SOFTFIRE

# SoftFIRE

## Software Defined Networks and Network Function Virtualization Testbed within FIRE+

Grant Agreement Nº 687860

# D2.3

# SoftFIRE (v2) usage manual for NFV/SDN/MEC and 5G experimenters

# WP2

## Infrastructure Programmability, Security and Experimentation Enablement

**Version:** 1.2.
**Due Date:** January 31st, 2017
**Delivery Date:** February 9th, 2017
**Type:** Report (R)
**Dissemination Level:** PU - public
**Lead partner:** TUB
**Authors:** All Partners (See List of Contributors below)
**Internal reviewers:** Roberto Minerva (TIM/EIT Digital)

# 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 |
|---|---|---|---|---|
| 0.1 | 10.11.2016 | Lorenzo Tomasini | TUB | ToC |
| 0.2 | 17.01.2017 | Lorenzo Tomasini | TUB | Integration of partners contributions |
| 1.0 | 20.1.2017 | Bjoern Riemer | FOKUS | Internal Review Version |
| 1.1 | 02.02.2017 | Giuseppe Carella | TUB | Internal Review, update and consolidation |
| 1.2 | 09.02.2017 | Susanne Kuehrer<br>Roberto Minerva<br>Giuseppe Carella | EIT Digital<br>TUB | Final consolidation and editing |

**Annexes:**

| Nº | File Name | Title |
|---|---|---|
| 1 | | Router Mongo DB VNFD |

**Contributors:**

| Contributor | Partner |
|---|---|
| Lorenzo Tomasini | TUB |
| Giuseppe Carella | TUB |
| Daniele Carmignati | Reply |
| Gerry Foster | UoS |
| Serdar Verul | UoS |
| Umberto Stavato | Ericsson |
| Marco Persichini | Ericsson |
| Bjoern Riemer | FOKUS |
| Roberto Minerva | TIM/EIT Digital |
| Susanne Kuehrer | EIT Digital |
| | |

**Deliverable Title:** SoftFIRE (v2) usage manual for NFV/SDN/MEC and 5G experimenters

**Deliverable Number** D2.3

**Keywords:** OpenBaton; jFED; FITEagle; NFV

# Executive Summary

The SoftFIRE project aims 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, or as defined in the previous version of this deliverable, D2.1 (D2.1 SoftFIRE, July 2016), layers, each of them providing a different feature to the whole infrastructure.

Since the important goals of this project have to be reached in a quite short timeframe, the SoftFIRE infrastructure is in constant development in order to enable specific new features and to improve what is already implemented after the first year of the project.

For that reason, this deliverable provides a description of how the experimenters have to interface with the infrastructure. Being a work in progress could still be updated until the end of the project as the infrastructure is continuously evolving and improving. Even if some improvements in the infrastructure taking place unnoticed from the external users, some of them could require modification of the usage manual thus to inform the experimenters.

# Table of Contents

# Table of Figures

# List of Tables

# 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.

Before presenting the different mechanisms, which could be used for interacting with SoftFIRE, a brief overview is provided, available also in D1.1 (D1.1 SoftFIRE, February 2017), presenting the SoftFIRE federated infrastructure architecture. Figure 1 shows the SoftFIRE federated infrastructure architecture. This architecture enhances the one already presented in deliverable D1.1.

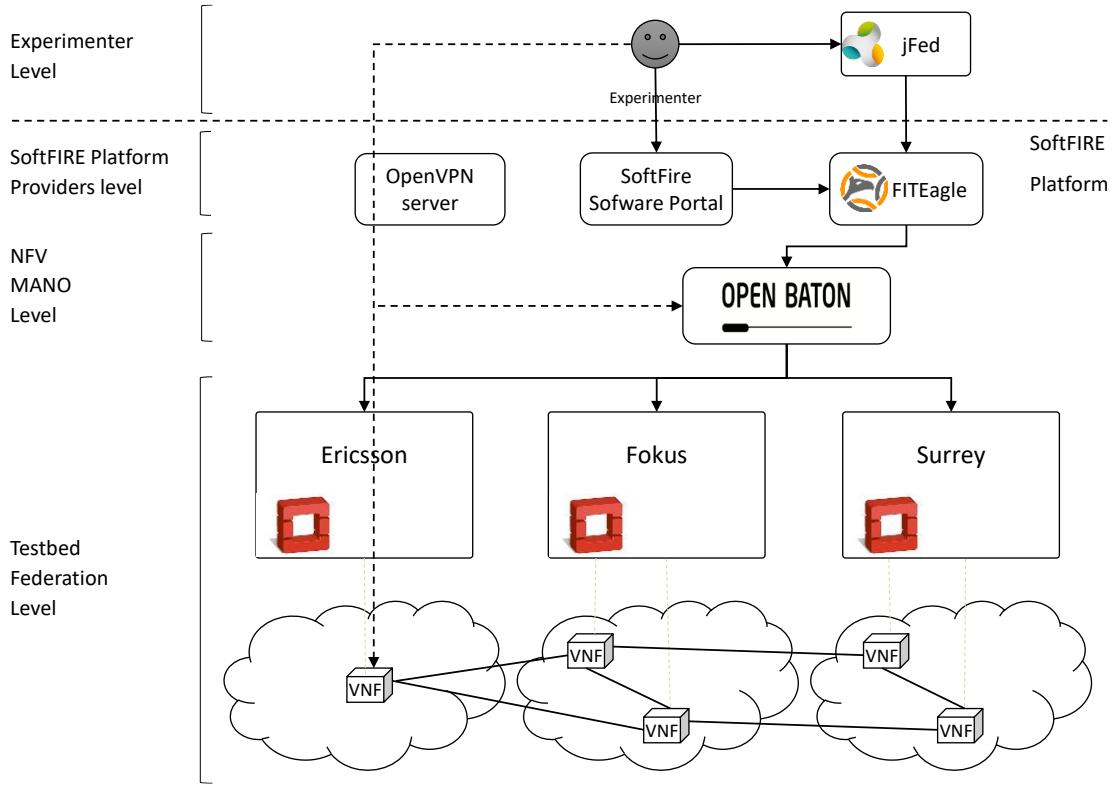


**Figure 1: Architecture v2**

There are four layers constituting the SoftFIRE federated architecture:

- **Experimenter level**:
  - jFed is in charge of interfacing with FITeagle, providing a SFA client

  - Direct access to OpenBaton on MANO NFV (ETSI MANO, 2014) level[1]

- **SoftFIRE Platform Providers level:**
  - FITEagle provides SFA capabilities
  - SoftFIRE Software Portal aims to fulfil the provisioning of the experimenter software into the SoftFIRE infrastructure
  - OpenVPN server allows the experimenter to access the VMs of his experiments
- **NFV MANO level:**
  - Open Baton provides NFV MANO capabilities to the infrastructure
- **Testbed Federation level:**
  - Multiple testbeds that act as multiple Point of Presences (PoPs) managed by the NFV MANO level

For what concerns The NFV MANO level, we used the open source NFV MANO implementation, Open Baton (Open Baton, 2016), in particular an enhanced version starting from 3.0.x.

In the SoftFIRE Platform Provider level, the SoftFIRE Software Platform was implemented as well as the Open Baton adapter for FITEagle that is able to manage and translate the SFA API to the NFV API and to ensure the separation of environments for all the SoftFIRE experimenters.

The following content is split in two main sections:

the first provides the concrete usage manual describing how and what the experimenter should do for running the experiment during the experiment lifecycle,

the second one describes to the readers the capabilities that the testbeds provide to the experiments out-of-the-box.

Finally, before the conclusion, a complete example is provided, also available for experimenters in the SoftFIRE infrastructure, ready to start with simple deployments on top of the infrastructure.

[1] Direct access to the Open Baton APIs has been allowed during the last month. Main reason, also explained later on in following sections, is that experimenters required additional capabilities not supported directly by the FIRE APIs.

# 2 Experiment Lifecycle

The experiment lifecycle comprehends different phases:

1. Access to the SoftFIRE infrastructure
2. Design of the experiment
3. Validation of the experiment
4. Gathering of the results of the experiment

The first step is realised with accessing the SoftFIRE federated infrastructure. As stated in deliverable D1.1 (D1.1 SoftFIRE, February 2017), SoftFIRE follows the FIRE architecture guidelines, and provides the SFA API (GENI) as main mechanism for allowing usage of the federated testbed infrastructure.

After getting access to the SoftFIRE federated infrastructure, the experimenter can start with the development of its own experiment. The experiment usually consists of a design phase, followed by a validation phase in which the designed solution is implemented and tested on top of the federated infrastructure. During the design phase, the experimenters should design their solutions considering the technologies provided by the SoftFIRE infrastructure, while during the validation phase they have to deploy/execute their developed software artefacts on top of the federated infrastructure.

Finally, results can be gathered using the supporting monitoring services provided by the SoftFIRE infrastructure, in order to prove the correct execution of the experiment and validate the solutions, which have been proposed.

All these phases are described more in details in the following subsections and are followed by some practical examples for simplifying their interaction with the platform.

## 2.1 Access the portal

The first entry point for the experimenter is the SoftFIRE Access Portal, which is reachable at https://portal.softfire.eu. Figure 2 shows the landing page from where the user can sign up to the portal. After approval of the sign up, the experimenter can log in.

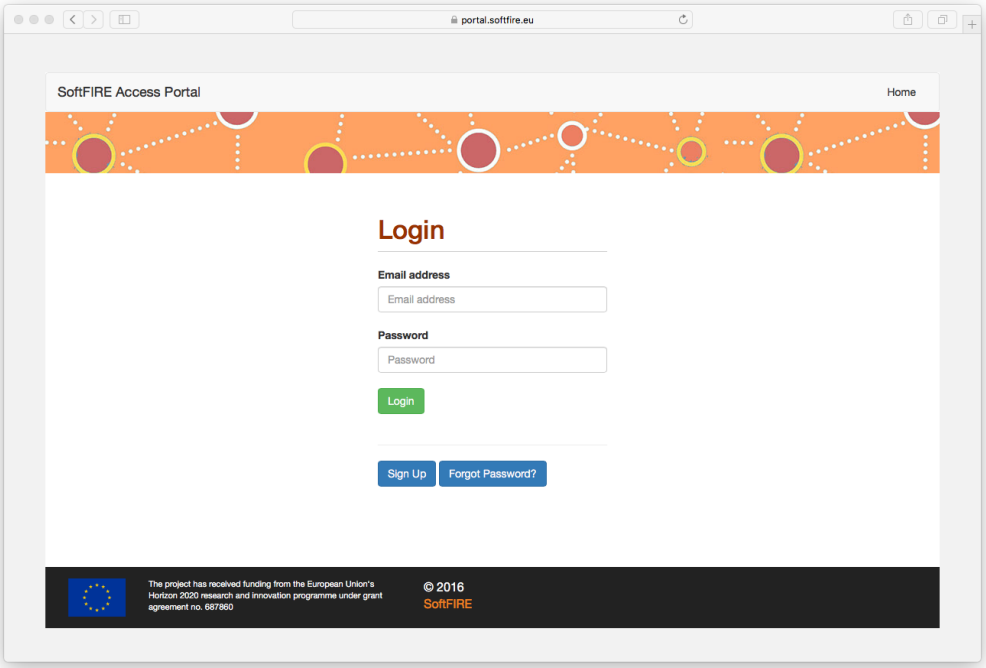


**Figure 2: SoftFIRE Access Portal**

The welcome page (Figure 3) provides additional information about the steps, which have to be executed, such as install Java 8, download the jFed Experimenter GUI and where to retrieve and afterwards put important configuration files.

