed
500	Internal Server Error

Sample Request

To delete the VNF "vTC" from the Middleware API the following cURL command can be used:

```
curl -X DELETE 0.0.0.0:1234/vnf_api/<vTC_VNFR_ID>/ -u admin:changeme -v
```

2.3. Southbound Interfaces

The southbound interface is used to connect the Middleware API to the various VNFs. This interface will use one of the available Drivers to interface with the VNFs.

The interface requirements defined for the interface between the Orchestrator, more specifically the VNFM, and the VNFs are the following:

Table 10. Interface Between VNFM and VNF Requirements

Req. Id	Alignment	Domain(s)	Requirement Name	Requirement Description
T-Vnfm-Vnf.01	T_NOVA_02	VNFM, VNF	Secure Interfaces	All the interfaces between the VNFM and the VNF shall be secure, in order to avoid eavesdropping (and other security threats)
T-Vnfm-Vnf.02		VNFM, VNF	Instatiate/Terminate VNF	The VNFM shall use this interface to instantiate a new VNF or terminate one that has already been instantiated
T-Vnfm-Vnf.03	T_NOVA_46, T_NOVA_48	VNFM, VNF	Retrieve VNF instance run-time configuration	The VNFM SHALL use this interface to retrieve the VNF instance run-time information (including performance metrics)
T-Vnfm-Vnf.04	T_NOVA_23, T_NOVA_33	VNFM, VNF	Configure a VNF	The VNFM SHALL use this interface

				to (re-)configure a VNF instance
T-Vnfm-Vnf.05	T_NOVA_24, T_NOVA_35, T_NOVA_58	VNFM, VNF	Manage VNF State	The VNFM SHALL use this interface to collect/request from the NFS the state/change of a given VNF (e.g. start, stop, etc)
T-Vnfm-Vnf.06	T_NOVA_36, T_NOVA_37, T_NOVA_38, T_NOVA_39, T_NOVA_42, T_NOVA_43, T_NOVA_44, T_NOVA_45	VNFM, VNF	Scale VNF	The VNFM SHALL use this interface to request the appropriate scaling (in/out/up/down) metadata to the VNF

Since, the Middleware API implements the Interface between the VNFM and the VNFs, the above requirements must be supported. Although the current version of the middleware API only supports two protocols to interface the VNFs, SSH and HTTP; other plugins can be developed to add support to other protocols.

2.3.1. SSH Driver

The specification for the SSH driver is summarized below:

- Authentication:
  - Private-Key: the private-key enables authentication in virtual machines when using the SSH protocol. The public key needs to be injected in the VM image prior the VNF onboarding.
- Operations:
  - Copy file to host: this operation allows the middleware API to send the configuration file to the VM. This operation enables the exchange of configuration parameters with the VNFs.
  - Run remote command: this operation is used to trigger a lifecycle event / reconfiguration of the VNF. With this operation VNF developers can specify not only commands that should be run inside the VM but also scripts.

2.3.2. HTTP Driver

The specification for HTTP driver is summarized below:

- Authentication:
  - Basic access authentication: for T-NOVA, when using HTTP only basic authentication will be supported. This authentication is realized by

sending a username and password via HTTP header when making a request.

- Operations:
  - File upload: simple HTTP file upload operation using "multipart/form-data" enconding type. POST and PUT operations must be supported to enable a single step operation (upload file and trigger reconfiguration).
  - Send POST request: POST request only applies to the start operation following the RESTful architecture best practices.
  - Send PUT request: PUT requests are used in all lifecycle events with the exception of the "start" event.
  - Send DELETE request: this request will map the "destroy" lifecycle event.

3. VNF DESCRIPTOR

The VNFD supported in T-NOVA covers the complete description of VNFs, ranging from business to operational requirements and processes. The latter also includes the lifecycle event description and management procedures, which are presented in this section. The following lifecycle events are supported in T-NOVA:

- Start - Initial configuration and start of VNF operation
- Restart – Restart the VNF operation after Pause/Stop events.
- Pause / Stop - Temporary pause without termination of VNF
- Scale Out - Increase of virtual resources
- Scale In - Decrease of virtual resources
- Destroy - Termination of VNF

More information about these lifecycle events can be found in [2].

3.1. Middleware API Related Sections

Bellow you can find an example of a partial VNF Descriptor (with only the lifecycle management related sections):

```
{
   "id":"localvNF",
   "vnf_lifecycle_events":{
     "driver":"SSH",
     "authentication":"-----BEGIN RSA PRIVATE KEY-----
suqoRaisvxwIWs71VI8UkNhQ6t (...) QKSr2V7/mIWfpu9Yde93EFt/rPI2Jc+rw==-----
-END RSA PRIVATE KEY-----",
     "authentication_type":"private key",
     "authentication_username":"ubuntu",
     "vnf_container":"/home/ubuntu/container/",
     "events":{
       {
         "start":{
           "command":"python /home/ubuntu/local_vnf/start.py",
           "template_file_format":"json",
           "template_file":"{
\"controller\":\"get_attr[vdu1:vnfc0:mngt0, ip]\",
\"vdu1\":\"get_attr[vdu1:vnfc0:mngt0, ip]\",
\"vdu2\":\"get_attr[vdu2:vnfc0:data0, ip]\" }"
         },
         "stop":{
           "command":"python /home/ubuntu/local_vnf/stop.py"
         },
         "halt":{
           "command":"python /home/ubuntu/local_vnf/halt.py",
           "template_file_format":"json",
           "template_file":"{\"controller\":\"get_attr[vdu1:vnfc0:mngt0,
ip]\", \"vdu1\":\"get_attr[vdu1:vnfc0:mngt0, ip]\",
\"vdu2\":\"get_attr[vdu2:vnfc0:data0, ip]\"}"
```

```
            }
        }
    }
}
```

3.1.1. VNFD Parameters for Lifecycle Management

A detailed description about the parameters in the VNFD relevant with the Middleware API is provided below.

id: The VNF instance Id (VNFR Id). It is used to register a new VNF in the Middleware. This ID is used by the Middleware API to identify the specific VNF API.

driver: The protocol used by the Southbound interface to communicate with the VNFs. Currently SSH and HTTP are supported.

