

Slovak University of Technology
Faculty of Informatics and Information Technologies

Team Project

NetCell – Technical Document

Bc. Andrej Mlynčár

Bc. Martin Janočko

Bc. Tomáš Hermánek

Bc. Tomáš Mikuška

Bc. Lukáš Kleščinec

Bc. Ľubomír Kaplán

Academic Year: 2015/2016

Degree Course: Software Engineering

Table of Contents

| | |
|---|----|
| 1 Introduction | 1 |
| 2 Overall View of System | 1 |
| 3 Project Goals for Winter Semester | 2 |
| 4 Project Goals for Summer Semester | 3 |
| 5 System Modules..... | 4 |
| 5.1 Analysis..... | 4 |
| 5.2 Design..... | 5 |
| 5.2.1 Management Core Modularization | 6 |
| 5.2.2 Node Type Modularization | 8 |
| 5.2.2.1 Node Client for VM-based VNFs | 8 |
| 5.2.2.2 Management Server Modules for VNFs | 9 |
| 5.2.2.3 VM-based VNFs and Management Server Communication | 11 |
| 5.3 Implementation | 11 |
| 5.3.1 NetCell Management Server..... | 11 |
| 5.3.1.1 Configuration Module | 12 |
| 5.3.1.2 Database Module | 13 |
| 5.3.1.3 Web Server Module..... | 13 |
| 5.3.1.4 Event Logger Module | 14 |
| 5.3.1.5 Identity Service Module | 15 |
| 5.3.1.6 OpenStack Manager Module | 17 |
| 5.3.1.7 Task Manager Service..... | 17 |
| 5.3.1.8 NetCell Launcher Module | 19 |
| 5.3.2 NetCell Node Client Core..... | 19 |
| 5.3.2.1 NetCell Node Client..... | 20 |

| | |
|---|----|
| 5.3.2.2 Node Type Configuration Definition..... | 20 |
| 5.3.2.3 Netcell Node Client Core Post Install Script | 21 |
| 5.3.2.4 Netcell Node Client Interface Configuration..... | 22 |
| 5.3.3 Netcell Node Client Modules..... | 24 |
| 5.3.3.1 Application Server Implementation | 25 |
| 5.3.3.2 Database Server Implementation | 25 |
| 5.3.3.3 DNS Module Implementation..... | 26 |
| 5.3.3.4 OpenVPN module Implementation | 27 |
| 5.3.3.5 Webserver module implementation | 28 |
| 5.3.4 NetCell Node Configuration Core | 29 |
| 5.3.5 NetCell Web Portal | 29 |
| 5.4 Testing..... | 30 |
| 5.4.1 Unit Testing..... | 30 |
| 5.4.1.1 Node Client Core Testing | 30 |
| 5.4.1.2 Node Configuration Core Testing | 31 |
| 5.4.2 Acceptance Testing | 32 |
| 5.4.2.1 Definition of acceptance tests..... | 32 |
| 5.4.2.1.1 Test Case 01: Create user | 32 |
| 5.4.2.1.2 Test Case 02: Create new project..... | 33 |
| 5.4.2.1.3 Test Case 03: Deploy Nameserver node client | 33 |
| 5.4.2.1.4 Test Case 04: Deploy Application server node client | 34 |
| 5.4.2.1.5 Test Case 05: Deploy Webserver node client | 35 |
| 5.4.2.1.6 Test Case 06: Deploy Database server node client | 36 |
| 5.4.2.1.7 Test Case 07: Deploy VPN server node client | 37 |
| 5.4.2.1.8 Test Case 08: Node client management | 38 |

| | |
|--|----|
| 5.4.2.1.9 Test Case 09: Other web-portal functionality | 38 |
| 5.4.2.1.10 Test Case 10: Tasks | 38 |
| 5.4.2.2 Acceptance testing results..... | 39 |
| 6 Prototypes | 40 |
| 6.1 NetCell Management Server..... | 41 |
| 6.2 NetCell Node Clients..... | 43 |
| 7 Installation Guidelines | 44 |
| 7.1 OpenStack | 44 |
| 7.2 NetCell Management Core..... | 44 |
| 7.3 NetCell Management Portal | 45 |
| 7.4 NetCell Node Client | 45 |
| 8 Appendices | 47 |
| 8.1 Netcell Node Client Modules Configuration Definitions..... | 47 |
| 8.1.1 Application Server Configuration Definition..... | 47 |
| 8.1.2 Database Server Configuration Definition..... | 47 |
| 8.1.3 DNS Configuration Definition | 50 |
| 8.1.4 VPN Configuration Definition | 54 |
| 8.1.5 Web Server Configuration Definition..... | 59 |

List of Tables

- Table 1** VNFs provided by Netcell system 2
- Table 2: List of Identity Service API endpoints 16
- Table 3: List of OpenStack manager API endpoints..... 17
- Table 4:Netcell Code Client Core interface configuration API functions..... 23

Table of Figures

- Figure 1** Management Server Architecture Overview 6
- Figure 2** Node Client Core Dependency Diagram 9
- Figure 3** Node Type Manager Loaded Modules Diagram..... 10
- Figure 4** VM-based VNF and Management Server Communication Diagram 11

1 Introduction

Documentation is containing detailed information about software system created during Team Project courses in academic year 2015/2016.

NetCell system is a web-based system networking application which provides options for drawing network topologies, which can be automatically deployed to production environment. NetCell provides automatic deployment and configuration of virtual network functions such as firewalls, web-servers, database servers, etc.

NetCell is developed by following team members:

- **Hermánek Tomáš, Bc.** – Python development, VPN service development,
- **Janočko Martin, Bc.** – Python development, Web-sever service development,
- **Kaplán Ľubomír, Bc.** – Java development, OpenStack administration, Core development,
- **Kleščinec Lukáš, Bc.** – Python development, DNS service development,
- **Mikuška Tomáš, Bc.** – Python development, Database and Application service development,
- **Mlynčár Andrej, Bc.** – Java development, Core development.

2 Overall View of System

Application is implemented as a web-based interface built on Java Maven platform. System core is running as jar file executed directly from server filesystem. System is using **OpenStack** cloud computing system for management and deployment of virtual network topologies. Client nodes, which are responsible for configuring virtual network functions (VNF) are implemented in Python programming language.

Following functionality is provided by NetCell system:

- **Topology Deployment** - as of user's decision to launch an instance of topology template, the template including all its VNFs and network connections is built on

top of OpenStack and put in functional state. This topology can be later modified and reconfigured and also saved as a template.

Following virtual functions are implemented in NetCell system:

Table 1 VNFs provided by Netcell system

| Name | System | Type | Scope | Description |
|---------------------------|---------------|-------------|----------------------|---|
| VPN | OpenVPN | Service | Linux-based Image | Secure remote network connection for remote users using OpenVPN. |
| Database Server | MariaDB | Service | Linux-based Image | Database service for other services. |
| Name Server | Bind9 | Service | Linux-based Image | Provide DNS server supporting forward and reverse resolution. |
| Application server | GlassFish | Service | Linux-based Image | Service that provides facilities to create web applications and a server environment to run them. |
| Web Server | Httpd | Service | Linux-based Image | Allows for web service provisioning using HTTP(s) protocols |

3 Project Goals for Winter Semester

The winter semester of 2015/2016 academic year covers the inception of the project development containing project organizational tasks as well as development tasks. Development tasks should result in a semi-functional product, which allows users to perform basic operations over the system:

1. Configure the system.

2. Work with basic system functions.
3. List types of virtual network functions, which may be deployed.

Overall goals for the end of winter semester of 2015/2016 academic year are:

1. Choose a cloud environment for which the network function virtualization application will be developed.
2. Analyze options for system modularization and design the foundations for system core, which will take care of user operations and communication and provisioning for virtualized network functions.
3. Design environment in which virtual network functions will run, the way of configuration provisioning and other management communication.
4. Develop the basic foundations for the core of the application.
5. Develop the foundation for virtual network functions management and provisioning,
6. Choose virtual network functions (service) to be developed by the project team.
7. Choose scope of functionality supported by the developed services.

4 Project Goals for Summer Semester

The summer semester of academic year 2015/2016 was mainly focused on development of management core module and node type services defined in previous semester.

Overall goals for the end of summer semester and also end of team project are:

1. Development of remaining components of management core
 - a. Identity Services
 - b. Event Logger Services
 - c. Task Management Services
 - d. OpenStack Services
 - e. Topology Management Services
2. Development of functionality of proposed node types – functionality should include basic configuration and operation options like network configuration, port

configuration, basic configuration specific for each of the proposed node type components.

3. Implementation of user-friendly interface designed control of management core system.
4. Integration of node types with management core and OpenStack network.

5 System Modules

5.1 Analysis

Since the application of this team project consists of several functional elements, it requires a structured split into system modules and sub-modules. The primary goals of modularization are:

1. Establishment of flexible environment for team cooperation,
2. Allowing for easy system extensibility in the future.

Final product needs to be split into several modules, which separate the core functionality from the functional layer. Next modular separation is aimed to separate particular elements of the application core into smaller parts, which can be extended in the future in a simple and efficient way.

Services to run on target VNF instances are developed in Python programming language with common core providing the basic functionality. Particular services developed as node types for the application are separate python modules using the common core. This forms are the first modularity layer. Node client core uses Flask REST API framework. This framework was chosen due to its small footprint in comparison to other REST frameworks such as Django, which has much bigger overhead and requires data modelling. These features are not required in this project, since we are trying to push the requirements for the VNF instance to the minimum. To achieve this, the Node Client application run within VNF instances needs to be as lightweight as possible.

Core of the application is developed in Java 8 programming language, which provides several modularity bases and modularization options such as:

1. Creation of simple custom plugin manager,
2. Container systems as **Jetty**

At the beginning of the project development, OSGi infrastructure of main web-interface system was chosen. OSGi due to its service-based nature, which works using the SOA model. OSGi has also lower overhead in comparison with other container frameworks. Unfortunately, during development of OpenStack OSGi services, problems with OpenStack OSGi bundles occurred and because of the missing compatibility, project needed to be migrated into standard maven architecture and every OSGi bundle will be implemented as separated maven module.

5.2 Design

The whole project needs to be separated in multiple modules and sub-modules on multiple tiers of modularization. First tier covers the following:

1. Management core modularization
2. Node type modularization.

5.2.1 Management Core Modularization

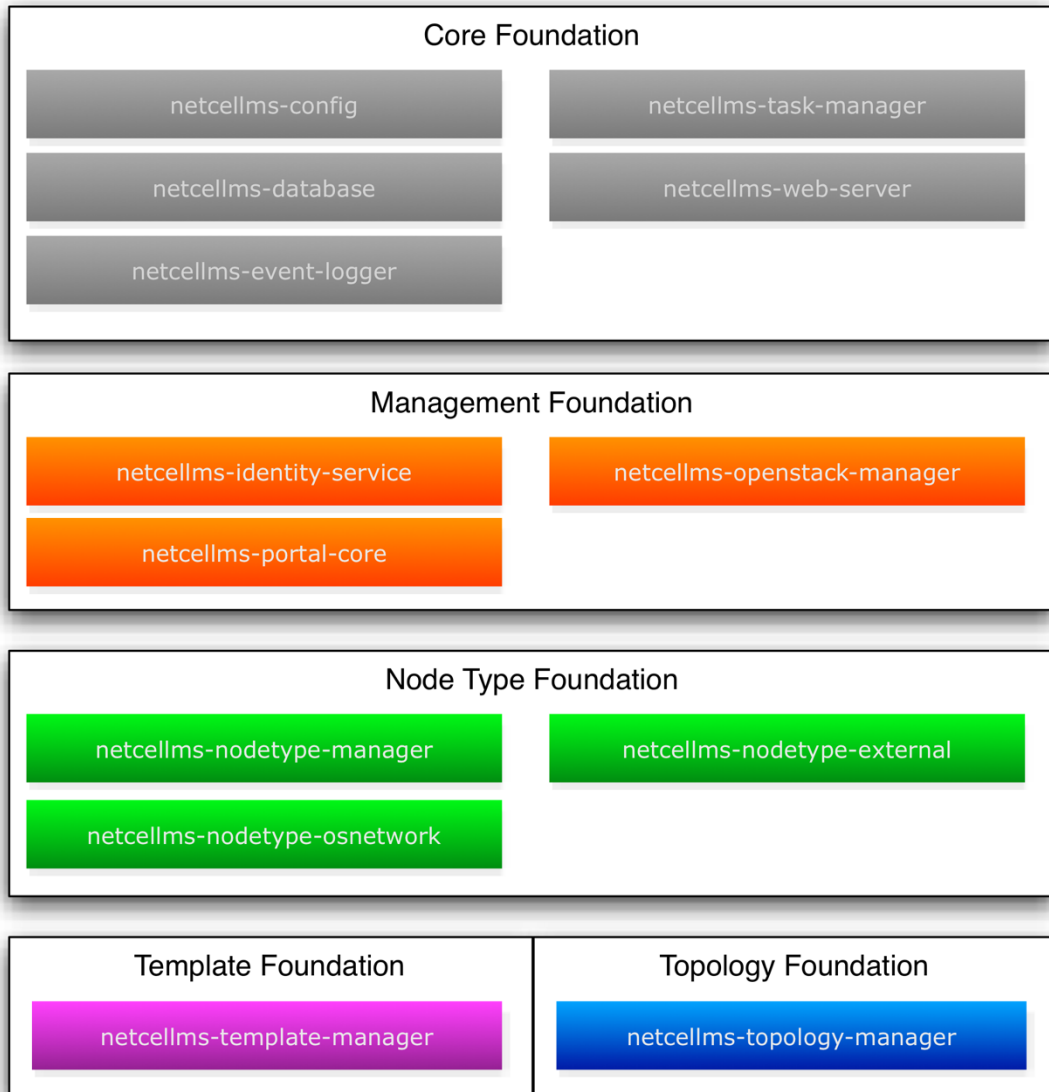


Figure 1 Management Server Architecture Overview

Figure 1 depicts the structure of the management server. The application is split into several foundations providing smaller portions of functionality in the entire system. The foundations provided by the management server are the following:

1. Core Foundation

- Provides basic functionality for the system such as logging, database access, configuration access, task management and web services provider.

- This foundation is written in generic and reusable way, allowing for use as a foundation of other applications as well.

2. Management Foundation

- Basic identity functionality covering user management, user session management, permission management and project management.
- Projects are the workspaces for users, this module does not provide any functionality within projects – only creates them as containers (workspaces).
- Primary component allowing communication with the underlying cloud platform is part of this foundation.
- Basic API methods are registered at this foundation, as well as core of the web portal.

3. Node Type Foundation

- This foundation takes care of enumeration, configuration and deployment metadata retention for node types, which can be deployed within the virtual topologies or designed in templates.
- Primary goal of this foundation is maintenance of registry with all node types and access to functions allowing deployment and management of nodes based on these node types.

4. Template Foundation

- Templates allow users to design network topologies and this foundation takes care of this functionality.
- Extends the API with methods related to template management.
- Extends the web portal (primarily project base) with views and functionality required for template management.

5. Topology Foundation

- Allows for deployment of templates and for live drawing of topologies.
- Basically a functional core elements of the system are parts of this foundation.

Each colored container (as depicted in Figure 1) is one or more modules of NetCell Management application infrastructure. Application will run in Jetty web container.

5.2.2 Node Type Modularization

Methods of VNF provisioning may be different and methods being currently implemented are the following:

1. Use of cloud platform built-in networking functions (cloud-based), eg.:
 - Firewall as a Service (FWAAS),
 - Load-balancer as a Service (LBAAS),
 - IPsec Site-to-Site Tunnel (IPSEC)
2. Use of virtual machine templates (VM-based) in order to create a fully customized function based on operating system (mostly Linux in case of this project):
 - Application Server (glassfish),
 - Database Server (mariadb),
 - Name Server (bind),
 - VPN Server (openvpn),
 - Web Server (apache2).

5.2.2.1 Node Client for VM-based VNFs

A software image for a virtual network functions includes a operating system (Debian 8 Linux is chosen for this project), service application and a node client, which communicates with the management server allowing management of the node.

These software images are spawned in the cloud environment as instances and are manageable by the management server using management interface.

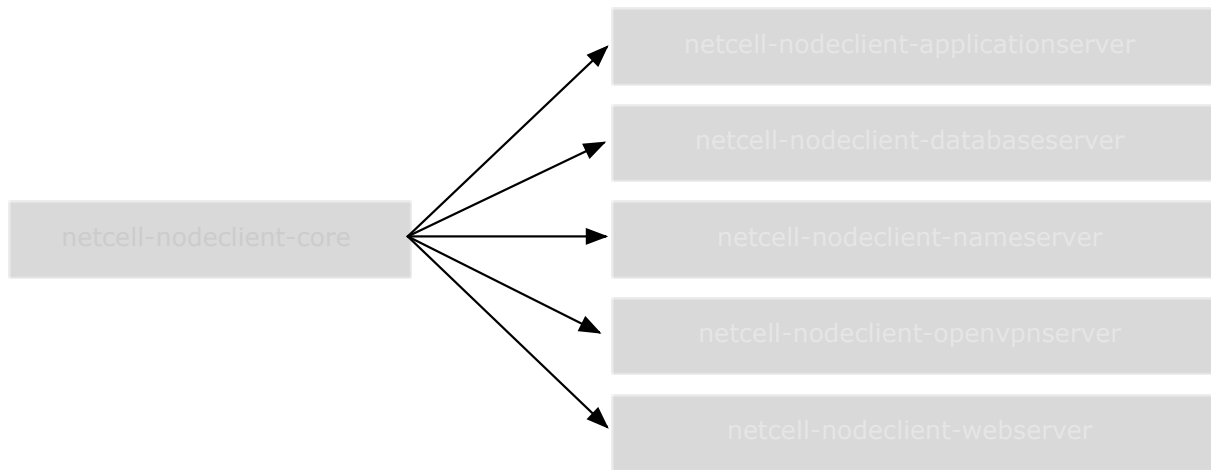


Figure 2 Node Client Core Dependency Diagram

Figure 2 show the dependency of different Node Client Application, which run within VNF instances and the core providing the common functionality. All these service Node Client Implementations use the same core, which communicates with the management server. This provides consolidated and consistent interface between spawned network functions and the management core, which provisions them

5.2.2.2 Management Server Modules for VNFs

Every VNF, either cloud-based or VM-based, requires a module within the management server, which allows for management of nodes of particular type. VM-based VNFs have their modules loaded within management server and allow for correct configuration and provisioning of the particular VNF type. Cloud-based VNFs also have the modules loaded within the management server. The difference in comparison with VM-based VNFs is, that cloud-based VNFs do not spawn instances in the underlying cloud platform, rather use the cloud platform’s built-in function to provide the services.

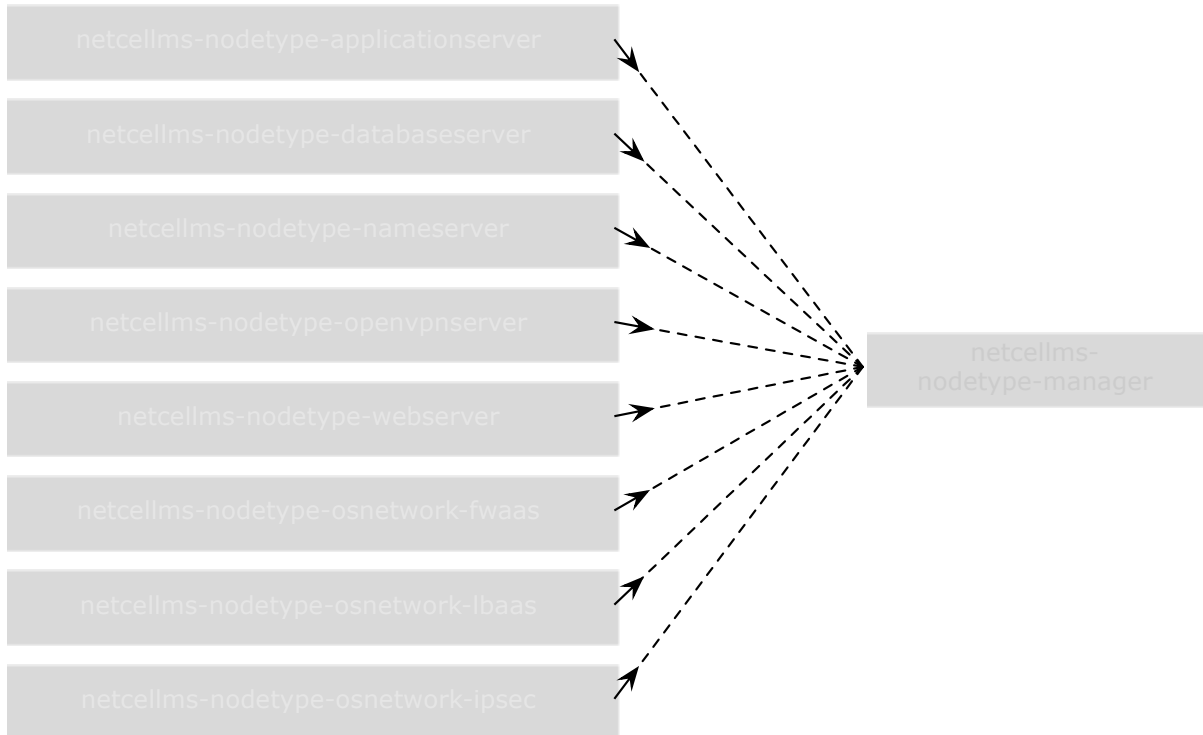


Figure 3 Node Type Manager Loaded Modules Diagram

Figure 3 depicts the Node Type Manager component of the management server and different Node Type Modules loaded in it. Node Type Manager provides services allowing other bundles to load a Node Type extensions, which provide VNF types implementation. Orange node types are VM-based VNFs, green ones are cloud-based VNFs. This diagram is an example depicting the currently developed modules.

5.2.2.3 VM-based VNFs and Management Server Communication

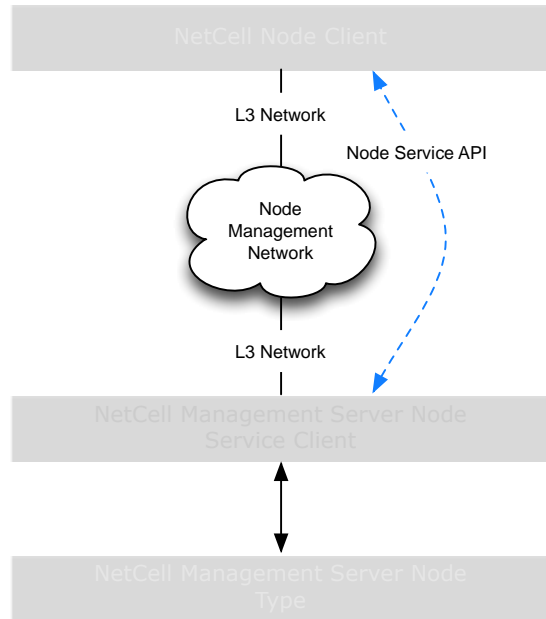


Figure 4 VM-based VNF and Management Server Communication Diagram

In Figure 4 the communication stack providing VM-based node management is depicted. Requests are made from the management server using the Node Type-specific library (blue box) using the Service Client library, which works as communication gateway for VM-based VNFs running the Node Client. The communication gateway uses L3 IP network to manage VNFs. Communication takes place using REST API named Node Service API (NSAPI).

5.3 Implementation

Chapter describes how NetCell modules are implemented, what kind of technologies is used and what kind of functionality is provided in various system components.

Currently, following NetCell main components are under development or are already released as finished parts of system:

5.3.1 NetCell Management Server

Management server is developed as service application developed on Java maven platform. Currently core foundation described in design is under development.

Following modules were successfully developed in management server component:

5.3.1.1 Configuration Module

The configuration component for the **NetCell Management Server** takes care of provisioning of the primary application configuration to other modules within the Jetty container, in which the server application runs. Primary configuration file is only a simple **.properties** file, which contains basic configuration information for other components. Configuration component consists of two modules **Service Provider Interface (SPI)** and **Implementation** for the configuration management.

