FedUp! Cloud Federation as a Service

Paolo Bottoni, Emanuele Gabrielli, Gabriele Gualandi, Luigi Vincenzo Mancini, Franco Stolfi

FedUp! Cloud Federation as a Service

Paolo Bottoni, Emanuele Gabrielli, Gabriele Gualandi,
Luigi Vincenzo Mancini, and Franco Stolfi

Dipartimento di Informatica - Sapienza University of Rome (Italy)
(bottoni,gabrielli,gualandi,mancini,stolfi)@di.uniroma1.it

Abstract. Current solutions for establishing federations of clouds require applications to be installed on the individual members of the federation, which have to devote a certain amount of resources to services for federation managing. Moreover, additional interoperability requirements may need to be satisfied by individual clouds in order to join a federation. This situation may negatively affect the decision whether to join a federation. In this paper we propose an alternative approach by viewing creation and management of a cloud federation as cloud services themselves, thus allowing a drastic simplification in the federation set-up process and the decoupling of the federation management services from the technologies adopted by the individual clouds, minimising technological complexity and intrusiveness in the individual cloud infrastructures, while increasing the flexibility and scalability of resources. We also point out that existing technologies, in particular containers, microservices, configurators, clusters and orchestrators, can be the basis for implementing a platform for generation and management of federations of individual clouds, in a way which facilitates optimisation of workload and scaling of applications via resource aggregation, and makes deploying and joining federations fast, easy, and transparent.

1 Introduction

In the last few years, cloud-based solutions have become of interest for public and private organizations, due to the possibilities they offer for achieving: (1) greater cost reductions, by moving part of the budget from fixed to variable costs; and (2) greater resilience, and by increasing the flexibility and scalability of resources in response to changing business needs. Nevertheless, some elements –such as customers' perception that they are losing control over infrastructure resources, or the risk of vendor lock-in, stemming from a pervasive use of provider's services– are still an obstacle to the use of cloud solutions.

In order to mitigate these problems, many organizations are trying other cloud computing strategies including the creation of federations of individual clouds, from which they expect optimization of workload, increased availability of resources, probably even at more competitive costs and on an as-needed basis, and high levels of security and quality of service, probably better calibrated on the needs of an individual cloud joining a federation [3].

Current approaches to the construction of federations of clouds, see e.g. Fogbow[1], Zentera[2], Reservoir [7], handle federation services through some specific applications or frameworks. Hence, specific components must be installed on the infrastructure of the federation members, which are required to provide a certain amount of resources to run these components. In addition, when federating heterogeneous individual clouds, one has to consider the presence of: (1) distinct domains; (2) different security policies and service levels; and (3) different repositories of accounts. Other issues include the technological compatibility between the framework and the infrastructure of the individual member clouds, which may require adaptation to join a federation. All of this brings increased time and cost, which could make joining a federation inconvenient or impossible.

Faced to these problems, we argue that *construction* and *management* of federations of individual clouds should be seen as cloud services in turn, thus allowing a drastic simplification in the federation set-up process, the decoupling of the federation management services from the technologies adopted by the individual clouds, and minimal technological complexity and intrusiveness in the individual cloud infrastructures, while increasing the flexibility and scalability of resources. A further advantage is that joining a federation and contributing or obtaining resources to and from a federation could be achieved at a fine grain directly by end-users or with reduced intervention of cloud administrators.

In this paper we outline the basic requirements and the available technologies –in particular containers, configurators and orchestrators– which allow us to introduce the **FedUp!** approach to lean deployment and management of federations of heterogeneous clouds, by devising some simple mechanisms for setting up a federation and for allowing individual clouds to join an existing federation.

Paper organisation. After concluding the introduction with a brief overview of related work, Section 2 presents the requirements and features on which the approach is based. Section 3 discusses the organisation of the **FedUp!** platform as a collection of microservices, possibly allowing a same cloud to join different federations, or even to participate in one federation in multiple ways, with different levels of service, Section 4 provides a description of how the approach can be realised with current technologies. The interactions involved in the execution of the main services for creating and managing federations of clouds are described in Section 5. Finally, Section 6 provides conclusions and points to future work.

1.1 Related Work

A layered model was proposed for the management of applications in a federated cloud in [9], where communication among clouds in the federation occurs at the respective layers (SaaS, PaaS, and IaaS[3]). In their proposal, a service request to a cloud, for an application at the SaaS level, will traverse layers and contact

[1] http://www.fogbowcloud.org/
[2] http://zentera.net/
[3] Acronyms for [Software, Platform, and Infrastructure] as a Service.

brokers at the different layers as needed, but the model does not consider the offering of specific services at federation level.