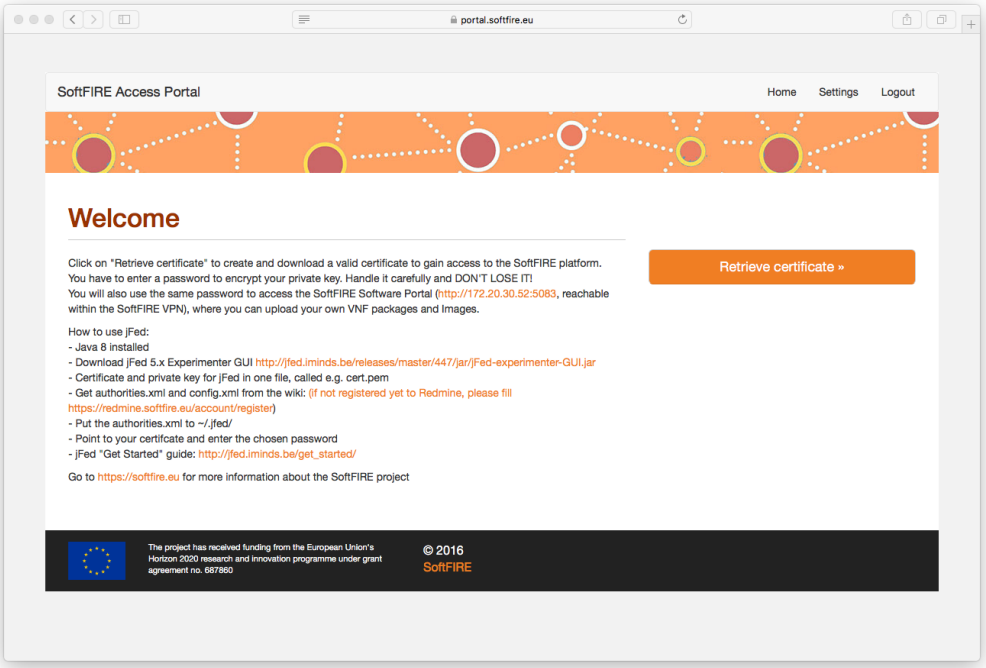


**Figure 3: Access Portal Welcome Page**

Prerequisite for the next steps is the retrieval of the personal certificate and OpenVPN configuration files. The process is triggered by clicking on the button on the right.

The certificate is protected with the password provided by the user during the generation of the certificate. At this stage an experimenter has full access to the SoftFIRE network (refer to deliverable D1.1 (D1.1 SoftFIRE, February 2017) for more details about the SoftFIRE networking infrastructure).

## 2.2 Design Phase

This section describes the most complex phase of the experiment execution: the design phase. One of the key points provided by the SoftFIRE federated infrastructure is the management and orchestration set of components for automating the execution of the experiment. Considering that the scope of the SoftFIRE project is to design solutions for the NFV and SDN domain, it was decided from the very beginning of the project to provide an ecosystem where experimenters could deal with state of the art technologies in these fields. Although the automation of the management and orchestration procedures brought many improvements in the time-to-market, system integration, decreased costs, along with the others, network function providers still need to determine a good network function “virtualization” process automation, also due to the variety of the NFV Management and Orchestration solutions available in the open ecosystem. The SoftFIRE infrastructure technologies are based on standard compliant solutions, like OpenStack as standard de-facto Virtualized Infrastructure Manager (VIM) and Open Baton as ETSI NFV MANO v1.1.1 compliant framework. Those technologies have been exposed to experimenters also via FIRE interfaces, and experimenters implementing their own solutions on top of SoftFIRE, could later on replicate this environment installing on their local premises those technologies.

### 2.2.1. Design the experiment

This section describes how the experiment should be designed and implemented for being executed on top of the SoftFIRE infrastructure. A practical example of this process is defined in Section 4, where a complete example for creating and executing a “hello world” kind of application is provided.

In order to be aligned with the ETSI NFV terminology, each experiment can be seen as a description of a Network Service (NS). A NS is by definition of the ETSI NFV MANO specification, a combination of multiple Virtual Network Functions (VNFs). A VNF is the combination of a virtual compute resource and the software artefact of the network functions. In order to deploy on an automated way those resources, it is required to provide a template, which will be named from now on Network Service Descriptor (NSD) in line with the terminology used by the ETSI NFV specification v1.1.1 (ETSI MANO, 2014). Therefore, the experimenters who would like to deploy their innovative solutions on top of SoftFIRE, should basically build a NS containing one or more VNFs, and provide for each of them their descriptors and software artefacts (more details will be given in the following sections about those points). Multiple instances of the same NSD could be launched by an experimenter, and each instance of such descriptor is named Network Service Record (NSR) in line with the terminology used by the ETSI NFV specification v1.1.1 (ETSI MANO, 2014).

#### *2.2.1.1. Design the NS solution*

As already mentioned in the previous paragraph, the design of an experiment corresponds to the design of a solution solving a particular problem in the area of SDN and NFV. An experimenter has to provide its solution as NS that could be executed on top of the SoftFIRE infrastructure using the APIs provided by the SoftFIRE software stack.

As mentioned already, SoftFIRE provides FIRE APIs on top of the NFV and SDN technologies used in the underneath testbeds for facilitating the management and orchestration operations required for the execution of an experiment. In addition, the enablement of these APIs allows the experimenter to use some kind of tools, such as jFed (iMinds), helping delineating the experiment by graphically composing the involved nodes (VNFs) as network services.

The following section focuses on the particular steps for building a VNF using mechanisms and technologies provided by SoftFIRE.

##### 2.2.1.1.1. VNF design

A VNF is by definition the combination of a virtual compute resource and the software artefact of the network functions. What makes quite different a VNF from a standard network function is the agile mechanism used for deploying, configuring and starting it. Thus, a VNF provider, in this case an experimenter, has to implement the lifecycle management procedures in order to deploy on demand its solution.

The combination of those lifecycle procedures (typically implemented with scripting languages), and the VNFD is called VNF Package. Alongside this lifecycle procedure, there are other resources which are carried together with the VNF to conceive: the description of the VNF, the description of the VNF package and the Virtual Network Function (VNF) software artefact. The VNF descriptor, the VNF Package descriptor and the lifecycle management procedures will create the VNF Package itself that will be uploaded in the SoftFIRE Software Portal (SSP) together with the VNF image.

The VNF image is supposed to contain the software artefact, which can be used for creating a running virtual machine on top of the SoftFIRE infrastructure. Usually experimenters can decide to have vanilla images (meaning containing only the basic operating system) and install their software on demand using orchestration procedures, or images where the software is already installed but not configured. The VNF code can be public or not, but, in both cases, it is suggested to prepare an image starting from one of the available ones that can be found in the Redmine wiki page[2] containing it. This process will be explained in details in the following sections.

To summarize, the VNF is described by the VNF Package, and the experimenter, in order to be able to use a VNF in its experiment, has to upload the VNF Package and related image in the SoftFIRE platform. The VNF Package is composed by:

- VNF descriptor (VNFD)
- Package descriptor (Metadata.yaml)
- Lifecycle management procedures (scripts)

[2] https://redmine.softfire.eu/projects/softfire/wiki/ImageCreation

The VNF Descriptor (Table 1: VNF Descriptor) follows the information model as specified by the ETSI NFV group in the MANO v1.1.1 specification, which is fully compliant also with the Open Baton information model (Open Baton) (Open Baton Descriptor Model) (Open Baton VNF Descriptor). Examples can be also found in the Redmine wiki page. The following tables explain the fields that compose the VNFD (Table 2: Virtual Deployment Unit) and that has to be created by the experimenter (Table 3: Lifecycle Event).

**Table 1: VNF Descriptor**

| Field name | Description | Type |
|---|---|---|
| **name** | The name of the VNF. Unique in a NS | String |
| **vendor** | The provider of the VNF | String |
| **version** | The version of the VNF | String |
| **type** | The VNF type. This field is relevant for the dependencies configurations. Must be unique and cannot contain special chars only letter and digits | String |
| **endpoint** | The VNFM that will manage the VNF. Must be "generic" | String |
| **vdu** | The NFV requirement of the VNF. See Table 2 | List |
| **configurations** | The configuration specific for this VNF containing a name and a list of ConfigurationParameters composed by key, name | Json object |
| **virtual_link** | The network to use. If not specified differently by the SoftFIRE use { "name":"softfire-internal"} | Json object |
| **deployment_flavour** | The flavour key[3] to use: possibilities are:<br>• m1.tiny<br>• m1.small<br>• m1.medium | Json object |
| **lifecycle_event** | The lifecycle management procedures. See Table 3 | List |
| **requires** | The logical requirement that the VNF has. This is explained more in depth later on. | Json object |

[3] http://docs.openstack.org/admin-guide/cli-manage-flavors.html provides an overview of resources offered by each flavour. Please consider that the physical infrastructure is shared among experimenters. If the experimenter has any specific requirements, please contact the SoftFIRE Team.

**Table 2: Virtual Deployment Unit**

| Field name | description | Type |
|---|---|---|
| **vm_image** | The list of image names to use. This name will be available once the VNF image is uploaded to the SSP | List |
| **scale_in_out** | Number of maximum VNF component. One is minimum | Int |
| **vnfc** | The VNF Components that are reflected as VMs. It contains the list of Connection Points that are the network that this component must be connected to. Use<br>{<br>"floatingIp":"random",<br>"virtual_link_reference":"softfire-internal"<br>}[4] | List |
| **vimInstanceName** | A list of Point of Presence. If more than one is provided the NFVO will chose randomly. Please specifiy the PoP where deploy the VM in between:<br>• **vim-instance-fokus**, in order to deploy in Fokus testbed<br>• **vim-instance-er**, in order to deploy in Ericsson testbed<br>• **vim-instance-uos**, in order to deploy in Surrey testbed<br>• **vim-instance-ti**, in order to deploy in Telecom Italia testbed. | list |

**Table 3: Lifecycle Event**

| Field name | description | type |
|---|---|---|
| **event** | The lifecycle event code in which these procedures must be executed:<br>• INSTANTIATE<br>• CONFIGURE<br>• START | String |

[4] If the experimenter has any specific requirements, should contact the SoftFIRE Team

| | | |
|---|---|---|
| **lifecycle_events** | List of scripts names (procedures) that has to be called during this lifecycle event | List |

This describes the VNF that the experimenter wants to upload.

A network service is a combination of one or more VNFs. In most of the cases, there are dependencies between VNFs: for instance, considering that we are dealing with a very dynamic environment, it is usual that IPs are dynamically allocated to running instances. Therefore, it is important to have a mechanism for solving at runtime those dependencies. The simplest example is a client server kind of network service in which the client requires the IP of the server in order to start a connection. In this case, this dependency has to be specified in the NS. In SoftFIRE, the way the dependencies are defined is through the "**requires"** field in the VNFD. This field looks like the following:

```
"requires":{
        "type_from_which_the_vnf_requires":{
            "parameters":[ "list of parameter names that this VNF requires" ]
        }
}
```

The parameter names could be:

- The network name (virtual link) for obtaining the private IP
- The network name (virtual link) followed by "_floatingIp" for the public IP
- Configuration parameter keys

These parameters are automatically injected as environment variables in the lifecycle script. The dependencies scripts **must** start with the foreign type followed by underscore ("foreigntype_"), if not, the script will not be executed. More detailed information can be found here directly on the public documentation of the Open Baton website (Open Baton)

As stated above, the VNFs are connected to each other through dependencies. If some dependencies are not satisfied, the onboard of the NS Descriptor is not allowed and an error will be thrown.

If all the VNF will be placed in the same datacenter, then the experimenter needs "public" IPs only for the virtual resources to which he wants to be able to log in. This can be achieved by specifying the "floatingIp" fields in the VNF Descriptor (see Table 2). The floating IP to choose is always "random"[5].

As last information in the VNF Package it is needed to describe the package itself using a Metadata.yaml file as follows:

[5] There are special cases where this is not valid, for instance in case of necessity of usage of RAN. In this case contact the SoftFIRE Team for instructions.

**name**: the name of the package
**description**: "a text description of the package"
**provider**: the provider
**shared**: false
**image**:
  **upload**: false
  **names**:
    - The name of the image. This is known only after uploading he image to the SSP

In order to create a package, all these resources have to be packed in a .tar file together with the scripts representing the lifecycle events in a folder called scripts (Open Baton).

The VNF must run on top of a Virtual Machine that is deployed based on an image. Already provided images can be used or it is also possible to create a specific one and upload it in the SoftFIRE infrastructure.

#### 2.2.1.1.2. Image design

A Software artefact, which is deployed as VNF on the SoftFIRE infrastructure, may have some external dependencies from third parties' libraries. For instance, while installing a particular binary version of a software component, you may use a package manager, which automatically downloads and installs dependent libraries.

