Marketplace (MARK)

1.1. Scope of this document

This document is to describe the purpose of the demonstration and the architecture of the test infrastructure that was used by the NECOS consortium. This test infrastructure is not precluding that any other can be used to run de demonstration. The guide to install the software in the substrate resources is provided in the README file, in the same repository as the software.

1.2. Introduction

This demonstration requests multiple slice allocations using one marketplace hosted by UOM and the Resources Providers from the FED4FIRE as presented in **Figure 1**.

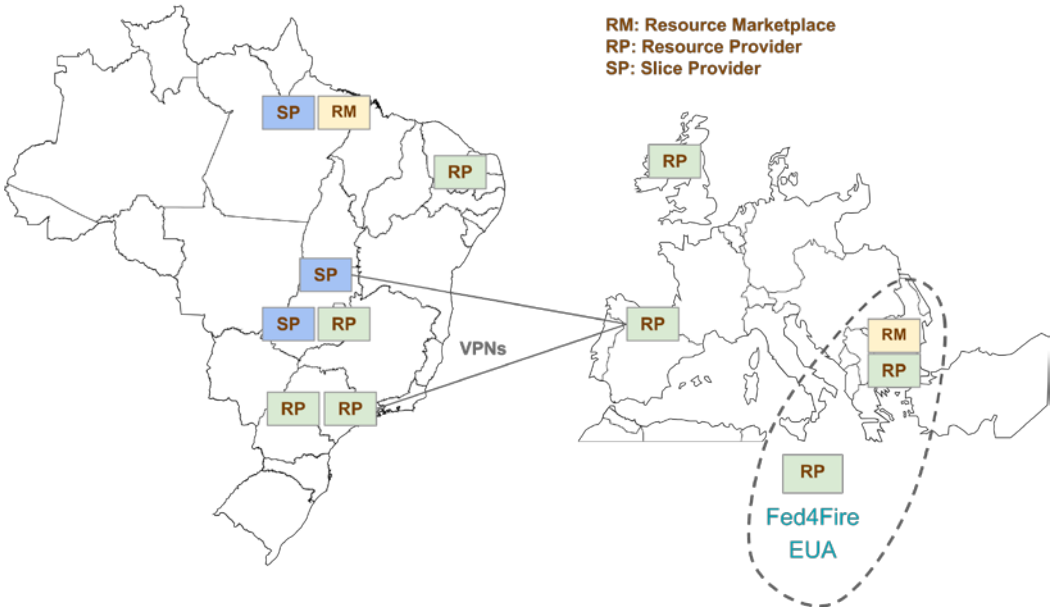

Figure 1. Instantiation of MARK on the experimental infrastructure.

1.3. License

All the source code developed within the NECOS project, and made available as OSS, is released under the Apache License – Version 2.0[1]

1.4. Objectives

The main objective of this demo is to demonstrate that the marketplace concept introduced in NECOS as a dynamic resource discovery mechanism can cope with slices of significant size and multiple geographically distributed resource providers. We assume

[1] http://www.apache.org/licenses/LICENSE-2.0

multiple slice requests and demonstrate the behavior of the marketplace components, the involved workflows and the relevant resource discovery performance.

In order to obtain real world data, namely the status of resources, we use 6 FED4FIRE testbeds (http://www.fed4fire.eu/testbeds). We developed a (Python) Translator component that is responsible to directly communicate with the corresponding test-bed control interface (e.g., jFed CLI), and to translate the response message into a uniform format. In practice, we maintain a local representation of the resources in JSON format from the following open-access test-beds: Virtual Wall 1, Virtual Wall 2, Cloudlab Utah, Grid5000, Cloudlab Wisconsin, and w-iLab2. This treatment of the resources' features is critical because FED4FIRE represents their resources through Resource Specifications (called RSPECs), but not in a uniform manner throughout the test-beds, e.g., the resources may have incomplete details or present different attributes.

FED4FIRE testbeds are organised in clusters of nodes, where nodes in a cluster are machines of identical configuration. Each node in a cluster is associated with a cost and in all experiments, we assume that one Virtual Deployment Unit (VDU) service is allocated to a single cluster, thus that cluster should “cover” the VDU Extended Platform Awareness (EPA) attributes.

We place the Slice Agents in Fed4Fire testbeds. To facilitate the experimentation process, we reserved three hosts in a subset of the aforementioned testbeds, and allocated our agents on these three hosts. The allocation is the following:

- Virtual Wall 1 (Europe): 4 DC Agents responsible for resource discovery of all the European Testbeds and one WAN Agent;
- Virtual Wall 2 (Europe): 6 DC Agents with “semi-artificial” resource data, and one WAN Agent;
- Cloudlab Utah (USA): 2 DC Agents responsible for resource discovery of all the USA Testbeds and one WAN Agent.

Each DC Agent is responsible for a testbed, communicating with the latter through the translator. Each Slice Agent has a different cost for the hosts it offers. Since FED4FIRE is an open platform, this cost was generated using random values. Since the time required by the Slice Agents to report back testbeds availability information is far greater than the actual time required to deploy them, we consider that the allocation of the Slice Agents in different testbeds would not significantly change our results. In order to further investigate further the proposed approach, we have generated “semi-artificial” data regarding resource providers, that are variations in terms of resource availability and host characteristics based on the real data obtained by the FED4FIRE testbeds. We also included 3 WAN-Provider Agents that offer connectivity between DC slice parts; in our tested we considered a fully connected graph between our slices, i.e. all WAN Agents are able to connect to all the testbeds albeit at a different cost.