Reference to the cloud layers is made in [2] to introduce a distinction between *horizontal* (same layer) and *vertical* (across layers) federations, focusing on horizontal ones, where two fundamental scenarios are considered, viz. *redundant deployment* of services and *service migration* from a cloud provider to another, the service becoming accessible through a different endpoint. They provide a reference architecture describing services to be offered by a federation, but do not discuss dynamic creation and join/leave on the individual cloud side

When considering federations of heterogeneous clouds with some common level of trust, the notion of community cloud arises, as proposed in [4] and elaborated on from the point of view of security in [6]. Here, each node in the participating clouds can play both the roles of producer and consumer, and a specific layer for coordination is proposed, including management of virtual machines, identity, networking, and transactions.

The kind of flexibility and the fine granularity that the **FedUp!** approach grants is also in the direction of Resources as a Service, as advocated at the cloud level in [1], where users could subscribe to the usage of resources maintained by a cloud for as much as needed, instead of declaring beforehand the amount of resources they expect to need. By making joining and leaving federations easy, shorter turnaround times can be achieved, thus enabling greater responsiveness by federations to the requests coming from individual clouds.

2 FedUp! Overview and Requirements

We are developing the **FedUp!** approach (and the homonymous platform) to setting up and managing federations of resources from individual clouds in a simple and non-intrusive way, based on cloud services, i.e. services (e.g. Paas, SaaS, and IaaS) made available to users on demand via the Internet and fully managed by a provider. In particular, **FedUp!** has been designed as a platform by which to create federations in a PaaS style, based on the following activities.

1. Manage the entire life cycle for a federation (creation, management of individual cloud membership, management of feature descriptions and of feature-based bidding and awarding of membership, dismissal). This will in turn exploit mechanisms based on cloud services, setting the basis for the new notion of *Federation as a Service*, thus enabling fastest federation start-up time, decoupling of services from technologies for service distribution, smaller intrusion in the technological choices for member clouds (though not completely technology-agnostic, the approach can be targeted to various implementations), greater flexibility in management of single federations.
2. Allow a cloud to: (1) become a member of several federations at a time (through multi-tenancy); (2) apply for becoming a member of any federation based on a description of the resources it can contribute and/or of the resources it needs to acquire; (3) easily migrate from one federation to

another. This will grant ample possibilities to individual clouds for choosing the federation to join, greater flexibility in managing the membership agreement, and a better appreciation of its resources.
3. Maintain a central repository storing the records concerning the generated federations. Each federation is described by a set of features (publicly represented in the form of *tags*). This will allow the generation of theme-based federations offering a set of specific services to the member clouds.

We state a list of requirements for lean mechanisms for establishing, managing and granting access to federated clouds, classified under three categories:

General (GR)	Overall view of the system
Federation (FR)	Resource-sharing and management of the federation.
Usability (UR)	Point of view of the user of the framework, either as administrator or as end-user.

Table 1 expresses the identified requirements for the two basic mechanisms of *generating* and of *managing* a federation with **FedUp!**.

Table 1. General requirements for generating and managing federations

GR1	**FedUp!** must support heterogeneous clouds.
GR2	**FedUp!** must have low impact on the existing infrastructure of individual clouds.
GR3	**FedUp!** must allow the generation and dismissal of federations in a transparent way with respect to individual clouds.
FR1	**FedUp!** must support the join of new member clouds and their abandoning.
FR2	**FedUp!** must support an automatic generation of federations.
FR3	**FedUp!** must generate federations orchestrated by an automatic system.
UR1	**FedUp!** must provide the generated federation with mechanisms to allow access control to federation resources.
UR2	**FedUp!** must allow a generated federation owner to specify the values of a pre-defined set of tags.

Table 2. Specific features required on a microservice implementation

F1	In order to be realistically orchestrable, microservices must work in isolation without requiring a dedicated VM each.
F2	In order to be efficiently orchestrable (e.g. to support redundancy or migration), microservices have to be properly defined, by separating different data domains into different services, as expected in a microservice scenario.

We argue that such requirements, especially **GR3**, **FR2** and **FR3**, are best met if the deployed Federations are realized via a microservice approach, thus automating both generation and management of federations (e.g. **FedUp!** can provide these functionalities as services). Table 2 collects the resulting requirements on implementation of Infrastructure Services composing a Federation.