While most of the datacenters provide Internet connectivity from a VM, some not, thus in some cases it may be necessary to prepare a software image in advance containing the preinstalled dependencies (and binary code of the VNF software). This also helps the deployment phase because it accelerates the installation (INSTANTIATE, see Table 3) since all the code is already installed and does not need to be downloaded on demand.

The SoftFIRE platform already provides some preconfigured images (Table 4: Preconfigured images). In case the VNF does not have special requirements, it is then possible to use one of the following images just by putting the name in the VNFD→VDU (see Table 2).

**Table 4: Preconfigured images**

| Name | Description |
|---|---|
| **Ubuntu 14.04 Cloud based** | This image contains the ubuntu 14.04 OS without any further dependencies |
| **ob-ems-mongo-iperf** | This image contains the EMS agent and some preinstalled tools such as mongodb and iperf |
| **OpenIMSCore_preConf_ems-16** | This image contains the preinstalled code of OpenIMS. |

In case the VNF needs a particular software installed or there is the need to reduce the deployment phase time, starting from one of these images it is possible to create a new one.

The process is the following:

1. Download an image from the list of supported OpenStack cloud images: http://docs.openstack.org/image-guide/obtain-images.html

2. Upload it to a local openstack (OpenStack).
3. Launch a VM using this image
4. Log in to this image using Secure Shell (SSH) and the private key selected while launching
5. Install all the needed software
6. Then log out
7. Shutdown the instance
8. Create a snapshot and download it choosing a unique name

There are also other advanced mechanisms that could be used for automating the preparation of an image. For instance, Packer[6] allows you to build images for any virtualization technology using template files.

### 2.2.2. Upload experiment resources

The snapshot (Figure 4) created can be directly uploaded to the SoftFIRE Software Portal (SSP). For accessing the SSP, the same credentials of the SoftFIRE Web Portal are valid.

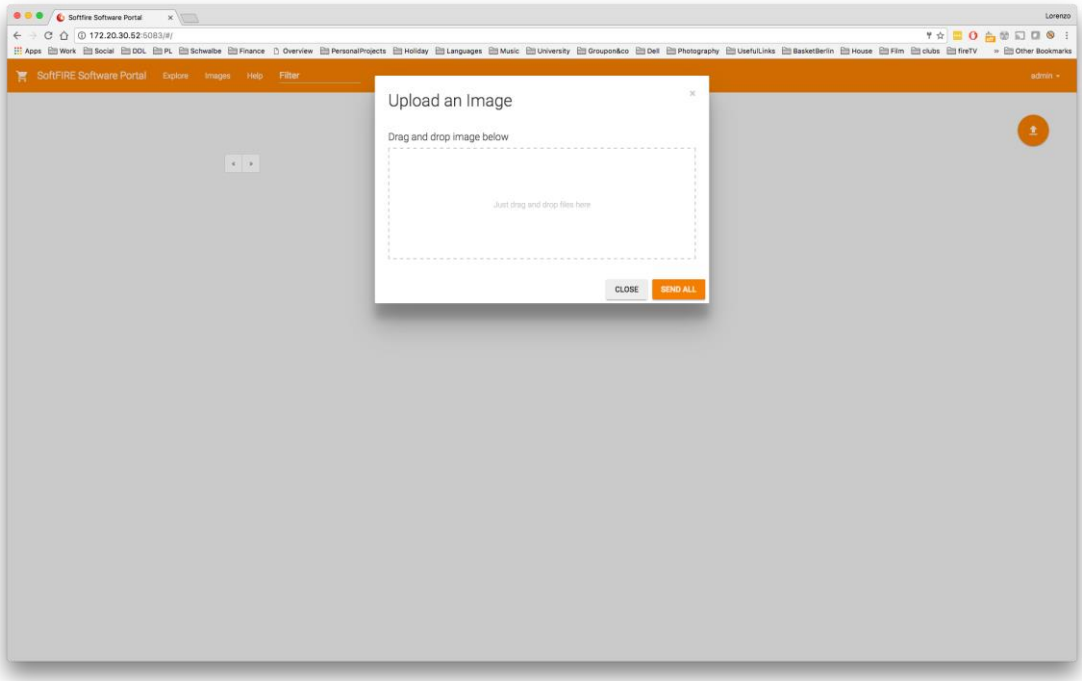


**Figure 4: SSP upload image**

When uploaded, the image will be then transferred to the datacenters and after some minutes it will be available for being used for on demand deployments. Only after this process is successfully concluded, it is possible to upload the VNF Package in the SSP, specifying the image name just uploaded in the VNF Package Metadata.yaml.

[6] https://www.packer.io/

## 2.3 Running the experiment

In order to be compliant with the FIRE interfaces (see Deliverable D1.1 for more details), SoftFIRE decided to expose the SFA APIs to experimenters for executing their experiment. An experimenter could use the jFed Experimenter GUI (provided by (iMinds)) for interacting with defining an experiment interacting with the SoftFIRE platform.

Prerequisites for using jFed are:

- Java 8 installed
- The config.xml and authorities.xml available in the Redmine wiki page[7]
- The personal certificate obtained from the SoftFIRE Access Portal

The authorities.xml file has to be placed in the jFed configuration folder, which is usually:

- for UNIX machines: ~/.jFed/
- for Windows machines: c:\Users\<user name>\AppData\Roaming\jFed\

To start jFed (the use of the console/Terminal/PowerShell is recommended):

"java -jar jFed-experimenter-GUI.jar --config=<path to your config.xml"


**Figure 5: jFed Login Screen**

Figure 5 shows the Login Screen of jFed. For the user certificate, the experimenter has to point to his personal certificate and enter the password selected in the certificate creation process.

[7] https://redmine.softfire.eu/projects/softfire/wiki/ImageCreation

After login, a new experiment can be initiated by clicking the “New” button in the top left corner; this will open the topology editor where the user can drag nodes, which are to be used.

In the current state only Generic Nodes are supported by the SoftFIRE platform (Figure 6). A Generic Node has a 1:1 mapping with a VNF in the SoftFIRE infrastructure.

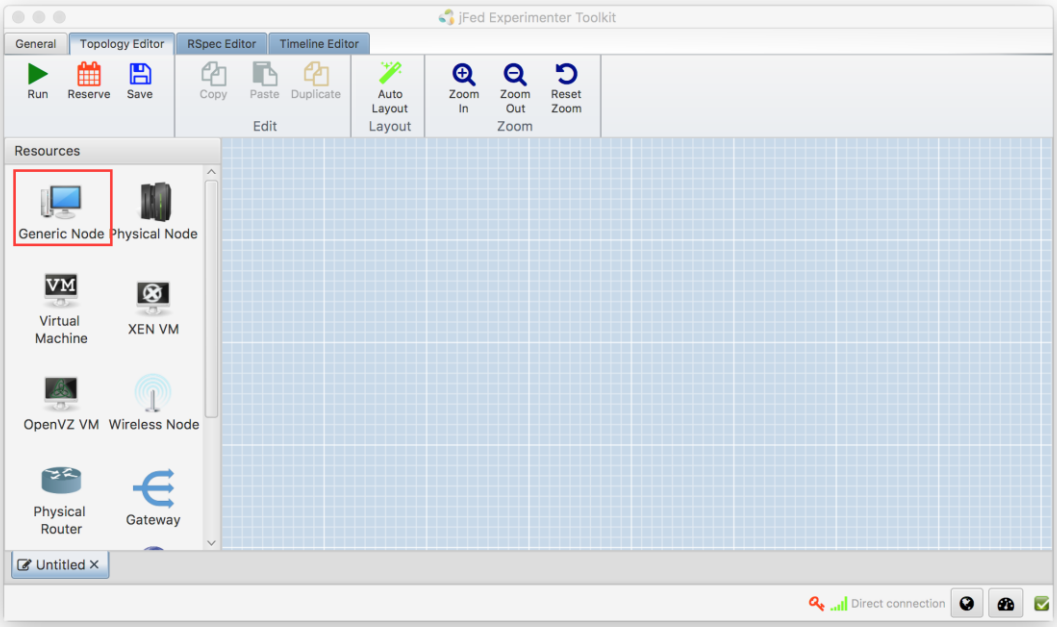


**Figure 6: jFed Topology Theater**

When right clicking on a node placed in the topology editor, the experimenter can configure the node (Figure 7).

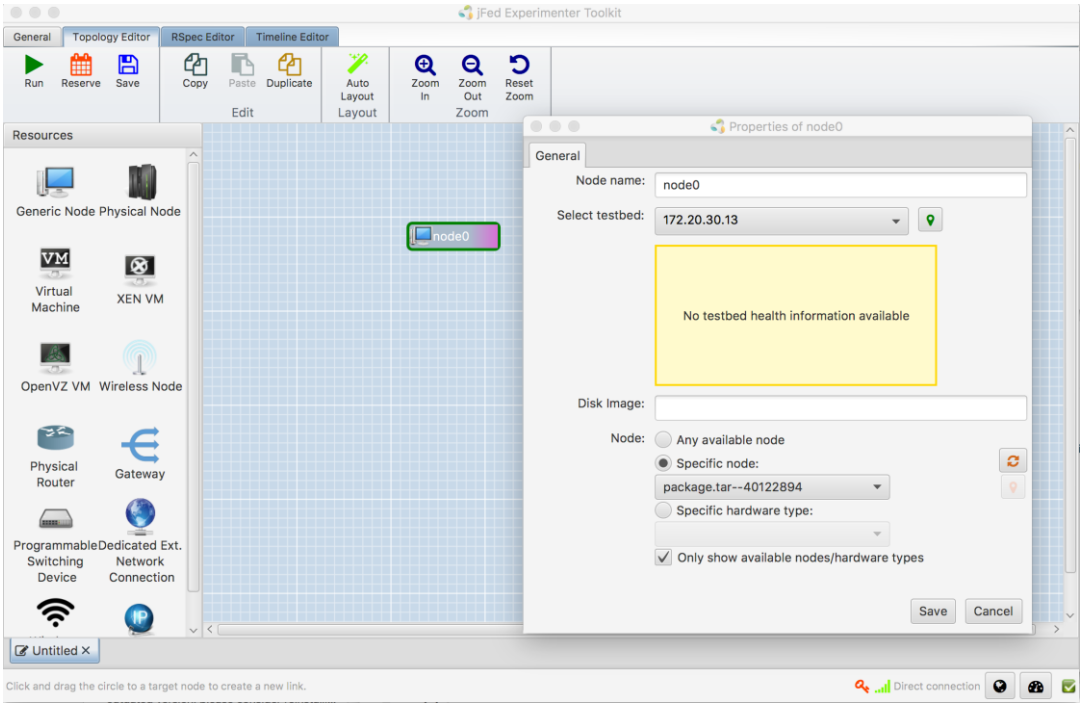


**Figure 7: jFed Configure Node**

Figure 8 shows the node configuration window. The node name can be assigned by the experimenter according to his preferences. The testbed 172.20.30.13 has to be selected from the according dropdown. The VNF package uploaded earlier can be selected from the “Specific node” dropdown. The “Save” button saves the specific configuration. When every node contained in the experiment has been drawn to the Topology editor and configured, clicking on “Run” triggers the deployment of the VNF. Figure 8 shows the dialogue where the experiment’s name and duration can be selected afterwards.

