SoftFIRE

Software Defined Networks and Network Function Virtualization Testbed within FIRE+

Grant Agreement № 687860

Handbook: programming and using the SoftFIRE middleware

Version:	1.0.
Due Date:	November 3rd, 2017
Delivery Date:	November 3rd, 2017
Type:	Report (R)
Dissemination Level:	PU
Lead partner:	TUB
Authors:	All Partners (See List of Contributors below)
Internal reviewers:	Roberto Minerva (EIT), Giuseppe Carella (TUB)

Disclaimer

SoftFIRE has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement no. 687860.

Version Control:

Version	Date	Author	Author's Organization	Changes
0.9	31.10.2017	Giuseppe Carella	TUB	Generate final version of deliverable D2.4
1.0	03.11.2017	Roberto Minerva	EIT Digital	Adaptation to become a Handbook

Contributors:

Contributor	Partner
Lorenzo Tomasini	TUB
Giuseppe Carella	TUB
Bjoern Riemer	FOKUS
Roberto Minerva	EIT
Thomas Briedigkeit	FOKUS
Massimiliano Romano	ADS
George Kamel	UoS
Serdar Vural	UoS
Marco Rossi	Ericsson
Tiziano Lazzeri	Ericsson

Executive Summary

The SoftFIRE project goal is to connect together different testbeds under the standard de facto technologies in the field of Network Function Virtualization (NFV) and Software Defined Networks (SDN), providing to experimenters an easy access based on common federated APIs. In order to achieve this goal, it has been decided to develop, improve and combine multiple software stacks.
In particular, SoftFIRE aims at (i) integrating and extending different open source technologies in order to provide a comprehensive reference software middleware for federating heterogeneous individual testbeds, (ii) providing a live federated infrastructure based on these technologies developed and integrated, and ease the possibility to running different experiments on the federated platform.
For these reasons, this deliverable first provides a description of the SoftFIRE middleware main functions and their design, and second it shows how the experimenters can interact with the running infrastructure. Considering that the SoftFIRE middleware will continuously evolve beyond the current version proposed in this document, a live version of the documentation has been offered at the URL: http://docs.softfire.eu/.

This document (wholly based on Deliverable D2.4 – "SoftFIRE (v3) usage manual for NFV/SDN/MEC and 5G experimenters") provides a complete description of the new SoftFIRE middleware that has been developed in order to overcome some issues in using older tools of the FIRE community. The SoftFIRE project initially adopted FIRE interfaces (i.e. SFA) based on the FITeagle toolkit. In fact, SoftFIRE project, starting from the second open call, has undertaken a new approach. After the first open call, based on the feedbacks received from the experimenters, the SoftFIRE team decided to move towards standard NFV/SDN interfaces (i.e. TOSCA) in order to simplify the way experimenters, interact with the platform. This document describes the middleware developed according to the newer technologies and developments related to the evolution and standardization of NFV/SDN. Some details about the migration from SFA to TOSCA are explained in section 2.1.
This document is particularly relevant for participants to both, SoftFIRE's third Open Call and to the SoftFIRE Challenge.

Table of Contents

List of Figures

1 Introduction

The meaning of this document is to provide, as the title states, the experimenters with a usage manual that explains how to interact with the federated testbed infrastructure and in the end also with their experiments. The experimenters should use this document as guideline, not only for running their experiments but also for accessing the platform and understanding how to design the experiment on top of this federated infrastructure, result of the combination of multiple technologies such as NFV/SDN. Thus, this document acquires an important role for experimenters and it should be constantly consulted by them while running their experiments, from the beginning to the end of the experimenters' projects.

This document (fully based on D2.4 – "SoftFIRE (v3) usage manual for NFV/SDN/MEC and 5G experimenters") provides a complete description of the new SoftFIRE middleware that has been developed in order to overcome some issues in using older tools of the FIRE community. The SoftFIRE project initially adopted FIRE interfaces (i.e. SFA) based on the FITeagle toolkit. This middleware is developed according to the newer technologies and developments related to the evolution and standardization of NFV/SDN. In fact, SoftFIRE project, starting from the second open call, has undertaken a new approach. After the first open call, based on the feedbacks received from the experimenters, the SoftFIRE team decided to move towards standard NFV/SDN interfaces (i.e. TOSCA) in order to simplify the way experimenters, interact with the platform. This document describes the middleware is developed according to the newer technologies and developments related to the evolution and standardization of NFV/SDN. Some details about the migration from SFA to TOSCA are explained in section 2.1.

During the course of the project, the spread of knowledge about the platform has been a constant goal of the SoftFIRE team. The experience has thought that if experimenters have a good basic know-how of the platform, they can readily start working on the platform. In order to share the knowledge of the platform, other initiatives have been conducted like tutorials and open days. The interested reader can have a look to the project website: www.softfire.eu for extending his/her understanding of the platform. However, this document is a consolidated basis on top of which to build experimentations and projects using the SoftFIRE federated testbed.

The intended readers are experimenters and users of the SoftFIRE Federated Platform. It is suggested to read it carefully before starting any experimentation. It could be convenient to consider to install local instances of the SoftFIRE Middleware in order to experiment locally before deploying on the real platform. Some hints are provided in docs.softfire.eu and in particular in http://docs.softfire.eu/install-softfire-middleware/.

This document is particularly relevant for participants to both, SoftFIRE's third Open Call and to the SoftFIRE Challenge.

2 SoftFIRE Middleware architecture

The SoftFIRE Middleware is the central software component of the SoftFIRE federation. It exposes a REST API that can be consumed either via a CLI or a dashboard. The experimenter is supposed to use the dashboard for running its experiments on demand on the SoftFIRE platform. The middleware's southbound interface is directed towards the available testbeds of Deutsche Telekom, Fraunhofer FOKUS, Ericsson, the University of Surrey and Assembly Data System S.p.A, managed by individual OpenStack controllers and exposing their functions through remote APIs.

2.1 From SFA to TOSCA-based experimentation as a service

The Slice-based Federation Architecture (SFA) 2.0 high level interface specification [1] draft was published in July 2010, before the advent and consolidation of current successful technologies such as Network Function Virtualization and Software Defined Networking. As defined in the draft, a "resource (RSpec) describes a component in terms of the resources it possesses and constraints and dependencies on the allocation of those resources" and the lifecycle is defined elsewhere.

In previous projects, i.e. FED4FIRE, the resource definition was done in different steps: the RSpec defined the actual resource to be used and the physical location of it while the "automated" execution of the experiment is defined through an OEDL script, executed via OMF.

This example shows that the concept of lifecycle of the experiment has gained a more complex definition. Nowadays, the genre of resources could widely vary from each other. SFA was designed for mainly provisioning compute resources (physical or virtual) to the experimenter. The experimenter was then in charge of handling the lifecycle of the actual experiment, via SSH or in some cases, via OEDL script, if the platform was providing OMF support.

SoftFIRE, aware of the lessons learned from the past, adapted its approach to the new technical requirements coming from the new generation of experimenters, maintaining implemented the FIRE functional requirements that are the concrete definition of a FIRE based project.

The recent evolution in the technology state of the art brought the Experimenters to include in their experiments and to require from the used platform some key NFV/SDN functionalities. The SoftFIRE platform has the intend to meet these requirements, while keeping on provisioning the FIRE required features.

The OASIS industry group defined in November 2013 a standard called Topology and Orchestration Specification for Cloud Applications (TOSCA) that aims to provide an efficient and precise tool able to model the cloud applications and associated IT services having a particular focus on telecom operators' requirements. This industry related standard is suitable for modelling NFV and SDN applications (described in addition in the *TOSCA Simple Profile for Network Functions Virtualization (NFV) Version 1.0* specification draft) and it has been chosen by the SoftFIRE consortium to be the reference modelling language for defining Experiments (*Topology Template*) as a set of different kinds of resources (*Node Template*) strictly formalized by the SoftFIRE *Node Type* definition.

As a matter of fact, TOSCA excellently adapts to experiment definition purposes, maintaining the new NFV/SDN requirements met, keeping abstract the experiment configuration and making the definition highly portable.

2.2 SoftFIRE Middleware

The SoftFIRE Middleware exposes a northbound API towards the platform's users. This API is exposed by the Experiment Manager (EM), the central component in the SoftFIRE Middleware. The SoftFIRE Middleware's northbound API consumes files following the TOSCA format which has been employed in this context for defining the user's experiments templates. Since TOSCA aims to be capable of modelling cloud applications and is focused on support the requirements of cloud applications operators, it is well suited for the needs of defining SoftFIRE experiments. In addition, the northbound API, namely the Experimenter API, are invoked directly from a Dashboard, reducing the complexity for the Experimenters to interact with the platform for deploying their experiments. Nevertheless, the dashboard consumes the REST APIs exposed by the EM component, thus experimenters have the freedom to use them directly in order to operate their experiments in a programmable way.

2.3 Functional architecture

The SoftFIRE Middleware is based on a microservice-oriented architecture where the EM plays the central role dispatching incoming requests and events to several managers in charge of handling different types of resources (see Figure 1).

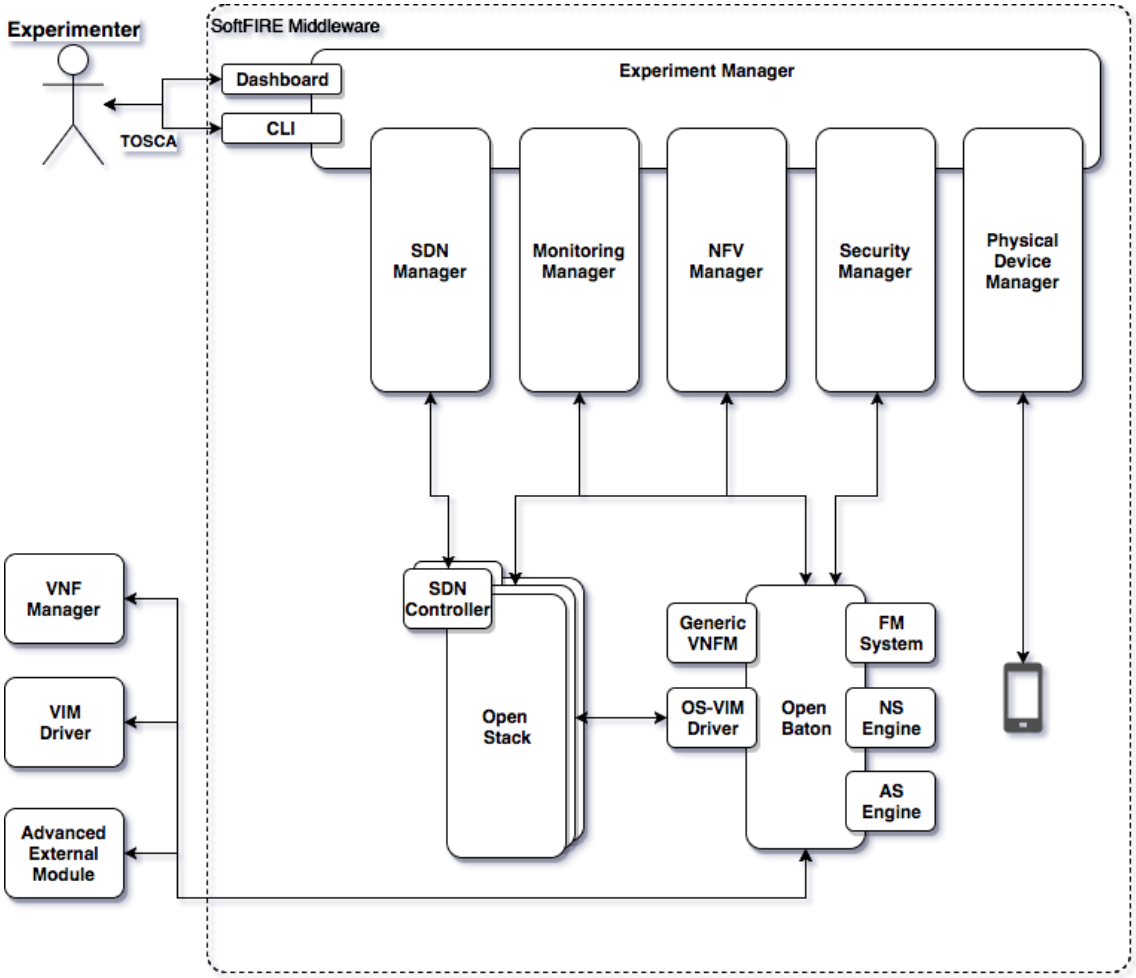


Figure 1. SoftFIRE Middleware Architecture

In particular, the current version of SoftFIRE comprises the following managers:

- **SDN Manager**: manages SDN resources
- **Security Manager**: for the Security resources
- **NFV Manager**: manages NFV resources integrating external NFV frameworks into the SoftFIRE middleware
- **Monitoring Manager**: provides experimenter monitoring resource access
- **Physical Device Manager**: handles the access to the physical resources

The main component of the middleware is the EM. It is responsible for distributing operations of an experiment's deployment to the responsible manager and keeping the overall status of the experiment. Other managers register to the EM, so that they are known and so that the EM is able to send to them requests required for an experiment and to get status information about the resources they are charge of.

The internal communication between the resource managers and the EM is realized using the **gRPC** [2] protocol.

2.3.1. Experiment Manager

The EM provides different features and operations starting from resource management to user management. In the following sections are presented those features in details.

2.3.1.1. User Authentication and Authorization

The EM uses Cork [3] as library in order to handle the user authentication and authorization on the northbound API.

The Experiment Manager exposes a dashboard for the users, which directly invokes the REST API. Depending on the type of user (admin, portal or experimenter) he can access different parts of the dashboard. Among other things, admin users have access to the creation and deletion of users (as shown in section 2.3.1.2) and has an overview of the registered users and managers. Experimenter users can only see the experimenter site where they have the possibility to reserve resources by uploading experiments and to deploy and remove them.

The dashboard sends requests to the EM's REST API, which exposes functions for all of the previously mentioned use cases.