Implementation takes care of processing of the configuration operations. This library examines system property **netcellms.configuration**, which needs to point to a configuration properties file, usually named **netcellms.properties**. This configuration file is usually placed in `conf/` folder where NetCell Launcher component is placed. This parameter can be configured by passing **-Dnetcellms.properties=path/to/file** to the Java executable.

Configuration module provides following service functionality:

- **Get configuration version.** Current version of configuration is returned. Version is incremented when new configuration is successfully stored.
- **Get configuration date.** Date, when configuration was last stored is returned. Date is returned as *java.util.Date* type.
- **Reload configuration.** Configuration is reloaded from configuration file.
- **Store configuration.** Newly stored configuration properties are stored into configuration file.
- **Get property.** Reads configuration property from loaded configuration file. Method can return configuration property as one of the following types:
 - String
 - Integer
 - Long
 - Date

- **Set property.** Sets configuration property to a new value. Method does not store property to the file.

5.3.1.2 Database Module

Database modules provide functionality of connecting and using of remote MongoDB NoSQL database. Database configuration is loaded by using configuration module. Database component consists of two modules Service Provider **Interface (SPI)** and **Implementation** for the configuration management. Database implementation is realized by using maven custom library called **Simple NoSQL Database (sldb)**.

Database components provides following services:

- **Database Configuration Service.** Service provides registration of java interfaces annotated with **DatabaseObject**. After registration of interface, collection space is created for registered database object. During startup of service, database connection is established. In case connection is not successfully created, service is not available for other core components.
- **Database Service.** Database services is designed to provide functionality for storing, modifying, deletion and listing of stored data in MongoDB database. Following dynamic methods are available for usage:
 - **isDatabaseConnected** – returns false or true status of database connection.
 - **object** – returns instance of database object, which can be used for insertion, modification or deletion of record from database.
 - **list** – returns multiple instances of database objects stored in java.util.List collection. Collection can be filled based on condition – if provided match key is containing provided matched value.

5.3.1.3 Web Server Module

Web service module provides a base for creation of API and portal services within the NetCell Management Server.

NetCell API Services allow for management of the following:

- API processing pipeline manipulation,
- API route manipulation,
- API response factoring.

API processing pipeline allows for extension of API processing line. In case other bundles will require special processing of API requests, the pipeline may be modified by the other modules. This will be used for example to integrate authentication mechanism in the API.

API routes represent paths on which REST objects reside within the API server. System uses RESTful standard of API based on the CRUD concept with the following methods and their respective HTTP alternatives used by the API server:

- CREATE → HTTP POST
- READ → HTTP GET
- UPDATE → HTTP PUT, HTTP PATCH
- DELETE → HTTP DELETE

API response factory is an internal system element allowing all API route handlers to generate well-formed responses having the same structure in all cases.

Routes to components to within the API fulfill the format **/api/v1/path/to/resource**. Here are examples of API routes:

- /api/v1/template/ (lists templates, allowed READ, CREATE)
- /api/v1/template/dfd43e7e-f67f-445f-905a-022b6935f8ef (shows template details, allowed READ, UPDATE, DELETE)

5.3.1.4 Event Logger Module

Event Logger component modules are responsible for storing, listing and registering events into management server database. Designed event is containing following properties:

- Event Severity. Event can be evaluated with following severities:
 - Emergency
 - Alert

- Critical
- Error
- Warning
- Notification
- Informational
- Debugging
- Event Type. Event type is defined with following properties:
 - Event Description
 - Event Name
 - Event Title
- Event Timestamp. Date and time of event occurrence stored in timestamp format.

Event logger service is providing functionality below:

- a) Register of Event type. New event type is created and stored into database.
- b) Store Event. Created event is stored into database.
- c) List of Events. Events are loaded by from database based on severity or event type.
- d) Load of one specified event type. One event type is loaded from database.

5.3.1.5 Identity Service Module

Identity service component is part of NetCell management server system, which is responsible for user and project management. Component provides services API calls designed for creation, deletion or modification of users and projects. With Identity Service Component we can also associate created users with projects.

Following API endpoints are implemented in this component:

Table 2: List of Identity Service API endpoints

| Endpoint | Methods | Description |
|--|---|--|
| /v1/identity/login |  | Performs user login and returns a session ID, which is used as an authenticator for other API calls. |
| /v1/identity/users |   | Allows creation and listing of users registered in system. |
| /v1/identity/users/<UID> |    | Allows information dump, update and deletion of particular user. |
| /v1/identity/users/<UID>/sessions |  | Listing of user sessions. |
| /v1/identity/users/<UID>/sessions/<SID> |   | Detail dump or session removal. |
| /v1/identity/projects |   | Endpoint allowing creation and listing of projects in the system. |
| /v1/identity/projects/<PID> |    | Detail, update and removal of existing projects. |
| /v1/identity/projects/<PID>/members |   | Listing and addition of project members. |
| /v1/identity/projects/<PID>/members/<UID> |    | Detail, update and removal of project membership |




5.3.1.6 OpenStack Manager Module

OpenStack manager module is component of NetCell system, which provides connection obtainment and management of OpenStack remote connection. OpenStack service stores information about connected OpenStack network in following DB records:

- Discovery Date
- Boolean if discovered network is management network.
- Boolean if discovered network is external network.

Following API endpoints are implemented in this module:

Table 3: List of OpenStack manager API endpoints

| Endpoint | Methods | Description |
|---|---|---|
| /v1/openstack/networks |  | Detailed information about all registered OpenStack |
| /v1/openstack/networks/<NID> |  | Possibility to update information about discovered OpenStack network. |
| /v1/openstack/connection |  | Provides details about managed OpenStack connection. |

5.3.1.7 Task Manager Service

Task manager is a dynamic service providing asynchronous execution long-running tasks. The service consists of the following components:

1. Task Queue,
2. Task Execution Thread Pool,
3. Tasks,

4. Task Steps,
5. Task Management API.

Task queue receives tasks, which consist of task steps and are queued for processing in task execution thread pool, which in a serial or parallel mode with concurrency limits, depending on current configuration.

Every task consists of task steps. Each step executes a portion of entire task. Task steps have a lifecycle consisting of the following states:

- **Created** – a created task step, which is not yet being executed,
- **Waiting** – task step, which is queued and will be executed,
- **Executing**,
- **Finished** – step, which had finished successfully,
- **Failed** – step on which the execution has failed.

As the execution of single task consists of multiple steps, each task has a lifecycle consisting of the following states:

- **Created** – a task has been created, but not yet queued for execution,
- **Queued** – task is waiting for execution,
- **Executing**,
- **Finished** – task, which had finished successfully,
- **Failed** – task execution has failed.

All tasks, which execution has finished are stored in a history database and are recallable at any point in time. All execution data including exact state change times are stored. This data may be obtained using API methods, which allow the following:

- Listing of queued and running tasks,
- Listing of history tasks,
- Retrieval of task or history task detailed information.

5.3.1.8 NetCell Launcher Module

NetCell Launcher module is main module, which sequentially launches rest of management core modules. Deployment of full management core application is made by launching this module compiled into java archive.

5.3.2 NetCell Node Client Core

Node Client Core is an application launched by the node in order to provide manageability by the NetCell Management Server.

NetCell node client core implements following functions:

Table 2 NetCell node client functions

| Function Name | Input Arguments | Returned Data | Description |
|-------------------------------|-----------------|---------------|--|
| nc_get_node_type_info | none | Dictionary | Method returns information about the current node type. These fields are required in the response: <ul style="list-style-type: none"> • nodeTypeName • nodeTypeDescription • nodeTypeVersion • nodeManagementIpAddress • nodeManagementPort • nodeCurrentTimestamp |
| nc_get_node_config_def | none | Dictionary | Method returns the configuration definition processed from file containing the definition. |
| nc_push_config | Element Tree | none | Method received the parsed XML in form of element tree and performs tasks required to load |

| | | | |
|---------------------------|--------|--------|--|
| | | | the new configuration to the node instance and restarts all required services. |
| nc_control_service | String | String | Starts, restarts, stops the service or returns the current service status. The following values for action are allowed: <ul style="list-style-type: none"> • start • stop • restart • status |

5.3.2.1 NetCell Node Client

Node type represents a type of instance, which can be deployed in the virtual topology or infrastructure. Different services offer different configuration basis. To provide a flexible environment for extension of the node types available for the NetCell IaaS, a dynamic configuration method has been chosen. Node type configuration definition declares a structure for configuration of the particular service. For instance, web server configuration has a different configuration than a VPN server. Every node can provide its own configuration definition (a template) using which the management server will construct display the configuration interface.

5.3.2.2 Node Type Configuration Definition

Basic principle of configuration definitions provides an extendible structure in JSON format offering basic field types, field groups and lists. The following field types are included in the version 1 of configuration definition syntax:

- List
- Group
- String
- Integer
- Boolean

- IP Address
- File
- Byte Size
- Password
- Options

Basic Field Definition

Fields are represented as JSON objects having minimal configuration or in special cases - more advanced configuration. Every field needs to contain the following information:

- **fieldName** - name of the field (lowercase, dash-separated is recommended)
- **fieldTitle** - human-readable title for the field
- **fieldDescription** - simple description on what the field configures
- **fieldType** - type of the field (may be one of listed above)

Every field can also contain following optional information fields:

- **fieldDefaultValue** - default value to be used for the field.
- **fieldRequired** - defines if configuration field is mandatory or optional. If field is not provided, configuration definition field is by default required.

All fields, but list, group, byte size, password and options currently comply to this configuration standard with these exceptions:

- For type Integer the fieldDefaultValue must be an integer JSON value.
- For type Boolean the fieldDefaultValue must be a Boolean JSON value.
- Field of type File cannot contain property fieldDefaultValue.

5.3.2.3 Netcell Node Client Core Post Install Script

Automatic post install script was developed to run application automatically after virtual machine will boot up. This scrip is compliant with Debian 8 **systemctl** method. It is system init and run manager, which allows running daemons. Our startup script supports methods these methods:

- Start - starts application and creates PID (process identification) file,

- Stop - reads PID file and runs kill command to stop application,
- Restart - do start and stop method,
- Status - do start and stop method.

Also it saves logging messages to /var/log/netcell.log file.

After a new node is spawned, post install script runs as a setup for the netcell node client module. It installs necessary packages and sets node module in a node client core module configuration json. Node modules consist of a 2 files: control file and parser. Control file provides basic functionality for webserver like start, stop, restart and status. Parser file serves as a parser for the incoming configuration files. Configuration files are in a form of xml files and are defined by configuration definition files available fully in Appendix 8.1.

5.3.2.4 Netcell Node Client Interface Configuration

Node Client Core supports configuration of virtual machine interfaces. API uses simple interface object in JSON format for communication with client server. Example of interface object is shown below:

```
{
  "eth0": {
    "netmask": "255.0.0.0",
    "ip_address": "127.0.0.1"
  }
}
```

All configured objects are stored into file named **nodeclient_interfaces.json**, which is dynamically created after first interface configuration. Netcell node client core after start runs Linux console command **ifconfig** for each interface object stored in this file. This process ensures persistent interface configuration.

Supported functions are listed in Table 4.

Table 4: Netcell Code Client Core interface configuration API functions

| HTTP method | Route | Parameters | Description |
|-------------|-----------------------------|--|--|
| GET | /interfaces/<interface> | interface - <i>optional parameter</i> , it is used for filtering specific interface configuration | Function return list of all (or specific) configuration added by administrator |
| POST | /interfaces/<interface> | interface - <i>optional parameter</i> , if it is used, specified interface will be updated | Function allows to send whole configuration to NetCell Node (old one will be removed). If interface parameter is used, interface will be updated (configuration of other interfaces stays same). |
| DELETE | /interfaces/<interface> | interface - <i>optional parameter</i> , if it is used, specified interface will be deleted | Using this function, whole configuration setted up by administrator will be cleared. If interface parameter is used, specific interface configuration will be deleted (configuration file + running configuration). |
| GET | /vps_interfaces/<interface> | interface - <i>optional</i> | Returns list of all interfaces in VPS |

| | | | |
|--|--|--|---|
| | | <i>parameter</i> , it is used for filtering specific interface configuration | containing information about interfaces, which was not configured by administrator using API functions (interfaces configured by DHCP, unconfigured interfaces etc.) |
|--|--|--|---|

5.3.3 Netcell Node Client Modules

The core of module implementation is dynamically loaded from **NetCell Node Client Core**. This core contains a method, which allows to:

- Get node type information – returns node information such as name, description, version, IP address, management port and timestamp,
- Get node type definition configuration – returns JSON file,
- Push node configuration – takes from argument XML file, which contains information for node configuration,
- Control service – allows starting, stopping, restarting and getting status of module service.