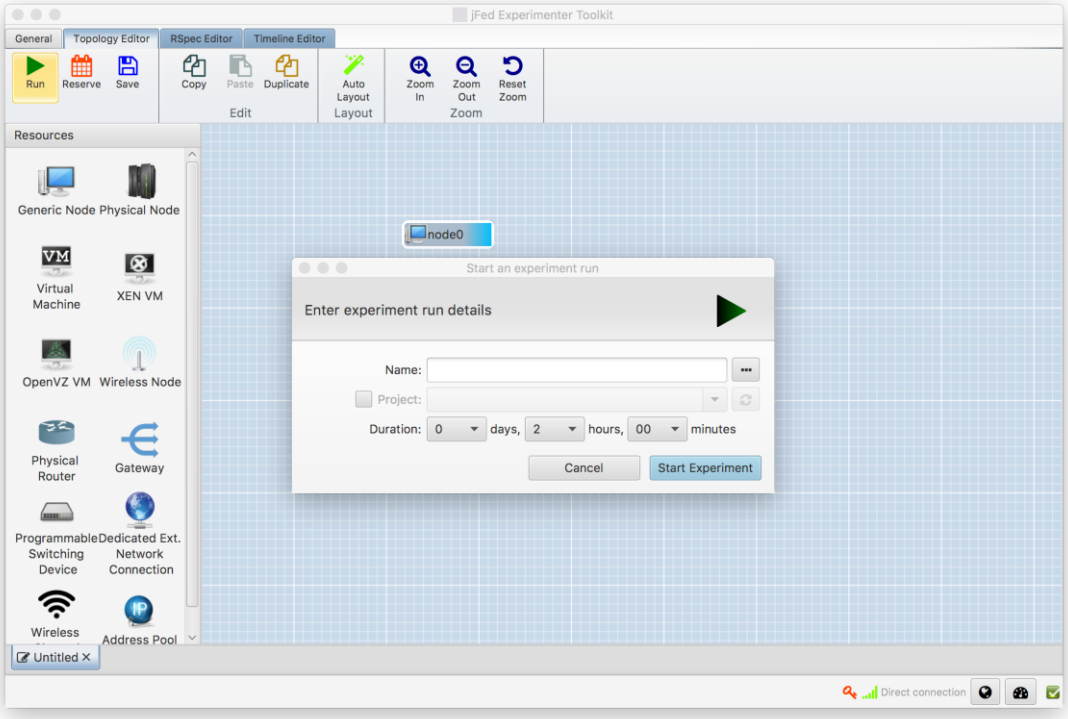


**Figure 8: jFed Run Experiment**

When the deployment of the VNF finishes, which is indicated by a deep green colouring of the node, the user can enter the machine by right clicking on the node Open SSH terminal, which logs in to the node with the credentials provided by the certificate (Figure 9).

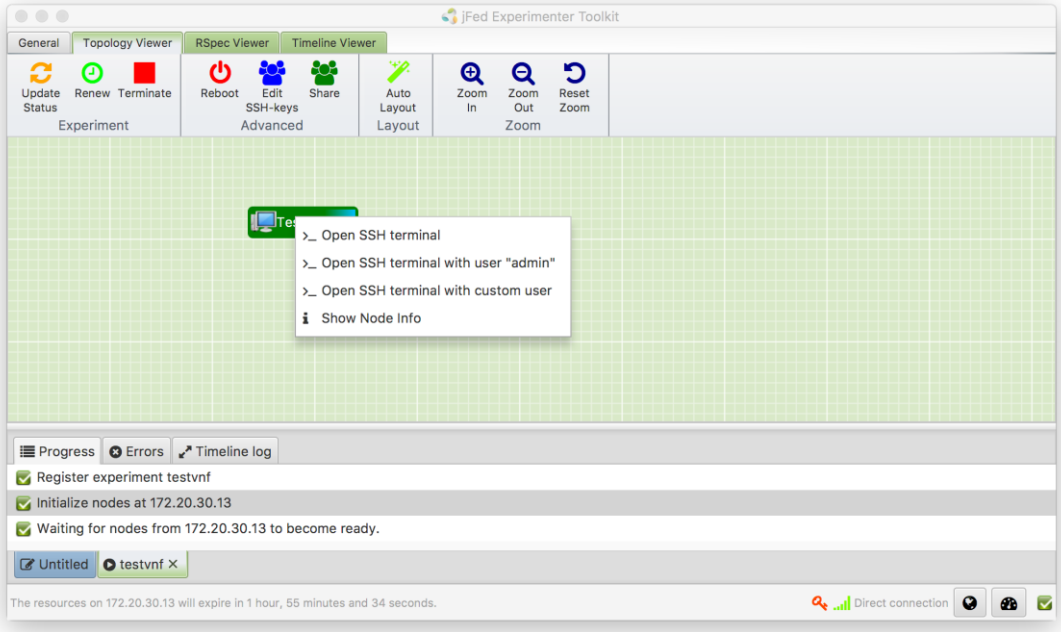


**Figure 9: jFed SSH Login**

## 2.4 Gathering results

A possible way to gather results and monitor the execution of the experiment is to add one VNF to the Network Service: a Zabbix server (Zabbix description).

A Zabbix server collects monitoring information of a system by installing an agent in the system under monitor. For doing this, it is necessary to add one lifecycle management procedure (script) in the VNF the experimenter wants to monitor. The link to the script is available in the Redmine wiki page[8] as well as the Zabbix Server VNF Package.

The Zabbix Server VNF Package is treated by the SoftFIRE platform as any other package provided by the experimenter, meaning that the experimenter has to upload to the SSP the Zabbix Server package as all the others. It has then to be included in the NS definition (RSpec) inside the jFed tool.

Hence, the experimenter has full control on the Zabbix Server, enabling the possibility to create all the required resources or specific items without interfere or without need to share a resource with other experimenters.

Following this approach gives the experimenter the ability to use any monitoring system he/she is familiar with, without being tied to a specific implementation.

[8] https://redmine.softfire.eu/projects/softfire/wiki/ImageCreation

# 3 Testbed functions as a service

The following section gives a rough overview of the functions as a service provided by the SoftFIRE testbed with focus on aspects that are needed for the design of experiments. For the sake of having a complete document for the experimenters, this section could contain information that is already presented in different Deliverables. These are mainly (D1.2 SoftFIRE, February 2017) for the operation and monitoring of the SoftFIRE infrastructure and (D1.1 SoftFIRE, February 2017) for the Infrastructure and its architecture.

## 3.1 Monitoring Functions

SoftFIRE offers the ability to get monitoring data about the physical machines composing the infrastructure providing the services to the whole project.

This service is called "*Infrastructure Monitoring*".

Most requested infrastructure metrics are: CPU load, total and free memory, free swap memory, disk IO, disk read/write, incoming and outgoing traffics on interfaces.

The Zabbix Infrastructure Monitoring System (Figure 10) is available via a web portal accessible at the following address: https://zabbix.softfire.eu

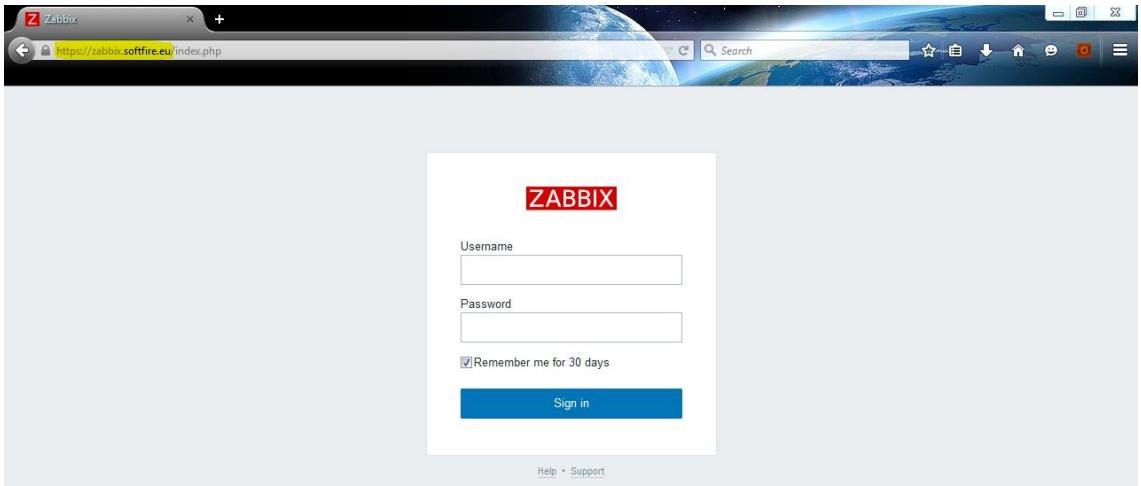


**Figure 10: Zabbix Access Page**

The Zabbix infrastructure for SoftFIRE has been built providing two types of UserGroups:

- *Providers*
- *Experimenters*

The users belonging to the Providers group have admin privileges and are grouped in order to customize the monitoring (templates, metrics) according to their Domain responsibilities.

The users belonging to the Experimenters group have fewer privileges that permit to have a restricted access to the functions provided by the tool, in particular they have access to collecting, triggering and events data.

Each experimenter selected in the different Open Calls will be provided the proper credentials to access the SoftFIRE infrastructure monitoring portal.

### 3.1.1. Dashboard

The "core" page of the Zabbix portal is the Dashboard (Figure 11) where each user can easily find a summary of the information related to the status of the SoftFIRE infrastructure, and can be reached selecting *Monitoring →Dashboard*

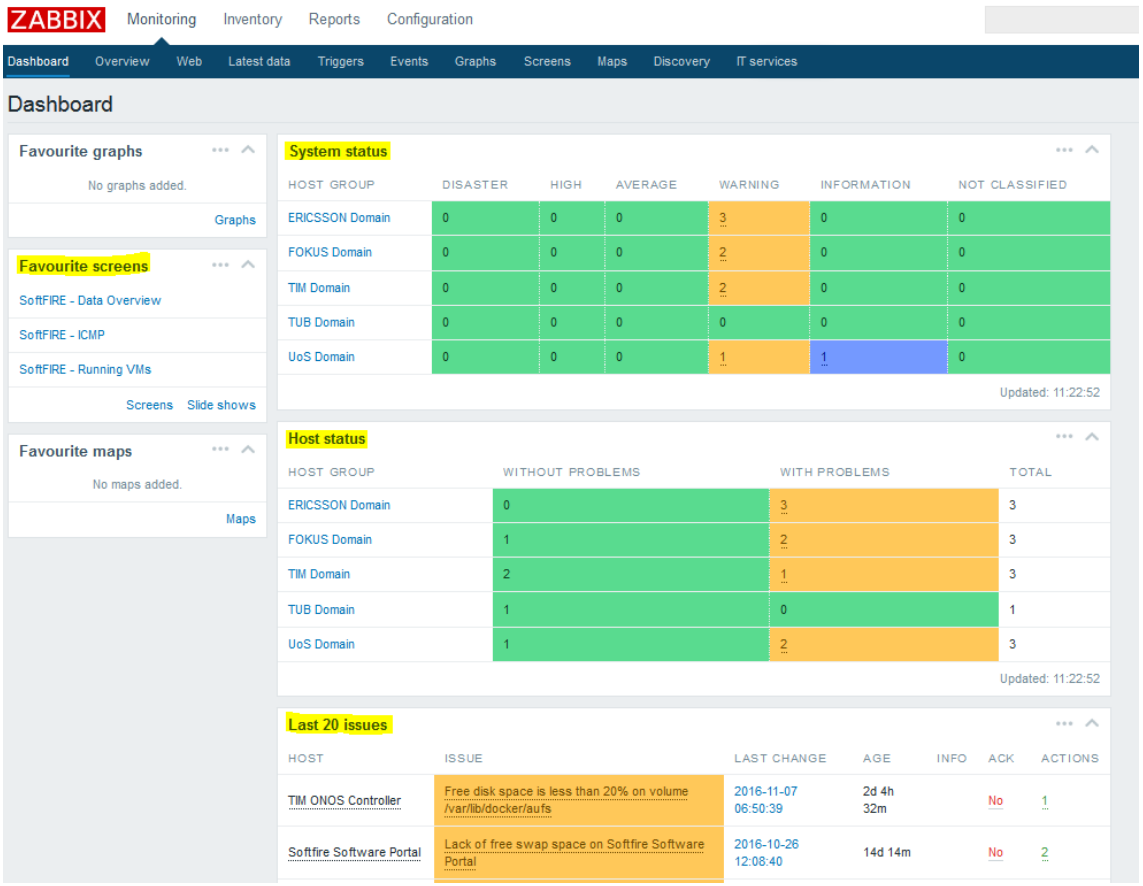


**Figure 11: Zabbix Dashboard**

As shown in the above Figure 11, the physical machines (hosts) used as infrastructure for the SoftFIRE project have been grouped for each testbed in so called "Domains", in particular:

- UoS Domain
- TIM Domain
- Ericsson Domain
- TUB Domain
- FOKUS Domain

The Dashboard contains some sections reporting summary information.

The "***System Status***" section shows for each testbed the number of physical machines with some active issues divided by severity.

Clicking over one of the numbers of this section a pop-up window is opened showing the list of last issues with a description related to that severity (Figure 12).