2.3.1.2. Experiment isolation on sub-components

In addition to the user authentication and authorization from the northbound perspective, the EM is in charge of preparing an environment for the experimenter also on the sub-components, in this case Open Baton and OpenStack. Thus, when a user creation is invoked, the EM prepares with the help of the sub managers the required logical tenants in the sub-components in order to logically isolate each experimenter workflow. This, it creates projects at the infrastructure level, and generates project/users inside sub-components. For instance, it requests to the Open Baton framework, through the NFV Manager, to create project and user, and it uploads the correct Virtual Infrastructure Manager (VIM) instances to it. Hence an experimenter is identified in Figure 2:

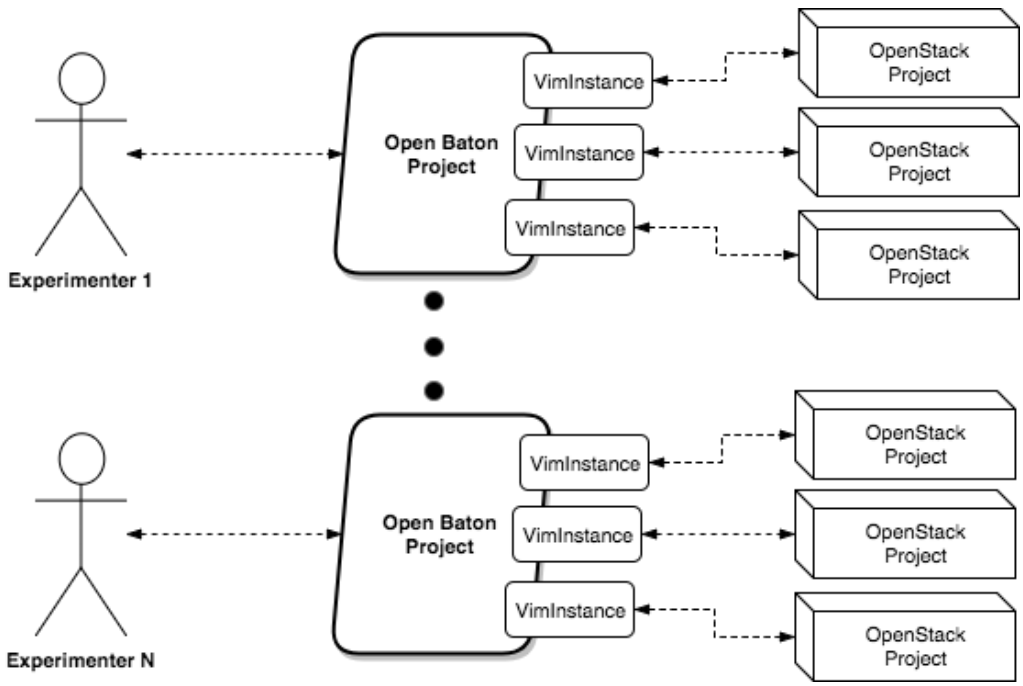


Figure 2. Experimenter Logical Mapping

The process of creating an experimenter environment is done automatically as soon as the experimenter gains access to the platform through the public portal.

2.3.1.3. Resource discovery

The EM provides to the Experimenter a list of available resources offered as a service. It gathers this list from the individual resource managers. Each resource is defined with:

- an **id** as unique identifier
- a **cardinality** that defines the number of instance of this resource that can be deployed
- a **testbed** that indicates the in which testbed this resource can be deployed
- a **description** of the resource

2.3.1.4. Resource reservation

The EM is in charge of calculating the availability of the resources based on a reservation. As will be further explained in Section 3.2, while booking a resource is also mandatory to define a start and end date. In this way, the EM is able to calculate the resource available in a precise moment.

2.3.1.5. Resource provisioning

The EM receives the request of provisioning a resource and transmits to the correct sub manager the "*provide_resource"* message and returns the result to the experimenter.

2.3.2. NFV Manager

The NFV Manager takes care of handling the experiment's NFV resources. It communicates with Open Baton's Network Function Virtualization Orchestrator (NFVO) using its REST API. In case of an experiment deployment it uploads a Network Service Descriptor (NSD) in TOSCA format to the NFVO and triggers the deployment of a Network Service Record (NSR). When the experiment is deleted it removes the NSR and NSD from the NFVO. The NFV Manager sends the NSR's status periodically to the EM.

Furthermore, the NFV Manager is involved in the creation and removal of SoftFIRE users. For this it interfaces the OpenStack testbeds. In the process of user creation, it creates a new OpenStack user and project and a new Open Baton user and project. When removing a SoftFIRE user, it removes also the Open Baton and Open Stack users and projects. It also makes sure that there are no leftovers in Open Stack after deleting the user and project.

The NFV Manager provides two NFV resources to the experimenters, which they can use out of the box. One is a VNF for deploying an iPerf server and client in two virtual machines and the other one is the Open IMS Core which is an Open Source implementation of IMS Call Session Control Functions (CSCFs) and a lightweight Home Subscriber Server (HSS), which together form the core elements of all IMS/NGN architectures as specified today within 3GPP, 3GPP2, ETSI TISPAN and the PacketCable initiative. Besides the two provided Network Services (NS) the experimenters can use their own CSAR files for describing the NS they want to deploy.

2.3.3. SDN Manager

The SDN Manager is in charge of the SDN functions provided by individual Testbeds. It communicates with the specific SDN proxy instance on each testbed using a REST API.

It keeps track of the API endpoints towards the SDN proxy services that are used to filter requests from experimenters to enable multi tenancy that is by default not provided by the used SDN controllers. The REST communication between the SDN manager and the individual

SDN proxy services is authorized by a secret that needs to be passed as a HTTP header field with every request. The URL endpoint used for REST communication between manager and proxy is statically stored inside the configuration file of the SDN manager.

The SDN manager is involved in the following Experiment Lifecycle phases:

1. User Creation
2. Resource Discovery
3. Validation
4. Provision
5. Release

The following Figure 3 shows the individual steps of the lifecycle events in details.

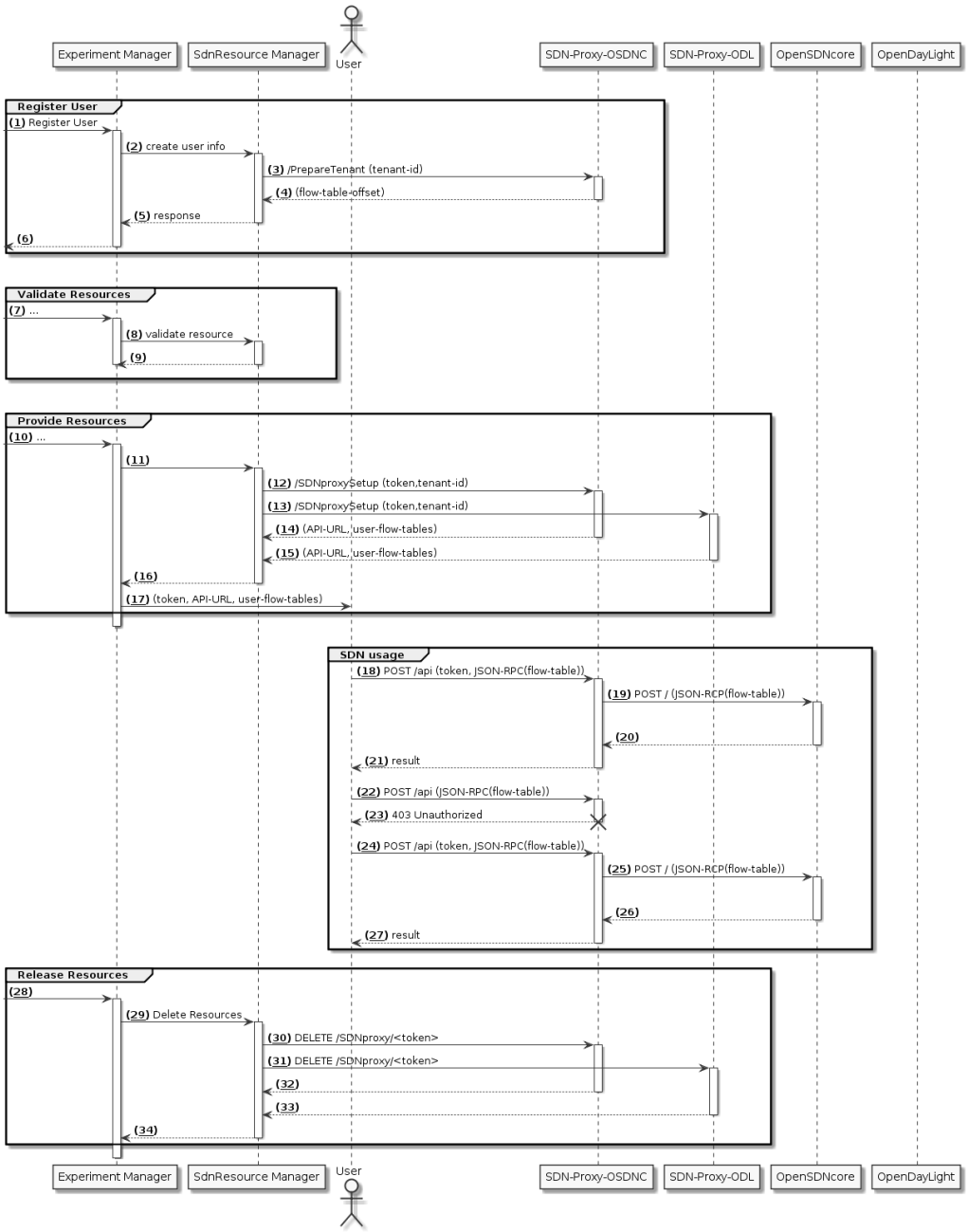


Figure 3. SDN Manager Lifecycle events

2.3.3.1. User Creation

This Event is triggered by the Experiment Manager when the user is initially created inside the SoftFIRE middleware. The SDN manager receives details about the user and the created OpenStack tenants of the user, which are used to prepare the SDN-proxy for OpenSDNcore in the FOKUS testbed.

2.3.3.2. Resource Discovery

Upon list resources event the SDN manager returns a list of available SDN endpoints to the Experiment Manager.

2.3.3.3. Resource Validation

The validation event is used by the SDN manager to check if the requested Resources are available at the moment.

2.3.3.4. Resource Provision

The resource Provision event triggers communication between the SDN manager end the actual SDN-proxy responsible for the selected testbed. Each experiment is assigned a unique token that is send together with the tenant-id to the SDN-proxy. The proxy assigns a number of flow tables to those experiments and returns them. The list of allowed flow tables are returned together with the experiment token and the API endpoint URL to the Experiment manager.

2.3.3.5. Resource Release

The release Resource event is used by the SDN manager to notify the associated SDN-proxy to shut down the experiment and to delete the assigned flow tables.

2.3.4. Monitoring Manager

The SoftFIRE platform offers a monitoring system as a service in order to monitor performances of virtualized resources and applications deployed via the EM.

The Monitoring manager is a software component that provides the capability to request to the SoftFIRE platform, the installation of a pre-configured Zabbix Server VM inside a specific experimenters' project.

Furthermore, in coordination with the other components of the SoftFIRE middleware, it also installs a Zabbix Agent on each VM requested to be instantiated by a particular experimenter.

2.3.4.1. The Zabbix monitoring system

The Zabbix monitoring system, Figure 4 is an enterprise open source software providing a monitoring solution for network functions and applications.

It is designed to monitor and track the status of various network services, servers, and others network hardware.

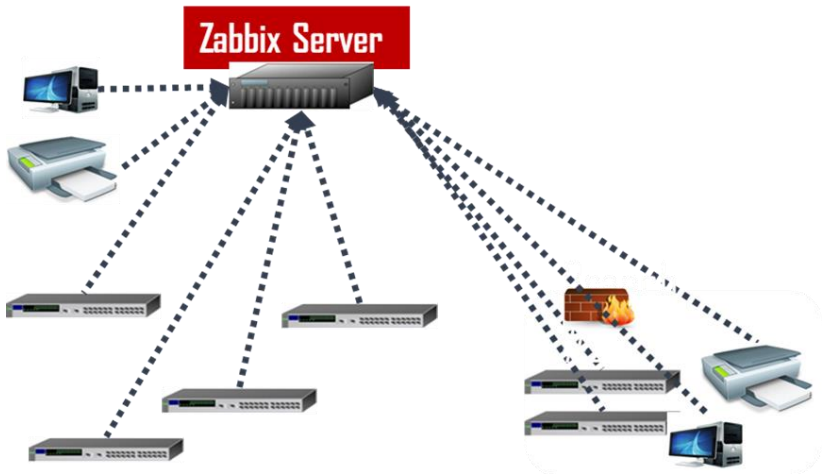


Figure 4: Zabbix Monitoring

In SoftFIRE, Zabbix uses MySQL as a backend database to store data. Its backend is written in C and the web frontend is written in PHP. Zabbix offers several monitoring options:

- Simple checks can verify the availability and responsiveness of standard services such as SMTP or HTTP without installing any software on the monitored host.
- A Zabbix agent can also be installed on UNIX and Windows hosts to monitor statistics such as CPU load, network utilization, disk space, etc.
- As an alternative to installing an agent on hosts, Zabbix includes support for monitoring via SNMP, TCP and ICMP checks, as well as over IPMI, JMX, SSH, Telnet and using custom parameters. Zabbix supports a variety of near-real-time notification mechanisms, including XMPP.

Once instantiated on your Tenant environment you can access to Zabbix server GUI opening the following url http://ip-of-the-vm/zabbix.

To access the Zabbix server, Figure 5, you need to use the following default credentials:

User name :*Admin*

Passwrod :zabbix

Figure 5: Zabbix login page

You can also access the Zabbix server via SSH, using the following credentials: *appliance/zabbix*

As administrator, you have full rights on Zabbix Server to create/modify your own monitoring items and personalized views, Figure 6.

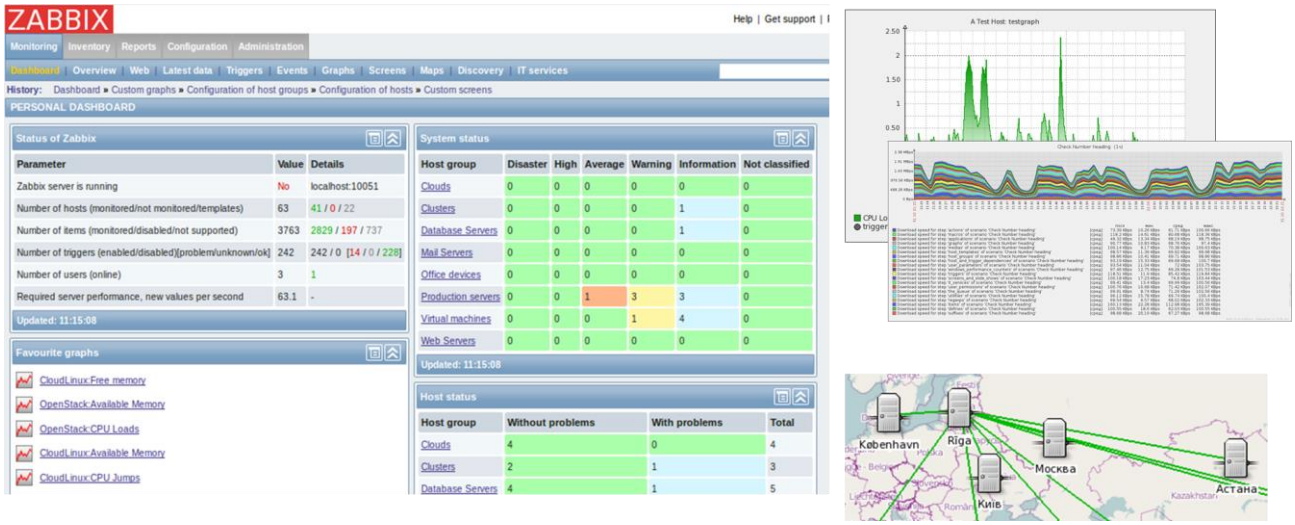


Figure 6: Zabbix Dashboard

2.3.5. Security Manager

The Security Manager inside the SoftFIRE Middleware makes available to the Experimenters a set of security related functionalities that they might decide to include and use within their activities on the SoftFIRE platform.

Here is the list of the available features for every type of Resource:

1. The Experimenters can deploy a Security Resource;
2. The Experimenters can statically configure the Security Resource by means of its descriptor;
3. The Experimenter can statically configure some features on her Resource;
4. The Experimenters can dynamically configure the Resource once it has been deployed;

Features not available for Resource *pfsense*:

1. The Experimenters can enable logs collection from their Resource;
2. The Experimenters can see Resources logs in a web dashboard;
3. The Experimenters can perform searches among the Resources logs in a web dashboard;
4. The Experimenters can see statistics related to the Resources logs in a web dashboard

A Security Resource is a commonly used security agent that the Experimenters can include in their experiment.