Important part is class **NodeConfigParser**, which has one public method **parse_configuration()**, which allows to iterate over XML elements and create service valid configuration file.

Design of parser allows improving module configuration possibilities in two steps:

- Modify **node_config_definition_file.json** file,
- Create protected method in **NodeConfigParser** class, which is mapped to **node_config_definition_file.json** using **validators** dictionary. This method must return line (or lines) of final configuration file. If this new options are file

type, method must also do conversion from base64 format and store it into file system.

5.3.3.1 Application Server Implementation

Node configuration definition for application server node defines only administration port, on which the service is manageable. Every command to glassfish goes through administration port and of course the service configuration commands executed by node. The administration port change needs to be executed by stopping the service with old, unchanged port, and started with the new port. For this purpose a configuration database was setup. The database contains table with listeners IP address and port. Node configuration definition is represented by `node_config_definition_file.json` file and can be viewed in appendix 8.1.1.

5.3.3.2 Database Server Implementation

Node configuration definition for the database server node is divided into logical groups defined by Mariadb configuration. The configuration is parsed and validated by node configuration parsing module. Then group settings are written into configuration file of the database server. Node configuration definition is represented by `node_config_definition.json` file and can be viewed in appendix 8.1.2. Node configuration definition is divided into following groups:

- Client settings - define default character set used by client.
- Client-server settings - define settings between client and server, such as TCP port of the Mariadb service or socket path for local client connections.
- Mysql settings - define default character set for the server.
- Mysqld settings - define server settings such as maximal size of cache and buffer memories or packet size limits.

Configuration generated by node parsing method rewrites the default Mariadb configuration. So the configuration given by user can be empty.

5.3.3.3 DNS Module Implementation

One of the Netcell Node Client type is also Domain Name Server service. This project use bind9 implementation for DNS. Nameserver Node Client implementation allows users to configure basic functionality as network functionality, forward and reverse zones with basic DNS records. This functionality is defined in file `node_config_definition.json` in Nameserver repository and can be viewed in attachment 8.1.3.

- **Network** – network configuration
 - **IP Address** – IP address on which is service operating
 - **TCP Port** – TCP on which is service listening
 - **UDP Port** – UDP port on which is service listening
- **Forward Zones** – configuration of forward zones
 - **Forward Records** – configuration of basic records in forward zone
 - **Type** – type of record
 - **Value** – value of record
- **Reverse Zones** – configuration of reverse zone
 - **Reverse Records** – basic reverse records configuration
 - **Host number** – host number (part of IP address)
 - **Host name** – host name – domain address of record
 - **NS records** – configuration of NS (subdomains)
 - **Host** – host number
- **SOA** – configuration of SOA record, this must be defined for each forward and reverse zone separately
 - **server** – domain of record
 - **admin** – admins mail address
 - **serial** – serial number
 - **refresh** – time to refresh
 - **retry** – time to retry
 - **expire** – time to expire
 - **ttd** – minimum time to live

5.3.3.4 OpenVPN module Implementation

VPN stand for virtual private network. In our project OpenVPN implementation is used for creation of VPN network node. At this point, node allows do configure basic VPN server configuration, which is represented by **node_config_definition_file.json** file and can be viewed in appendix 8.1.4.

Actual implementation allows to configure VPN node by following statements:

- Ip address – IP address to listen for connections on,
- Port – Port to listen for connection on,
- Proto – type of transport protocol,
- Dev – used network device (tun/tap) which determines layer of VPN tunnel (L2 or L3),
- SSL configuration – allows to configure cryptosystem by uploading public key, private key, certificate file and Diffie-Hellman parameters file,
- Network pool – prefix pool for allocation ip addresses to clients,
- Log level - verbosity log level,
- Compression – allows to use LZO compression,
- Kee-alive – hello and dead timers configuration,
- Topology mode – different IP/prefix addresses allocation,
- Nobody user and Nogroup configuration – change of user and group after VPN server starts,
- Tunnel persistency mode – persistent tunnel mode can be configured,
- Server network routes – configuration of server routing table,
- Client network routes – allows to configure clients routing tables,
- Cipher – different kinds of encryption ciphers can be used (allowed cyphers are dynamically loaded from **settings.py** file using command **vpn –show-ciphers**).

Module contains post install script **netcell_nodeclient_openvpnserver_postinsall.sh**, which will set up Debian instance as valid netcell node, which can be part of network topology.

5.3.3.5 Webserver module implementation

Webserver module provides nodes, created with this module, with capabilities of a standalone webserver. Core of the webserver is Apache server. Configuration through a configuration file of a webserver module is limited compared to all the options Apache offers, but there is still enough to set up. It offers basic configuration of a webserver plus various configurations of multiple virtual hosts. It is possible to configure following things:

- IP Address – IPv4 Address to listen for connections on,
- Port – Port to listen for connections on,
- Webserver General Configuration – Basic configuration of an underlying Apache server. Following parameters are configurable: Timeout, KeepAlive, MaxKeepAliveRequests, KeepAliveTimeout and HostnameLookups,
- Virtual Host – Provides options to set up various virtual hosts at once during deployment,
 - Virtual Host Type – There are 4 types of virtual hosts being configurable: Standard, Reverse, Proxy, Redirect and Load Balancer,
 - Configuration Type – Offers Name, IP, Port based configuration,
 - Basic configuration – It is possible to set up following configuration for each host separately with some limitation due to configuration type: hostname, IP, ports, aliases, log paths, directories and set of rules for each and ssl

Besides basic configuration shared among all virtual hosts it is possible to set up a virtual host with a specific purpose. These types are defined in Virtual Host Type and are: standard, reverse-proxy, redirect and load balancer virtual host.

Redirect virtual host offers redirection capabilities that can be applied to different path. Each redirection set of rules must contain: source path, destination path and redirection code. Similarly, Reverse Proxy configuration of a virtual host must contain paths for redirection. For a Load Balancer virtual host there must be set up a list of balance members. However, these balance members are not created by the node or module.

5.3.4 NetCell Node Configuration Core

NetCell node configuration core is library which provides methods designed for configuration definition and configuration handling.

Library functionality is available via static class **ConfigUtils** which provides following methods:

- **Configuration serialization.** Java configuration entities are serialized to JSON format. Serialized configuration is containing all information stored as parameters in configuration entities. Method returns JSON stored in **javax.json.JsonObject**.
- **Configuration deserialization.** Reverse process of serialization. Serialized JSON configuration is transformed into Java configuration entities.
- **Configuration XML generation.** Configuration parsed as a parameter to method **buildConfig** is transformed to XML format, which can be in future send to node clients. Method is using XML parsers and transformers provided by libraries **javax.xml**. XML is returned from method as **String**.
- **Default configuration creation.** Based on configuration definition, method creates **default configuration**. Configuration fields values are filled with default values which are part of JSON configuration definitions. If field default value is not provided, field in configuration is not created.
- **Configuration parsing.** Method is used for transforming **JSON configuration definition** received from Node clients to Java configuration definition entities. This method is also used in configuration deserialization. Validation of configuration definition is part of this functionality. In case configuration is not created correctly, parse configuration definition exception is thrown.

5.3.5 NetCell Web Portal

Netcell web portal is a web interface for Netcell product. Netcell web portal is developed for easier usage of the whole system. Technologies used in Netcell web portal are PHP, JavaScript, jquery, HTML and CSS. Netcell Web Portal is dived on several web subpages, which are accessible from dashboard, located on the left side. Netcell Web Portal contains following subpages:

- Dashboard – home page
- Profile – profile of current user
- User Management – list of created users – after clicking on line with user, this users profile can be modified in a new popup window
- Projects
- Projects – list of created projects – after clicking on line with project, this project can be modified in a new popup window
- Task History – list of the executed tasks – after clicking on line with tasks, new popup window with task details is shown
- Node Types – info page containing information about Node Clients Services
- About – information page about NetCell project
- Logout – log out from system

Users can access Netcell Web Portal from login page. This means, that users need to have login credentials and without them the system is unreachable.

5.4 Testing

Chapter describes information about implemented tests, which were realized during and after product development.

5.4.1 Unit Testing

To validate if developed components provide requested functionality, most of the modules of netcell architecture contain set of unit tests. Unit testing implementation is realized separately in every netcell component. Unit testing significantly helps us to catch special cases of component behavior, helps us implement exception and integration between components, such as management core, node-config core, node-client core etc.

5.4.1.1 Node Client Core Testing

Unit testing in node client core is realized with python framework **unittest**. Test part of node client core provides basic test cases which verifies, if various functionality is implemented correctly without any faults and bugs.

Following tests are implemented in this module:

NetcellNodeclientCoreTestCase – Basic test which is used in every other test cases in this module, raw data are extracted from response and validation of node type name and node type version is realized. Raw data are transformed to object and returned for further testing.

GetNodeTypeInfoTestCase – Checks if response which should return call *get node type info* is containing correct information – node type description, node management IP address, node management port, node current timestamp.

GetNodeConfigDefTestCase – Verifies if requesting *node configuration definition* is working correctly. In concrete, it tests if configuration version is extracted correctly, if configuration definition format version is correct and it also test error messages in case of invalid configuration definitions.

PushConfigTestCase – Test verifies if *push configuration* functionality is working correctly. Test covers how core behaves when valid XML is pushed to client core. It also provides tests of invalid XML formats – missing XML headers, XML with missing field etc.

ControlServiceTestCase – Testing of control messages. Functionality of start request, stop request, restart request, status request and unknown command request are successfully tested.

5.4.1.2 Node Configuration Core Testing

As it is described in design and implementation chapters, node configuration core is set of features which will be used in management core to handle configurations and configuration definitions. Testing is realized with **Junit framework**.

Following tests are implemented in this module:

ConfigBuilderTest. Checks if configuration stored as java object is correctly transformed to XML format with method **buildConfig**. Created XML is compared with stored sample XML.

ConfigDefParserTest. Checks if configuration definition stored in JSON format is correctly parsed to java entities with utility **parseConfigDef**. It also tests how program

behaves in case incorrect configuration definitions is being parsed – checks if exceptions are correctly thrown.

ConfigDeserializerTest. Verifies if serialized configuration definitions are correctly deserialized into java configuration entities via method **deserializeConfig**. Testing of exception functionality is included.

ConfigEntityTest. Provides tests which are responsible for checking if configuration definition entities are initialized correctly – all parameters are correctly set. Configuration navigation test cases are also part of this test.

ConfigSerializerTest. Testing of serialization functionality provided by method **serializeConfig**. Serialized configuration definition samples are compared with serialized entities.

DefaultConfigurationBuilderTest. Verification of building default configuration from configuration definition is part of this test. This functionality is provided by method **buildDefaultConfig**. Created default configuration is similarly as in previous tests compared with sample files included in test folder of the project.

5.4.2 Acceptance Testing

5.4.2.1 Definition of acceptance tests

5.4.2.1.1 Test Case 01: Create user

Prerequisites: none

Steps:

1. Login into system with admin user credentials:
 - a. username: admin
 - b. password: Password1
2. Go to “User Management” tab. This subpage should contain information about created users.
3. Try to modify some created user by clicking on line with specific user.
4. Create user without admin rights. Credentials:

- a. username: test1
 - b. password: Password1
5. Log off the system.