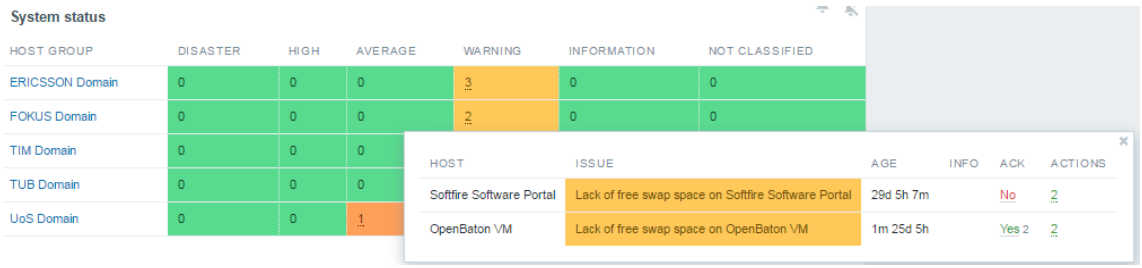


**Figure 12: Issue pop-up window**

The "***Host Status***" section shows for each testbed the number of physical machines (with/without problems and total).

Clicking over one of the numbers (for "with problems") of this section a pop-up window is opened showing a detailed list of the hosts affected by some issues and relative severity (Figure 13).

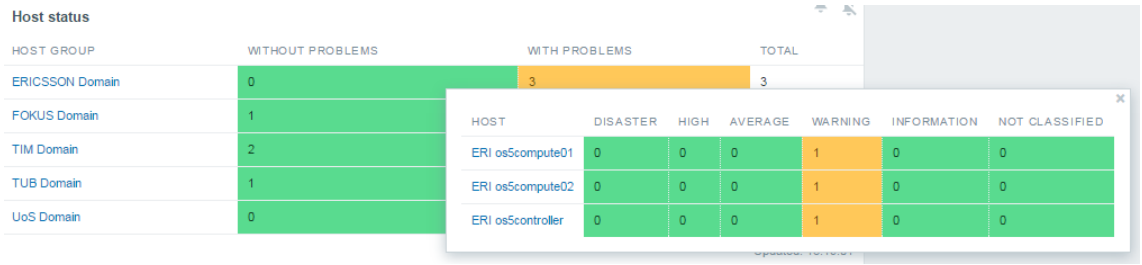


**Figure 13: Host pop-up window**

The "***Last 20 issues***" section shows the list of the last 20 "active" triggers detected on the whole SoftFIRE infrastructure.

Clicking over the description of an issue in this section a pop-up window is opened showing a list of the last triggers with their status related to the same event (Figure 14). SoftFIRE has used the triggering mechanisms as defined by Zabbix and has customized them to its needs. In particular, SoftFIRE has defined the severity level for events according to the process that has been put in place for monitoring the platform and the lifecycle of the experiments.


**Figure 14: Last issues pop-up window**

### 3.1.2. Collected Data

Another important section of the Zabbix portal is the *Monitoring → Latest Data* page where it is possible to view latest values gathered by items as well as to access various graphs for the items (Figure 15).

When you open this page for the first time, nothing is displayed.

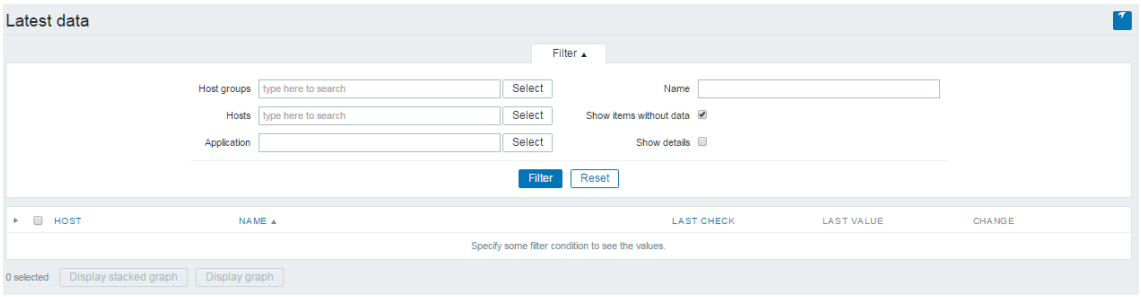


**Figure 15: Latest Data initial page**

To access data, you need to make selections in the filter such as *host groups, host, application or item name*. In such a case, some pop-up windows will be opened to enable the selections.

Below an example, selecting the Host groups, a list of the "Domains" in SoftFIRE is shown (Figure 16), giving also the possibility to perform a multi-selection:

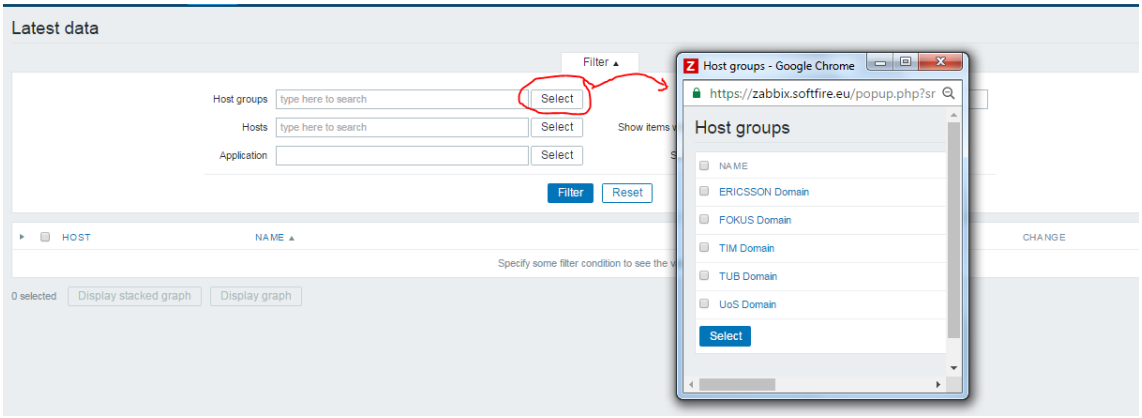


**Figure 16: Data Filtering**

Pressing the "*Filter*" button the relative data are shown in the bottom part of the window.

Moreover, *Show details* allows to extend displayable information on the items (Figure 17). Such details as refresh interval, history and trends settings, item type and item errors (fine/unsupported) are displayed.

You can expand all data (related to host or application), thus revealing all items by clicking on arrow in the header row.

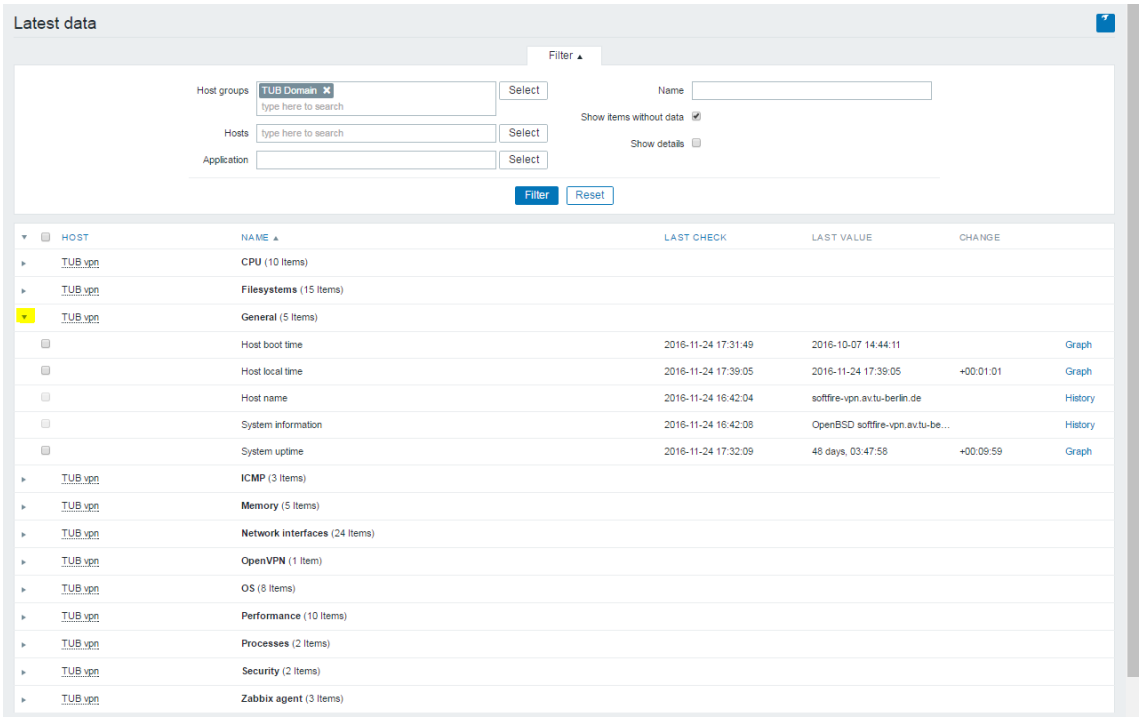


**Figure 17: Collected Data (example)**

The last column in the latest value list offers:

- a History link (for all textual items) - leading to listings (Values/500 latest values) displaying the history of previous item values.
- a Graph link (for all numeric items) - leading to a simple graph (Figure 18). However, once the graph is displayed, a dropdown on the upper right offers a possibility to switch to Values/500 latest values as well.

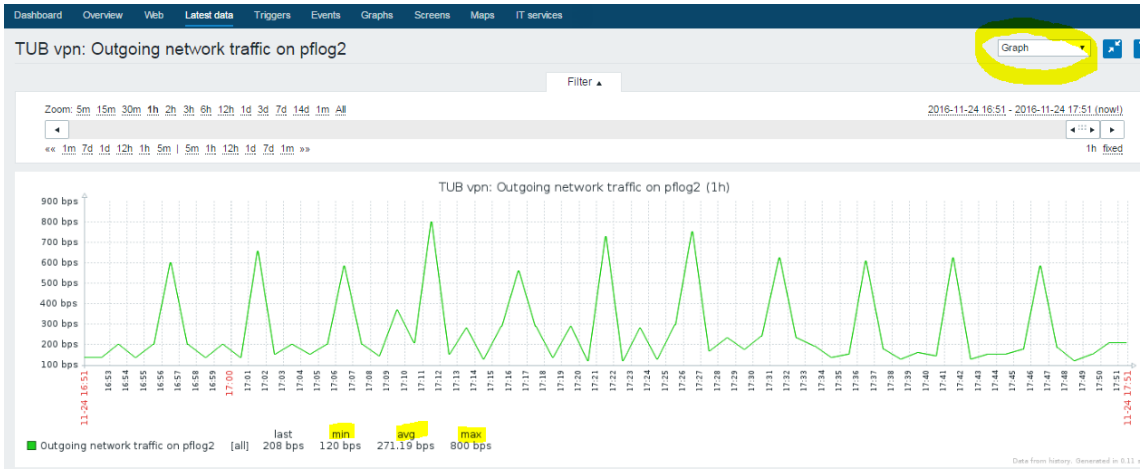


**Figure 18: Item Data (Graph)**

In the graph window (Figure 19) it is possible to have also data related to "**min**", "**max**", and "**avg**".