They can access and configure it through a static initial configuration, included in the TOSCA description of the Experiment, or, once deployed, through the interfaces that expose its main services.

These interfaces can include SSH, a dashboard, or ReST APIs.

Depending on the type of Resource, Experimenters can also ask the Security Resource to send its log messages to a remote log collector, which makes them available in a simple web page reserved to them.

The Experimenters could easily access it through its web browser and check the behaviour of all their security agents, and to see some related statistics.

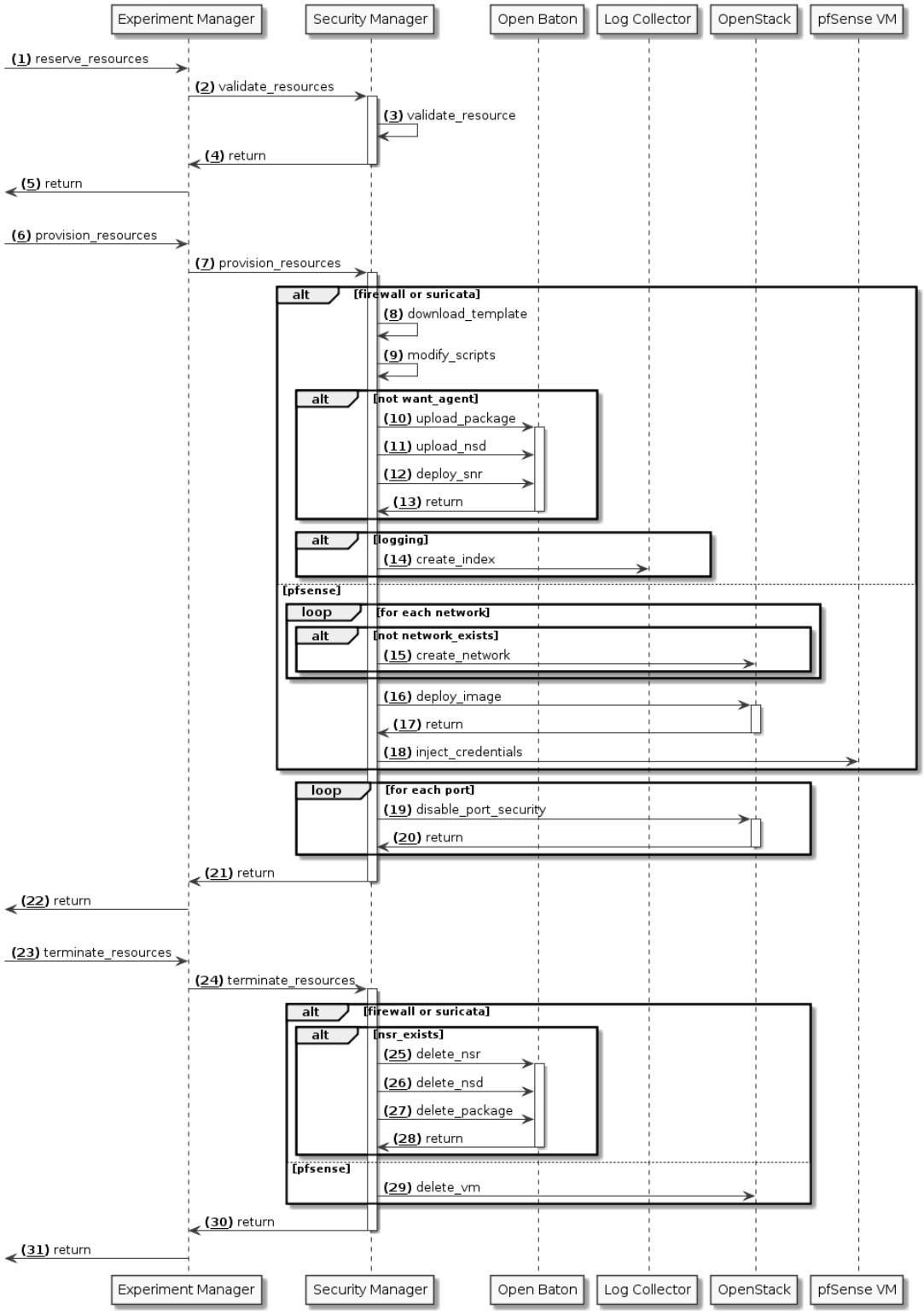
The Experimenters can get the Security Resource in two different formats:

- As an agent directly installed in the VM that they want to monitor. The system will provide them a script that the Experimenters have just to run inside the VM. It will be already configured as required in the TOSCA description of the resource. The output of the script will provide to the Experimenters information on how to access the deployed resource (URLs, etc.)
- As a standalone VM. The Security Resource will be deployed directly by the Security Manager in the testbed chosen by the Experimenter. The Security Manager will take care of the initial configuration of the resource. The Experimenters have to set up on their own the redirection of the network traffic that they want to control through the Security Resource VM (by means of OS configuration, or SDN capabilities provided by the SoftFIRE platform).

The Security Manager provides three types of resources:

- *Firewall*
- *Suricata*
- *pfsense*

This sequence diagram specifies the operations performed by the Security Manager based on the inputs received by the Experimenter.



2.3.5.1. Firewall Resource

A firewall is a network security system that monitors and controls the incoming and outgoing network traffic based on predetermined rules.
The available firewall resource is built upon Ubuntu UFW (Uncomplicated FireWall), to which a

control system, based on a ReST interface, has been added. The firewall agent is available for Ubuntu OS only.
The rules that can be defined on this type of firewall are stateless (they do not maintain information about the context). It works as a packet filter, which looks at network addresses, ports and protocols.

Services specifically available for the firewall Resource are:

1. The Experimenter can statically define a list of allowed IP addresses (or CIDR masks)
2. The Experimenter can statically define a list of denied IP addresses (or CIDR masks)
3. The Experimenter can statically define the default behaviour of the firewall
4. The Experimenter can get the status of the firewall
5. The Experimenter can get the rules installed on the firewall
6. The Experimenter can dynamically add a rule to the firewall
7. The Experimenter can dynamically update a rule on the firewall
8. The Experimenter can dynamically remove a rule from the firewall

Further information and the UFW documentation can be found at https://help.ubuntu.com/community/UFW.

2.3.5.2. Suricata Resource

Suricata is a free and open source network threat detection engine.

The Suricata engine is capable of real time intrusion detection (IDS), inline intrusion prevention (IPS), network security monitoring (NSM) and offline pcap processing.

Suricata inspects the network traffic using a powerful and extensive rules and signature language, and has powerful Lua scripting support for detection of complex threats.

The Suricata project and code is owned and supported by the Open Information Security Foundation (OISF), a non-profit foundation committed to ensuring Suricata's development and sustained success as an open source project.

Suricata is provided in SoftFIRE on top of an Ubuntu VM, and the Suricata Resource offers following Services:

1. The Experimenters can statically define a list of rules that will be inspected by Suricata
2. The Experimenters can view Suricata log messages on a dedicated dashboard
3. The Experimenters can exploit all Suricata features.

The official documentation about Suricata can be found at
`http://suricata.readthedocs.io/en/latest/`.

2.3.5.3. PfSense Resource

PfSense is an open source firewall/router computer software distribution based on FreeBSD.

It can be installed on either physical or virtual machines, and it offers a high-level configuration interface, by means of a web dashboard, as well as a low-level interface by means of SSH.

Furthermore, pfSense supports the installation of third-party packages, that can be included. Exploiting this mechanism, the resource made
available for SoftFIRE experimenters provides a ReST interface through which it is possible to configure and modify every property of the pfSense.
The packet used to provide the ReST interface is the FauxAPI extension (please refer to the official `repository`
for more details about).

Moreover, the credentials of the Experimenter are pushed in the Virtual Machine, so that user only is able to access each of the available interfaces.

This is a list of the main features offered by pfSense OS:

- Firewall with stateful packet inspection
- IPv4 and IPv6 support
- Traffic Shaping
- NAT
- Load Balancing
- VPN - IPsec, OpenVPN, L2TP
- Dynamic DNS
- DHCP Server and Relay

More detailed information about pfSense can be found on the official `website` and `documentation`.

2.3.6. Physical Device Manager

The Physical Device Manager is in charge of handling resources of type *PhysicalResource*. Since the physical resource is an LTE cell located at Fraunhofer FOKUS and cannot be accessed remotely, the Physical Resource Manager's tasks are limited. It only takes care of reserving and configuring the physical resource so that experimenters can be assigned exclusively to a certain experiment per time.

2.3.6.1. User Equipment (UE) Reservation Engine

The UE Reservation Engine takes care of the allocation of User Equipment resources to experimenters, providing them with full remote access to mobile UEs. It receives instructions from the Physical Device Manager over a REST API, in order to:

1. create users,
2. allocate a UE for a user, and
3. terminate all UE allocations of the user.

Based on these instructions received from the Physical Device Manager, the UE Reservation Engine communicates with the TeamViewer [6] UE application through its own REST API to create a user within the SoftFIRE corporate TeamViewer account, and respectively share and unshare UEs with it.

A UE that is shared with a user can be accessed remotely through the online TeamViewer console of that user. Once connected to a UE, the user can control it as if he/she physically accessing the UE, and therefore can install and control applications on the UE.

2.3.6.1.1. Message Flows for Procedures of UE Reservation Engine

The three procedures, i.e. user creation, UE allocation, and termination of UE allocations, are performed via message exchanges between the Experiment Manager and the UE Reservation Engine. Figure 7 below illustrates these message flows. The Engine implements a REST API to create users, reserve a UE, and terminate all reservations. As can be seen in the figure, the Physical Resource Manager acts as a message router, relaying messages between the Reservation Engine and the Experiment Manager. At the background in a UE, the TeamViewer app receives its supported API calls from the UE Reservation Engine. This transparent operation between the User and TeamViewer provides the User with remote control of the app, and hence the UE itself.

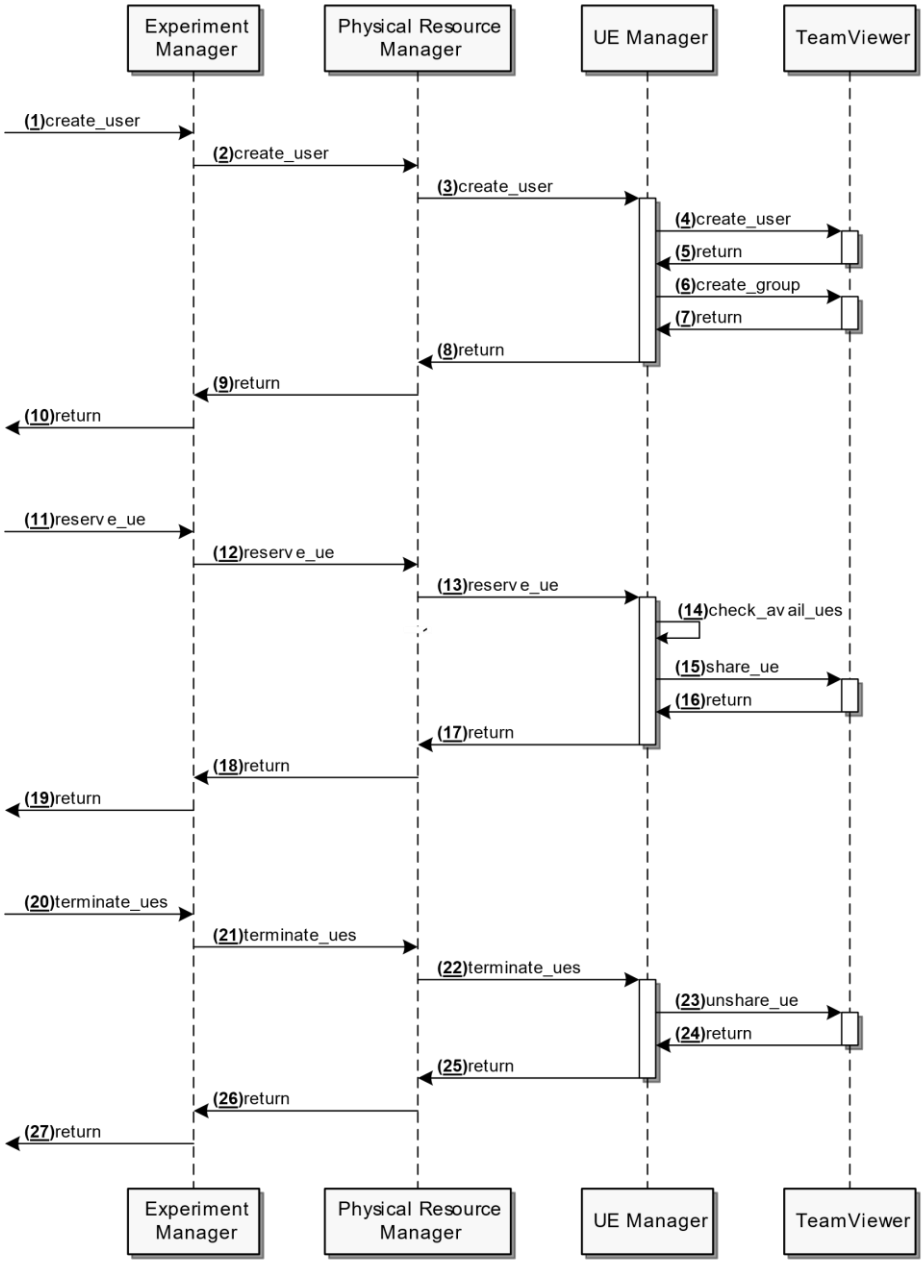


Figure 7. Message Flows for UE Reservation Engine Procedures

2.3.6.1.2. API Calls to the UE Reservation Engine

All API calls that are used to interact with the UE Reservation Engine accept and return JSON data, and require a Bearer Authorization token in the message header.

2.3.6.1.2.1 Create User

This API call is used to create a TeamViewer user account under the SoftFIRE corporate account. This is performed as part of a user's registration to the Experiment Manager. The following are the parameters to this API call:

- **email** – the e-mail address with which to create a TeamViewer account under the SoftFIRE corporate TeamViewer account. As such, the e-mail address that is provided must ideally be one that it not already registered with TeamViewer, and one for which the User is willing to surrender all TeamViewer account privileges to SoftFIRE. If the provided e-mail address is already associated to a TeamViewer account, then an error message will be returned to indicate that the account is already registered, and that the User needs to manually join the SoftFIRE account using the returned URL (see below).
- **password** – the password to use with the TeamViewer account. This must meet the following TeamViewer-imposed requirements:
    - At least six characters long
    - Must contain at least two of the following:
        - Lower case letter
        - Upper case letter
        - Special character
        - Number

The API call returns the following values:

- **email** – the email that was used to register by the User, confirming the account has been successfully created
- **password** – the password that was used by the User to register
- **url** – the URL used to access the User's TeamViewer console

The API call may return the following error messages:

- **no_access_token** (error_code: 0)
  "Access token is missing."
- **invalid_token** (error_code: 1)
  "The access token provided is invalid."
- **database_connection_error** (error_code: 2)
  "Could not connect to the UE manager database." – Please note that UE manager in this message refers to the UE Reservation Engine.
- **insecure_password** (error_code: 6)
  "The password provided is insecure. Your password must be at least six characters long and must meet at least two of the following criteria: (1) Lower case letter; (2) Upper case letter; (3) Special character; (4) Number."
- **email_already_in_use** (error_code: 4)
  "This e-mail address already has a TeamViewer account associated with it. Please go to https://login.teamviewer.com/cmd/joincompany and join the SoftFIRE TeamViewer account (g.kamel@surrey.ac.uk) or use another e-mail address which does not have a TeamViewer account already associated with it."
- **user_already_registered** (error_code: 5)
  "User is already registered and set up."