The Slice Brokers (that use the RabbitMQ service) are hosted on a server on the UOM premises. We have generated multiple slice request cases, by varying the number of slice parts, services hosted in each slice part and their tenant requested geographic constraints. In all cases, we investigated the number of messages exchanged, the number of slice parts

alternatives and the total number of alternative slice instantiations and their total cost, from which we derived the slice instantiation of the minimum cost.

Finally, since the testbed deployment is time consuming and in order to further investigate the scalability of the proposed approach, we report also on single host experiments, that would allow us to conduct (more easily) an experimental evaluation. This single host experiments are reported in the corresponding section below. It should be stressed that the implementation tested in the Fed4Fire Demo and Single Host experiments is identical.

1.5. Workflow

The Slice Broker and Slice Agents prototypes implement fully the NECOS Marketplace functionality. In particular, the current implementation is based on the workflow depicted in **Figure 2** (Deliverable D5.1).

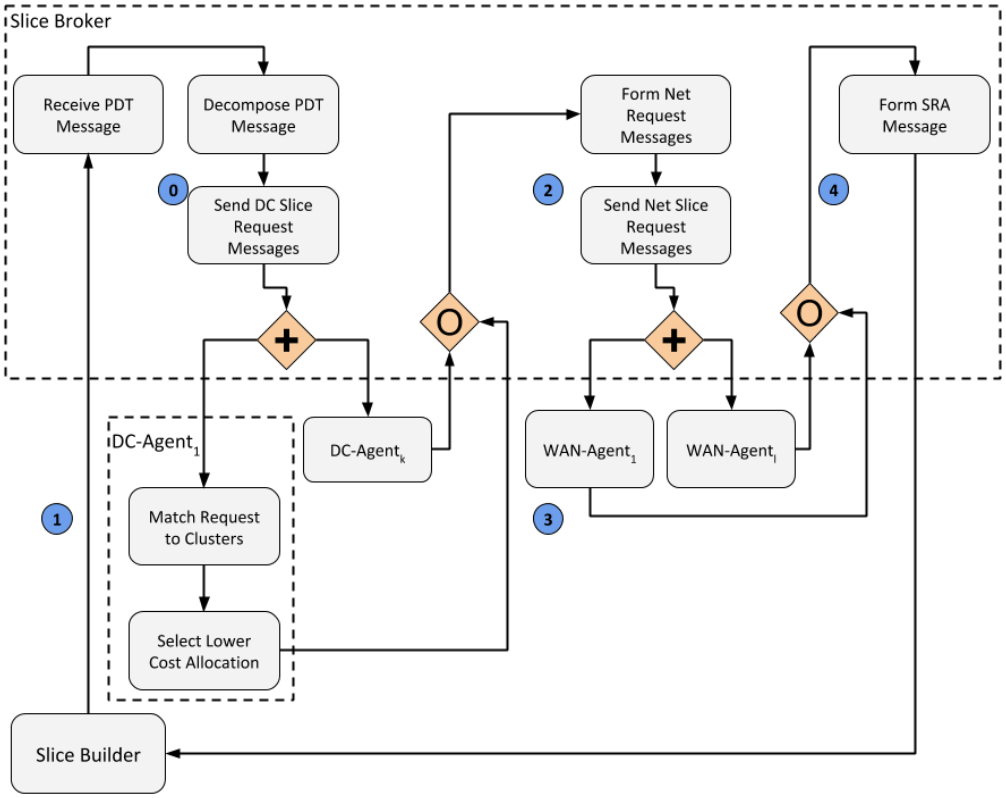

Figure 2. Marketplace Resource Discovery Workflow.

- **Step 1:** The Broker receives the PDT Message from the Slice Builder and extracts from it the different DC slice parts. Each of such DC slice parts is enriched with the necessary information regarding its host constraints, obtained by the corresponding service specification that is included in the PDT Message (please see deliverables D4.2 and D5.2) and is forwarded to the agents via the RabbitMQ messaging platform, annotated with the corresponding geographic constraints.
- **Step 2:** A Slice agent that receives a slice part request message, computes the clusters that can host DC part requested. Since multiple clusters can host a VDU in a slice part (please see D5.2) the agent computes its minimum cost answer,

based on any allocation constraints regarding host availability. Finally, this answer is communicated back to the Broker.

- **Step 3:** After having collected all available DC-Agent answers for all the DC slice parts, the Broker forms requests addressed to Net-Agents. In order to do so, the Broker extracts connectivity information from the slice graph described in the PDT message. Thus, for each net-(slice) part it forms a request annotated with the network details (IP) of the providers at the ends of each connection. Each such message is sent to all the WAN-Agents.
- **Step 4:** WAN-Agents report on the availability and the cost of providing the specific network connection.
- **Step 5:** After receiving all requests, the Slice Broker combines them into a single SRA message and sends it to the Builder.