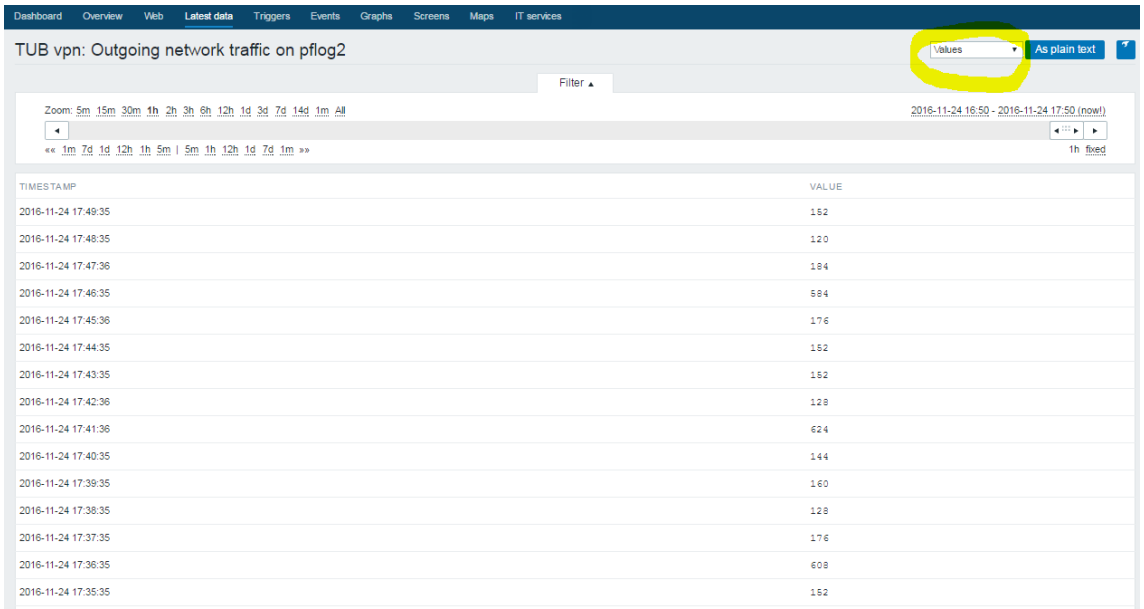


**Figure 19: Item Data (Values)**

For further details, the experimenter can refer directly to the official Zabbix documentation at http://www.zabbix.com/documentation.php.

### 3.1.3. Infrastructure Metrics

The main metrics collected by the "Infrastructure Monitoring" system are those ones typically associated to physical machine with Linux Operating System.

Any host monitored in the SoftFIRE has been associated to the default "*Template OS Linux*" provided by the Zabbix tool.

The metrics (*items*) that are measured and therefore will also be supported in the SoftFIRE infrastructure monitoring are the following ones grouped for "*Application Type*" (Table 5).

**Table 5: Metric items**

| **Application Type** | **Number of Items** |
|---|---|
| CPU | 13 |
| Memory | 5 |
| Operating System | 8 |
| Processes | 2 |
| ICMP | 3 |
| FileSystem | 1 for each File System discovered |
| Network | 1 for each network interface discovered (incoming and outgoing traffic) |

Moreover, some ***customized metrics*** have been developed, related to some specific type of machines working within the SoftFIRE project and for retrieve data concerning the Virtual Machines deployed in the OpenStack platform.

**OpenStack metrics:** these metrics (Figure 20) can be retrieved selecting the hosts acting as OpenStack controller (one in each testbed/Domain), or simply writing "**OS_Controller**" in the Application's filter field:

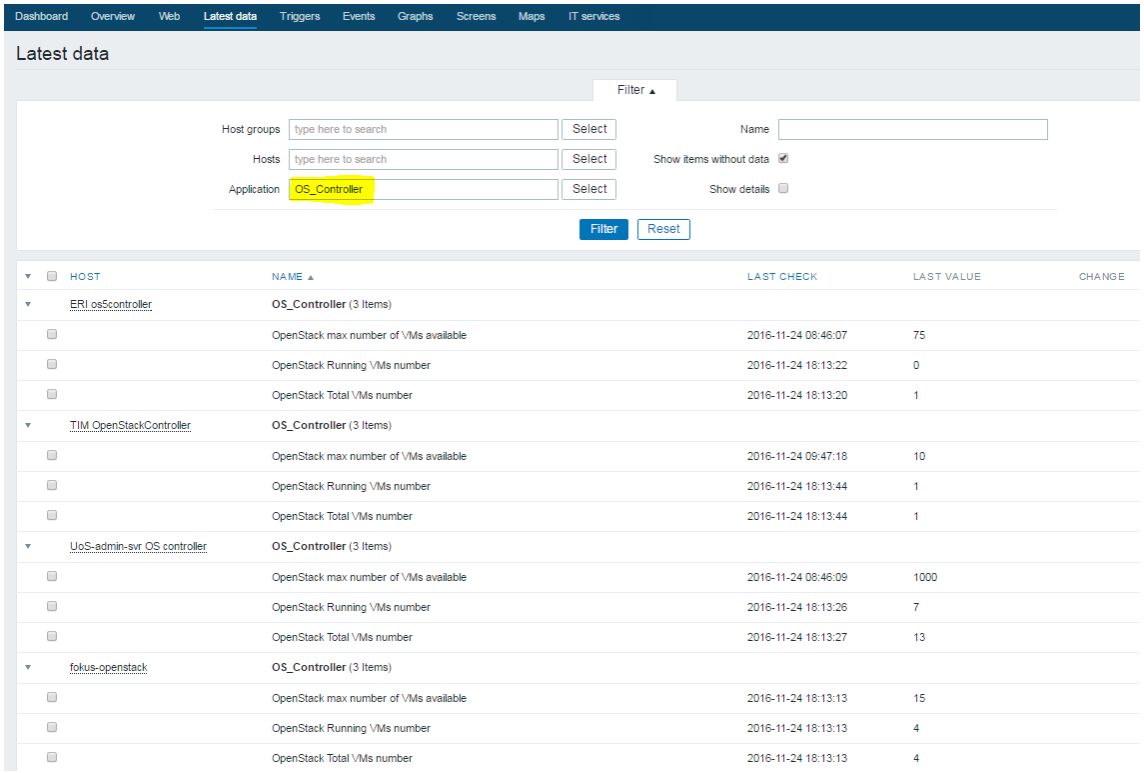


**Figure 20: OpenStack metrics**

These metrics represent:

- The maximum number of Virtual Machines available in a testbed
- The number of Virtual Machines instantiated on a testbed
- The number of Virtual Machines instantiated on a testbed and in *Running* state

**OpenBaton metrics:** these metrics (Figure 21) can be retrieved selecting the hosts acting as OpenBaton controller (in FOKUS Domain), or simply writing "**cloud**" in the Application's filter field:

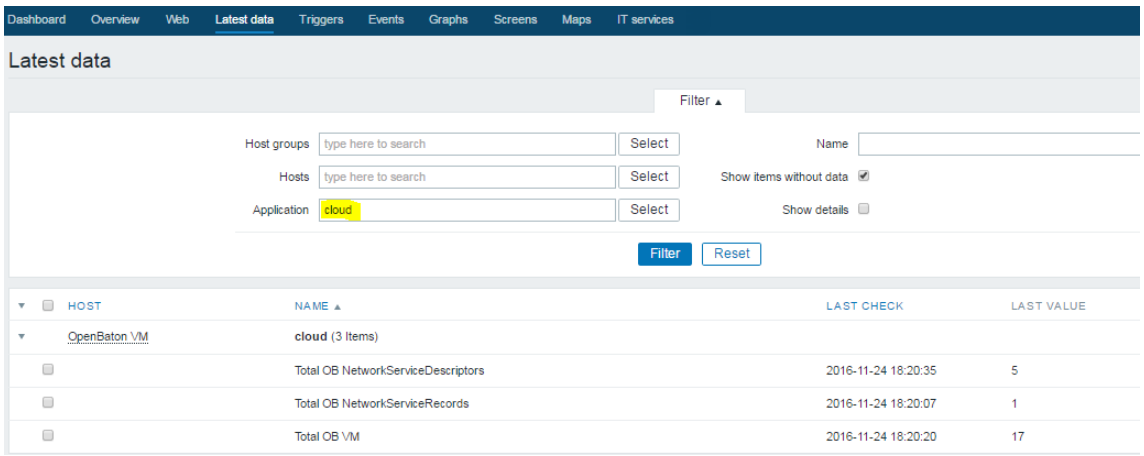


**Figure 21: OpenBaton metrics**

These metrics represent:

- The number of Virtual Machines running on Open Baton orchestrator
- The number of Network Service Descriptors on Open Baton
- The number of Network Service Records on Open Baton

**TUB OpenVPN:** this metric (Figure 22) can be retrieved selecting the hosts acting as OpenVPN controller (in TUB Domain), or simply writing "**OpenVPN**" in the Application's filter field:

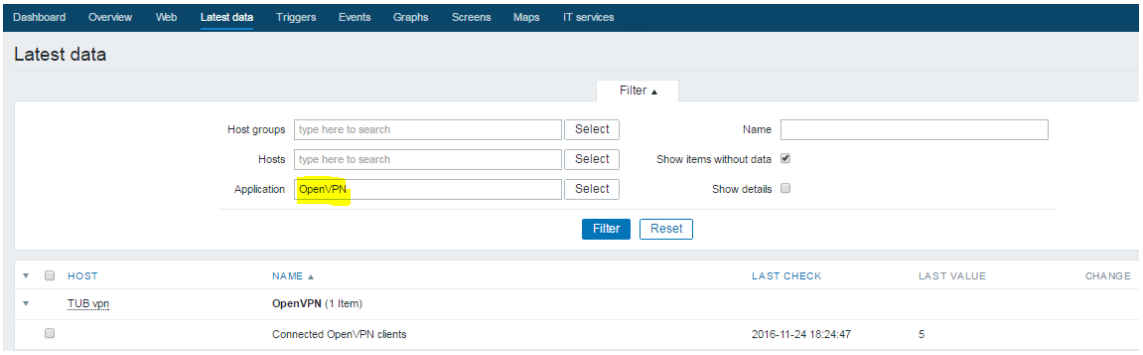


**Figure 22: OpenVPN metric**

This metric represents:

- The number of clients connected to OpenVPN

### 3.1.4. Triggers

The Monitoring → Triggers section displays the status of triggers (Figure 23).

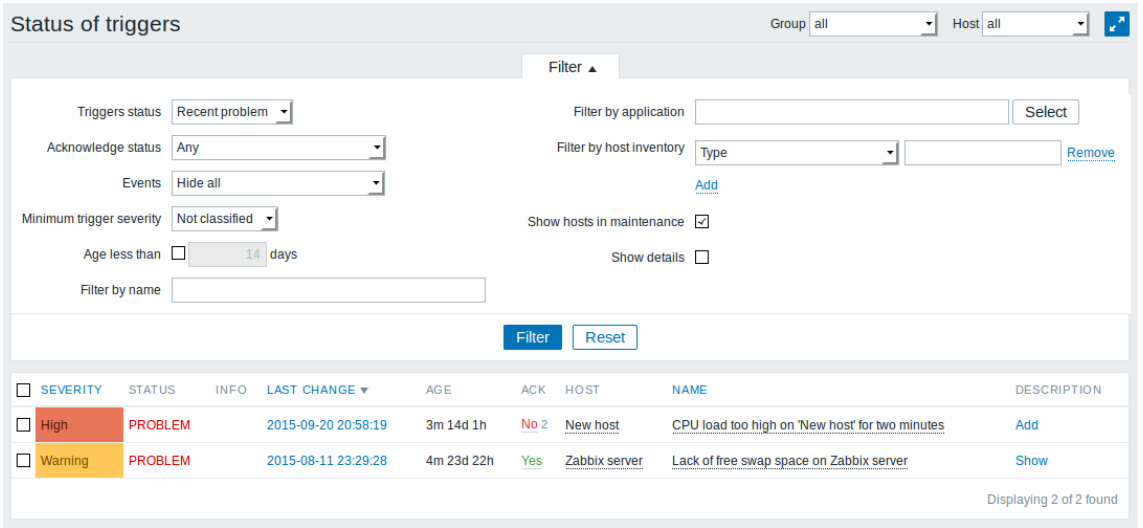


**Figure 23: Status of triggers**

Table 6 describes the triggers parameters as presented in Figure 23.

**Table 6: Trigger paramenter description**

| Column | Description |
|---|---|
| *Severity* | The trigger severity is displayed. The color of the severity is used as cell background for problem triggers. For OK triggers, green background is used. |
| *Status* | The trigger status is displayed - OK or PROBLEM. By default, it will be blinking for 30 minutes for triggers that have recently changed their state. Additionally, triggers that have recently changed their state to OK, will be displayed for 30 minutes even if the filter is set to show only problems. |
| *Info* | A grey icon with a question mark indicates that there is some relevant information to be displayed. If you move your mouse pointer over it, the message will be displayed. |
| *Last change* | The date and time of last trigger status change is displayed. |
| *Age* | The age of last trigger status change is displayed. |
| *Acknowledged* | The acknowledgement status of the trigger is displayed: **Yes** - green text indicating that the trigger is acknowledged. A trigger is considered to be acknowledged if all events for it are acknowledged. **No** - a red link indicating unacknowledged events. If you click on the link you will be taken to a bulk acknowledgement screen where all events for this trigger can be acknowledged at once. *Note*: If you wish to acknowledge a single event only, go to *Monitoring→Events*. **No events** - if there are no events for the trigger. |
| *Host* | The host of the trigger is displayed. |
| *Name* | The name of the trigger is displayed. It is also a link to the trigger event list and the trigger configuration page, as well as to a simple graph of item data. |
| *Description* | *A link to trigger description* |

In case a trigger with "PROBLEM" severity is detected in a host it has been implemented an automatic alert messaging service to inform the testbed's responsible (provider) that something is not properly working on that host due to a reason as reported in the Monitoring tool.