3 Generating and Managing Federations with FedUp!

In this section we describe the main components and functionalities of the **FedUp!** platform and the naming conventions used in this paper.

FedUp! is an innovative solution for generating and managing multiple cloud federations using an approach based on cloud services. A federation is conceived as a set of contributions from individual clouds with the aim of sharing and optimizing their own resources, together with a number of services for the dynamic proposal, acquisition and withdrawal of these contributions.

We refer to any federation generated via the **FedUp!** platform as a *Federation*. The owner of the individual cloud or tenant starting the generation process for a Federation, is referred to the *Federation owner*.

The actors which may interact with the platform have been identified as:

- ***FedUp!** Administrator*, responsible for **FedUp!** platform management.
- *Cloud* or *tenant administrator*, using the **FedUp!** utilities to create a Federation, or to make an individual cloud (or tenant) join an existing Federation generated with **FedUp!**.
- *User* of a cloud member of a Federation, who can request resources or services made available to the Federation.

Table 3. The List of utilities and actions in **FedUp!**

Utility and Action	Description
`FedUp.Fed`	Services for federation management.
`FedUp.Fed.Create`	To create a new Federation. The cloud creating the new Federation becomes the owner.
`FedUp.Fed.Update`	To change/update a Federation features, tags included.
`FedUp.Fed.Dismiss`	To dismiss a Federation. All clouds, except the owner, need to quit.
`FedUp.Fed.Acquire`	To integrate a cloud that wants to adhere to a Federation.
`FedUp.Fed.Search`	To search for active Federations that satisfy the requirements of an individual cloud. This option is available for clouds that are looking for Federations to adhere to.
`FedUp.Cloud`	Services for cloud/tenant management.
`FedUp.Cloud.Join`	To adhere to a named Federation or to Federations characterized by given tags.
`FedUp.Cloud.Join(target)`	To adhere to a targeted Federation.
`FedUp.Cloud.Join(tags[])`	To adhere to a Federation that matches the given requirements (tags).
`FedUp.Cloud.Update`	To change/update a cloud features, tags included.
`FedUp.Cloud.Leave`	To allow a cloud to leave a Federation, releasing the resources that cloud allocated to that Federation.
`FedUp.Cloud.Search`	To search for available clouds matching the requirements of a Federation that wants to acquire clouds.

As **FedUp!** is based on cloud services, in order to avoid ambiguities, we call "Utilities" the services that manage **FedUp!**. Each utility consists of a set of operations named "Actions" (see Table 3). **FedUp!** presents two main utilities:

- `FedUp.Fed`: this utility supports the generation and management of the Federations created with the **FedUp!** platform. Each Federation can consist of the aggregation of specific features established by the Federation owner at creation time (e.g. based on requested or granted quality of service, the level of safety on economic factors of political, institutional, geographical, etc.). The Federation features are described as tags. A dictionary of accepted tags is defined by **FedUp!** and the owner can choose the tags with their associated values, characterizing the new Federation at creation time. Tags are used by search functions available for the individual clouds to select the federations that match the membership requirements.
- `FedUp.Cloud`: this utility supports the communication with individual clouds requesting to join a Federation. Each individual cloud can join a specific Federation or can apply to join any Federation that matches specific tags.

These utilities rely in turn on the following structures:

- `FedUp.Registry` maintains information on the generated Federations and on the individual clouds that want to offer or use services offered by **FedUp!**.
- `FedUp.ServiceRegistry` maintains the information on services in terms of configuration and set of microservices.
- `FedUp.ContainerHub` maintains the images of the containers relative to the various microservices.
- `FedUp.ConfiguratorMaster` is responsible for ensuring that Federations generated with **FedUp!** be properly configured in terms of the presence of files, installed packages and services running to manage that Federation. More information about this component is given in Section 4.

In particular, `FedUp.Registry` contains:

- Data on generated Federations together with the corresponding tags describing their main features. The tags are assigned by the Federation owner at generation time, but can be changed during its construction and are used by the search functions made available for the individual clouds to search for Federations matching the membership requirements of the individual cloud.
- References to the individual clouds that want to share resources through the **FedUp!** platform with the corresponding tags describing their main features. Tags are assigned by the cloud owner during the registration to **FedUp!** and are used by search functions made available for the Federations to select individual clouds matching the Federation membership requirements.
- The map of participant clouds to individual Federations and the map of clouds available to federate.