5.4.2.1.2 Test Case 02: Create new project

Prerequisites: Test Case 01

Steps:

1. Login into system with created user “test1”
2. Create new project name: “VPN Zones”
3. Try to modify some created project by click on specific project line. Project details are open in new subpage.
4. Go back to project page. Associate current user “test1” with created project “VPN Zones”

5.4.2.1.3 Test Case 03: Deploy Nameserver node client

Prerequisites:

- Instance with deployed Node Configuration Tool – Documentation of this tool can be found here:
<https://confluence.ctrdn.net/display/VNET/Node+Configuration+Tools>
- Empty virtual instance with installed Linux
- Test files: nameserver_test1.txt, nameserver_test2.txt

Steps:

1. Deploy Nameserver server Node client by running install script:
post_install_script_db.sh which is included in Nameserver Node Client repository. Instance should be restarted and service should be started after post install script execution.
2. Try get node status with Node Config tool - ./netcell-nodeconfig-tools.sh -c status -H “hostIP“ -p “port“
3. Try get node configuration definition - ./netcell-nodeconfig-tools.sh -d -H “hostIP“ -p “port“

4. Try push prepared XML test configuration file into instance with node configuration tool
 - a. Stop service - `./netcell-nodeconfig-tools.sh -c stop -H "hostIP" -p "port"`
 - b. Push configuration from prepared files for database - `./netcell-nodeconfig-tools.sh -P "test file" -H "hostIP" -p "port"`
 - c. Check nameserver service configuration.
5. Do following action with service
 - a. Start service - `./netcell-nodeconfig-tools.sh -c start -H "hostIP" -p "port"`
 - b. restart service - `./netcell-nodeconfig-tools.sh -c restart -H "hostIP" -p "port"`
 - c. get service status - `./netcell-nodeconfig-tools.sh -c status -H "hostIP" -p "port"`

5.4.2.1.4 Test Case 04: Deploy Application server node client

Prerequisites:

- Instance with deployed Node Configuration Tool – Documentation of this tool can be found here:
<https://confluence.ctrdn.net/display/VNET/Node+Configuration+Tools>
- Empty virtual instance with installed linux
- Test files: app_test1.txt, app_test2.txt

Steps:

1. Deploy Application server Node client by running install script: `post_install_script_db.sh` which is included in Application Server Node Client repository. Instance should be restarted and service should be started after post install script execution.
2. Try get node status with Node Config tool - `./netcell-nodeconfig-tools.sh -c status -H "hostIP" -p "port"`
3. Try get node configuration definition - `./netcell-nodeconfig-tools.sh -d -H "hostIP" -p "port"`

4. Try push prepared XML test configuration file into instance with node configuration tool
 - a. Stop service - `./netcell-nodeconfig-tools.sh -c stop -H "hostIP" -p "port"`
 - b. Push configuration from prepared files for database - `./netcell-nodeconfig-tools.sh -P "test file" -H "hostIP" -p "port"`
 - c. Check application server service configuration.
5. Do following action with service
 - a. Start service - `./netcell-nodeconfig-tools.sh -c start -H "hostIP" -p "port"`
 - b. restart service - `./netcell-nodeconfig-tools.sh -c restart -H "hostIP" -p "port"`
 - c. get service status - `./netcell-nodeconfig-tools.sh -c status -H "hostIP" -p "port"`

5.4.2.1.5 Test Case 05: Deploy Webserver node client

Prerequisites:

- Instance with deployed Node Configuration Tool – Documentation of this tool can be found here:
<https://confluence.ctrdn.net/display/VNET/Node+Configuration+Tools>
- Empty virtual instance with installed linux
- Test files: webserver_test1.txt, webserver_test2.txt

Steps:

1. Deploy Web Server server Node client by running install script: `post_install_script_db.sh` which is included in Web Server Node Client repository. Instance should be restarted and service should be started after post install script execution.
2. Try get node status with Node Config tool - `./netcell-nodeconfig-tools.sh -c status -H "hostIP" -p "port"`
3. Try get node configuration definition - `./netcell-nodeconfig-tools.sh -d -H "hostIP" -p "port"`

4. Try push prepared XML test configuration file into instance with node configuration tool
 - a. Stop service - `./netcell-nodeconfig-tools.sh -c stop -H "hostIP" -p "port"`
 - b. Push configuration from prepared files for database - `./netcell-nodeconfig-tools.sh -P "test file" -H "hostIP" -p "port"`
 - c. Check Web server service configuration.
5. Do following action with service
 - a. Start service - `./netcell-nodeconfig-tools.sh -c start -H "hostIP" -p "port"`
 - b. restart service - `./netcell-nodeconfig-tools.sh -c restart -H "hostIP" -p "port"`
 - c. get service status - `./netcell-nodeconfig-tools.sh -c status -H "hostIP" -p "port"`

5.4.2.1.6 Test Case 06: Deploy Database server node client

Prerequisites:

- Instance with deployed Node Configuration Tool – Documentation of this tool can be found here:
<https://confluence.ctrdn.net/display/VNET/Node+Configuration+Tools>
- Empty virtual instance with installed linux
- Test files: database_test1.txt, database_test2.txt

Steps:

1. Deploy Database server Node client by running install script: `post_install_script_db.sh` which is included in Database Node Client repository. Instance should be restarted and service should be started after post install script execution.
2. Try get node status with Node Config tool - `./netcell-nodeconfig-tools.sh -c status -H "hostIP" -p "port"`
3. Try get node configuration definition - `./netcell-nodeconfig-tools.sh -d -H "hostIP" -p "port"`

4. Try push prepared XML test configuration file into instance with node configuration tool
 - a. Stop service - `./netcell-nodeconfig-tools.sh -c stop -H "hostIP" -p "port"`
 - b. Push configuration from prepared files for database - `./netcell-nodeconfig-tools.sh -P "test file" -H "hostIP" -p "port"`
 - c. Check database service configuration.
5. Do following action with service
 - a. Start service - `./netcell-nodeconfig-tools.sh -c start -H "hostIP" -p "port"`
 - b. restart service - `./netcell-nodeconfig-tools.sh -c restart -H "hostIP" -p "port"`
 - c. get service status - `./netcell-nodeconfig-tools.sh -c status -H "hostIP" -p "port"`

5.4.2.1.7 Test Case 07: Deploy VPN server node client

Prerequisites:

- Instance with deployed Node Configuration Tool – Documentation of this tool can be found here:
<https://confluence.ctrdn.net/display/VNET/Node+Configuration+Tools>
- Empty virtual instance with installed linux
- Test files: vpn_test1.txt, vpn_test2.txt

Steps:

1. Deploy VPN server Node client by running install script: `post_install_script_db.sh` which is included in VPN Server Node Client repository. Instance should be restarted and service should be started after post install script execution.
2. Try get node status with Node Config tool - `./netcell-nodeconfig-tools.sh -c status -H "hostIP" -p "port"`
3. Try get node configuration definition - `./netcell-nodeconfig-tools.sh -d -H "hostIP" -p "port"`
4. Try push prepared XML test configuration file into instance with node configuration tool

- a. Stop service - `./netcell-nodeconfig-tools.sh -c stop -H "hostIP" -p "port"`
 - b. Push configuration from prepared files for database - `./netcell-nodeconfig-tools.sh -P "test file" -H "hostIP" -p "port"`
 - c. Check VPN server service configuration.
5. Do following action with service
- a. Start service - `./netcell-nodeconfig-tools.sh -c start -H "hostIP" -p "port"`
 - b. restart service - `./netcell-nodeconfig-tools.sh -c restart -H "hostIP" -p "port"`
 - c. get service status - `./netcell-nodeconfig-tools.sh -c status -H "hostIP" -p "port"`

5.4.2.1.8 Test Case 08: Node client management

Prerequisites: One of test cases 03-07

Steps:

1. Pick one of the created Node client modules
2. Try reboot Node client
3. Try shutdown Node client
4. Try turn on Node client

5.4.2.1.9 Test Case 09: Other web-portal functionality

Prerequisites: none

Steps:

1. Log in to the system
2. Go to "About" tab. This tab should contain information about whole project and web-portal.
3. Go to "Node Type" tab. This tab should contain information about all implemented network services in Node Clients.

5.4.2.1.10 Test Case 10: Tasks

Prerequisites: none

Steps:

1. Log into system.
2. Go to “Tasks” tab. This tab should contain list of executed tasks in ordered by time of execution.
3. Try to get detail information about one selected task by click on this task. Detail info should contain all executed steps in selected task.

5.4.2.2 Acceptance testing results

External person, who created following test result report, made acceptance testing.

Title: Test Specification

Testing user: Bc. Dušan Matejka –DevOps Admin Eset s.r.o.

Date: 30.4.2016

| Test Case | Steps | Result | Comment |
|-----------|-------|--------|---------------------------|
| 01 | 1 | OK | |
| | 2 | OK | |
| | 3 | OK | |
| | 4 | OK | |
| | 5 | OK | |
| 02 | 1 | OK | |
| | 2 | OK | |
| | 3 | OK | |
| | 4 | OK | |
| 03 | 1 | OK | |
| | 2 | OK | |
| | 3 | FAILED | 500 Internal Server Error |
| | 4 | OK | |
| | 5 | OK | |
| 04 | 1 | OK | |
| | 2 | OK | |
| | 3 | OK | |
| | 4 | OK | |
| | 5 | OK | |
| 05 | 1 | OK | |
| | 2 | OK | |
| | 3 | OK | |

| | | | |
|----|---|----|--|
| | 4 | OK | |
| | 5 | OK | 404 Not Found, Test case command need to be changed. |
| 06 | 1 | OK | |
| | 2 | OK | |
| | 3 | OK | 404 Not Found, Test case command need to be changed. |
| | 4 | OK | |
| | 5 | OK | |
| 07 | 1 | OK | |
| | 2 | OK | |
| | 3 | OK | |
| | 4 | OK | |
| | 5 | OK | |
| 08 | 1 | OK | |
| | 2 | OK | |
| | 3 | OK | |
| | 4 | OK | |
| 09 | 1 | OK | |
| | 2 | OK | |
| | 3 | OK | |
| 10 | 1 | OK | |
| | 2 | OK | |
| | 3 | OK | |

According to acceptance tests, following changes were made in NetCell system:

In Test case number 03, the testing of task number 3 failed with http message Internal Server Error and status 500. In test cases number 05 and 06 were some commands defined incorrectly and the service responded with http message Not Found and status 400. When I fix these commands, the server returned expected http message OK and status 200.

6 Prototypes

Several prototypes exist out of the development. System is basically separated in three parts:

1. NetCell Management Server
2. NetCell Portal
3. NetCell Node Clients

6.1 NetCell Management Server

NetCell Management Server takes care of entire system orchestration and prototype already exists provided as a testbed for the management server. Server contains following list of compiled Java packages.