authentication: For the SSH driver, it is the RSA private key which is used for authentication with the VNFs. The VNF developer is responsible to create a pair of RSA private and public keys. The private key is provided in the "authentication" field while the public key is added on the VNF controller inside the VM. For the HTTP driver, the password is provided.

authentication_type: In case the SSH driver is used, this field should contain the "private key" otherwise the "basic" should be provided when HTTP is used.

authentication_username: The name of the user who is able to run remote commands on the VNF. It is used with both drivers.

authentication_port: The port used with HTTP.

vnf_container: It can be either the path of the folder in the VNF controller in which specific configuration parameters, e.g. the IP of the VNF controller, are transferred by the middleware API or a partial URI for SSH or HTTP respectively.

events: A list of lifecycle events.

event: The name of the lifecycle event. T-NOVA currently supports the start, restart, stop, scale out, scale in, destroy events. However, the mAPI could support other lifecycle events.

command: The remote command to be executed on the VNF controller in order to trigger the VNF re-configuration.

template_file_format: The format of the template file e.g. JSON

template_file: The template file which is used by the VNFM to create the configuration file. It defines parameters that will be obtained during runtime. For example, the IP of the VNF controller will be obtained using the get_attr. Those parameters could be transferred to the VNF controller. This field is optional.

3.1.2. Lifecycle Events Description

In this section a description of the lifecycle events in terms of API calls is provided.

The "start" event is executed using the API call "POST /vnf_api/<vnfr_id>/config/". A JSON file should be passed including the event name, the IP address of the VNF controller and an optional list of parameters. The API call is detailed described in section 2.2.2.2.

The API call "DELETE /vnf_api/<vnf_id>/" triggers the "destroy" event. No parameters are needed to be sent. The API call is detailed described in section 2.2.2.4.

The events "scale out", "scale in", "stop", "restart" are executed using the API call "PUT /vnf_api/<vnf_id>/config/". The event name, the IP of the VNF controller and an optional list of parameters are defined in the JSON file which is passed as a parameter. The API call is detailed described in section 2.2.2.3.

3.2. Get Attributes Function

In IaaS deployment templates sometimes it is needed to reference attributes (e.g. IP Addresses) that are only known after deploying the virtual resources, usually the problem is addressed with the use of "get attributes" functions. In the NFV world the requirements are very similar. We need to have a way to reference these attributes in sections where specific instantiation parameters are important, e.g. lifecycle events.

Example

In this example we have a VNF composed by two VNFCs. To configure the VNF, only the IP addresses assigned to each VNFC is needed. When the VNF developer is building the VNFD for its VNF, it needs to have a way to reference the VNFCs IP addresses in the lifecycle events section.

Although there are various solutions to this problem, here only the most notorious will be highlighted:

- **Amazon Web Servic**es - AWS is one of the most widespread IaaS platforms and has their own version of orchestration templates in which OpenStack HOT is based on.
- **OASIS** - the OASIS consortium has specified the Topology and Orchestration Specification for Cloud Applications (TOSCA), which is being adopted by a lot of platforms recently due to their simple and versatile approach to IaaS templates. Recently they have been specifying a new version of TOSCA to support NFV.
- **OpenStack** - OpenStack is currently the reference IaaS platform for the VIM role. As such, it also has its own definition of a cloud template, HOT, to be used in the HEAT project.

3.2.1. AWS implementation

The following highlights the Get Attributes function defined for the AWS Cloud Formation Template which is defined in [3].

Declaration

"Fn::GetAtt" : ["logicalNameOfResource", "attributeName"]

In which:

logicalNameOfResource - The logical name of the resource that contains the attribute you want.

attributeName - The name of the resource-specific attribute whose value you want.

3.2.1.1. Example

This example returns a string containing the DNS name of the LoadBalancer with the logical name MyLB.

```
"Fn::GetAtt" : [ "MyLB" , "DNSName" ]
```

3.2.2. TOSCA implementation

The definition for the TOSCA Template can be found in [4]. Nevertheless, the information related with the Get Attribute function is shown below.

Declaration

get_attribute: [<modelable_entity_name>,
<optional_req_or_cap_name>,<attribute_name>,
<nested_attribute_name_or_index_1>, ...,<nested_attribute_name_or_index_n>]

In which:

modelable_entity_name - The required name of a modelable entity (e.g., Node Template or Relationship Template name) as declared in the service template that contains the named attribute definition the function will return the value from.

optional_req_or_cap_name - The optional name of the requirement or capability name within the modelable entity (i.e., the <modelable_entity_name> which contains the named attribute definition the function will return the value from. Note: If the attribute definition is located in the modelable entity directly, then this parameter MAY be omitted.

attribute_name - The name of the attribute definition the function will return the value from.

nested_attribute_name_or_index_# - Some TOSCA attributes are complex (i.e., composed as nested structures). These parameters are used to dereference into the names of these nested structures when needed. Some attributes represent list types. In these cases, an index may be provided to reference a specific entry in the list (as named in the previous parameter) to return.

3.2.3. OpenStack HEAT Implementation

This function works by referencing in the template the logical name of the resource and the resource-specific attribute. The attribute value will be resolved at runtime. The syntax for the function can be found at [5], and is as follows:

```
{ get_attr: [<resource_name>, <attribute_name>, <key/index 1>,
<key/index 2>, …] }
```

In which:

resource_name – The resource name for which the attribute needs to be resolved.

attribute_name - The attribute name to be resolved. If the attribute returns a complex data structure such as a list or a map, then subsequent keys or indexes can be specified. These additional parameters are used to navigate the data structure to return the desired value.

3.2.3.1. Example

This example returns the name of the node containing the mysql database.

```
{ get_attr: [ mysql_database, name ] }
```

This example returns the port used by the database service. SELF is a special keyword which can be used instead of the entity name (other keywords: source, target, host).

```
{ get_attr: [ SELF, database_endpoint, port ] }
```

3.2.4. T-NOVA Get Attributes Function

In T-NOVA a function based on HEAT will be used with slight differences in the declaration.

The declaration for the attributes function in T-NOVA is the following:

```
{ get_attr: [ <resource identification>, <resource attribute
identification> ] }
```

In which:

- **Resource identification** – this parameter will identify the specific resource using a tree representation, each level is divided by ':'

- **Resource attribute identification** – each resource in Openstack has several attributes that VNF developers might find useful, with that in mind a specific id of the attribute has to be present

3.2.4.1. Example

This example returns the IP address for the datapath network of a specific VNF:

```
{ get_attr: [ vdu0:vnfc0:data0, ip ] }
```

The vdu0, vnfc0 and data0 labels are all ids of specific resources presented in the VNFD.