The information provided by `FedUp.ServiceRegistry` fully defines a Federation infrastructure service, in terms of the information necessary to a cluster manager to correctly deploy and maintain the service. In particular, the following queries are handled:

– given the *id* of a service, the *id*s of the microservices composing that service;
– given the *id* of a microservice, the *id* of a container image relative to that microservice.

Finally, `FedUp.ContainerHub` can be queried to retrieve, given the *id* of a microservice, the correspondent image of a container.

An overview of the proposed solution is drawn in Figure 1, while more information about how its realization is given in Section 4.

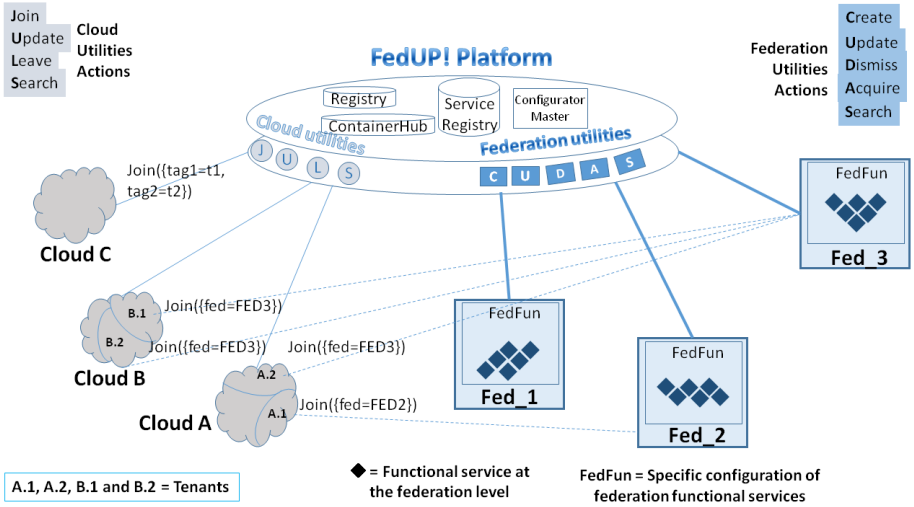

Fig. 1. A conceptual overview of the proposed solution

4 Implementation Aspects

In this section we provide some information about how to define a service configuration for a Federation and how to deploy the corresponding set of microservices using containers based on Docker[4], Saltstack[5] and Kubernetes[6] technologies.

We briefly introduce the following notions.

– *Microservice*: a term referring to a way of designing an architecture as a set of independently deployable services [5]. A single service can be thought as composed of a set of loosely-coupled microservices.

[4] https://www.docker.com/
[5] http://saltstack.com/
[6] https://github.com/kubernetes/kubernetes/blob/master/docs/design/README.md

- *Configuration Management Engine*: a software responsible of ensuring that a remote Operating System is properly configured in terms of the presence of files, installed packages and running services.
- *Container*: an isolated, resource-controlled, portable operating environment.
- *Container Cluster Manager*: a software capable of orchestrating Containers among a set of nodes, managing a so called *Container Cluster*.

SaltStack is a well-known Configuration Management Engine based on a master-slave architecture [8]. It is also classifiable as a Remote Execution Engine since it allows the execution of remote executable files. The SaltStack master configures the Operative Systems of a set of machines (called *minions*) such that they comply with certain desired states. Among other configuration management engines, Saltstack is characterized by the use of asynchronous message queues. Containers are widely used to deploy software, since they are a more efficient alternative to virtual machines.

Docker is a common implementation of containers. Kubernetes is a Container Cluster Manager based on a master-slave architecture. It is defined as a system for managing containerized applications across multiple hosts, providing basic mechanisms for deployment, maintenance, and scaling of applications. Kubernetes automatically manages a cluster of containers by means of a scheduler, changing the state of the cluster to keep it consistent with respect to a set of declarative primitives, codified as YAML[7] formulas, and regarding topological, workload and policy aspects. Kubernetes organizes containers in a multi-level, hierarchical way. In Kubernetes terminology, a *cluster* is composed of a set of nodes, each *node* is composed of a set of pods, and a *pod* is composed of a set of containers. A *container* is intended to host a single microservice, realizing a part of the functionalities of a specific service. A service is not statically associated with a particular pod, but its traffic may be routed to different pods depending on the activities of the scheduler. This mechanism transparently decouples the containers from the services, providing robust availability.

The **FedUp!** platform is intended to automatically deploy *Federation Infrastructures* (which have been declaratively specified) via Kubernetes. This implies a possible coupling between the development of a Federation Infrastructure and Kubernetes, based on the following observations.