2.3.6.1.2.2 Reserve UE

This API call is used to reserve a single UE for the User. It accepts the following parameters. Please note that the User must provide their correct e-mail and password pair, originally used to create their user account.

- **email** – the e-mail address used in user creation by the User
- **password** – the password used in user creation by the User for the purpose of password-protecting access to the UE

The API call returns the following values:

- **email** – the email of the User to which a UE has now been allocated to. This is a confirmation to the User to indicate that the UE has been reserved for a particular User identified by this email address
- **assigned_devices** – the aggregate number of UEs associated to the User. This is a counter that indicates how many UEs the User has associated so far.

The API call may return the following error messages:

- **no_access_token** (error_code: 0)
  "Access token is missing."
- **invalid_token** (error_code: 1)
  "The access token provided is invalid."
- **database_connection_error** (error_code: 2)
  "Could not connect to UE manager database." – Please note that UE manager in this message refers to the UE Reservation Engine.
- **no_free_ue** (error_code: 7)
  "There are no more UEs available, as they are all in use by other experimenters.". Please note that "experimenter" refers to "user" in this context.
- **invalid_user** (error_code: 3)
  "User does not exist. Please first create a user account."

2.3.6.1.2.3 Terminate UE reservations

This API call is used to terminate the any UE reservations a User has. It has the following parameters:

- **email** – the e-mail address used in user creation by the User,
- **password** – the password used in user creation by the User for the purpose of password-protecting access to the UE.

Please note that the User must provide their correct e-mail and password pair, originally used to create their user account.

The API call returns the following values:

- **email** – the email of the User to which a UE was allocated. This is a confirmation to the User to indicate that the UE has been reserved for a particular User identified by this email address,

- **assigned_devices** – the aggregate number of UEs associated to the User. This is a counter that indicates how many UEs the User has associated so far.

The API call may return the following error messages:

- **no_access_token** (error_code: 0)
  "Access token is missing."
- **invalid_token** (error_code: 1)
  "The access token provided is invalid."
- **database_connection_error** (error_code: 2)
  "Could not connect to UE manager database." – Please note that UE manager in this message refers to the UE Reservation Engine.
- **invalid_user** (error_code: 3)
  "User does not exist. Please first create a user account."

2.4 The SoftFIRE SDN components

The SoftFIRE platform was extended for supporting SDN functionalities in some of the individual testbeds, in particular for providing experimenters the possibility to interact on demand with the SDN controller and apply different traffic paths inside their virtual networks. Hence on these testbeds the networking is not implemented by the OpenStack Neutron module, but leverages on a SDN backend which provides more advanced functionalities using the standard OpenFlow protocol. These SDN backends allow flow manipulation directly on the network level.

However, the used SDN controller implementations do not provide user separation based on a per tenant schema. This would lead to the ability for each to interfere with the network configuration of other Experiments that are run at the same time on the same Testbed. In order to overcome these issues, the SoftFIRE project introduced a controller specific sdn-proxy which is put in between the SDN controller and the experimenter. Figure 8 shows the relation between Experimenter, SDN Manager, sdn-proxy and SDN controller. The separation between experiments is realized by the assignment of user-specific flow tables that are inserted into the normal packet flow of the SDN switch. The sdn-proxy analyses the controller specific protocol and makes sure that write or modification operations are only done in the flow tables that where assigned to the experiment.

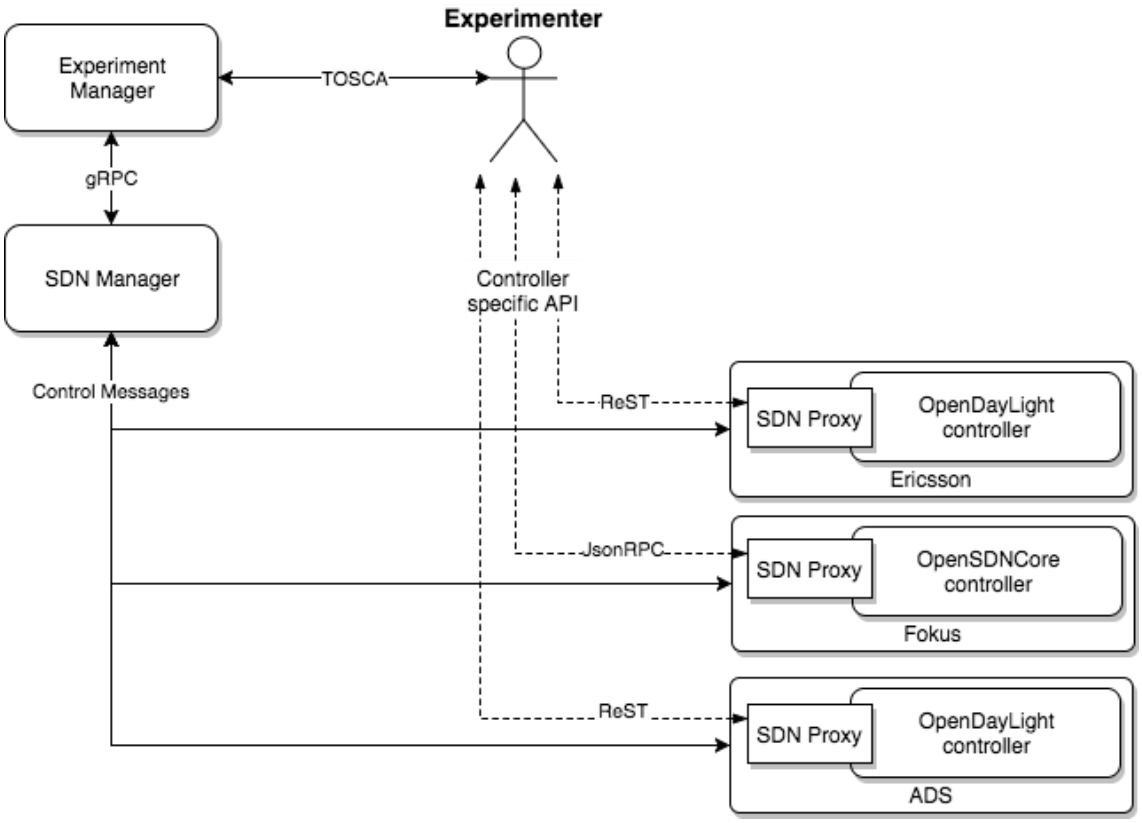


Figure 8. SoftFIRE SDN Infratructure

The SoftFIRE platform provides two types of SDN controllers and switches:

- OpenDaylight / openVswitch
- OpenSDNCore controller and switch

OpenDaylight is one of the most known open source SDN controller, with an extensible architecture based on OSGI plugins. It has been integrated with OpenStack Neutron service which enables control and configuration of the attached Open Virtual Switch (OVS). This replaces the L3 Agent typically found in OpenStack installations. ODL provides a REST API in order to allow the experiments to program flows of OVS switches. Using Openflow the experiments can implement advanced networking functionalities like Load Balancing, Firewalling or Traffic Mirroring.

The experimenters have not direct access to the Opendaylight controller, but their request must be done towards another component called ODLProxy. ODLProxy function is to filter the requests in order to provide experimenter/tenant isolation assigning some tables to the experimenters. Further information can be found at paragraph 4.4.

OpenSDNcore is the highly flexible SDN controller and OpenFlow switch developed by Fraunhofer FOKUS with the targeting the 5G Mobile Network. The OpenSDNcore controller is integrated into the Neutron component of OpenStack to replace the OpenVSwitch and the L3 Agent that is typically handling the Network services in an OpenStack installation. OpenSDNcore provides a JSON-RPC API towards the experimenter to control the operation of the attached switches. Using those APIs, OpenFlow rules can be loaded into the processing pipeline of the switches. The API can be used to implement advanced Features like Service

Function Chain (SFC) or GPRS Tunneling Protocol (GTP) tunnels handling right into the switches used by OpenStack.

The User separation based on dedicated flow tables is already supported by the OpenSDNCore OpenStack agent implementation. The missing user separation on the JSON-RPC API was added by the SDN-proxy-osdnc [7] implementation of the SoftFIRE project.

2.5 How to extend the platform

The SoftFIRE middleware can be easily extended and customized for any particular kind of scenario. This can be done by adding an external Open Baton service, for instance an auto-scaling engine, a fault management system or a monitoring plugin. Furthermore, another Virtual Network Function Manager (VNFM) could be registered to the NFVO.

All these extensions can be applied without the need to change the SoftFIRE middleware, resulting in an easy and effortless extension process. Old experiments are therefore still supported and the experimenters can make use of the extensions by adjusting their experiments if they want to.

This is possible using the Open Baton NFVO SDK that allows the service to register for some events and execute reactions. An example in Java (also available in Go and Python) on how to register for REST events follows:

```
package org.openbaton.event.module.main;
import org.openbaton.catalogue.nfvo.Action;
import org.openbaton.catalogue.nfvo.EndpointType;
import org.openbaton.catalogue.nfvo.EventEndpoint;
import org.openbaton.sdk.NFVORequestor;
import org.openbaton.sdk.api.exception.SDKException;
import org.openbaton.sdk.api.rest.EventAgent;
public class EventModule {
  /* This is the Username used to connect to the NFVO */
  private static String obUsername = "admin";
  /* This is the Password used to connect to the NFVO */
  private static String obPassword = "openbaton";
  /* This is the Project ID used to connect to the NFVO */
  private static String obProjectId = "cef9283a-de4b-47e3-a221-d1192ce9e5bd";
  /* This must be true if during the NFVO installation the ssl was enabled */
  private static boolean isSslEnabled = false;
  /* This is the NFVO Ip */
  private static String obNfvoIp = "127.0.0.1";
  /* This is the NFVO port */
  private static String obNfvoPort = "8080";

  public static void main(String[] args) {

    NFVORequestor  requestor =  new NFVORequestor(obUsername, obPassword,
obProjectId, isSslEnabled, obNfvoIp, obNfvoPort, "1");
    /* Now the Event Agent needs to be retrieved */
    EventAgent eventAgent = requestor.getEventAgent();
    /* Define your endpoint */
    EventEndpoint eventEndpoint = new EventEndpoint();
    eventEndpoint.setName("MyEvent");
    eventEndpoint.setDescription("My event endpoint");
    /*Register to all the event describing the correct instantiation of NSR*/
    eventEndpoint.setEvent(Action.INSTANTIATE_FINISH);
    eventEndpoint.setType(EndpointType.REST);
    eventEndpoint.setEndpoint("http://<SOFTFIREVPNIP:4554>/event/module");
    /* Now register the endpoint */
    try {
      eventAgent.create(eventEndpoint);
    } catch (SDKException e) {
      e.printStackTrace();
      System.err.println("Got an exception :(");
    }
  }
}
```

For making use of the NFVO SDK, it is possible to add the dependency using gradle [8] or maven [9] as follows:

```
<dependency>
   <groupId>org.openbaton</groupId>
   <artifactId>sdk</artifactId>
   <version>3.2.0</version>
```

```
compile 'org.openbaton:sdk:3.2.0'
```

The following sequence diagram is shown in order to understand the lifecycle of an experimenter *custom* service.

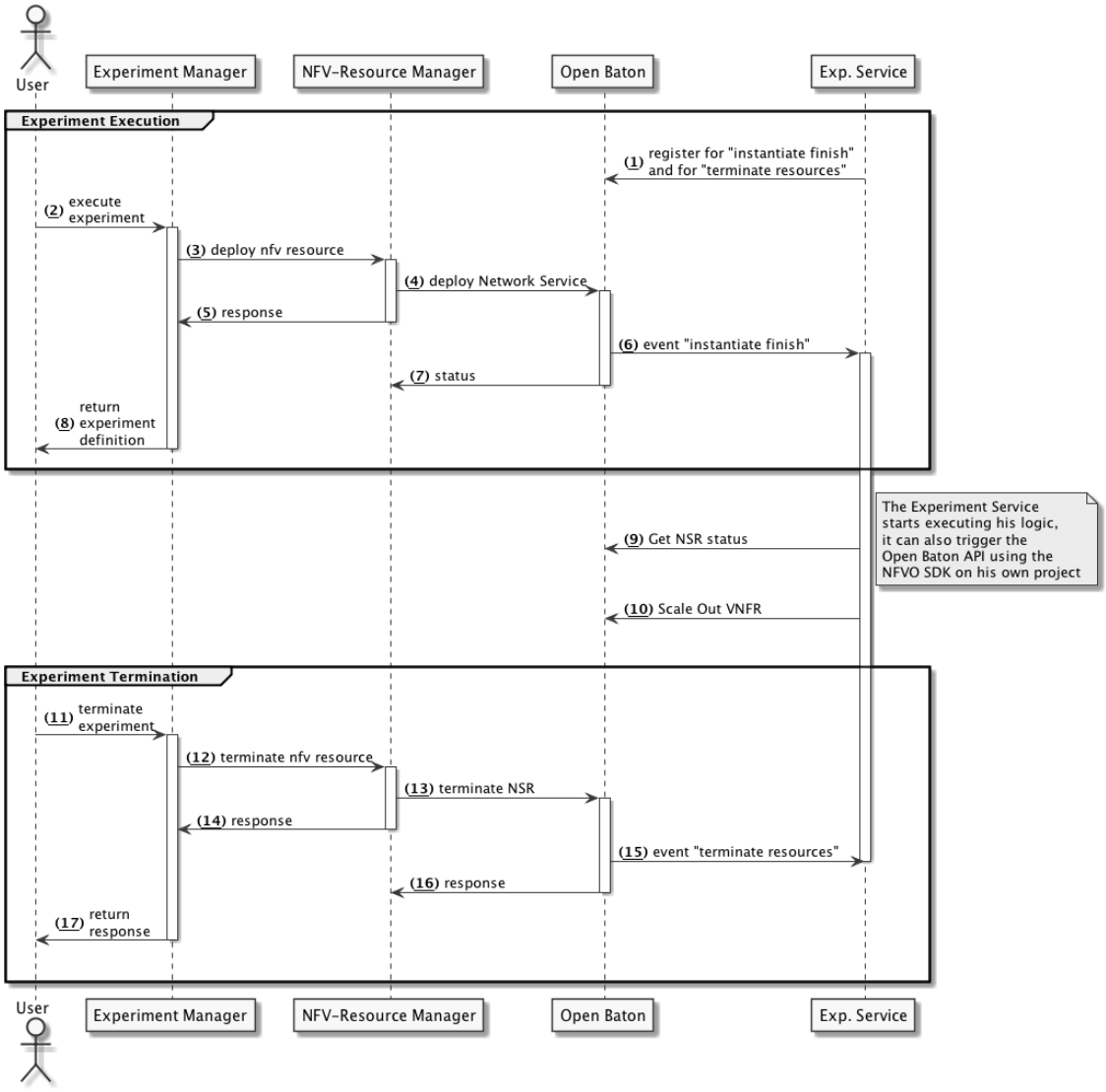


Figure 9. SoftFIRE external module sequence diagram

As we can see from the sequence diagram in Figure 9, the external service must register for specific events to the NFVO. The credentials necessary to use the NFVO API are specific for Experimenter and each experimenter cannot retrieve or modify any other resources of another experimenter. The event to register depends from the application logic and goals, but there are events for any modification of a specific NSR or VNFR status.