3.2.4.2. How it works

Below is a sequence diagram (with a simplified version of resource instantiation) showcasing the deployment of a VNF:

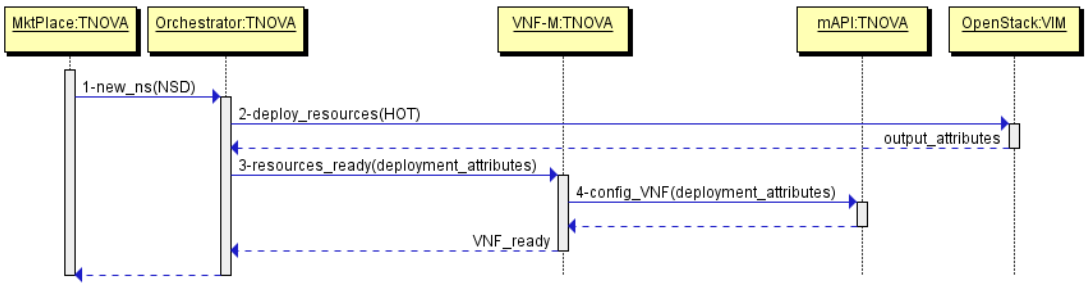

Figure 2. VNF Deployment Sequence Diagram

Steps:

1. The market place requests the Orchestrator to instantiate a new network service
2. After retrieving all the information from the catalogues, the Orchestrator sends a HOT file to OpenStack.
    a. OpenStack will use the HOT to build the necessary virtual resources (for this VNF these include two VDUs)
    b. After deploying the resources, OpenStack notifies the Orchestrator with the specific parameters of this deployment (e.g. IP addresses)
3. Afterwards, the Orchestrator will notify the VNF-M that the resources are ready and it can start the lifecycle management operations
    a. Here, the Orchestrator also provides the VNF-M the deployment attributes
4. To configure the VNF, VNF-M will use the VNFD to know which deployment attributes it needs to send to the mAPI (for this VNF it will be the IP addresses)

4. TECHNOLOGIES

In this section the technologies identified as viable candidates for the implementation of the middleware API will be presented. In this case the most viable candidates are:

- Rundeck
- Saltstack

4.1. Rundeck Overview

Rundeck is a software that enables the automation of remote configuration procedures. A single or a group of commands aggregated in a workflow is called a job. Jobs can be triggered by the scheduler or on-demand.

Rundeck alone provides much of the functionality required to implement the VNF middleware API. In [6] the main use-cases for Rundeck are identified:

- share standard operating procedures
- job scheduler
- automated deployment after a build
- self-service test environments
- coordinate data processing jobs in the cloud
- build a custom operations platform

Some of these use-cases are in line with the VNF middleware API use-case.

Rundeck provides the service logic for the mAPI and as such, when registering a new VNF it is necessary to create new resources in Rundeck. The table below defines the mapping between mAPI entities and Rundeck entities.

Table 11. Rundeck and mAPI entity mapping

Rundeck	mAPI	Description
Project	VNF Instance	Each VNF instance registered in mAPI is mapped in a single Rundeck project and is referenced in mAPI by its VNFR Id.
Job	VNF Instance Lifecycle Event	Each lifecycle event defined in the VNFD is mapped in a single Job within a project in Rundeck
Node	VNF Controller	Each VNF Instance has a single VDU that is identified as the controller in the VNFD. This controller is in charge of receiving the configuration requests from mAPI and perform the configuration of other VNFCs.
Plugin	Driver	A driver identifies a communication protocol to be used by mAPI when interacting with VNFs, e.g. SSH

4.1.1. Jobs in Rundeck

Jobs represent a configuration workflow that can be assigned to one or multiple nodes (VMs). Each Job is defined by:

- Options: dynamic parameters which can be specified before running a job. Also, each option also supports:
  - Restrictions (e.g. match regular expressions, list of allowed values) on possible values
  - Multiple values in a single option
  - Each option can be used in any script or command that is part of the workflow
- Workflow: list of steps that define the configuration procedure. Generally the workflow can be configured to either stop at a failed step or run the remaining steps before failure and also if it is node or step-oriented.
  - Each step in a workflow can be a command, script, local command or a file copy action
  - In terms of error handling, each step can have a specific error handler which is basically another step.
- Notifications: after running a job, Rundeck can notify the result via web hook or e-mail
- Additional functionality:
  - Supports scheduling, timeout and retry
  - Each job can be applied on a single or multiple nodes (simultaneously or sequentially)

4.1.2. Rundeck Driver Support

The standard driver in Rundeck and which is already available is SSH. Nevertheless additional drivers can be added to Rundeck via plugins. The plugins can be either be developed in Java or using Scripts. In case of Java a specific class has to be used to implement the plugin. In the case of scripts it is necessary to provide the command to execute the script (you can use script interpreters such as Python or Bash). The script needs to implement an exit code of 0 in case of success and all STDOUT and STDERR is captured for the Job's output.

For every plugin it is necessary to provide a context which details the specifics for this plugin (e.g. optional arguments).

In the case of T-NOVA the plugin mechanism can be used to implement a REST API driver.

4.2. Saltstack

4.2.1. What is Saltstack

Saltstack [7] is a software that follows the "Infrastructure as a Service" paradigm. It is considered to be the next generation of traditional configuration management software (like Chef or Puppet), on par with Ansible.

With Saltstack, the operator expresses the state of the system using yaml scripts. This state is enforced by a "Salt master" program which is responsible for applying required modification to the system. Among others DevOps tasks, Salt can install packages, run services, copy files, alter system configuration or execute commands. These operations can be done through an installed agent (salt minion) on VMs or through an ssh connection. Saltstack is designed to handle hundreds or thousands of servers. Is uses ZeroMQ [8], a many-to-many networking library developed for the banking world, to achieve security and high-speed message transmission.