- aopalliance-repackaged-2.3.0-b05.jar
- commons-codec-1.10.jar
- guava-17.0.jar
- hk2-api-2.3.0-b05.jar
- hk2-locator-2.3.0-b05.jar
- hk2-utils-2.3.0-b05.jar
- jackson-annotations-2.4.0.jar
- jackson-core-2.4.1.1.jar
- jackson-databind-2.4.1.3.jar
- jackson-dataformat-yaml-2.4.1.jar
- jackson-jaxrs-base-2.3.2.jar
- jackson-jaxrs-json-provider-2.3.2.jar
- jackson-module-jaxb-annotations-2.3.2.jar
- javassist-3.18.1-GA.jar
- javax.annotation-api-1.2.jar
- javax.inject-2.3.0-b05.jar
- javax.json-1.0.4.jar
- javax.servlet-api-3.1.0.jar
- javax.ws.rs-api-2.0.jar
- jersey-client-2.10.1.jar
- jersey-common-2.10.1.jar
- jersey-guava-2.10.1.jar

- jersey-media-json-jackson-2.11.jar
- jetty-http-9.3.0.v20150612.jar
- jetty-io-9.3.0.v20150612.jar
- jetty-security-9.3.0.v20150612.jar
- jetty-server-9.3.0.v20150612.jar
- jetty-servlet-9.3.0.v20150612.jar
- jetty-util-9.3.0.v20150612.jar
- jsr305-2.0.0.jar
- log4j-1.2.17.jar
- mongo-java-driver-3.1.0.jar
- netcellms-commons-1.0.0.jar
- netcellms-config-impl-1.0.0.jar
- netcellms-config-spi-1.0.0.jar
- netcellms-database-impl-1.0.0.jar
- netcellms-database-spi-1.0.0.jar
- netcellms-event-logger-impl-1.0.0.jar
- netcellms-event-logger-spi-1.0.0.jar
- netcellms-identity-impl-1.0.0.jar
- netcellms-identity-spi-1.0.0.jar
- netcellms-openstackmanager-impl-1.0.0.jar
- netcellms-openstackmanager-spi-1.0.0.jar
- netcellms-taskmanager-1.0.0.jar
- netcellms-webservices-impl-1.0.0.jar
- netcellms-webservices-spi-1.0.0.jar
- package-launcher-1.0.0.jar
- openstack4j-2.11.jar
- openstack4j-core-2.11.jar
- openstack4j-jersey2-2.11.jar
- org.apache.servicemix.bundles.reflections-0.9.10_3.jar
- org.osgi.core-4.3.1.jar

- osgi-resource-locator-1.0.1.jar
- pica_rit.txt
- slf4j-api-1.7.7.jar
- slf4j-log4j12-1.7.7.jar
- sndb-1.0.0.jar

6.2 NetCell Node Clients

NetCell Node Clients consists of Node Client Core, which is common for all VM-based node types. The other part is Node Client Implementation Module, which is specific for every node type. Currently prototypes supporting the following functionality exist:

- Retrieval of node type information,
- Retrieval of node configuration definition,
- Control of service status (start, stop, restart, status).
- Application Server Node Client configuration,
- Database Server Node Client configuration,
- Nameserver Node Client configuration,
- OpenVPN Node Client configuration,
- Web Server Node Client configuration.

7 Installation Guidelines

7.1 OpenStack

To achieve successful installation of NetCell infrastructure, following prerequisites needed to be secured in OpenStack network.

- Openstack network needs to be available for server where NetCell management core is located.
- OpenStack needs to contain Jessie Debian Image available for deployment to OpenStack network.
- OpenStack need to have available RAM and CPU capacity for successful management of OpenStack topologies.

7.2 NetCell Management Core

Netcell management core needs to be deployed into server environment with at least 515MB available RAM space for Java Virtual Machine. Server also needs to have Java Runtime Environment 1.8 installed in this server environment. It is highly recommended to run this system only on stable Linux environment.

Start of management server can be done by executing following command. Command needs to be executed with root or sudo permissions.

```
java -Dnetcellms.configuration=/path/to/config/file netcellms-launcher-1.0.0.jar
```

Following configuration directives needs to be configured in config file, which is located in execution command:

- **database.server.host.** Hostname of MongoDB database server.
- **database.server.port.** Port of MongoDB database server.
- **database.authentication.** Boolean value, if database authentication is required.
- **database.authentication.username.** Username of MongoDB user.
- **database.authentication.password.** Password of MongoDB user.

- **database.name.** Name of database where management core will store its data.
- **webservices.listen-host.** Hostname where management core will be run.
- **webservices.listen-port.** Port where management core will be run.
- **session.expire.time.** Time defined in seconds, when user session will expire after inactivity.
- **password.min.lenght.** Minimal password length.
- **password.complexity.required.** Boolean value, if password complexity is required.
- **openstack.keystone.endpoint.** Address of OpenStack keystone.
- **openstack.keystone.domain.** Domain of OpenStack keystone.
- **openstack.keystone.project.** Name of OpenStack project, which should be managed by management-core
- **openstack.keystone.username.** Username of OpenStack user.
- **openstack.keystone.password.** Password of OpenStack user.

7.3 NetCell Management Portal

NetCell Management Portal component needs to be deployed into server environment, which has stable connection to server where management-core module is located and deployed. Server also needs to have Web Server system (preferably Apache HTTP Web Server) and PHP interpreter installed.

To achieve stable connection with management core following configuration directive needs to be configured. Configuration is located in directory backend/portal_configuration.inc:

\$MANAGEMENT_PORTAL_URL. URL address where NetCell Management Core is deployed and running.

7.4 NetCell Node Client

This chapter contains basic information about preparation and starting service in the node. Currently possible operating systems are Debian and Ubuntu, but we recommend using Debian Jessie, which was the solution that was tested. Current Debian Image is

possible to download at <http://cdimage.debian.org/cdimage/openstack/>. Node services require at least 1 CPU, 512 MB of RAM memory and 8 GB of disk space. Network connection is also required for using implemented API.

After node deployment is possible to run post-install script which complete node service installation automatically. It is possible to run this script with node deployment in OpenStack. Each of supported node services delivers own post install script. These scripts are in appendixes below. The service is manageable by running following command:

```
/etc/init.d/netcell [start, stop, restart, status]
```

8 Appendices

8.1 Netcell Node Client Modules Configuration Definitions

8.1.1 Application Server Configuration Definition

```
{
  "configDefinitionVersion": 1,
  "configDefinitionFormatVersion": 1,
  "fields": [
    {
      "fieldDefaultValue": 4848,
      "fieldDescription": "TCP port for glassfish management",
      "fieldName": "admin-port",
      "fieldTitle": "Administration TCP Port",
      "fieldType": "integer"
    }
  ],
  "nodeTypeName": "applicationServer",
  "nodeTypeVersion": "1.0.0"
}
```

8.1.2 Database Server Configuration Definition

```
{
  "configDefinitionVersion": 1,
  "configDefinitionFormatVersion": 1,
  "fields": [
    {
      "fieldDescription": "This group information is passed to all MariaDB clients",
      "fieldName": "client-settings",
      "fieldTitle": "Client settings",
      "fieldType": "group",
      "fields": [
        {
          "fieldDefaultValue": "utf8",
          "fieldDescription": "Determines the character set for queries arriving from
the client",
          "fieldName": "client-charset",
          "fieldTitle": "Client charset",
          "fieldType": "string"
        }
      ]
    },
    {
      "fieldDescription": "This group is read both both by the client and the server",
      "fieldName": "client-server-settings",
      "fieldTitle": "Client-server settings",
      "fieldType": "group",
      "fields": [
        {
          "fieldDefaultValue": 3306,
          "fieldDescription": "TCP port to listen for HTTP connections on",
          "fieldName": "listen-port",
          "fieldTitle": "Listen TCP Port",
          "fieldType": "integer"
        }
      ]
    }
  ]
}
```

```

    },
    {
      "fieldDefaultValue": "/tmp/mysql.sock",
      "fieldDescription": "Path to socket file used for local client connections",
      "fieldName": "socket-path",
      "fieldTitle": "Socket path",
      "fieldType": "string"
    }
  ]
},
{
  "fieldDescription": "This group is only read by MariaDB servers",
  "fieldName": "mariadb-settings",
  "fieldTitle": "MariaDB server settings",
  "fieldType": "group",
  "fields": [
  ]
},
{
  "fieldDescription": "This group contains mysql settings",
  "fieldName": "mysql-settings",
  "fieldTitle": "Mysql settings",
  "fieldType": "group",
  "fields": [
    {
      "fieldDefaultValue": "utf8",
      "fieldDescription": "Character set used by the default server",
      "fieldName": "server-character-set",
      "fieldTitle": "Server character set",
      "fieldType": "string"
    }
  ]
},
{
  "fieldDescription": "This group is only for the mysqld standalone daemon",
  "fieldName": "mysqld-settings",
  "fieldTitle": "Mysqld settings",
  "fieldType": "group",
  "fields": [
    {
      "fieldDefaultValue": 2000,
      "fieldDescription": "Number of open tables for all threads",
      "fieldName": "table-open-cache",
      "fieldTitle": "Table open cache",
      "fieldType": "integer"
    }
  ],
  {
    "fieldDescription": "Group containing buffer settings",
    "fieldName": "buffer-settings",
    "fieldTitle": "Buffer settings",
    "fieldType": "group",
    "fields": [
      {
        "fieldDefaultValue": 134217728,
        "fieldDescription": "TCP port to listen for HTTP connections on",
        "fieldName": "key-buffer-size",
        "fieldTitle": "Key buffer size",
        "fieldType": "bytesize",
        "bytesizeOptions": {
          "minimumValue": 16777216,
          "maximumValue": 1073741824
        }
      }
    ]
  }
},

```

```

    {
      "fieldDefaultValue": 1048576,
      "fieldDescription": "Maximum size in bytes of a packet or a
generated/intermediate string",
      "fieldName": "max-allowed-packet",
      "fieldTitle": "Max allowed packet",
      "fieldType": "bytesize",
      "bytesizeOptions": {
        "minimumValue": 1024,
        "maximumValue": 1073741824
      }
    },
    {
      "fieldDefaultValue": 134217720,
      "fieldDescription": "Size of the buffer allocated when creating or sorting
indexes on a MyISAM table",
      "fieldName": "myisam-sort-buffer-size",
      "fieldTitle": "Myisam sort buffer size",
      "fieldType": "bytesize",
      "bytesizeOptions": {
        "minimumValue": 4096,
        "maximumValue": 18446744073709547520
      }
    },
    {
      "fieldDefaultValue": 16384,
      "fieldDescription": "The starting size for the connection and thread
buffers for each client thread",
      "fieldName": "net-buffer-length",
      "fieldTitle": "Net buffer length",
      "fieldType": "bytesize",
      "bytesizeOptions": {
        "minimumValue": 1024,
        "maximumValue": 1048576
      }
    },
    {
      "fieldDefaultValue": 131072,
      "fieldDescription": "Each thread performing a sequential scan allocates a
buffer of this size for each table scanned.",
      "fieldName": "read-buffer-size",
      "fieldTitle": "Read buffer size",
      "fieldType": "bytesize",
      "bytesizeOptions": {
        "minimumValue": 8200,
        "maximumValue": 2147479552
      }
    },
    {
      "fieldDefaultValue": 262144,
      "fieldDescription": "Size of the buffer used when reading rows from a
MyISAM table in sorted order after a key sort",
      "fieldName": "read-rnd-buffer-size",
      "fieldTitle": "Read rnd buffer size",
      "fieldType": "bytesize",
      "bytesizeOptions": {
        "minimumValue": 8200,
        "maximumValue": 2147479552
      }
    },
    {
      "fieldDefaultValue": 2097152,

```

```

        "fieldDescription": " Each session performing a sort allocates a buffer
with this amount of memory",
        "fieldName": "sort-buffer-size",
        "fieldTitle": "Sort buffer size",
        "fieldType": "bytesize",
        "bytesizeOptions": {
            "minimumValue": 16384,
            "maximumValue": 4294967296
        }
    }
}
]
}
]
},
"nodeTypeName": "databaseServer",
"nodeTypeVersion": "1.0.0"
}

```

8.1.3 DNS Configuration Definition

```

{
    "configDefinitionFormatVersion": 1,
    "configDefinitionVersion": 1,
    "fields": [
        {
            "fieldDescription": "Group containing network settings for DNS",
            "fieldName": "network",
            "fieldTitle": "Network settings",
            "fieldType": "group",
            "fields": [
                {
                    "fieldDefaultValue": 53,
                    "fieldDescription": "UDP port on which DNS is listening",
                    "fieldName": "udp-port",
                    "fieldTitle": "UDP Port",
                    "fieldType": "integer"
                },
                {
                    "fieldDefaultValue": 53,
                    "fieldDescription": "TCP port on which DNS is listening",
                    "fieldName": "tcp-port",
                    "fieldTitle": "TCP Port",
                    "fieldType": "integer"
                },
                {
                    "fieldDefaultValue": "0.0.0.0",
                    "fieldDescription": "IPv4 address to listen for connections on",
                    "fieldName": "listen-ip",
                    "fieldTitle": "Listen IP Address",
                    "fieldType": "string"
                }
            ]
        },
        {
            "fieldDescription": "Forward Zones for addresses",
            "fieldName": "forward-zones",
            "fieldTitle": "Forward Zones",
            "fieldType": "list",
            "listDefinition": {
                "childFieldName": "forward-zone",
                "displayFieldName": "zone-name",
            }
        }
    ]
}