In the sequence diagram above, it is shown the logic of an external service making use of the Open Baton APIs for triggering scaling operations towards the VNFR part of the NFV Resource Manager as soon as the NSR is deployed. When the NSR goes in ACTIVE state, the NFVO sends an event to the service registered, containing the NSR deployed. The external service can, using the NFVORequestor, invoke the NFVO API for:

1. retrieving the NSR
2. choose which VNFR → VDU to scale
3. Invoke the NFVORequestor scale out method passing the ids chosen in the previous step
4. wait for the action to finish
5. when the NSR is deleted, the external service can stop

2.6 How to install the Middleware

The SoftFIRE Middleware already provides a set of bash functions[1] that will help you in case you want to install your *private* SoftFIRE Middleware. There are two options:

- codeinstall: install the code of all the managers and the python package of the SDK. This procedure is meant to be for development purposes
- install: install the code of all the managers and the python package of the SDK. This procedure is meant to be for production purposes

In case you want just to play around with the Experiment Manger, you can use the docker installation.

2.6.1. Prerequisites

Both procedures need to have git installed:

```
sudo apt install git
```

and to run:

```
git clone https://github.com/softfire-eu/bootstrap.git
```

for instance, in your home directory. After the clone, you should have a folder called bootstrap containing:

```
bootstrap
├─── LICENSE
├─── README.md
├─── bootstrap.sh
```

[1] The SoftFIRE Middleware is OS independent, however the bootstrap procedure assumes that the underlying OS is Debian based.

2.6.2. Installation

Go into the directory and run the bootstrap commands:

```
$-> cd bootstrap
$-> ./boostrap.sh

./bootstrap.sh <action>

actions:    [install|update|clean|start|stop|codestart|codeupdate|codeinstall|purge]

install:      install the SoftFIRE Middleware python packages
update:       update the SoftFIRE Middleware python packages
clean:        clean the SoftFIRE Middleware
start:        start the SoftFIRE Middleware via python packages
stop:         stop the SoftFIRE Middleware
codeinstall:  install the SoftFIRE Middleware source code
codeupdate:   update the SoftFIRE Middleware source code
codestart:    start the SoftFIRE Middleware via source code
```

2.6.2.1. Source Code installation

For installing the source code just run:

```
./boostrap.sh codeinstall
```

2.6.2.2. Release installation

For installing the python packages just run:

```
./boostrap.sh install
```

2.6.2.3. Installation Procedure

After running these commands the script will:

1. install the debian packages required by the middleware
2. creating the configuration folders
3. downloading the source code or installing python packages of all the managers (depending on what installation procedure you chose)
4. downloading configuration files

Start the Middleware¶

If everything went well, you are able to start the SoftFIRE Middleware by running

```
./boostrap.sh codestart
```

in case you installed via source code, or

```
./boostrap.sh start
```

in case you installed via python packages.

In both cases, a tmux session will run in background and you can check the output by attaching to it:

```
tmux a
```

2.7 Deploy the SoftFIRE Middleware using docker compose

2.7.1. Prerequisites

You need to install:

- docker (that includes also docker-compose)
- git

For having a real example (fully reproduce the SoftFIRE Middleware), you will also need:

- an OpenStack instance where executing deployment

2.7.2. Docker Compose content

This deployment will be composed by these containers:

- Experiment Manager
- Nfv Manager
- Sdn Manager
- Security Manager
- Monitoring Manager
- Physical Device Manager
- Open Baton Standalone

Note: the Nfv Manager and Monitoring Manager containers must be able to reach the OpenStack endpoints.

2.7.3. Get the docker compose folder

Just clone the repository containing the docker compose file and the configurations:

```
git clone https://github.com/softfire-eu/docker-softfire.git
```

after this command, you can go in the folder and check that everything is there:

```
cd docker-softfire
```

Before running docker compose we need to correctly configure the Middleware.

and you should have something like this:

```
.
├── LICENSE
├── docker-compose.yaml
├── ex-man
│   ├── Dockerfile
│   └── softfire
│       ├── experiment-manager.ini
│       ├── softfire-ca.p12
│       ├── softfire-key
│       ├── softfire-key.pem.pub
│       ├── template_openvpn.tpl
│       ├── users
│       └── views
├── mon-man
│   ├── Dockerfile
│   └── softfire
│       ├── monitoring-manager.ini
│       └── openstack-credentials.json
├── nfv-man
│   ├── Dockerfile
│   └── softfire
│       ├── available-nsds.json
│       ├── nfv-manager.ini
│       ├── openstack-credentials.json
│       ├── packages
│       ├── softfire-key
│       ├── softfire-key.pem.pub
│       └── start.sh
├── pd-man
│   ├── Dockerfile
│   └── softfire
│       ├── physical-device-manager.ini
│       └── physical-resources.json
├── sdn-man
│   ├── Dockerfile
│   └── softfire
│       ├── sdn-manager.ini
│       └── sdn-resources.json
└── sec-man
    ├── Dockerfile
    └── softfire
```

2.7.4. Configuration

Each manager has its own configuration. Some are very simple some are more complex. we will have a look into all of them.

2.7.4.1. NfvManager Configuration

In the nfv-man folder

```
cd nfv-man
```

we need to configure the nfv manager itself by changing the nfv-manager.ini file inside the softfire dir.

```
vim softfire/nfv-manager.ini
```

Here you have to do some modifications:

the Open Baton endpoint should be already correctly configured to point to the openbaton container DNS entry.

```
...
[nfvo]
ip = openbaton
username = admin
password = openbaton
port = 8080
https = False
```

Then we need to set the OpenStack endpoint

```
vim softfire/openstack-credentials.json
```

modify the file in order to match your openstack endpoint.

Note: At the moment only v3 is supported

```
{
  "fokus": {
    "username": "admin",
    "password": "password",
    "auth_url": "http://openstack:5000/v3",
    "ext_net_name": "whatever",
    "admin_tenant_name": "admin",
    "allocate-fip": 0,
    "api_version": 3,
    "admin_project_id": "ea45bf4462864832a75ece4c4cc33c11",
    "user_domain_name": "default"
  }
```

Note: please let as key *fokus* since it is needed to be one of the SoftFIRE testbed names.

2.7.4.2. Monitoring Manager Configuration

As per the Nfv Manager, we need to configure the Monitoring Manager, so just copy the file of the nfv manager

```
cp nfv-man/softfire/openstack-credentials.json mon-
man/softfire/openstack-credentials.json
```

Then you also have to configure some options in the .ini file:

```
vim mon-man/softfire/monitoring-manager.ini
```

in particular, the openstack related configurations must be changed based on your particular installation:

```
[openstack-params]
image_name=Zabbix_Server_image
flavour=m1.small
security_group=default
instance_name=Zabbix_Server_Instance
```

2.7.5. Deploy

Now it is time to deploy:

```
docker-compose up --build -d --remove-orphans
```

The Experiment Manager is available at *http://localhost:5180/*. You can access the admin portal by using admin/admin.

The next step is to create an experimenter. By creating a user, a long chain of calls will be performed. In particular, the Nfv Manager will create a user in OpenStack and then upload the right vim to Open Baton.

If it goes well, then you are able to logout and then log in with the create username and password and you should be able to see all the available resources.

2.8 Integration tests

Additional work has been realized for providing a Continuous Integration / Continuous Development (CI/CD) system allowing executing integration tests on top of the federated testbed. This CI/CD system allows testing realistic use cases of all the components of the SoftFIRE Middleware on an automated way. Jenkins [10] is used for the automation of the tests. Every night a Jenkins job runs a comprehensive test. The test is a separate project which exists only for the purpose of testing the SoftFIRE middleware. The test executes a comprehensive scenario on different testbeds utilizing every type of resource in an experiment.

These are the steps of the integration test:

1. Creation of a new experimenter
2. Upload of an experiment making use of all the available resources
3. Deployment of the experiment

4. Validation of the deployed experiment
5. Removal of the experiment
6. Removal of the experimenter

Steps one and six are optional. The number of experimenters can be increased. This means that each step is executed multiple times concurrently for several experimenters. In this case, not all the resources can be used by each experimenter since the physical resource can only be used by one user at a time.

The validation of the deployed experiment is done by validating specific indicators for each resource.

3 Experiment Lifecycle

In this section, are explained in details all the steps that an Experimenter shall follow in order to access the platform, define an experiment and deploy/provision it. The information provided below are also available as part of the official documentation online: http://docs.softfire.eu/getting-started/

3.1 Get Started

The first step to be executed by an experimenter who wants to access the SoftFIRE infrastructure is to register at the Web Portal, Figure 10, available at: https://portal.softfire.eu/login/.

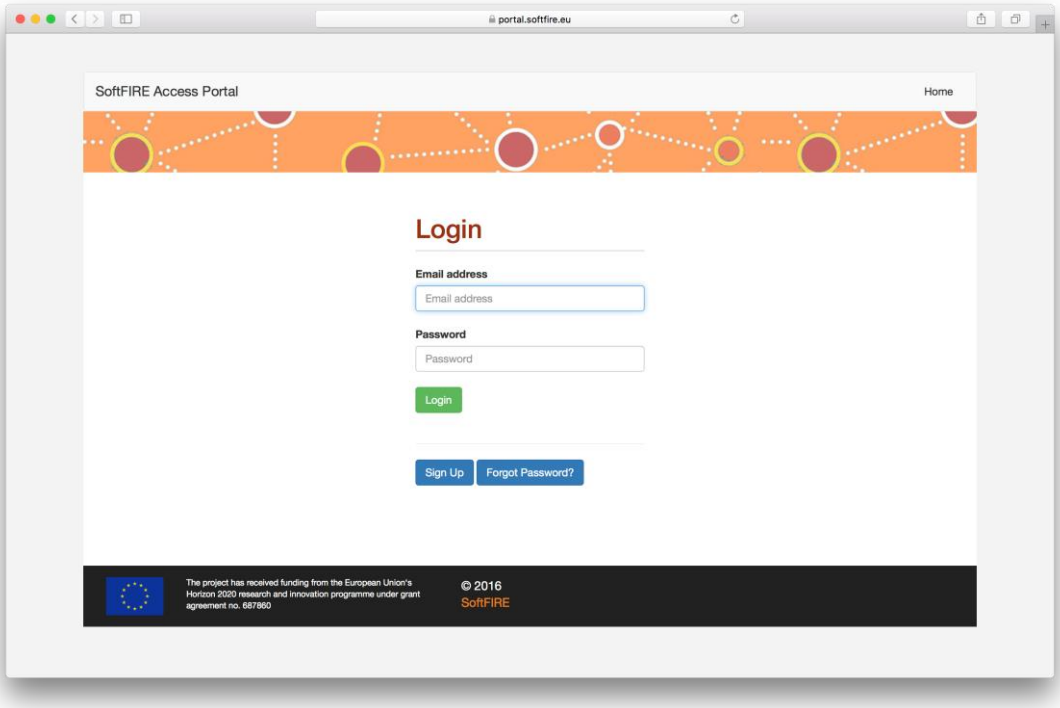


Figure 10. The SoftFIRE Portal

From its personal page it will be possible to download the OpenVPN certificate configuration file that will allow entering the SoftFIRE VPN. The second step required is to install the OpenVPN client and start it using the certificate configuration file previously downloaded.

Once the SoftFIRE VPN is active it is possible to reach the Experimenter Manager dashboard at: http://experiment.vpn.softfire.eu:5080/login. Figure 11 shows the page:

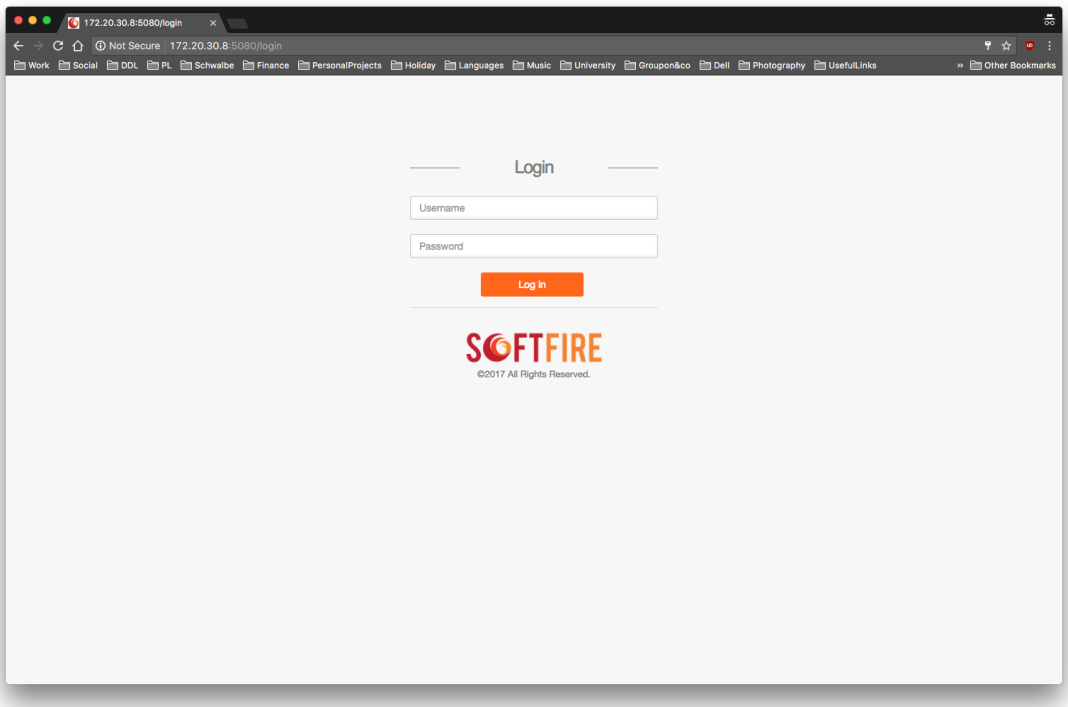


Figure 11. SoftFIRE Experiment Manager dashboard

After providing the username and password, the password is the same used in the SoftFIRE Web Portal and the username is your *name+surname*. The *Signup* is currently disabled If the login works correctly you will be redirected on the Experimenter page that looks like the following Figure 12.