4.2.2. Configuration

Each salt minion holds the specific configuration files that allow the salt master to configure the VMs with its specificities. For example, one could specify that VM1 needs to have a firewall allowing TCP ports 22 and 80 whereas VM2 should have UDP and TCP ports 554. To know which specific configuration is needed for which VM, the salt master often relies on "Salt Grains" or "Salt Pillars" configuration files, which are installed on the VM beforehand.

4.2.3. Decision

In a nutshell, for T-Nova, it could provide the ability to send lifecycle commands to VNFs in a secure (by sharing private keys), reliable (by configuring high available salt masters), and flexible (via the state abstraction) way.

However, to achieve scalability, one must install salt minion agents into VNFs as well as maintaining a specific set of salt related configuration files on each VM. This would cause an adherence to a particular system, which is not desirable to attract new VNF developers. Even if using Saltstack over SSH could be possible, we would lose most of its added value wrt other solutions.

Having experienced this technology has been useful however, to design the VNF internal configuration management system, where SaltStack is used by some VNF developers, see [9].

5. IMPLEMENTATION

5.1. Middleware API Implementation

5.1.1. Northbound API

The northbound API is built using Python and the Bottle framework. The latter is a very light web framework without external dependencies other that Python itself. This framework enables a fast implementation of APIs using simple syntax. More information regarding Bottle can be found in [10].

The specification of the northbound API is described in section 2.2.

5.1.2. Service Logic

The core of the Service Logic component in mAPI is realised through Rundeck. Nevertheless, to integrate Rundeck in mAPI a wrapper was implemented. With this in mind, the purpose of this wrapper is to convert non-functional elements such as VNF instances and lifecycle events into resources that Rundeck understands (e.g. Projects, Jobs etc).

With the exception of "add node" operation the mAPI interacts with Rundeck using its RESTful API that is described in [3]. The following operations related with Projects and Jobs are used:

- create Project
- delete Project
- create Job
- execute Job
- check if Job is running
- delete Job

The "add node" operation must be performed using another method because Rundeck does not support this operation in the API. Nodes belonging to a Project are declared in a XML file within the Project folder. To overcome this issue, mAPI accesses the folder and updates the node declaration file by adding the VNF controller.

5.1.3. VNF Registry

This element constitutes the database of the Middleware API and is built using MySQL. In this database the information regarding the deployed VNFs and lifecycle events is persistently stored.

The database is composed by two tables, “VNFs” and “Events”, and can be seen in Figure 3. Also, it is shown the relationship between these two tables, one to many, where one or more rows in “Events” are associated with a single row in the “VNFs” table. The “VNFs” table stores general information about the VNF that can't be mapped into a specific lifecycle event: the "id" which is both the row's primary key and also the T-NOVA's VNF instance id (VNFR Id), the "projectUrl" column which stores the Rundeck project URL and the "username" which is needed for authentication. The “Events” table

stores information regarding lifecycle events such as: the Rundeck Job that enforces the configuration procedures, the event name and the key to a row stored in the VNFs table.

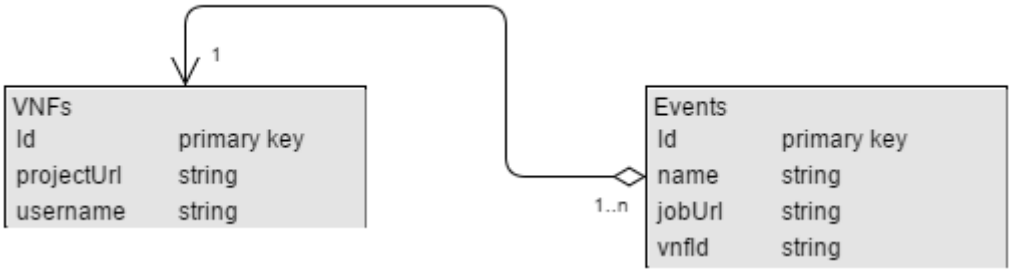

Figure 3. Middleware API Database

Finally, since the mAPI is built using Python, a Python MySQL Object Relational Mapper (ORM), SQL Alchemy [11], was used to interface it. ORM's, such as SQL Alchemy, enables mapping the database into Python classes easing database operations within applications.

5.1.4. VNF Drivers

The drivers are part of the Rundeck application and currently only the SSH and HTTP drivers are supported. The SSH driver is already part of the Rundeck standard version, while the HTTP driver implementation is part of this task's work.

5.1.4.1. Using the HTTP driver

Two RunDeck plugins are available: httpCall and httpUpload.

httpCall plugin

Through httpCall plugin a simple HTTP request command can be sent.

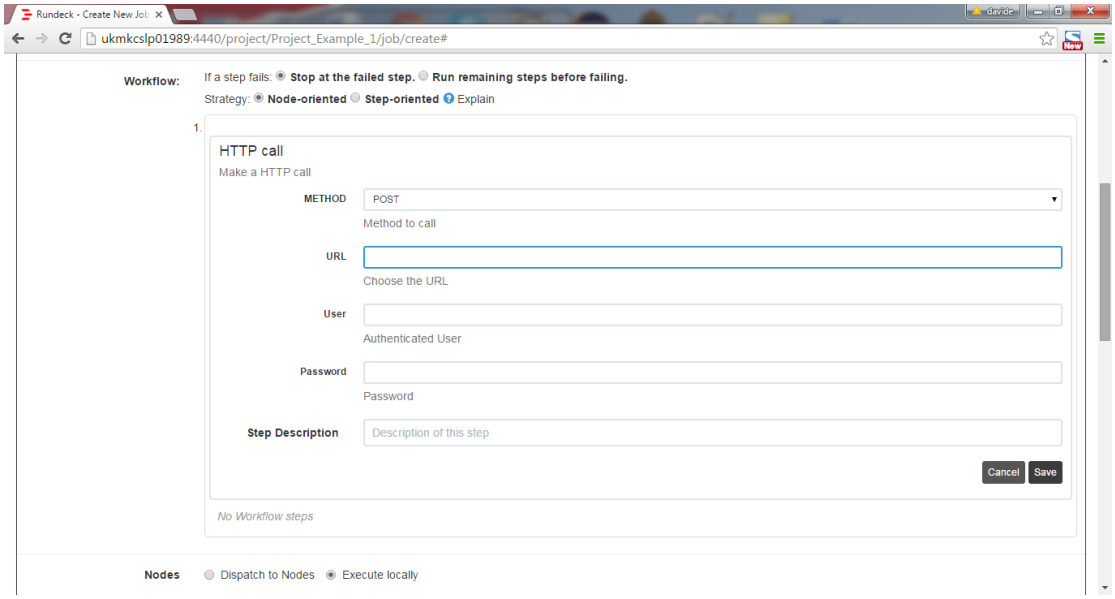

Figure 4. HTTP Command Inside Rundeck Workflow Node

Through the METHOD's menu POST, PUT or DELETE http-method can be chosen. Basic Authentication's User and Password fields are implemented by the server.

http Upload plugin

Through Upload plugin an HTTP request with file upload can be sent.