```

```

"fields": [
  {
    "fieldDefaultValue": "example",
    "fieldDescription": "Name of the zone",
    "fieldName": "zone-name",
    "fieldTitle": "Zone name",
    "fieldType": "string"
  },
  {
    "fieldDescription": "Group containing settings of SOA record",
    "fieldName": "soa-record",
    "fieldTitle": "SOA Record",
    "fieldType": "group",
    "fields": [
      {
        "fieldDefaultValue": "example.com",
        "fieldDescription": "Name server for SOA record",
        "fieldName": "name-server",
        "fieldTitle": "Name Server",
        "fieldType": "string"
      },
      {
        "fieldDefaultValue": "admin.example.com",
        "fieldDescription": "Admin contact in SOA record",
        "fieldName": "admin-contact",
        "fieldTitle": "Admin Contact",
        "fieldType": "string"
      },
      {
        "fieldDefaultValue": 2,
        "fieldDescription": "Serial Number",
        "fieldName": "serial",
        "fieldTitle": "Serial",
        "fieldType": "integer"
      },
      {
        "fieldDefaultValue": 604800,
        "fieldDescription": "Time to refresh",
        "fieldName": "refresh",
        "fieldTitle": "Refresh",
        "fieldType": "integer"
      },
      {
        "fieldDefaultValue": 86400,
        "fieldDescription": "Time to retry",
        "fieldName": "retry",
        "fieldTitle": "Retry",
        "fieldType": "integer"
      },
      {
        "fieldDefaultValue": 2419200,
        "fieldDescription": "Time to expire",
        "fieldName": "expire",
        "fieldTitle": "Expire",
        "fieldType": "integer"
      },
      {
        "fieldDefaultValue": 604800,
        "fieldDescription": "Negative cache Time To Live",
        "fieldName": "minimum-ttl",
        "fieldTitle": "Minimum TTL",
        "fieldType": "integer"
      }
    ]
  }
]

```



```

    ]
  },
  {
    "fieldDescription": "Records in forward zone",
    "fieldName": "forward-records",
    "fieldTitle": "Forward Records",
    "fieldType": "list",
    "listDefinition": {
      "childFieldName": "forward-record",
      "displayFieldName": "forward-record-name",
      "fields": [
        {
          "fieldDefaultValue": "Default name",
          "fieldDescription": "Record name",
          "fieldName": "forward-record-name",
          "fieldTitle": "Forward record name",
          "fieldType": "string"
        },
        {
          "fieldDefaultValue": "example.com",
          "fieldDescription": "URL Name",
          "fieldName": "name",
          "fieldTitle": "Name",
          "fieldType": "string"
        },
        {
          "fieldDefaultValue": "A",
          "fieldDescription": "Type of record",
          "fieldName": "record-type",
          "fieldTitle": "Record Type",
          "fieldType": "string"
        },
        {
          "fieldDefaultValue": "0.0.0.0",
          "fieldDescription": "Translated value",
          "fieldName": "value",
          "fieldTitle": "value",
          "fieldType": "string"
        }
      ]
    }
  }
]
},
{
  "fieldDescription": "Reverse zones for addresses",
  "fieldName": "reverse-zones",
  "fieldTitle": "Reverse Zones",
  "fieldType": "list",
  "listDefinition": {
    "childFieldName": "reverse-zone",
    "displayFieldName": "zone-name",
    "fields": [
      {
        "fieldDefaultValue": "example",
        "fieldDescription": "Name of the zone",
        "fieldName": "zone-name",
        "fieldTitle": "Zone name",
        "fieldType": "string"
      }
    ]
  }
},
{
  "fieldDescription": "Group containing settings of SOA record",

```

```

"fieldName": "soa-record",
"fieldTitle": "SOA Record",
"fieldType": "group",
"fields": [
  {
    "fieldDefaultValue": "example.com",
    "fieldDescription": "Name server for SOA record",
    "fieldName": "name-server",
    "fieldTitle": "Name Server",
    "fieldType": "string"
  },
  {
    "fieldDefaultValue": "admin.example.com",
    "fieldDescription": "Admin contact in SOA record",
    "fieldName": "admin-contact",
    "fieldTitle": "Admin Contact",
    "fieldType": "string"
  },
  {
    "fieldDefaultValue": 2,
    "fieldDescription": "Serial Number",
    "fieldName": "serial",
    "fieldTitle": "Serial",
    "fieldType": "integer"
  },
  {
    "fieldDefaultValue": 604800,
    "fieldDescription": "Time to refresh",
    "fieldName": "refresh",
    "fieldTitle": "Refresh",
    "fieldType": "integer"
  },
  {
    "fieldDefaultValue": 86400,
    "fieldDescription": "Time to retry",
    "fieldName": "retry",
    "fieldTitle": "Retry",
    "fieldType": "integer"
  },
  {
    "fieldDefaultValue": 2419200,
    "fieldDescription": "Time to expire",
    "fieldName": "expire",
    "fieldTitle": "Expire",
    "fieldType": "integer"
  },
  {
    "fieldDefaultValue": 604800,
    "fieldDescription": "Negative cache Time To Live",
    "fieldName": "minimum-ttl",
    "fieldTitle": "Minimum TTL",
    "fieldType": "integer"
  }
]
},
{
"fieldDescription": "Records in reverse zone",
"fieldName": "reverse-records",
"fieldTitle": "Reverse Records",
"fieldType": "list",
"listDefinition": {
  "childFieldName": "reverse-record",
  "displayFieldName": "reverse-record-name",

```

```

"fields": [
  {
    "fieldDefaultValue": "Reverse record name",
    "fieldDescription": "Name of the reverse record",
    "fieldName": "reverse-record-name",
    "fieldTitle": "Reverse record name",
    "fieldType": "string"
  },
  {
    "fieldDefaultValue": 10,
    "fieldDescription": "Address of Host from IP address",
    "fieldName": "host-address",
    "fieldTitle": "Host Address",
    "fieldType": "integer"
  },
  {
    "fieldDefaultValue": "example.com",
    "fieldDescription": "Name of Host - URL address",
    "fieldName": "host-name",
    "fieldTitle": "Host Name",
    "fieldType": "string"
  }
]
},
{
  "fieldDescription": "NS Records in reverse zone",
  "fieldName": "ns-records",
  "fieldTitle": "NS records",
  "fieldType": "list",
  "listDefinition": {
    "childFieldName": "ns-record",
    "displayFieldName": "ns-record-name",
    "fields": [
      {
        "fieldDefaultValue": "NS record name",
        "fieldDescription": "Name of the NS record",
        "fieldName": "ns-record-name",
        "fieldTitle": "NS record name",
        "fieldType": "string"
      },
      {
        "fieldDefaultValue": "something.",
        "fieldDescription": "Name of Host - URL address",
        "fieldName": "host-name",
        "fieldTitle": "Host Name",
        "fieldType": "string"
      }
    ]
  }
}
]
},
{
  "nodeTypeName": "NameServer",
  "nodeTypeVersion": "1.0.0"
}

```

8.1.4 VPN Configuration Definition

```
{
```

```

"configDefinitionVersion": 1,
"configDefinitionFormatVersion": 1,
"nodeTypeName": "OpenVPN server",
"nodeTypeVersion": "1.0.0",
"fields": [
  {
    "fieldDefaultValue": "0.0.0.0",
    "fieldDescription": "IPv4 address to listen for connections on",
    "fieldName": "listen-ip",
    "fieldRequired": false,
    "fieldTitle": "Listen IP Address",
    "fieldType": "string"
  },
  {
    "fieldDefaultValue": 1194,
    "fieldDescription": "Port to listen for connections on",
    "fieldName": "listen-port",
    "fieldRequired": true,
    "fieldTitle": "Listen Port",
    "fieldType": "integer"
  },
  {
    "fieldDefaultValue": "udp",
    "fieldDescription": "Type of protocol for OpenVPN service",
    "fieldName": "proto",
    "fieldRequired": true,
    "fieldTitle": "Transmission protocol",
    "fieldType": "options",
    "fieldOptions": [
      {
        "id": "udp",
        "title": "UDP",
        "description": "Uses UDP protocol for OpenVPN service"
      },
      {
        "id": "tcp",
        "title": "TCP",
        "description": "Uses TCP protocol for OpenVPN service"
      }
    ]
  },
  {
    "fieldDefaultValue": "tun",
    "fieldDescription": "Determining whether to use a routed or bridged VPN",
    "fieldName": "dev",
    "fieldRequired": true,
    "fieldTitle": "Device",
    "fieldType": "options",
    "fieldOptions": [
      {
        "id": "tun",
        "title": "Tun",
        "description": "Uses virtual ethernet adapter"
      },
      {
        "id": "tap",
        "title": "Tap",
        "description": "Uses virtual point-to-point IP link"
      }
    ]
  },
  {
    "fieldDescription": "Group containing SSL-related configuration",

```

```

"fieldName": "ssl",
"fieldRequired": true,
"fieldTitle": "SSL Security",
"fieldType": "group",
"fields": [
  {
    "fieldDescription": "PEM-formatted certification authority public key",
    "fieldName": "ca",
    "fieldRequired": true,
    "fieldTitle": "SSL Certificate File",
    "fieldType": "file"
  },
  {
    "fieldDescription": "PEM-formatted public-key signed by certification
authority",
    "fieldName": "cert",
    "fieldRequired": true,
    "fieldTitle": "SSL server public-key File",
    "fieldType": "file"
  },
  {
    "fieldDescription": "PEM-formatted private-key signed by certification
authority",
    "fieldName": "key",
    "fieldRequired": true,
    "fieldTitle": "SSL server private-key File",
    "fieldType": "file"
  },
  {
    "fieldDescription": "PEM-formatted diffie-hellman file",
    "fieldName": "dh",
    "fieldRequired": true,
    "fieldTitle": "DH parameters",
    "fieldType": "file"
  }
]
},
{
  "fieldDescription": "Prefix pool for allocation ip addresses",
  "fieldName": "network",
  "fieldTitle": "Network pool",
  "fieldRequired": true,
  "fieldType": "group",
  "fields": [
    {
      "fieldDefaultValue": "10.8.0.0",
      "fieldDescription": "IP address of network pool for allocation of IP addresses
for clients",
      "fieldName": "prefix-address",
      "fieldRequired": true,
      "fieldTitle": "IP address of network pool",
      "fieldType": "string"
    },
    {
      "fieldDefaultValue": "255.255.255.0",
      "fieldDescription": "Subnet mask of network address pool",
      "fieldName": "prefix-mask",
      "fieldRequired": true,
      "fieldTitle": "Subnet mask",
      "fieldType": "string"
    }
  ]
}
],
},

```

```

{
  "fieldDefaultValue": 3,
  "fieldDescription": "Verbosity log level",
  "fieldName": "verb",
  "fieldRequired": false,
  "fieldTitle": "Verbosity",
  "fieldType": "integer"
},
{
  "fieldDefaultValue": false,
  "fieldDescription": "Use fast LZO compression, may add up to 1 byte per packet for
incompressible data",
  "fieldName": "comp-lzo",
  "fieldRequired": false,
  "fieldTitle": "Compression",
  "fieldType": "boolean"
},
{
  "fieldDescription": "Group for keep alive mechanism using ping",
  "fieldName": "keep-alive",
  "fieldRequired": false,
  "fieldTitle": "Keep alive",
  "fieldType": "group",
  "fields": [
    {
      "fieldDefaultValue": 10,
      "fieldDescription": "Ping interval in seconds",
      "fieldName": "ping-interval",
      "fieldRequired": true,
      "fieldTitle": "Ping interval",
      "fieldType": "integer"
    },
    {
      "fieldDefaultValue": 60,
      "fieldDescription": "Trigger a SIGUSR1 restart after n seconds pass without
reception of a ping or other packet from remote.",
      "fieldName": "dead-interval",
      "fieldRequired": true,
      "fieldTitle": "Dead interval",
      "fieldType": "integer"
    }
  ]
},
{
  "fieldDefaultValue": "net30",
  "fieldDescription": "Several network topologies exist for servers configured to
accept multiple client connections.",
  "fieldName": "topology",
  "fieldRequired": false,
  "fieldTitle": "Topology",
  "fieldType": "options",
  "fieldOptions": [
    {
      "id": "net30",
      "title": "Net30",
      "description": "Allocation of virtual /30 subnet for each client"
    },
    {
      "id": "subnet",
      "title": "Subnet",
      "description": "Addressing is done by IP & netmask"
    }
  ]
}