### 3.1.5. Configuration

All the users with "Provider" profiles have access to another menu (not visible for experimenter profile).

The "***Configuration***" menu contains sections for setting up major Zabbix functions, such as hosts and host groups, data gathering, data thresholds, sending problem notifications, creating data visualisation and others.

For further details, on how to configure the different functions you can refer directly to the official Zabbix documentation at http://www.zabbix.com/documentation.php.

## 3.2 IMS Functions

The federated testbed provides an IMS image called 'OpenIMSCore_preConf_ems-16' as detailed earlier in this document. This image contains all of the component IMS functions, required to deploy a test IMS network and can be used to provides IMS call capabilities to either a wired or Wi-Fi based client device.

The IMS system provides Text, Audio and Video media service capabilities by instantiating an IMS Monster Client on the device.

The SoftFIRE federated testbed provides the standard IMS functional elements illustrated in Figure 24 below, from the standardised 3GPP 23.002 architecture document.

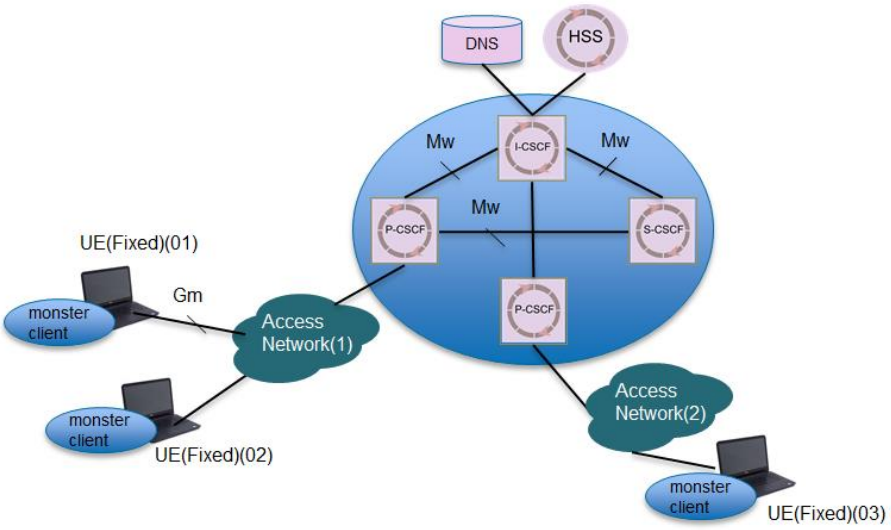


**Figure 24: The 3GPP IMS Architecture**

The SoftFIRE project has made available all of the IMS functions required to be able to setup a VNF instantiated IMS core as ETSI MANO compliant packages. The IMS packages are available at the OpenBaton orchestrator catalogue at TUB. The IMS component package functionalities are provided as per the VNFDs listed in Table 7 below:

**Table 7: SoftFIRE IMS VNFD's**

| SoftFIRE VNFD | IMS Function |
|---|---|
| icscf | I-CSCF |
| pcscf | P-CSCF |
| scscf | S-CSCF |
| fhoss | HSS |
| bind9 | DNS |

The 5x VNFDs that are listed in Table 7 are typically deployed as one Network Service Description Package for an IMS core deployment. The core image is available via the SoftFIRE OpenVPN/ jFed interface and called NSD (OpenIMSCore).

For more information on the SoftFIRE IMS core description, please see (D1.1 SoftFIRE, February 2017), section 4.1.

## 3.3 SDN Functions

This section describes the SDN functions that can be used by SoftFIRE users during experiments. For the first Open Call SoftFIRE provides physical SDN capabilities only in some isolated islands located in two separated testbeds: 5G Core Network and JoLNet.

The NFV based 5G Network at 5GIC/UoS testbed exploits 1Gbps switches to connect Virtual Machines instantiated on top of compute nodes to the Radio Access Network (RAN) and to the Internet. Traffic steering on these switches is managed by an OpenDaylight controller, which performs also software NAT on traffic coming from the VMs. The UoS testbed will be shortly upgraded to operate using SDN Layer 3 flow based switches. This is expected to be done by the second SoftFIRE Open Call, for first quarter 2017.

TIM's JoLNet exploits OpenFlow capable switches to provide network flexibility and programmability. Thanks to the open source virtualization tool FlowVisor, it allows experimenters to operate simultaneously and seamlessly on the same infrastructure avoiding conflicts and collisions among them. In fact, it exports different network slices, each of them ruled by a different OpenFlow controller. In JoLNet each network slice contains all physical switches and is identified by one or more VLAN IDs, but the controller does not need to know the VLAN tags associated to its slice, since FlowVisor shows a flat network. However, network flows must not cross slices, so if a controller tries to force a VLAN tag that is not in its flowspace it will receive an error message. JoLNet provides slices with already deployed SDN controllers from open source projects, such as ONOS and OpenDaylight that experimenters can access through REST API, graphic interface and command line interface. Every application already included in the original version of the controller could be activated and used by experimenters. In addition, it is possible to install new applications on them exploiting the Karaf-OSGi framework. If experimenters have their own OpenFlow controller, they can choose

to deploy it and to connect to a slice. However, FlowVisor is only able to operate with OpenFlow v1.0, restricting the flexibility that can be provided.

The target of the SoftFIRE consortium for next Open Calls is to upgrade the federation in order to provide SDN paths, which spread over different testbeds and cover most of the entire infrastructure.

## 3.4 5G functions

### 3.4.1. 5G Core Packages Available to Experimenters

The federated SoftFIRE testbed provides 5G Core Network, Virtual Network Functions (VNF) and Network Services (NS) for deployment on the UoS LTE-A RAN testbed at the University of Surrey, in the UK. These packages are listed in Table 8following, with their respective required resource allocations.

**Table 8: 5G Packet Core Package Resource Requirements**

| Package | #Cores | RAM (GBytes) | Image Storage (Gbytes) | HDD (Gbytes) |
|---|---|---|---|---|
| VNF(EPC_CPN) | 8 | 16 | 3 | 20 |
| NS(EPC_UPN_CC) | 4 | 8 | 3 | 20 |
| NS(EPC_UPN_CC) | 4 | 8 | 3 | 20 |

These VNF/NS components are made available as VNFD and NSD, ETSI MANO compliant packages for orchestration from the SoftFIRE, OpenBaton Orchestrator catalogue and are exposed to Experimenters via the OpenVPN/jFed interface.

The following explanations illustrate which 3GPP standardized entities have been packaged into which packages during the VNF and NS virtualization packaging process:

VNF(EPC_CPN) includes the following software component functionality: HSS, MME, and integrated (SGWc + PGWc).

VNF(PPE) is a Packet Processing Entity for the LTE packet core User Plane (UP) that comprises integrated (SGWu + PGWu) integrated into a Media Gateway (MGW) software component.

VNF(CC) Is a 5G evolved Cluster Controller which provides intelligent context handling to the packet core network at the Cluster-wide level which maps to Macro-Cell coverage in order to control and optimize cluster-wide user behaviour for best served QoE. E.g. the best place for a UE to camp on the cellular RAN network within RF strength and quality performance bounds according to mobility Entropy.

VNF(CM) Is a 5G evolved Cluster Member entity which provides intelligent context handling to the packet core network at the small cell, within a cluster level

NS(EPC_UPN_CC) includes the following functionality: PPE and CC.

NS(EPC_UPN_CM) includes the following functionality: PPE and CM

The 5GIC, UoS testbed component has a pre-prepared and installed VNFD(EPC_CPN) on the compute server at the 5GIC building. This provides CP packet core service to up to 4 UP slices on the testbed.

The 5GIC, UoS testbed component has a pre-prepared and installed NSD(EPC_UPN_CC) and NSD(EPC_UPN_CM) instances on its compute server, this provides a UP packet core service slice for the UoS staff to use to monitor progress on the testbed 5G network and benchmark experimenters metrics.

The 5GIC, UoS testbed component is further setup so that it can provide up to three experimenters with further NSD(EPC_UPN_CC) and NSD(EPC_UPN_CM) instances to enable instantiation of an experimenter UP packet core service slice on the testbed 5G network.

### 3.4.2. User Equipment Support

In order for Experimenters to test on the UoS 5G packet core network at the UoS Testbed component they will need to purchase a mobile that is operable on the RAN with the right capabilities and install a copy of the 5gD (5 Device application) onto the mobile. The 5gD software can be obtained from the SoftFIRE Software Portal library and the recommended LTE-A device is a Nexus 6P mobile, which supports the bands operated at UoS, which are namely LTE B38, B41 and Wi-Fi Eu bands.

SIMs for test LTE-A devices may be obtained from the UoS mentor (g.foster@surrey.ac.uk)

### 3.4.3. MEC Support

SoftFIRE provides a capability to place Mobile Edge Computing (MEC) applications within the intranet of the 5G/4G+ test access network at UoS, in order for experimenters to try out edge applications.

The experimenter needs to virtualize their application and then mount inside the intranet of the cellular test network as a MEC application on the UoS compute server provided, via the nomal OpenBaton packaging process as ETSI MANO compliant VNFD and/or NSD.

For more information on the SoftFIRE IMS core description, please see (D1.1 SoftFIRE, February 2017), section 4.3.

# 4 Examples and tutorial

In this section, we will describe some examples that were executed using the SoftFIRE platform during the SoftFIRE first Plug Test (SoftFIRE). The result of this example can be found in our VNF Package repository[9].

## 4.1 Example of MongoDB as simple Network Service

### 4.1.1. Introduction

(MongoDB, 2016) is an open-source non-relational database providing a simple server solution or a "sharded" solution suitable for NFV environments in which some VNFs may benefit from external databases.

MongoDB stores most of the records as structured data objects, under the JavaScript Object Notation (ECMA JSON). The binary documents are stored as BSON, a binary representation of JSON format. If the binary document is bigger than 16 Megabytes, then it is automatically stored in the MongoDB filesystem called GridFS.

In a simple scenario, an installation all-in-one could be enough, but when moving the solution to the NFV infrastructure, new requirements are involved and the MongoDB architecture can be spread in different servers with different roles (see Figure 25), in order to satisfy these new requirements:

- **Shard**: Each shard contains a subset of the data. Each shard can be also deployed as a replica set. A replica is a server that is not active in the running system but is a copy a running instance and can substitute the active server in case of failure, providing in this way high availability
- **Mongos (Router)**: The mongos router acts as a query *router*, providing an interface between client applications and the "sharded" cluster. The router is the interface to which the client can query for records. It is its task to route the request to the correct entity
- **Config Server(s)**: Config servers store metadata and configuration settings for the cluster. *Config servers* can be deployed as a replica set

[9] http://172.20.30.52/vnfpackages/

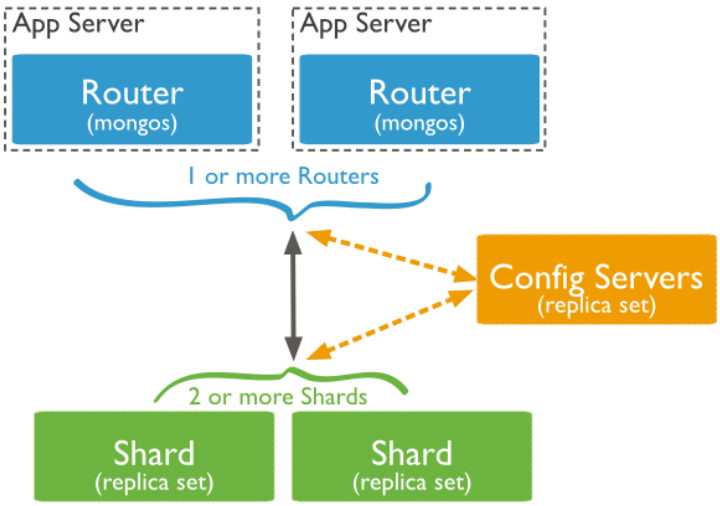


**Figure 25: MongoDB Sharded Architecture**

### 4.1.2. Network Service design

After having understood the architecture of VNF, it is clear that there will be needed at least three different VNF Components. This means that we need to create three VNF Packages, as described in section 2.2.1.1.1.