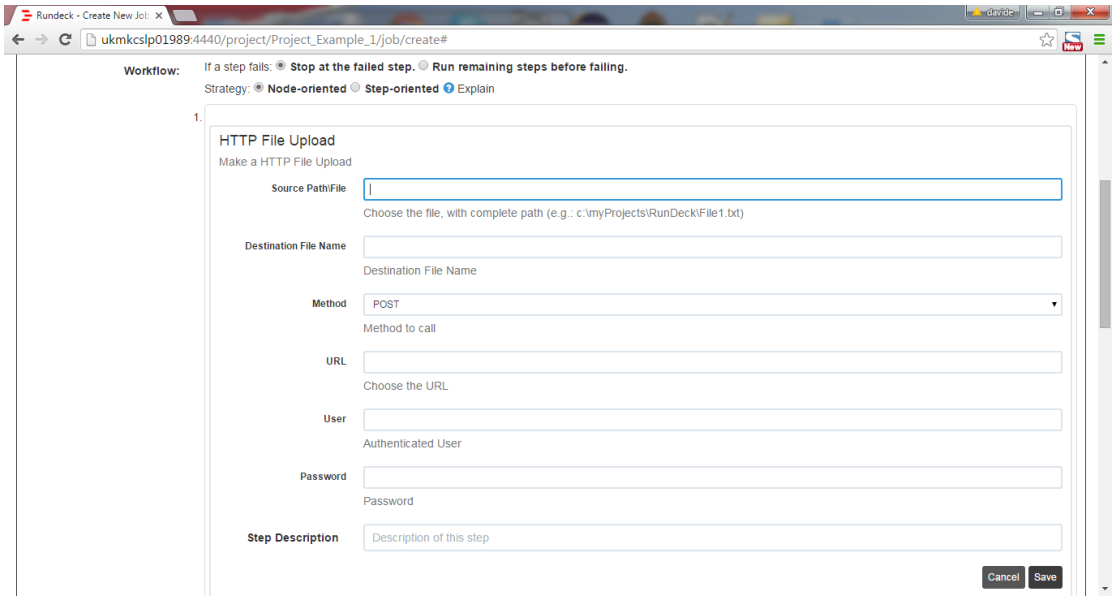

Figure 5. HTTP Upload Inside Rundeck Workflow

Through the METHOD's menu POST or PUT HTTP-method can be chosen. Basic Authentication's User and Password fields are implemented by the server.

5.2. Middleware API Operations

This section details the procedures for each operation of the mAPI. Moreover, a flow chart is shown for each subsection to better illustrate the sequence of steps presented in the operation.

5.2.1. VNF Onboarding

Figure 6 shows the internal procedures when onboarding a new VNF. The first step is the creation of the project in Rundeck which will hold all the jobs that map into lifecycle events, and after a successful creation the project URL is returned. Afterwards, mAPI stores the project URL and username (in case the driver is SSH) in the database. Moreover, a specific folder for the VNF is created to store the event templates and the current configuration info. Finally, the mAPI will run a loop for each lifecycle event defined in the VNFD to create the respective job in Rundeck and to store the necessary information in the database. It must be noted, that not all events will have an associated template (e.g. destroy) but at least a command must be available.