```

```

        "id": "p2p",
        "title": "Point-to-Point",
        "description": "Topology uses Point-to-Point networking"
    }
]
},
{
    "fieldDefaultValue": false,
    "fieldDescription": "drop privileges to user nobody",
    "fieldName": "user-nobody",
    "fieldRequired": false,
    "fieldTitle": "User nobody",
    "fieldType": "boolean"
},
{
    "fieldDefaultValue": false,
    "fieldDescription": "drop privileges to group nobody",
    "fieldName": "group-nobody",
    "fieldRequired": false,
    "fieldTitle": "Group nobody",
    "fieldType": "boolean"
},
{
    "fieldDefaultValue": false,
    "fieldDescription": "Don't close and reopen TUN/TAP device or run up/down scripts
across SIGUSR1",
    "fieldName": "persist-tun",
    "fieldRequired": false,
    "fieldTitle": "Persist tun/tap",
    "fieldType": "boolean"
},
{
    "fieldDefaultValue": false,
    "fieldDescription": "Don't re-read key files across SIGUSR1",
    "fieldName": "persist-key",
    "fieldRequired": false,
    "fieldTitle": "Persist key",
    "fieldType": "boolean"
},
{
    "fieldDescription": "Routes for OpenVPN server",
    "fieldName": "server-routes",
    "fieldRequired": false,
    "fieldTitle": "Server routes",
    "fieldType": "list",
    "listDefinition": {
        "childFieldName": "server-route",
        "displayFieldName": "server-route-address",
        "fields": [
            {
                "fieldDefaultValue": "0.0.0.0",
                "fieldDescription": "Network IP address",
                "fieldName": "server-route-address",
                "fieldRequired": true,
                "fieldTitle": "IP address for route",
                "fieldType": "string"
            },
            {
                "fieldDefaultValue": "255.255.255.255",
                "fieldDescription": "Subnet mask",
                "fieldName": "server-route-mask",
                "fieldRequired": true,
                "fieldTitle": "Subnet mask for route",
            }
        ]
    }
}

```



```

"fields": [
  {
    "fieldDefaultValue": 80,
    "fieldDescription": "TCP port to listen for HTTP connections on",
    "fieldName": "tcp-port",
    "fieldRequired": true,
    "fieldTitle": "TCP Port Number",
    "fieldType": "integer"
  },
  {
    "fieldDefaultValue": false,
    "fieldDescription": "Enable or disable SSL on the TCP port",
    "fieldName": "ssl-toggle",
    "fieldRequired": true,
    "fieldTitle": "Enable SSL",
    "fieldType": "boolean"
  }
]
},
{
  "fieldDescription": "Group containing General Webserver configuration",
  "fieldName": "gereral-config",
  "fieldRequired": false,
  "fieldTitle": "General Configuration",
  "fieldType": "group",
  "fields": [
    {
      "fieldDefaultValue": 300,
      "fieldDescription": "The number of seconds before receives and sends time
out",
      "fieldName": "server-timeout",
      "fieldRequired": false,
      "fieldTitle": "Timeout",
      "fieldType": "integer"
    },
    {
      "fieldDefaultValue": "On",
      "fieldDescription": "Wheter or not to allow persistent connections",
      "fieldName": "server-keepalive",
      "fieldRequired": false,
      "fieldTitle": "KeepAlive",
      "fieldType": "string"
    },
    {
      "fieldDefaultValue": 100,
      "fieldDescription": "Maximum number of request to be kept alive",
      "fieldName": "server-maxkeepalive-requests",
      "fieldRequired": false,
      "fieldTitle": "MaxKeepAliveRequests",
      "fieldType": "integer"
    },
    {
      "fieldDefaultValue": 5,
      "fieldDescription": "Timeout time for connected clients",
      "fieldName": "server-keepalive-timeout",
      "fieldRequired": false,
      "fieldTitle": "KeepAliveTimeout",
      "fieldType": "integer"
    },
    {
      "fieldDefaultValue": "On",
      "fieldDescription": "Enable Hostname Lookups",

```



```

    {
      "id": "reverse-proxy",
      "title": "Reverse Proxy",
      "description": "Reverse proxies services within internal network"
    },
    {
      "id": "redirect",
      "title": "Redirect",
      "description": "Redirects users accessing the portal to other URL"
    },
    {
      "id": "load-balancer",
      "title": "Load Balancer",
      "description": "Acts as a load balancer in network"
    }
  ]
},
{
  "fieldDefaultValue": "admin@example",
  "fieldDescription": "Name of the Virtual Host admin",
  "fieldName": "vh-server-admin",
  "fieldRequired": false,
  "fieldTitle": "Virtual Host Server Admin",
  "fieldType": "string"
},
{
  "fieldDefaultValue": "/path/to/doc/root",
  "fieldDescription": "Path to document root of the Virtual Host",
  "fieldName": "vh-document-root",
  "fieldRequired": true,
  "fieldTitle": "Virtual Host Document Root",
  "fieldType": "string"
},
{
  "fieldDefaultValue": "example.com",
  "fieldDescription": "Name of a Virtual Host",
  "fieldName": "vh-server-name",
  "fieldRequired": false,
  "fieldTitle": "Virtual Host Server Name",
  "fieldType": "string"
},
{
  "fieldDescription": "List of alternative names for Virtual Host",
  "fieldName": "vh-alias-list",
  "fieldRequired": false,
  "fieldTitle": "List of Virtual Host Aliases",
  "fieldType": "list",
  "listDefinition": {
    "childFieldName": "vh-alias-list-field",
    "displayFieldName": "vh-alias-record",
    "fields": [
      {
        "fieldDefaultValue": "example1.com",
        "fieldDescription": "Alternative name for Virtual Host",
        "fieldName": "vh-alias-record",
        "fieldRequired": false,
        "fieldTitle": "Virtual Host Alias",
        "fieldType": "string"
      }
    ]
  }
}
},
{

```

```

    "fieldDefaultValue": "/path/to/error/log",
    "fieldDescription": "Path to Virtual Host ErrorLog",
    "fieldName": "vh-errorlog-path",
    "fieldRequired": false,
    "fieldTitle": "Virtual Host ErrorLog Path",
    "fieldType": "string"
  },
  {
    "fieldDefaultValue": "/path/to/custom/log",
    "fieldDescription": "Path to Virtual Host CustomLog",
    "fieldName": "vh-customlog-path",
    "fieldRequired": false,
    "fieldTitle": "Virtual Host CustomLog Path",
    "fieldType": "string"
  },
  {
    "fieldDefaultValue": "10.0.0.1",
    "fieldDescription": "IP address on which the service is available",
    "fieldName": "vh-ip-address",
    "fieldRequired": false,
    "fieldTitle": "Virtual Host IP Address",
    "fieldType": "string"
  },
  {
    "fieldDefaultValue": "example.com",
    "fieldDescription": "Hostname on which the service is available",
    "fieldName": "vh-name",
    "fieldRequired": false,
    "fieldTitle": "Virtual Host DNS Name",
    "fieldType": "string"
  },
  {
    "fieldDefaultValue": 80,
    "fieldDescription": "Port number on which the service is available",
    "fieldName": "vh-port",
    "fieldRequired": false,
    "fieldTitle": "Virtual Host Port",
    "fieldType": "integer"
  },
  {
    "fieldDescription": "List of directory settings applied on Virtual Host",
    "fieldName": "vh-directory-list",
    "fieldRequired": false,
    "fieldTitle": "Virtual Host Directory Settings List",
    "fieldType": "list",
    "listDefinition": {
      "childFieldName": "vh-directory-list-field",
      "displayFieldName": "vh-directory-path",
      "fields": [
        {
          "fieldDefaultValue": "/path/to/directory",
          "fieldDescription": "Path to the directory",
          "fieldName": "vh-directory-path",
          "fieldRequired": false,
          "fieldTitle": "Directory Path",
          "fieldType": "string"
        },
        {
          "fieldDescription": "List of Virtual Host settings applied on the
directory",
          "fieldName": "vh-directory-option-list",
          "fieldRequired": false,
          "fieldTitle": "List of Virtual Host Directory Options",

```

```

    "fieldType": "list",
    "listDefinition": {
      "childFieldName": "vh-directory-option-list-field",
      "displayFieldName": "vh-directory-option",
      "fields": [
        {
          "fieldDefaultValue": "Option OptionSetting",
          "fieldDescription": "Option to be applied to the directory",
          "fieldName": "vh-directory-option",
          "fieldRequired": false,
          "fieldTitle": "Directory Option",
          "fieldType": "string"
        }
      ]
    }
  }
}
},
{
  "fieldDescription": "Group containing SSL-related configuration",
  "fieldName": "vh-ssl-conf",
  "fieldRequired": false,
  "fieldTitle": "VH SSL Configuration",
  "fieldType": "group",
  "fields": [
    {
      "fieldDefaultValue": false,
      "fieldDescription": "SSL enabled if checked",
      "fieldName": "ssl-enable",
      "fieldTitle": "Enable SSL",
      "fieldType": "boolean"
    },
    {
      "fieldDefaultValue": "aGVsbG8K",
      "fieldDescription": "PEM-formatted certificate",
      "fieldName": "ssl-certificate-file",
      "fieldRequired": false,
      "fieldTitle": "SSL Certificate File",
      "fieldType": "file"
    },
    {
      "fieldDefaultValue": "aGVsbG8K",
      "fieldDescription": "PEM-formatted certificate key",
      "fieldName": "ssl-certificate-key-file",
      "fieldRequired": false,
      "fieldTitle": "SSL Certificate Key File",
      "fieldType": "file"
    },
    {
      "fieldDescription": "Private key passphrase",
      "fieldName": "ssl-certificate-key-passphrase",
      "fieldRequired": false,
      "fieldTitle": "SSL Certificate Key Passphrase",
      "fieldType": "password",
      "passwordOptions": {
        "generatorEnabled": false,
        "generatorLength": 8
      }
    }
  ]
}
},
{

```

```

    "fieldDescription": "Group containing Virtual Host configuration for
redirection",
    "fieldName": "vh-redirect-group",
    "fieldRequired": false,
    "fieldTitle": "Virtual Host redirection",
    "fieldType": "group",
    "fields": [
      {
        "fieldDescription": "Redirection rules",
        "fieldName": "redirect-rule-list",
        "fieldRequired": false,
        "fieldTitle": "Redirection rules",
        "fieldType": "list",
        "listDefinition": {
          "childFieldName": "redirection-rule",
          "displayFieldName": "rule-setting",
          "fields": [
            {
              "fieldDefaultValue": "Redirect",
              "fieldDescription": "Rule Setting",
              "fieldName": "rule-setting",
              "fieldRequired": true,
              "fieldTitle": "Rule Setting",
              "fieldType": "string"
            },
            {
              "fieldDefaultValue": 302,
              "fieldDescription": "Redirection code configuration",
              "fieldName": "redirection-code",
              "fieldRequired": true,
              "fieldTitle": "Redirection Code",
              "fieldType": "integer"
            },
            {
              "fieldDefaultValue": "/path/to/be/redirected",
              "fieldDescription": "Path to be redirected",
              "fieldName": "redirection-src",
              "fieldRequired": true,
              "fieldTitle": "Redirection Path",
              "fieldType": "string"
            },
            {
              "fieldDefaultValue": "/redirection/destination",
              "fieldDescription": "Redirection destination",
              "fieldName": "redirection-dst",
              "fieldRequired": true,
              "fieldTitle": "Destination Address",
              "fieldType": "string"
            }
          ]
        }
      }
    ]
  },
  {
    "fieldDescription": "Reverse Proxy Virtual Host configuration",
    "fieldName": "vh-reverse-proxy-configuration",
    "fieldRequired": false,
    "fieldTitle": "Reverse Proxy configuration",
    "fieldType": "group",
    "fields": [
      {
        "fieldDefaultValue": "/path/to/be/redirected",

```