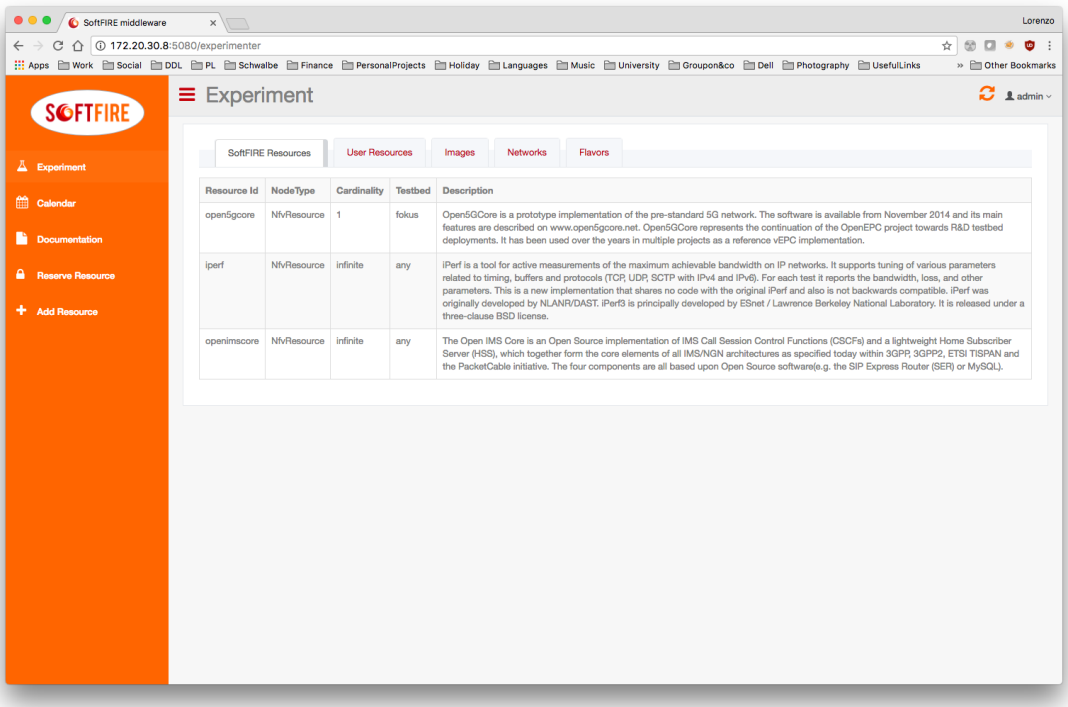


Figure 12. Experiment Manager Overview page

By reloading the page, the experimenter can also refresh the list of available resources. These resources have a detailed description and an id. The id has to be used while defining the experiment, for defining which resources should be reserved and provisioned.

As already mentioned in the previous chapter, the EM uses TOSCA as main format for the experiment definition, thus, for reserving resources, the experimenter must define an experiment description and package it as a TOSCA CSAR archive. This archive should be this archive file in the input box of the *Reserve resources* section. Once the experiment is reserved the experimenter will be able to see it under the *Defined experiment* section.

Once uploaded the experiment CSAR file, the experimenter will have a list of chosen resources in the bottom table. The ***value*** of the resources will be empty until deployed. By clicking on the "Deploy" button, it can trigger the deployment in the SoftFIRE middleware. The status will change to deploy and the content of the deployed resource will appear in the value column.

By clicking on the "Delete" button, the experimenter can trigger the removal of all the resources created. It is necessary to reserve the experiment again in case of re-deployment.

3.2 Experiment Definition

Experiments are the main entity used for communication between the experimenters and the SoftFIRE platform. With the help of experiment files, the experimenters can describe what they want to achieve and how their deployment should look like. Essentially, an experiment is made of the experiment's description and the SoftFIRE resources used in the experiment. Those resources can either be included directly inside the experiment file or stored by the

SoftFIRE platform. In the former case, the resources are stored on the SoftFIRE platform when the experiment is uploaded. In the latter case, the resources are either provided by SoftFIRE out of the box or they have been uploaded to the platform previously by the experimenter. In either case must the experiment's description reference the resources for using them.

The SoftFIRE Middleware expects experiments' definitions to be uploaded as CSAR packages, a format specified by OASIS TOSCA. This package has to contain the experiment's definition and the other files (e.g. scripts) needed for the deployment.

Below it is shown the experiment's directory structure:

```
├── Definitions
|   └── experiment.yaml
├── Files
|   └── nsd.csar
└── TOSCA-Metadata
    ├── Metadata.yaml
    └── TOSCA.meta
```

The **experiment.yaml** file is the TOSCA topology template describing the SoftFIRE experiment. The file describes the topology model of the experiment and consists of a set of *node templates*. Each *node template* describes a specific resource that shall be used in the experiment. A detailed description of the available node types can be found in the appendix A.

3.2.1. Files

NFV Resources may require additional files for their deployment. These files are CSAR files which contain the description of a NS and can be stored inside the Files directory and referenced from the experiment.yaml file.

3.2.2. Metadata.yaml

This file contains the experiment's name, start date and end date and looks like the following:

```
name: Experiment Name
start-date: "2017-07-28"
end-date: "2017-07-30"
```

3.2.3. TOSCA.meta

This file includes TOSCA specific information that are the TOSCA and CSAR versions used, the creator of the experiment and the reference to the TOSCA topology template which is, in the case of SoftFIRE, always the file Definitions/experiment.yaml (or as it was named).

```
TOSCA-Meta-File-Version: 1.0
CSAR-Version: 1.1
Created-By: MyCompany
Entry-Definitions: Definitions/experiment.yaml
```

3.3 Resource definitions

3.3.1. NfvResource node type

With *NfvResources* the experimenters are able to deploy NS compositions of Virtual Network Functions (VNF). They are orchestrated and managed by Open Baton through the NFV

Manager, as explained in the previous sections. NfvResources have different properties that can be set by the experimenter.

The properties are:

- **A public SSH key:** This key will be present in each of the NS virtual machines and allows the experimenter to connect to the virtual machine using SSH, given that the virtual machine has a floating IP.
- **A file name:** The experimenter can use his own CSAR files for describing NS. This property has to provide the path to that file which has to be stored inside the experiment's Files directory.
- **An NSD name:** This name will be used by Open Baton when storing the NSD for the resource.
- **A testbed mapping:** The experimenter can use this map for specifying, which Virtual Deployment Unit (VDU) shall be deployed on which testbed.

3.3.2. SdnResource node type

In order to get access to the SDN-Resources provided by individual testbeds the experimenters need to add *SdnResource* node type to their experiment description. The SdnResource object is handled by the SdnManager component, which selects the appropriate SDN Implementation based on the ResourceId.

The following properties are defined by the SdnResource on top of the properties inherited from BaseResource:

- **Resource_id:** Defines the type of the SDN Resource. Depending on this id the testbed that is used to provide the SDN resource implicit is chosen.

3.3.3. MonitoringResource

The experimenter should define a *MonitoringResource* node type in order to instantiate a monitoring system (i.e. Zabbix as mentioned before) as part of its experiment.

The MonitoringResource node type contains these two properties:

- **testbed** (type: string required: true) : Location where to deploy the monitoring server. In case the experimenter requires deployment of VMs on more than one testbed is it possible to define on which testbed the Zabbix Server VM will be deployed
- **lan_name** (type: string required: true) : Openstack lan name where to deploy the monitoring server.

3.3.4. SecurityResource

Every node has different properties. Here they are listed for each type of resource:

resource_id = *firewall*

- **testbed**: Defines where to deploy the Security Resource selected. It is ignored if want_agent is True
- **want_agent**: Defines if the Experimenter wants the security resource to be an agent directly installed on the VM that he wants to monitor

- **ssh_key**: Defines the SSH public key to be pushed on the VM in order to be able to log into it
- **lan_name**: Select the network on which the VM is deployed (if *want_agent* is False). If no value is entered, softfire-internal is chosen
- **logging**: Defines if the Experimenter wants the security resource to send its log messages to a collector and he wants to see them on a dashboard
- **allowed_ips**: List of IPs (or CIDR masks) allowed by the firewall. [allow from IP]
- **denied_ips**: List of IPs (or CIDR masks) denied by the firewall [deny from IP]
- **default_rule**: Default rule applied by the firewall (allow/deny)

resource_id = *suricata*

- **testbed**: Defines where to deploy the Security Resource selected. It is ignored if *want_agent* is True
- **want_agent**: Defines if the Experimenter wants the security resource to be an agent directly installed on the VM that he wants to monitor
- **ssh_key**: Defines the SSH public key to be pushed on the VM in order to be able to log into it
- **lan_name**: Select the network on which the VM is deployed (if *want_agent* is False). If no value is entered, softfire-internal is chosen
- **logging**: Defines if the Experimenter wants the security resource to send its log messages to a collector and he wants to see them on a dashboard
- **rules**: Defines the list of rules to be configured in Suricata VM. These rules follow the syntax of Suricata application

resource_id = *pfsense*

- **testbed**: Defines where to deploy the Security Resource selected
- **wan_name**: Selects the network on which the first interface of the VM is attached. It is configured as WAN on pfSense. It must be a network connected to the SoftFIRE-public network
- **lan_name**: Selects the network on which the second interface of the VM is attached. It is configured as LAN on pfSense

3.3.5. PhysicalResource

The *PhysicalResource* node type allows provisioning and reserving an LTE Cell located at Fraunhofer FOKUS connected to the Open5GCore NS. This cell cannot be used remotely so the experimenter has to be present for using it. The PhysicalResource inside an experimenter descriptor is needed for reserving the cell for a specific time period.

The *UeResource* node type allows provisioning a set of three UEs located at the University of Surrey, each located in close proximity to their own femto-cell. These UEs can be accessed and controlled remotely by an experimenter, allowing them to install and run any applications for their experiments.

4 Examples and Tutorials

This section provides to the reader some tutorials on how to get started immediately deploying and provisioning some “hello world” scenarios on top of SoftFIRE, making use of the different resources individually. The usage of two or more resources can be easily achieved combining the individual elements exemplified in the following sections.

4.1 NFV Resources

4.1.1. iPerf tutorial

This tutorial describes the mechanism for deploying an iPerf server and client kind of network service using the SoftFIRE middleware. For this, one of the NFV resources is used which is provided by the NFV Manager out of the box.

The experiment needs the typical file structure. This means that inside the experiment’s root directory are three directories called ‘Definitions’, ‘Files’ and ‘TOSCA-Metadata’.

The Definitions directory has to contain the ‘experiment.yaml’ file, which defines how the experiment looks like. You can find it in appendix B.

As you can see, has the experiment only one node template, which is called ‘iperftutorial’. The name can be chosen arbitrarily.

The experiment’s resource ID, defined by the property ‘resource_id’ is set to the value ‘iperf’. This is important since this is the resource ID of the NFV resource provided to us out of the box by the NFV Manager.

You can find the provided resources on the SoftFIRE dashboard under the tab ‘SoftFIRE Resources’.

The ‘testbeds’ property maps the value ‘fokus’ to the key ‘ANY’, which is a special keyword for saying that every VDU of the deployed VNF shall be deployed on the fokus testbed. If different VDUs have to be deployed on different testbeds, one can specify this by mapping the testbed’s identifier to the VDU’s name. For deploying a VDU named ‘vdu1’ on the fokus testbed and another VDU named ‘vdu2’ on the testbed of ADS one has to write the following:

```
vdu1: fokus
vdu2: ads
```

Please note that the virtual machines deployed on one testbed are not able to reach virtual machines on another testbed unless they have a floating IP.

The ‘ssh-key’ property is optional and defines the public SSH key that will be inserted into the authorized_keys file of virtual machines spawned for the NS.

The ‘nsd_name’ property sets the name of the NSD that will be used when creating the NSD for the resource in the NFVO.

There are other files, which are required by TOSCA inside the experiment’s directory. One of them is the TOSCA.meta file in the TOSCA-Metadata directory. Here is an example:

```
TOSCA-Meta-File-Version: 1.0
CSAR-Version: 1.1
Created-By: SoftFIRE
Entry-Definitions: Definitions/experiment.yaml
```

The only value which is supposed to be changed by the experimenter is the 'Created-By' property.

The other required file in the TOSCA-Metadata directory is the 'Metadata.yaml' file, which contains metadata about the experiment:

```
name: ExperimentIperf
start-date: "2017-7-2"
end-date: "2017-7-15"
```

The start date marks the date on which the experiment is supposed to start and the end date the final day of the experiment respectively. Only in this period it is possible to deploy the experiment.

The three directories 'TOSCA-Metadata', 'Files' and 'Definitions' can now be compressed into a single zip file with the '.csar' file extension. The dashboard's 'Reserve Resource' form can be used to upload the experiment to the SoftFIRE platform.

4.1.2. Custom VNF tutorial

This section describes how to create an experiment with a custom NFV resource. The previous tutorial explained how to use the iPerf resource that is provided by the NFV Manager. This tutorial however, describes how to create a custom NFV resource and how to use it in an experiment. For this, it shows how to create a self-build iPerf NFV resource and how to make the experiment use it.

The experiment's folder structure is the same as in the previous tutorial and can be taken from section 4.1.1. Also, the TOSCA.meta and Metadata.yaml files can be reused as they are.

The first difference from the previous tutorial is in the experiment.yaml file. You can find it in appendix C. The two important differences are the resource ID and the added 'file_name' property. The resource ID does not match one of the provided resources anymore. The 'file_name' property points to the location of the CSAR file which describes the NS.

Additionally, in contrast to the previous tutorial, this example uses its own NS definition stored in the experiment's 'Files' directory. Just like the experiment, the NS is described in TOSCA format and packed in a CSAR file. Here is the directory structure:

```
├─── Definitions
|   └── custom-iperf.yaml
├─── Scripts
|   └── client
|        ├─── install.sh
|        ├─── iperfserver_configure.sh
|   └── server
|        ├─── install.sh
|        └── start-server.sh
└── TOSCA-Metadata
    ├─── Metadata.yaml
```

The custom-iperf.yaml file can be found in appendix D. It contains the definition of the NS i.e. the description of the VNFs and their relations and dependencies to one another. There are two node templates of type openbaton.type.VNF, which describe an iPerf server and client. At

the bottom of the file is the relationship description, which specifies that the client needs the server's IP address on the network 'softfire-network'.

The 'Scripts' directory contains two subdirectories, one for each VNF. Their names correspond to the types of the VNFs. In these directories are the scripts stored which shall be executed in the different lifecycle phases of the VNF and install iPerf and start the connection. You can find the content of the scripts in appendix E.

The Metadata.yaml defines the essential properties of the NS. Here is the one for the iPerf example:

```
name: iPerf tutorial NSD
description: "NSD in TOSCA format describing a NS deploying an iPerf
client and server."
provider: SoftFIRE
shared: true
nfvo_version: 3.2.0
image:
    upload: false
    names:
            - Ubuntu-16.04
vim_types:
    - openstack
```

The image property says that the image used for deployment shall be 'Ubuntu-16.04'. The false value of the upload property implies that the image is already present on OpenStack.

The TOSCA.meta file's entry point has to reference the custom-iperf.yaml file in the 'Definitions' directory.

```
TOSCA-Meta-File-Version: 1.0
CSAR-Version: 1.1
Created-By: SoftFIRE
Entry-Definitions: Definitions/custom-iperf.yaml
```

4.1.3. How to write an Open Baton Network Service

This section explains some basic concepts of Open Baton Network Services, which are used in NFV resources. For this it uses the NS from section 4.1.2 which can be found in appendix D and describes a NS consisting of an iPerf client connecting to an iPerf server.

The TOSCA file describing the NS consists of three main parts: a metadata section, a topology section and a section about the relationships inside the NS.

The metadata section contains information about the vendor and the version of the NS.

The topology section describes the components from which the NS is built. In the iPerf example there are two VNFs inside the NS. They are represented by the two nodes 'iperf-server' and 'iperf-client'. They are both of type 'openbaton.type.VNF'. In their properties section you can see that they map the 'endpoint' property to 'generic'. This means that Open Baton will use the Generic-VNFM for deploying this VNF. Since this is the only VNFM used by SoftFIRE you should not change this value. The 'deploymentFlavour' property specifies the

deployment flavour used for the virtual machines of the NS. The 'type' property defines the VNF's type and will be important again in the later parts of this tutorial. The type can be chosen arbitrarily by the experimenter. The only restriction is that each VNF should have a different type. There is one property available, which is not included in the example NS and this is the 'configurations' property. This property allows the experimenter to define configuration parameters inside the NS description, which can be used as environment variables in the lifecycle scripts and can even be passed in dependencies to other VNFs of the NS. Here is an example of the 'configurations' property:

```
properties:
  configurations:
    name: config_name
    configurationParameters:
      - key: value
      - key2: value2
```

The next section of the VNF node is the 'requirements' section. As you can see it contains a reference to a VDU. In this example, the VNF only contains one VDU, but you can also specify that a VNF consists of multiple VDUs by listing the different VDUs. The VDUs themselves are defined later in this tutorial.