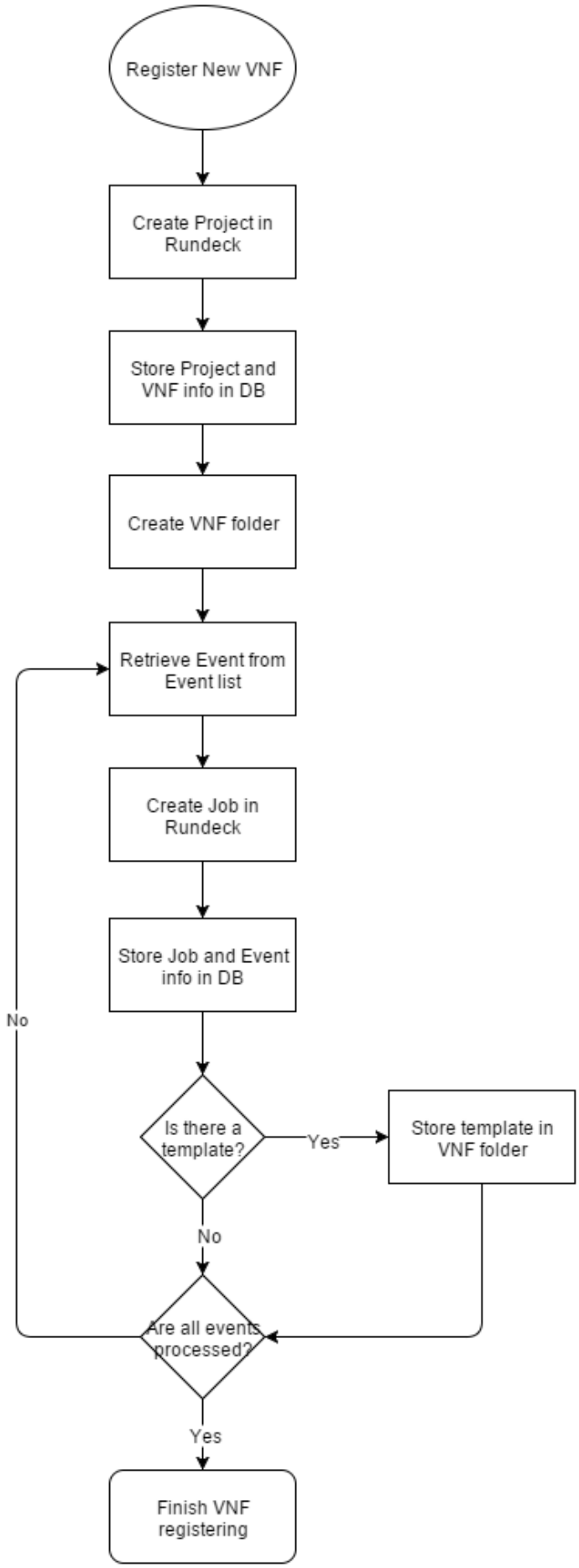

Figure 6. VNF Onboarding Workflow

5.2.2. Get VNF Current Configuration

With the exception of the registering operation, all other operations start by validating the VNFR Id. If no VNF has been registered with this id then the API will return a 404 Error (Not found). Afterwards, mAPI will try to fetch the current configuration from the VNF. This process doesn't involve interacting with the VNF, which will only be successful if at least the start event has been processed. If no configuration is found, it means this VNF is still waiting for the initial configuration and an error is returned (Error 400 - Bad Request). Finally, after a successful retrieval, the configuration is processed and returned. This procedure is illustrated in the figure below.

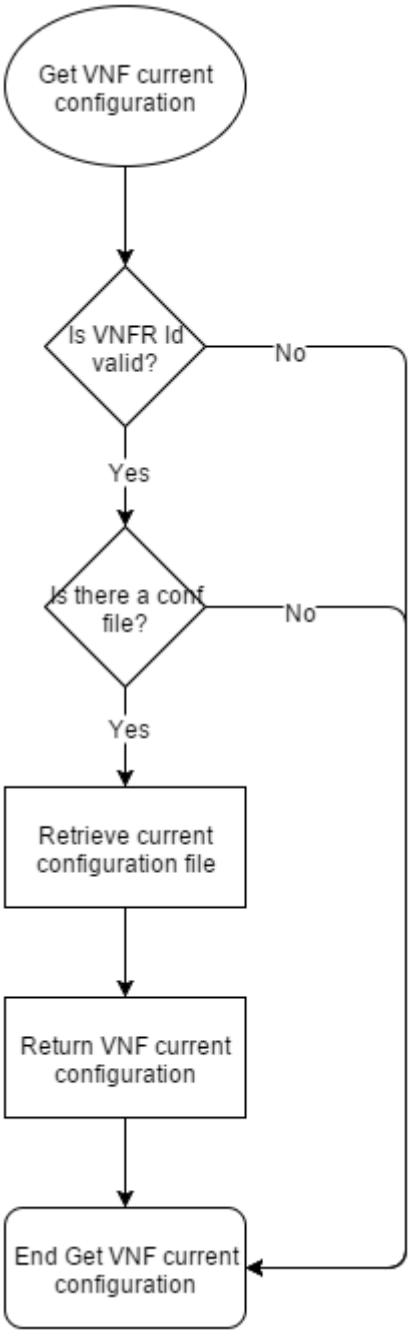

Figure 7. Get VNF Configuration Workflow

5.2.3. Set VNF Initial Configuration

Similar to other runtime management operations, this one also starts with VNFR Id validation. Afterwards, the mAPI will fetch all the VNF and Event information from the database. If deployment parameters are present in the Set Configuration request then mAPI will consume this information to build the configuration file. The configuration file is based on the template defined on the VNFD, which is populated with deployment values obtained through the Get Attribute function defined in section 3.2.3. It should be noted that when the VNF was registered in the mAPI, no deployment information was available and therefore no node was added. It is only during the "start" event that the VNF Controller, which is responsible to configure the remaining VNFCs, is added as a node to Rundeck. Finally, with the configuration file (if applicable) and the VNF controller added, the "start" event Job is executed and this procedure is concluded.

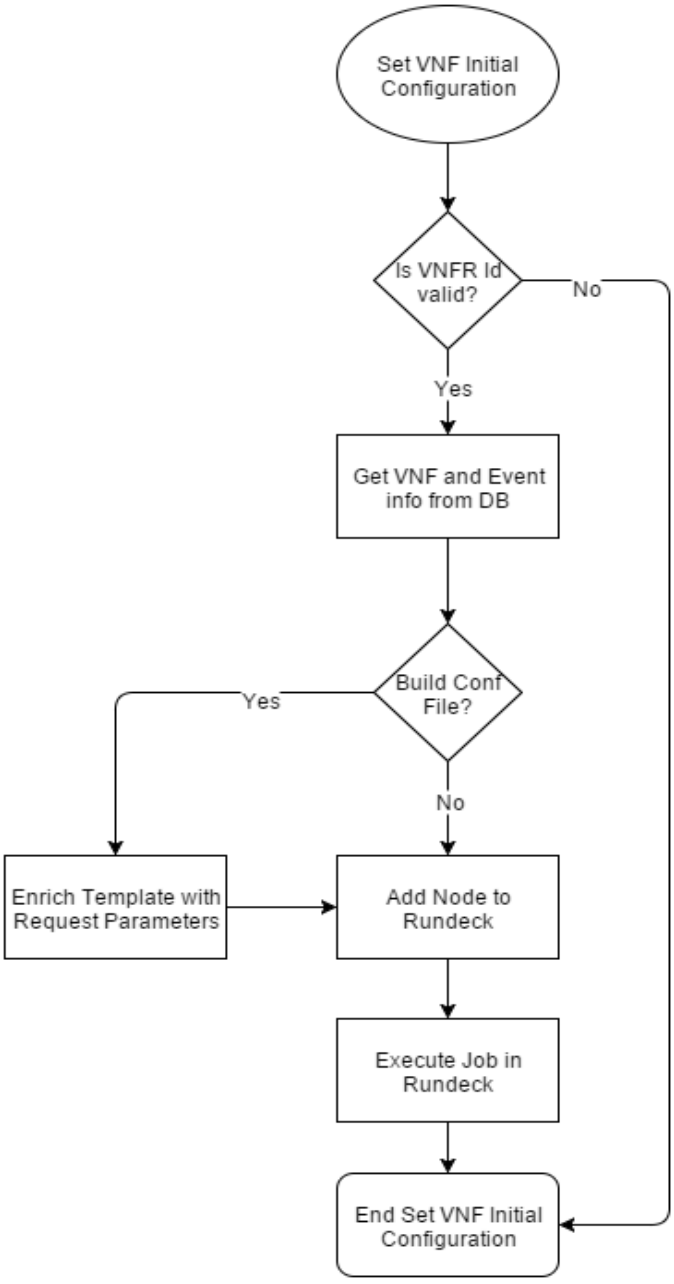

Figure 8. Set VNF Configuration Workflow

5.2.4. Update VNF Configuration

The *Update* VNF configuration procedure is very similar to the previous one, see Figure 9. The only exception is that there is no need to add a node to Rundeck, since it was previously added.