As first step, we need to create the installation, configuration and start scripts. The scripts of this example are available in the public repository: https://github.com/SoftFIRE-TUB/public-scripts.

Since the Router VNF needs to be configured with the IPs of the Shard and Config we will have a logical architecture of the NSD as follows in Figure 26.

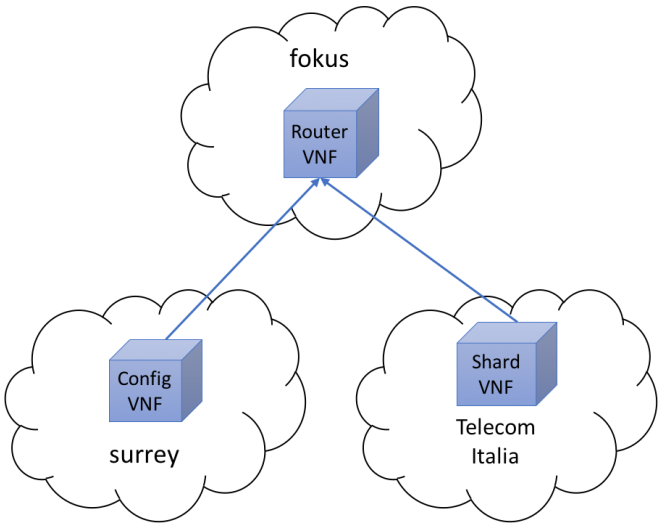


**Figure 26: MongoDB NSD Logical View**

Figure 26 shows some details on how the Network Service is described in particular regarding *where* the VNF will be placed as well as regarding the *dependencies* between each component.

***Config*** and ***Shard*** are not "target" of any relation, indeed only ***Router*** "*requires*" IPs from Shard and Config.

Thus, the ***Config*** and ***Shard*** only requires one (or more) installation script while the ***Router*** also requires two configuration scripts.

The installation script is the same for all of the packages, but needs anyway to be replicated inside all the tar files.

```
#!/bin/bash

if [ ! -z "$http_proxy" ]; then
  sudo apt-key adv --keyserver-options http-proxy=$http_proxy --keyserver hkp://keyserver.ubuntu.com:80 --recv 7F0CEB10
else
  sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv 7F0CEB10
fi

echo "deb http://repo.mongodb.org/apt/ubuntu trusty/mongodb-org/3.0 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-3.0.list

sudo apt-get update
sudo apt-get install -y --force-yes mongodb-org screen

sudo service mongod stop
```

**Code Snippet 1. install-mongo.sh**

The variable $**http_proxy** is injected by the MANO layer of the SoftFIRE platform and the value is retrieved from the ConfigurationParameter (see Table 1).

The same procedure applies to the configuration scripts, not only for "local" variables, but also for the dependency variables. Hence, we can define the configuration scripts for the ***Router*** as follow:

```
#!/bin/bash

export LC_ALL=C
shard_uri="$shard_softfire_internal_floatingIp:$shard_port"
echo $shard_uri
echo "sh.addShard('$shard_uri')" > addShard.js
if [ -z $database ]; then
        database=softfire
fi

echo "sh.enableSharding('$database')" >> addShard.js

mongo --port $port < addShard.js
```

**Code Snippet 2. shard_configure.sh**

The variable called $**shard_uri** is composed by two other environment variables that are automatically injected by the SoftFIRE MANO layer but differently from the previous ones, these are retrieved from a dependency, in particular from the dependency with the ***Shard*** VNF. This feature is already specified in section 2.2.1.1.1, the script must start with the foreign type followed by underscore and the name of the script, here **shard_configure.sh**, and in the code the experimenter can use the parameter specified in the requires field (see Table 1) just by specifying the variable name as $<foreign_type>_<parameter_key>, in this case $**shard_softfire_internal_floatingIp,** where the foreign type is *shard* and the parameter is *softfire_internal_floatingIp* pointing to the IP of the *softfire-internal* network. Note that the

character "-" is translated in "_". The same approach is followed for the configuration of the **Config** as shown in the following script:

As the previous Code Snippet 2, we use the foreign parameters with the same syntax.

```
#!/bin/bash

export LC_ALL=C

screen -d -m -S router mongos --port $port --configdb $config_softfire_internal_floatingIp:$config_port

nproc=`sudo netstat -tulpn | grep $port | wc -l`

COUNTER=1
while [ 1 -eq "$nproc" ]; do
 COUNTER=$(($COUNTER + 1))

 sleep 5 # wait for 1/10 of the second before check again
 if [ 100 -eq $COUNTER ]; then
  echo "mongodb not started"
  exit 99
 fi
done
echo "mongodb router is running"
```

**Code Snippet 3. config_configure.sh**

It is possible now to define the VNFD, the full example is shown in Annex A.

Important to mention are the "requires" field and the vimInstanceNames:

```
"requires":{
  "config":{
    "parameters":[
     "port",
     "softfire-internal",
     "softfire-internal_floatingIp"
    ]
   },
   "shard":{
    "parameters":[
     "port",
     "softfire-internal",
     "softfire-internal_floatingIp"
    ]
   }
```

This field contains all the needed parameters, in this case the *port* and the private and public IPs, identified by the network name.

The vimInstanceNames is a list of testbed where to deploy. The possible values are already defined in Table 2.

Then, before creating the tar file, the Metadata.yaml file must be created.

```
name: router
description: "The router module"
provider: TUB
shared: true
image:
   upload: false
   names:
     - ubuntu-14.04-cloud-amd64
```

In this case, we specify that the image to be used is one of the available ones in particular the Ubuntu Cloud image 14.04.

We are now able to create a tar file containing:

- Metadata.yaml
- Vnfd.json
- The scripts folder that contains the scripts defined before.

Finally, we can upload the VNF Package to the SSP and after that, it will be available in the jFed tool.

# 5 Conclusions

This document provides a usage manual of the SoftFIRE infrastructure. The complexity of the platform could lead to difficulties in understanding but also demonstrates the challenges that we are facing and overcoming.

The steps defined in this document are guidelines for all experimenters, thus the importance of this deliverable.

This document is published after one year of the project duration; it has evolved during this time not only because of the improvement applied to the infrastructure, but also thanks to a continuous communication between project partners and experimenters. The direct communication with each of the experimenters showed increased progresses with the work they were conducting. For this reason, in addition to the documentation provided as guidelines, the consortium decided to setup dedicated periodic calls with each experimenter in order to support them in understanding the complexity of the SoftFIRE infrastructure.

In fact, after such discussions and extensive internal testing (mainly related with security issues) we have also decided to provide direct access to the MANO layer, as in many of the experiments the capabilities offered via the SFA APIs were not satisfactory enough.

After these measures that have been applied, almost all experiments became successful and contributed with value to the innovation of the involved technologies. One of the more interesting experiment that can be cited is “NFV@EDGE”, that was able already at this first stage of the platform to extend it, increasing the number of the supported VIM, providing an extension able to transform the Universal Node into a NFV ready VIM and to integrate it in the SoftFIRE platform.

All the experimenter proposals have been analysed in order to understand the requirements and identify possible trends of evolution. The platform now has reached the first stage of consolidation and it is ready to be further extended with new features. Considering the trends represented by previous proposals is a means for identifying possible needs of other experiments.

Moreover, in the next phase we are planning to extend functionalities at almost every level of the platform, giving the experimenters more freedom to extend the platform, and more control on the experiments increasing the capacity currently provided by each individual testbed.

# 6 References

D1.1 SoftFIRE Consortium. (February 2017). *NFV/ SDN/MEC systems' integration with FIRE frameworks (FIRE + enhancements for NFV/SDN/MEC experimentation).*

D2.1 SoftFIRE Consortium. (July 2016). *Usage manual for NFV/ SDN/MEC and 5G experimenters V1.*

ECMA. (n.d.). *Standard ECMA-404, The JSON Data Interchange Format.* Retrieved from http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf

ETSI. (2014, 12 1). *ETSI GS NFV-MAN 001.* Retrieved from etsi.org: http://www.etsi.org/deliver/etsi_gs/NFV-MAN/001_099/001/01.01.01_60/gs_NFV-MAN001v010101p.pdf

GENI. (n.d.). *SFA.* Retrieved from http://groups.geni.net/geni/wiki/SliceFedArch

iMinds. (n.d.). Retrieved from http://jfed.iminds.be/

MongoDB. (n.d.). *open-source document database that provides high performance, high availability, and automatic scaling*. Retrieved from https://www.mongodb.com/

Open Baton. (2016). *Open Baton is an open source project providing a comprehensive implementation of the ETSI Management and Orchestration (MANO) specification.* Retrieved from OpenBaton: http://openbaton.github.io

Open Baton. (n.d.). *Open Baton dependency management between VNF.* Retrieved from http://openbaton.github.io/documentation/vnfm-generic/

Open Baton. (n.d.). *Open Baton Descriptor Model.* Retrieved from http://openbaton.github.io/documentation/ns-descriptor/

Open Baton. (n.d.). *Open Baton Virtual Network Function Descriptor.* Retrieved from http://openbaton.github.io/documentation/vnf-descriptor/

Open Baton. (n.d.). *Open Baton VNF Package how to.* Retrieved from http://openbaton.github.io/documentation/vnf-package/

Open Baton. (n.d.). *openbaton.* Retrieved from http://openbaton.github.io/

OpenStack. (n.d.). *Open source software for creating private and public clouds.* Retrieved from https://www.openstack.org/

SoftFIRE. (n.d.). *SoftFIRE InterOpTest.* Retrieved from https://www.softfire.eu/wp-content/uploads/SoftFIRE_InterOpTest_Agenda-20160930.pdf

Zabbix. (n.d.). *The Ultimate Enterprise-class Monitoring Platform.* Retrieved from http://www.zabbix.com/

# 7 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. |
| 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*<br>*(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 |

| | |
|---|---|
| VPN | Virtual Private Network |

# A. Annex: Router MongoDB VNFD

Here the full example as mentioned on page 43.


```
{
   "vendor":"TUB",
   "version":"0.1",
   "name":"router",
   "type":"router",
   "endpoint":"generic",
   "configurations":{
      "configurationParameters":[
         {
            "confKey":"port",
            "value":"5800"
         },
         {
            "confKey":"SECURITY",
            "value":"false"
         },
         {
            "confKey":"SMALLFILES",
            "value":"true"
         },
         {
            "confKey":"USERNAME_MD",
            "value":"mongo"
         },
         {
            "confKey":"PASSWORD",
            "value":"mongo"
         }
      ],
      "name":"mongo-configuration"
   },
   "vdu":[
      {
         "scale_in_out":1,
         "vm_image":[

         ],
         "vimInstanceName":[
            "vim-instance-fokus"
         ],
         "vnfc":[
            {
               "connection_point":[
                  {
                     "floatingIp":"random",
                     "virtual_link_reference":
                  "softfire-internal"
                  }
               ]
            }
         ]
      }
   ],
   "virtual_link":[
      {
         "name":"softfire-internal"
      }
   ],
   "lifecycle_event":[
      {
         "event":"INSTANTIATE",
         "lifecycle_events":[
            "install-mongo.sh"
         ]
      },
      {
         "event":"CONFIGURE",
         "lifecycle_events":[
            "config_configure.sh",
            "shard_configure.sh"
         ]
      }
   ],
   "deployment_flavour":[
      {
         "flavour_key":"m1.small"
      }
   ],
   "requires":{
      "config":{
         "parameters":[
            "port",
            "softfire-internal",
            "softfire-internal_floatingIp"
         ]
      },
      "shard":{
         "parameters":[
            "port",
            "softfire-internal",
            "softfire-internal_floatingIp"
         ]
      }
   }
}
```