The last important section of the VNF node is the 'interfaces' section. Here are the VNF's lifecycles defined. There are five lifecycle events supported by Open Baton: INSTANTIATE, CONFIGURE, START, TERMINATE and SCALE_IN. The INSTANTIATE lifecycle event is executed immediately after the virtual machines have been started. Afterwards the CONFIGURE lifecycle event runs, if the VNF is target of a dependency. Then the START lifecycle event is executed and the VNF becomes active. The TERMINATE lifecycle event runs shortly before the VNF is stopped and the SCALE_IN lifecycle event runs when the VNF is target of a dependency and a scale in operation is performed on the source VNF. The lifecycle events are not mandatory and are skipped if not defined. In the iPerf example you can see that the iPerf client VNF has two lifecycles and each lifecycle contains one name of a script, which will be executed in the lifecycle event. The iPerf server VNF has only one lifecycle event, but in this event two scripts are executed.

There are two options for storing the scripts: either inside the NS CSAR file itself or in a public git repository. The former option was used in section 4.1.2. For this there existed a directory called 'scripts' which contained two directories named after the types of the VNFs. These directories contained the lifecycle event scripts for the VNFs. The latter option can be realized by adding an additional property to the VNF, which contains a link to the git repository. Here is an example:

```
properties:
  vnfPackageLocation: "https://github.com/openbaton/vnf-scripts.git"
```

The next part describes the definition of the components that were referenced in the VNF nodes but not yet defined. We start with the VDUs. There are two VDUs in the iPerf example, one belonging to the iPerf client VNF and the other one to the iPerf server VNF. As mentioned earlier, a VNF may also have multiple VDUs. A TOSCA VDU node has the type 'tosca.nodes.nfv.VDU' and a properties and requirements section. The properties can specify the maximum number of virtual machines that may be started for this VDU with the

'scale_in_out' property. Theoretically, the properties may also contain a property for specifying the VIM on which the virtual machines should be launched, but since this is defined in the SoftFIRE experiment, this property should not be set. The 'virtual_link' property in the requirements section of the VDU points to one or more connection points.

A connection point node is of type 'tosca.nodes.nfv.CP' and one could say that for each connection point there will be one virtual machine when the NS is launched. A connection point is linked to a virtual link, which corresponds to a network. Furthermore the 'requirements' section needs a reference to the VDU for which the connection point is used, this is done with the 'virtualBinding' map. If the virtual machine should have a floating IP, the properties section has to contain a 'floatingIP' property. If you choose 'random' as a value, the floating IP will be assigned randomly.

The last node of the example's topology section is the virtual link node. It is of type 'tosca.node.nfv.VL' and represents a network. The network's name is the same as the node's name, in this case 'softfire-internal'. It is referenced by connection points.

After the topology section is explained, the relationship section remains. In this section are the dependencies between the VNFs defined. The iPerf example has one dependency with the iPerf server being the dependency's source and the iPerf client being the target. The dependency parameter is the IP address of the server. This dependency is necessary since the iPerf client does not know the server's IP address on start-up. The relationship node's type is 'tosca.nodes.relationships.ConnectsTo'. It has a 'source' and a 'target' map which reference the VNFs by their node name and a 'parameters' list. These parameters will be available to the iPerf client VNF as environment variables when executing its CONFIGURE lifecycle event scripts.

The lifecycle scripts can be found in appendix E. They simply install iPerf, start the iPerf server and establish a connection from the client to the server. The IP address of the server is passed because the relationship node listed the virtual link 'softfire-internal' to which the server VNF is connected. There are three types of variables that are made available automatically by Open Baton. That are the IP address of a virtual machine, the floating IP address if it exists and the hostname. They can be used in lifecycle scripts. As can be seen in the 'server_configure.sh' script the environment variable $server_softfire_internal is used for retrieving the server's IP address. This variable name can be broken down as follows: It begins with 'server_' which is necessary for showing that the IP address shall be retrieved from the VNF of type 'server'. The second part is 'softfire_internal'. This corresponds to the relationship parameter that was passed from the server to the client, except for the hyphen being replaced by an underscore. This is necessary since bash variables must not contain hyphens and is translated automatically by Open Baton. Furthermore, the script's name is important when using environment variables resolved from relationships. The script's name has to start with the type of the source VNF followed by and underscore. It is important to note, that relationship parameters are only available in CONFIGURE lifecycle event scripts.

4.2 Security Resources

The next sections present three individual scenarios providing an example of each of the available security resource types.

For each example, we choose to deploy the resource inside the ADS testbed. The newly created machine will be attached to the private network and OpenStack will provide a floating IP to reach the resource. In order to log on the machine, we provide our public SSH key.

Below there are the yaml definition of the security resources available.

4.2.1. Example 1

```
node_templates:
    fw_resource:
        type: SecurityResource
         properties:
             resource_id: firewall
              want_agent: false
              lan_name: private
              ssh_key: ssh-rsa AAAAAV******SZ9
              testbed: ads
              default_rule: allow
              denied_ips: [172.20.10.42]
              logging: true
```

In this scenario we specify to enable the logging functionality and to configure the firewall to allow all traffic except from 172.20.10.42.

Resource Id	Status	Value

firewall	DEPLOYED	-{
		"api_url": "http://172.20.30.111:5000",
		"dashboard_log_link": "http://172.20.10.138:8888/dashboard/77p59dkbbcs1z56",
		"ip": "172.20.30.111",
		"status": "ACTIVE"
		}

The Experiment Manager provides to the Experimenter the Floating IP address of the Firewall VM, and the URL of the ReST interface. The VM can be accessed by means of the SSH private-key coupled with the public one provided in the descriptor. Furthermore, it is shown the URL of the dashboard displaying log messages from the resource.

4.2.2. Example 2

```
node_templates:
    suricata_resource:
        type: SecurityResource
         properties:
             resource_id: suricata
              want_agent: false
              lan_name: private
              ssh_key: ssh-rsa AAAAAV******SZ9
              testbed: ads
              rules:
alert icmp any any -> any any (msg:"ICMP test"; sid:1000001; rev:1;
classtype:icmp-event;)
```

In this scenario we set a rule for Suricata to allow ping from any to any. Similar to the previous example we enable the logging functionality.

Resource Id	Status	Value

suricata	DEPLOYED	-{
		"status": "ACTIVE",
		"ip": "172.20.70.43",
		"rules": +[...]
		}

The Experiment Manager provides to the Experimenter the Floating IP of the Suricata VM, that can be accessed by means of the SSH private-key coupled with the public one provided in the descriptor.

4.2.3. Example 3

```
node_templates:
    pfsense_resource:
        type: SecurityResource
         properties:
             resource_id: pfsense
              lan_name: private
              wan_name: my_wan
              testbed: ads
```

The pfSense virtual machine is provided with two interfaces that are attached to two different OpenStack networks. The first one (*lan_name*) is connected to the LAN interface of pfSense. Instead the other one (*wan_name*) is connected to WAN interface of pfsense.

Resource Id	Status	Value

pfsense	DEPLOYED	-{
		"FauxAPI-ApiSecret": "uyir180rnrv79xttr9rmdt3doyj8dry1yyhwre11onuuehtmp8se0y3yjb21",
		"FauxAPI-ApiKey": "[PFFAexperimenter]",
		"ip": "10.20.70.18"
		}

The Experiment Manager provides to the Experimenter the Floating IP of the deployed pfSense VM, with APIkey and APIsecret needed to access the FauxAPI ReST interface. PfSense web dashboard and SSH are accessible by means of the already known Experimenter's username and password.

4.3 Monitoring Resources

In this section, we describe how an experimenter can deploy a Zabbix server for monitoring porpoise.

The Definitions directory contains the 'experiment.yaml' file, which defines the monitoring resources and it looks like the following example:

```
description: "Template for SoftFIRE yaml resource request
definition"
imports:
  - softfire_node_types:
"http://docs.softfire.eu/etc/softfire_node_types.yaml"
topology_template:
  node_templates:
    m:
      type: MonitoringResource
      properties:
        testbed: ericsson
        lan_name: Zabbix_Lan
        resource_id: monitor
```

This node type has two properties:

- testbed: in case the experimenter requires deployment of VMs on more than one testbed is it possible to define on which testbed the Zabbix Server VM will be deployed. In the example above ericsson testbed has been selected.
- lan_name: it possible to define on which Openstack lan the Zabbix Server VM will be deployed. In the example above the Zabbix server will be deployed on a lan named Zabbix_Lan.

On successful deployment, the Experiment Manager provides to the experimenter with the Floating IP address of the Zabbix server VM, and the URL of the GUI interface.

At this point the experimenter can access the Zabbix server opening the URL shown in the picture above (http://10.44.57.37/zabbix/index.php) and providing the following credentials:

Username: Admin

Password: zabbix

The Zabbix server VM can be also accessed by means of the SSH using the following credentials:

Username: appliance

Password: Zabbix

4.3.1. Zabbix agent example

The following example shows how manually install the Zabbix agent on a VM

Login the VM and install the Zabbix agent with the following commands:

```
$ sudo apt update
$ sudo apt-get install zabbix-agent
```

Edit the file /etc/zabbix/zabbix_agentd.conf

```
$ sudo vim /etc/zabbix/zabbix_agentd.conf
```

Set the Zabbix Server IP address and your VM Hostname accordingly to your experiment

```
Server=192.117.1.10
ServerActive=192.117.1.10
Hostname=project_vm1
```

Use the Zabbix server InternalIp address if your VM is co-located with the Zabbix server, otherwise use the floatingIp address. Both addresses are provided by the Experiment Manager (see picture above).

Restart the Zabbix agent with the following command:

```
sudo /etc/init.d/zabbix-agent restart
```

Wait a couple of minutes and your VM information are added to the Zabbix server Dashboard.

4.4 SDN Resources

This chapter provides examples on how to use the SDN resources provided by the SoftFIRE platform and some of its testbeds.

4.4.1. OpenSDNcore port mirror Example

This example will use the SDN features provided by the OpenSDNcore that is available in the Fraunhofer FOKUS development testbed. The SDN controller will be used to setup an experiment that uses a port mirror on the SDN switch to receive a copy of each network packet send to a specified VM instance.

Please follow the NFV tutorial in chapter 4.1.1 to create an experiment with iPerf server, iPerf client and a plain Ubuntu machine used for monitoring. Please make sure that the experiment uses the fokus-dev testbed as this is the only testbed that provides OpenSDNcore support. In order to request SDN features please add the *sdn-controller-opensdncore-fokus* resource to the *experiment.yaml* file.

```
---
description: "Template for SoftFIRE yaml resource request
definition"
imports:
  - softfire_node_types:
http://docs.softfire.eu/etc/softfire_node_types.yaml

topology_template:
  node_templates:
    iperf:
      type: NfvResource
      properties:
        resource_id: iperf
        nsd_name: The Iperf NSD
        testbeds: { ANY: fokus-dev }
    ericsson_ads:
      properties:
        resource_id: sdn-controller-opensdncore-fokus
      type: SdnResource
tosca_definitions_version: tosca_simple_yaml_1_0
```

After the successful deployment of the experiment via the experiment manager the status field of the *sdn-controller-opensdncore-fokus* resource includes the needed details how to access the OpenSDNcore API.

```
{
    "resource_id": "sdn-controller-opensdncore-fokus",
    "flow-table-range": [30, 31, 32],
    "token": "123d67576eab0853341ea467867a2530",
    "URI": "http://172.20.30.5:8001/api"
}
```

Copy the *token* value and navigate to the provided *URI* using a web browser. The website provides the needed information and a simple user interface to run JSON-RPC request against the OpenSDNcore Northbound-API [11]. Use the provided token value to identify your experiment when doing API requests.

1. Use the *ofc.list.channels* JSON-RPC method to find the dpid value used by the switch. In our case the setup uses value *0x0001*
2. Find the Port number (*port_no*) associated for each of the Virtual Machines by using the MAC address of the virtual NIC and the entries in flow table *0x04*. The entries of flow table 0x04 can be listet using *the ofc.send.multipart.flow* function as seen in the sniplet below:

```
{
  "jsonrpc": "2.0",
  "method": "ofc.send.multipart.flow",
  "params": {
    "dpid": "0x01",
    "ofp_multipart_flow": {
      "table_id": "0x4"
    }
  },
  "id": 2
}
```

3. use the discovered *dpid*, *port_no*, and the private IP-address of the instance which traffic should be duplicated to construct a openflow definition that will duplicate each network-packet to the target port of the monitoring instance.
4. Add the new flow via the following JSON-RPC query to the switch into one of the flow tables that are assigned to your experiment (ex. 30,31,32).

```
{
  "id":2342,
  "jsonrpc":"2.0",
  "method":"ofc.send.flow_mod",
  "params":{
     "dpid":"0x0000000000000001",   /* address of the target
switch */
     "ofp_flow_mod":{
        "command":"add",
        "flags":[
           "reset_counts",
           "send_flow_rem"
        ],
        "idle_timeout":1000,
        "ofp_instructions":{
           "apply_actions":{
              "output": {
                 "port": "0x05"   /* port number of the mirror
port */
              }
           },
           "write_actions":{
              "output": {
                 "port": "0x01"   /* port number of the
original destination instance */
              }
           }
        },
        "ofp_match":[
           {
             "match_class": "openflow_basic",
                "field": "ipv4_dst",
                "value": "192.168.66.22" /* the private ip
address of the target virtual machine */
           }
        ],
        "priority":400,
        "table_id":"0x1e"      /* flow_table 30 in hex notation
*/
     }
  }
}
```

Now the experiment can be verified by using *tcpdump* on the monitoring machine to capture all incoming packets and at the same time ping the target machine. The *tcpdump* output should show ICMP-EchoRequest packets with the destination address of the target machine.

4.4.2. OpenDayLight SDN usage

This section describes how to use one of the OpenDayLight controllers provided by SoftFIRE platform.