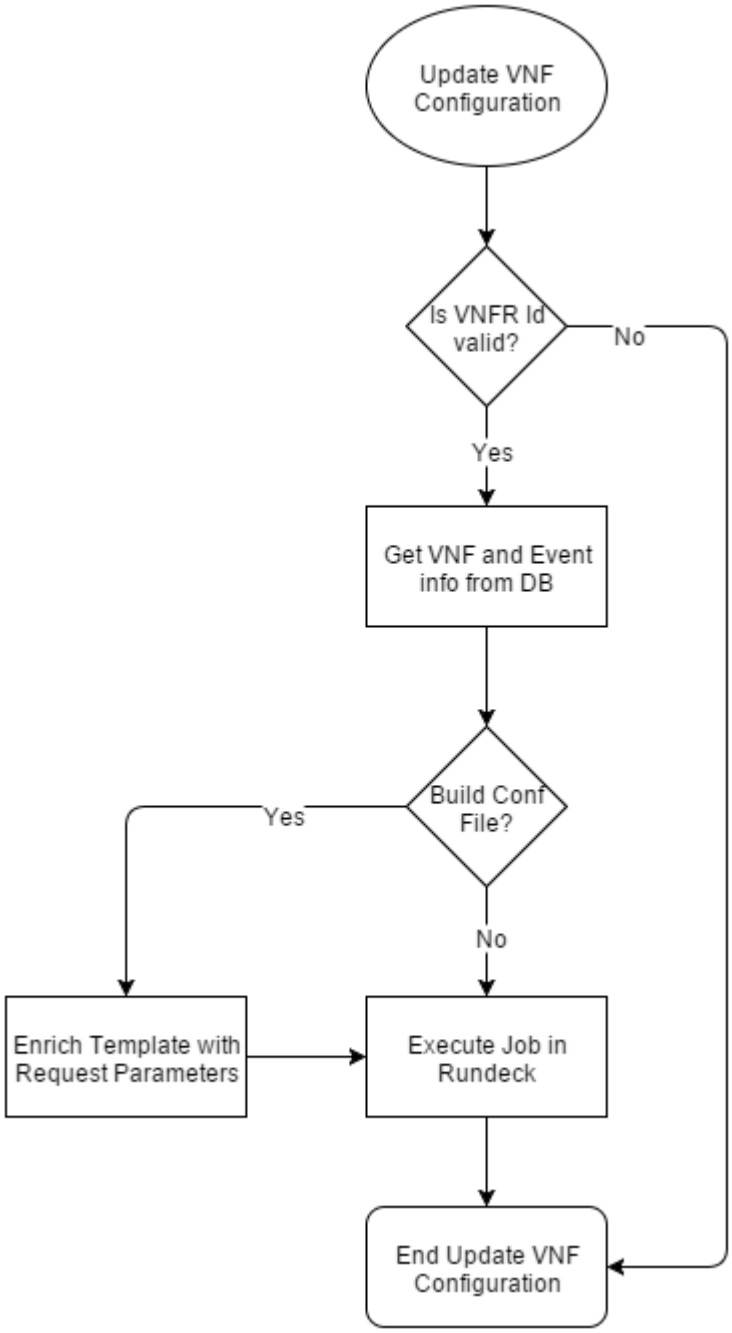

Figure 9. Update VNF Configuration Workflow

5.2.5. Delete VNF Configuration API

The *Delete* operation is divided in two parts, one is the lifecycle event in which the procedure is similar to *Set* and *Update* configuration procedures, and the other is the deletion of all the resources in mAPI. The lifecycle event itself does not involve the use of configuration and ends with a loop to check for completion before moving to the

second part. The deletion of the resources in mAPI is done by successfully deleting the Rundeck Project (by deleting the project all other Rundeck resources are also removed); deleting all the files present in the VNF folder; and finally by removing the VNF entry in the database.