In order to request the SDN controller usage the experimenter must declare the SdnResource node type in its experiment.yaml file referencing the OpenDayLight controller provided by a particular testbed as described in the following snippet:

```
sdn_ericsson:
  properties:
    resource_id: sdn-controller-odl-ericsson
    type: SdnResource
```

Below a full example of an experiment.yaml with NFV and SDN-ODL resources:

```
---
description: "Template for SoftFIRE yaml resource request
definition"
imports:
  - softfire_node_types:
http://docs.softfire.eu/etc/softfire_node_types.yaml

topology_template:
  node_templates:

    iperf:
      type: NfvResource
      properties:
        resource_id: iperf
        nsd_name: The Iperf NSD
        testbeds: { ANY: ads }

    ericsson_ads:
      properties:
        resource_id: sdn-controller-odl-ericsson
      type: SdnResource

tosca_definitions_version: tosca_simple_yaml_1_0
```

Once deployed an experiment with OpenDayLight resource, in the Experiment Manager Dashboard are showed the details of the SDN resource allocated:

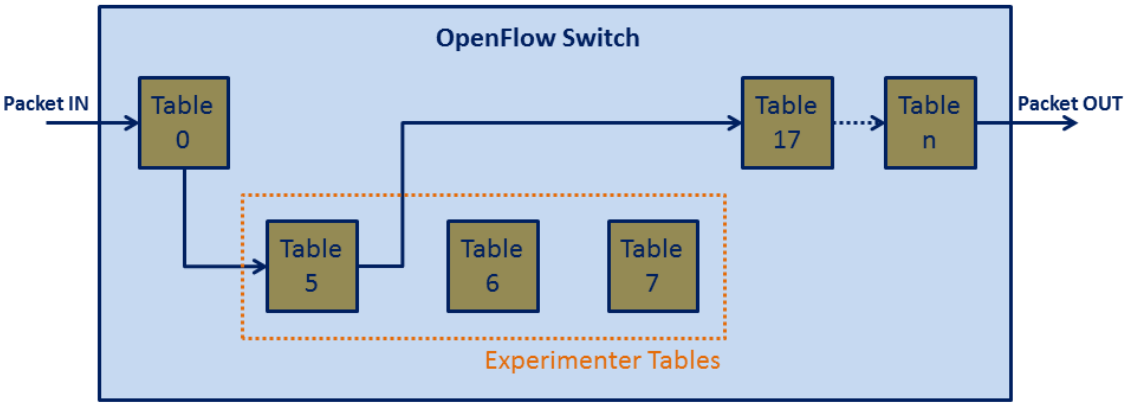
```
{
    "resource_id": "sdn-controller-odl-ads",
    "token": "bcbabcbabcbabcbabcbabcbabcbabcba",
    "flow-table-range": [
        2,
        3,
        4
    ],
    "URI": http://172.20.70.130:8001/
}
```

The details show:

- **resource_id**: the resource id requested
- **URI**: OpenDayLight endpoint used for RESTCONF API requests
- **token**: token to be inserted in the HTTP header of RESTCONF-API requests
- **flow-table-range**: OpenFlow tables assigned to the experimenter

After the OpenDaylight SDN controller is assigned, the experiment can call the OpenFlow plugin API of the controller using the URI received with the header `'api-token'` with the value equals to the token received

When the experiment starts, for every OpenFlow node, all the flows of table 0 regarding its VMs are redirected to the first table assigned to the user and then go to table 17



The new flows are created with the following naming convention:

`<TENANT_ID>_<TABLE_ID>_<OVS_PORT>_<NEUTRON_PORT_ID>_<NOVA_VM_ID>`

Below an example of query all nodes of OpenDaylight:

```
curl -X GET \
'http://<URI>/restconf/config/opendaylight-inventory:nodes/' \
  -H 'accept: application/json' \
  -H 'api-token: <token>' \
  -H 'cache-control: no-cache' \
```

Using the above query, the experimenter could have a clear idea of the topology of all OpenDaylight resources.

Below an example of a REST request to drop all packets with destination IP equals to 10.0.10.2/24

```
curl -X PUT \
  'http://<URI>/restconf/config/opendaylight-
inventory:nodes/node/openflow:72664714402125/table/2/flow/1' \
  -H 'accept: application/json' \
  -H 'api-token: <token>' \
  -H 'cache-control: no-cache' \
  -d '{
      "flow-node-inventory:flow": [
        {
          "id": "1",
          "flow-name": "Foo",
          "match": {
            "ipv4-destination": "10.0.10.2/24",
            "ethernet-match": {
              "ethernet-type": {
                "type": 2048
              }
            }
          },
          "priority": 2,
          "table_id": 2,
          "instructions": {
            "instruction": [
              {
                "order": 0,
                "apply-actions": {
                  "action": [
                    {
                      "order": 0,
                      "drop-action": {}
                    }
                  ]
                }
              }
            ]
          }
        }
      ]
    }'
```

NB: To ensure experiment isolation, some OpenFlow action cannot be applied. Inside the attribute “output-node-connector”, the allowed action attribute values are only "TABLE" and "IN_PORT". The values "ALL", "CONTROLLER", "ANY", "LOCAL", "NORMAL", "FLOOD" are not allowed.

For further information about these values, you can read the official OpenFlow documentation at https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.0.pdf

All the requests that not satisfy the above requirements, will be rejected with an http status code = 403

The official OpenDaylight OpenFlow Plugin documentation can be found at https://wiki.opendaylight.org/view/OpenDaylight_OpenFlow_Plugin:End_to_End_Flows

Examples for XML for various flow matches, instructions & actions can be found at https://wiki.opendaylight.org/view/Editing_OpenDaylight_OpenFlow_Plugin:End_to_End_Flows:Example_Flows, but remember that the SoftFIRE OpenDaylight controller accepts only JSON requests

5 Conclusions

This document is the Handbook of the SoftFIRE Middleware components and its actual instantiation across different locations in Europe. The steps defined in this document are guidelines for any experimenter either planning to use the SoftFIRE live infrastructure or making use of the SoftFIRE components released openly to the community.

This document provides a description about the SoftFIRE Middleware, a set of software components acting as microservices for providing on demand experimentation and federation. The designed and developed Experiment Manager (EM) represents a novel approach to utilize TOSCA as definition of experiments comprising heterogeneous resources in the context of NFV and SDN.

All experiments executed during wave 1 and wave 2 have been analysed in order to understand the requirements and identify potential solutions to include in the SoftFIRE Middleware. The current version of the platform has reached the first stage of consolidation and it is ready to be further extended with new features. The integration of a Continuous Integration / Continuous Development system allowed also the identification of potential issues in a shorter amount of time.

This document constitutes the basis for the development of experimentations and projects on the SoftFIRE platform. The intended audience is any experimenter of the platform and those that will be using it during the SoftFIRE Challenge. Therefore, it has been designed and edited in order to be self-contained. However, the SoftFIRE team suggests checking the web documentation for improvements and news about the middleware infrastructure.

References

[1] GENI, “SFA,” [Online]. Available: http://groups.geni.net/geni/wiki/SliceFedArch.

[2] “gRPC,” Google, [Online]. Available: https://grpc.io/.

[3] F. Ceratto, “Cork,” [Online]. Available: http://cork.firelet.net/.

[4] Open Baton, “Open Baton is an open source project providing a comprehensive implementation of the ETSI Management and Orchestration (MANO) specification.,” [Online]. Available: http://openbaton.github.io.

[5] OpenStack, “Open source software for creating private and public clouds.,” [Online]. Available: https://www.openstack.org/.

[6] TeamViewer, “Empowering secure remote access and support,” [Online]. Available: https://www.teamviewer.com/en/.

[7] Fraunhofer FOKUS, “oSDNc-proxy at GitHub,” 2017. [Online]. Available: https://github.com/softfire-eu/sdn-proxy-osdnc.

[8] Gradle Inc., “Gradle Build Tool,” [Online]. Available: https://gradle.org/.

[9] Apache, “Apache Maven Project,” [Online]. Available: https://maven.apache.org/.

[10] “Jenkins,” [Online]. Available: https://jenkins.io/index.html.

[11] Fraunhofer FOKUS, “OpenSDNcore Northbound-API documentation,” 2017. [Online]. Available: http://docs.softfire.eu/opensdncore-nb-api/.

[12] Open Baton, “openbaton,” [Online]. Available: http://openbaton.github.io/.

[13] Open Baton, “Open Baton Descriptor Model,” [Online]. Available: http://openbaton.github.io/documentation/ns-descriptor/.

[14] Open Baton, “Open Baton Virtual Network Function Descriptor,” [Online]. Available: http://openbaton.github.io/documentation/vnf-descriptor/.

[15] Open Baton, “Open Baton VNF Package how to,” [Online]. Available: http://openbaton.github.io/documentation/vnf-package/.

[16] Open Baton, “Open Baton dependency management between VNF,” [Online]. Available: http://openbaton.github.io/documentation/vnfm-generic/.

[17] Zabbix, “The Ultimate Enterprise-class Monitoring Platform,” [Online]. Available: http://www.zabbix.com/.

6 List of Acronyms and Abbreviations

Acronym	Meaning
API	Application Programming Interface
CC	FDC, Cluster Controller
CM	FDC, Cluster Member
CP	3GPP cellular Control Plane
CUPS	Is a 3GPP, Rel-14 feature that separates out the SGW and PGW functionality into CP and UP entities.
EM	Experiment Manager
EPC	3GPP, Enhanced Packet Core
EPC_CPN	EPC Control Plane Node
EPC_UPN_CC	EPC User Plane Node with CC control
EPC_UPN_CM	EPC User Plane Node with CM control
FDC	*Flat Distributed Cloud architecture* *(5G Core Network Architecture proposal from UoS)*
GUI	Graphical User Interface
MANO	ETSI Management and Orchestration
MGW	Media Gateway
NFV	ETSI, Network Function Virtualisation
NS	Network Service
NSD	Network Service Description
PoP	Point of Presence
PDN	Packet Data Network
PGW	3GPP cellular PDN Gateway
PGWc	is a, CUPS evolved PGW (CP) entity
PGWu	is a, CUPS evolved PGW (UP) entity
PPE	Packet Processing Entity
SDN	Software defined networking
SFA	Slice Facility Architecture
SGW	3GPP cellular Serving Gateway
VIM	Virtualized Infrastructure Manager
VNF	Virtual Network Function

VNFD	**MANO Virtual Network Function Description**
VNFM	**Virtual Network Function Manager**
VNFR	**Virtual Network Function Record**
VPN	**Virtual Private Network**

A. SoftFIRE node templates

```
description: "SoftFIRE Node Types definitions"
tosca_definitions_version: tosca_simple_yaml_1_0
node_types:
  MonitoringResource:
    derived_from: eu.softfire.BaseResource
    description: "Defines the Zabbix monitoring resource requested"
    properties:
      lan_name:
        description: "Network to attach the server"
        type: string
      testbed:
        description: "Location where to deploy the monitoring server"
        required: false
        type: string
  NfvResource:
    derived_from: eu.softfire.BaseResource
    description: "Defines a NFV resoruce request in the SoftFIRE
Middleware"
    properties:
      ssh_pub_key:
        required: false
        type: string
      file_name:
        required: false
        type: string
      nsd_name:
        type: string
      testbeds:
        entry_schema:
          description: "mapping between vnf types and testbed. Or
'all':<testbed_name> for all in one"
          type: string
        type: map
  SdnResource:
    derived_from: eu.softfire.BaseResource
    description: "Defines a SDN resource request in the SoftFIRE
Middleware"
  PhysicalResource:
    derived_from: eu.softfire.BaseResource
    description: "Defines a Physical resource request in the SoftFIRE
Middleware"
  SecurityResource:
    derived_from: eu.softfire.BaseResource
    description: "Defines a Security agent to be deployed. More
details on *docu_URL*"
    properties:
      allowed_ips:
        entry_schema:
          type: string
        required: false
        type: list
      default_rule:
        required: true
```

```
        type: string
      denied_ips:
        entry_schema:
          type: string
        required: false
        type: list
      logging:
        required: true
        type: boolean
      testbed:
        required: false
        type: string
      want_agent:
        required: true
        type: boolean
      lan_name:
        required: false
        type: string
eu.softfire.BaseResource:
  derived_from: tosca.nodes.Root
  properties:
    resource_id:
      required: true
      type: string
    start-date:
      required: false
      type: string
    end-date:
      required: false
      type: string
```

B. iPerf experiment.yaml

```
description: "Experiment containing an NFV resource for deploying an
iPerf server and client."
imports:
  - softfire_node_types:
http://docs.softfire.eu/etc/softfire_node_types.yaml
topology_template:
  node_templates:
    iperf-tutorial:
      properties:
        resource_id: iperf
        testbeds:
          ANY: fokus
        ssh_pub_key: "ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQDNyPHgao8TpwOuEYo497w7kKOztMiRQ9m59ZBZr7
Xnb6LWcIhjuqrxlFmNlluw7V1+hot6RA6psh2xPC/+urTfevIY6p8pOZmPtMToZmP2/5BP
VkBAHAUISU/BZxDAM75QX14CHg/4imcfLxzLyx4XY0SjwfrxtqTTJJW2khKJ5eNoMHnw9+
NwNiM1BY9A1khZ2WXZMAlG8+NkZU+UZhIiyHWYZQU8ZrC02qI/zFfaGFX7OC/yDGbZOeGV
/cuissvai4vn8gtS1Stdj+QJZ/Mcl3t2A65FlW8oSYzih+OrxLsvJ2w8dxBBdw39lPcihb
3e8Za29aiKMrUINleVJ82P"
        nsd_name: "iperf"
      type: NfvResource
tosca_definitions_version: tosca_simple_yaml_1_0
```

C. Custom experiment.yaml

```
description: "Template for SoftFIRE yaml resource request definition"
imports:
  - softfire_node_types:
http://docs.softfire.eu/etc/softfire_node_types.yaml
topology_template:
  node_templates:
    customnsd:
      properties:
        resource_id: custom-iperf
        testbeds:
          ANY: fokus
        nsd_name: "iperf-nsd"
        file_name: Files/custom-iperf.csar
      type: NfvResource
tosca_definitions_version: tosca_simple_yaml_1_0
```

D. Custom-iperf.yaml

```
tosca_definitions_version: tosca_simple_profile_for_nfv_1_0
description: "NS for deploying a clean ubuntu machine"
metadata:
  ID: Custom Iperf NSD
  vendor: SoftFIRE
  version: "1.0.0"
topology_template:
  node_templates:
    iperf-server:
        type: openbaton.type.VNF
        properties:
          vendor: SoftFIRE
          version: 1.0
          endpoint: generic
          type: server
          deploymentFlavour:
            - flavour_key: m1.small
        requirements:
          - vdu: VDU2
        interfaces:
          lifecycle:
            instantiate:
              - install.sh
              - start-server.sh

    iperf-client:
      type: openbaton.type.VNF
      properties:
        vendor: SoftFIRE
        version: 1.0
        type: client
        deploymentFlavour:
          - flavour_key: m1.small
        endpoint: generic
      requirements:
         - vdu: VDU1
      interfaces:
          lifecycle:
            INSTANTIATE:
              - install.sh
            CONFIGURE:
              - server_configure.sh

    VDU1:
      type: tosca.nodes.nfv.VDU
      properties:
        scale_in_out: 1
      requirements:
        - virtual_link: CP1

    VDU2:
      type: tosca.nodes.nfv.VDU
      properties:
```

```
        scale_in_out: 3
      requirements:
        - virtual_link: CP2

    CP1:
      type: tosca.nodes.nfv.CP
      properties:
        floatingIP: random
      requirements:
        - virtualBinding: VDU1
        - virtualLink: softfire-internal

    CP2: #endpoints of VNF2
      type: tosca.nodes.nfv.CP
      requirements:
        - virtualBinding: VDU2
        - virtualLink: softfire-internal

    softfire-internal:
      type: tosca.nodes.nfv.VL

relationships_template:
  connection_server_client:
    type: tosca.nodes.relationships.ConnectsTo
    source: iperf-server
    target: iperf-client
    parameters:
        - softfire-internal
```

E. Section 4.1.2 scripts

```
#!/bin/bash
sudo apt-get update && sudo apt-get install -y iperf screen
```

install.sh

```
#!/bin/bash
screen –d –m –S server iperf -s
```

start-server.sh

```
#!/bin/bash
screen –d –m –S client iperf –c $server_softfire_internal –t
```

server_configure.sh