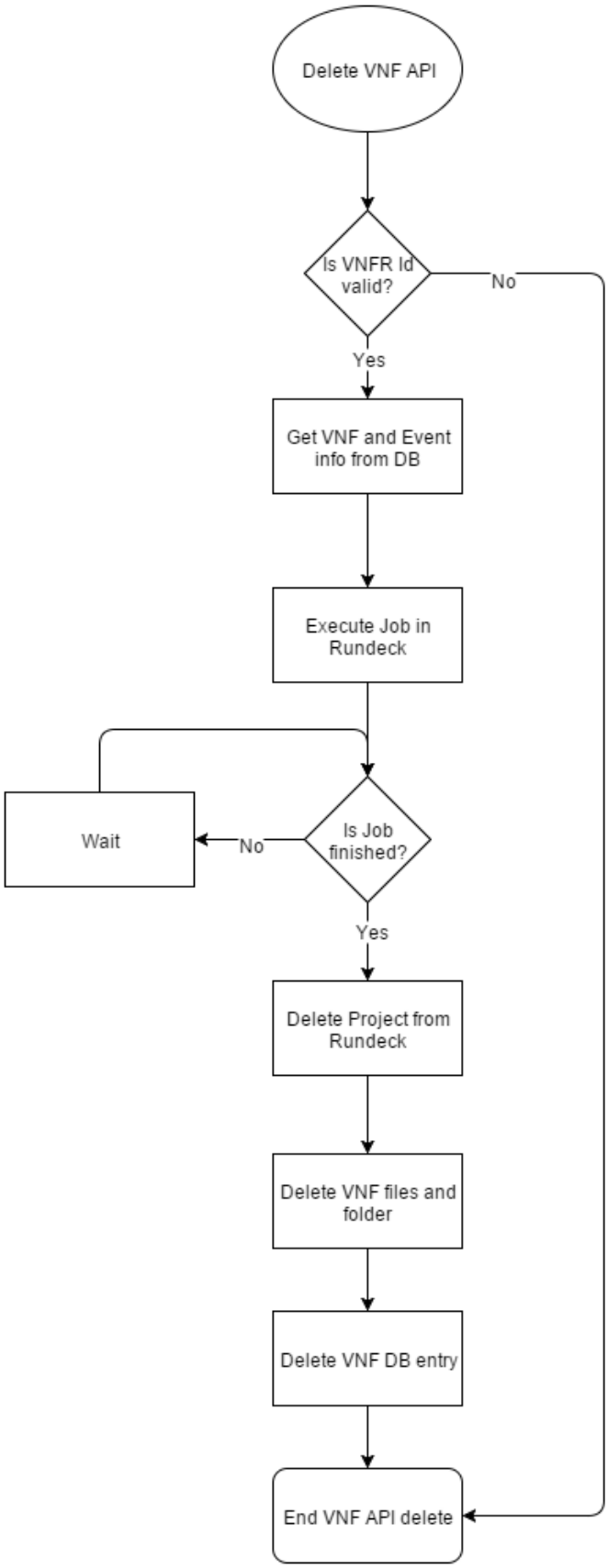

Figure 10. Delete VNF API Workflow

5.3. Code Repository

The Middleware API code and instructions on installation procedures are available at the following GIT repository:

https://github.com/T-NOVA/mAPI

In the repository is also available some examples on how to use the Middleware API.

A proof of concept of the Middleware API using the PXaaS VNF can be found at:

https://github.com/T-NOVA/PXaaS-mAPI

Specifically it presents the steps followed in order to integrate the PXaaS VNF with the mAPI.

6. Conclusion and Future Work

6.1. Conclusions

This document reports the outputs of Task 5.2, which aimed to define the middleware layer realizing the API for the deployment, configuration and management of the VNF.

Therefore the following key points have been identified:

- high level description of the internal architecture of the middleware API components and of the elements constituting the middleware API
- general guidelines for the implementation of the middleware API component, highlighting the benefits coming from the open source software used in this process. The technologies identified are Rundeck and Saltstack
- detailed description of :

– the northbound interface towards the T-NOVA Orchestration framework. This interface enables registering a new VNF, retrieving the last configuration from a specific VNF, setting the initial configuration to a specific VNF, updating a VNF configuration, stopping operations on a running VNF

– the southbound interface towards the various VNFs. The current version of the middleware API supports, by means of specific plugins, two communication protocols (SSH and HTTP)

- usage of commodity servers to host the VNFs. Specific APIs have been developed to allow the installation, configuration, modification and termination of these VNFs
- detailed description of VNFD parameters related to the lifecycle management of a network function, according to the ETSI specification. The T-NOVA project supports the following lifecycle events: Start, Restart, Stop, Destroy, Scale In/Out
- definition of the GIT repository where the Middleware API code and instructions on installation procedures are available.

Given the before mentioned key points, we can say that the initial goal of implementing a VNF API framework was fully achieved.

6.2. Future Work

Current implementation supports the transfer of instantiation parameters to VNFs and the triggering of state changes through commands or HTTP calls. In the near future, the middleware API should also support script based configurations (using the SSH driver), instead of configuration files plus command method, for added versatility.

Other drivers should also be implemented to shorten the time VNF developers need to integrate their VNFs in T-NOVA platform. With the increase of drivers, chances are that VNFs native management interface is already supported at the driver level and thus, VNF developers do not need to write adapters to integrate their VNFs.

Follow ETSI specification of Ve-vnfm-vnf interface and make the necessary adaptations to fully support ETSI NFV standardization effort.

7. REFERENCES

[1] Deliverable D5.01 - Report on Network Functions and associated framework. T-NOVA Project.

[2] Deliverable D2.41 - Specification of the Network Function Framework and T-NOVA Marketplace.

[3] Amazon AWS Cloud Formation User Guide. Available Online [Accessed 2015]. http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/cfn-ug.pdf.

[4] OASIS TOSCA Simple Profile in YAML Version 1.0. Available Online [Accessed 2015]. http://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.0/csd03/TOSCA-Simple-Profile-YAML-v1.0-csd03.pdf.

[5] OpenStack HEAT Template Guide. Available Online [Accessed 2015]. http://docs.openstack.org/developer/heat/template_guide/hot_guide.html.

[6] Rundeck. Available: [Accessed 2016] http://rundeck.org.

[7] Saltstack. Available: [Accessed 2016] http://saltstack.com/.

[8] Zero MQ. Available: [Accessed 2016] http://zeromq.org/.

[9] Deliverable D5.31 - Network Functions Implementation and Testing. T-NOVA Project.

[10] Bottle: Python Web Framework. Available: [Accessed 2016] http://bottlepy.org/docs/dev/index.html.

[11] SQL Alchemy: The Database Toolkit for Python. Available: [Accessed 2016] http://www.sqlalchemy.org/.

8. List of Acronyms

Acronym	Explanation
API	Application Programming Interface
AWS	Amazon Web Services
CPU	Central Processing Unit
DB	Database
DevOps	Development and Operations
DNS	Domain Name System
ETSI	European Telecommunications Standards Institute
GPU	Graphics Processing Unit
HOT	HEAT Orchestration Template
HTTP	Hypertext Transfer Protocol
IaaS	Infrastructure-as-a-Service
IMS	IP Multimedia Subsystem
IP	Internet Protocol
JSON	JavaScript Object Notation
mAPI	Middleware API
MRF	Media Resource Function
NFV	Network Functions Virtualization
NS	Network Service
OASIS	Organization for the Advancement of Structured Information Standards
ORM	Object-Relational Mapping
PXaaS	Proxy-as-a-Service
REST	Representational State Transfer
SQL	Structured Query Language
SSH	Secure Shell
STDERR	Standard Error
STDOUT	Standard Output
TCP	Transmission Control Protocol
TOSCA	Topology and Orchestration Specification for Cloud Applications

UDP	User Datagram Protocol
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
VDU	Virtualization Deployment Unit
VIM	Virtual Infrastructure Manager
VM	Virtual Machine
VNF	Virtual Network Function
VNFC	Virtual Network Function Component
VNFD	Virtual Network Function Descriptor
VNFR	Virtual Network Function Record
vTC	Virtual Traffic Classificator
XML	eXtensible Markup Language