1. Kubernetes uses its own network models for inter-service communications. For example, it offers a DNS service as a pluggable component. This technique transparently masks the dynamic routing from a service to potentially different pods. As a consequence, differently from traditional models, services need to query the DNS service frequently.
2. Kubernetes needs a declarative proposition to be prepared for every single definition of a container, service or pod used by the infrastructure.

In order to decrease the coupling between the development of services and (specific versions of) Kubernetes, we exploit the flexibility of its network model.

[7] YAML Ain't Markup Language,http://yaml.org/

In particular, **FedUp!** adopts a traditional network model (i.e. using static IP address and ports). This is realized by defining a formalized model for the definition of a Federation Infrastructure, and an automatic generator of declarative primitives to be used for the deployment of a federation.

With this solution, the declarative propositions influencing a deployed Federation Infrastructure (e.g. numbers of replicas for pods, minimum policies for machines) need not be considered by the developers of a Federation Infrastructure. Given that reasonable declarative propositions can be chosen by **FedUp!**, developers are lifted from the burden of acquiring the necessary knowledge of, and re-implement services to fits to, a specific Kubernetes network model.

5 Federating Heterogeneous Clouds

In this section, we focus on the two basic mechanisms for creating a Federation and for the joining/acquisition of a cloud (actually a tenant) to a Federation. The other actions provided by **FedUp!** are realized in analogous ways. Figure 2 provides an activity diagram describing the fundamental phases involved in these two processes, which are then detailed in the following subsections.

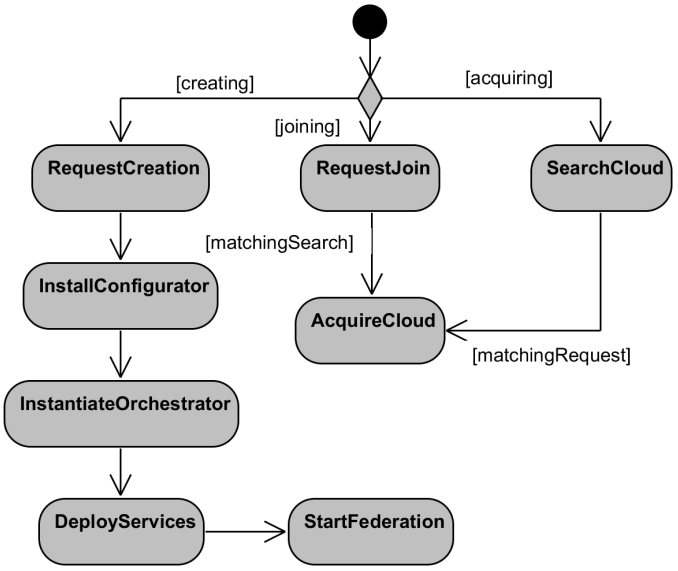

Fig. 2. An overview of the main processes in **FedUp!**.

While creation only involves the originator cloud and **FedUp!**, joining a Federation *Fed* is seen as a two-step process in which a cloud *Cld* (actually a tenant in a cloud) issues a request via the `Join` action, also providing information on what it can contribute, and *Cld* enters *Fed* when the latter accepts a specific

request, via the `Acquire` action, or interrogates the **FedUp!** `Search` service about the presence of candidate clouds characterised by specific features. To support this kind of mechanisms, join and acquire requests are associated with a collection of tags expressing cloud features, e.g. the types and quantities of resources or services a cloud can contribute to a Federation, the SLAs of the contributed services, or even the intention to join a specific Federation. **FedUp!** will therefore publish a dictionary of managed tags and their admissible values. For example, in Figure 1 in Section 3, we have indicated that clouds **A** and **B** had issued specific requests to participate in **Fed2** and **Fed3**, while cloud **C** has issued a request characterised by two specific values for two different tags.

Similarly, a Federation *Fed* can directly acquire a cloud, based on previous notifications of interest for some features, so that if a matching cloud issues a request to participate in *Fed*, it is automatically acquired. Criteria for accepting a new member are proper to any Federation and need not be public. For example, an initial set of governmental agencies can decide to set up a version of **FedUp!** and accept only clouds from other public agencies, while a European initiative for a federated cloud could accept only clouds managed by agencies at governmental level from EU nations.

Each cloud accepted in a Federation *Fed* will generate a partition of its resources to act as tenant for *Fed*. In principle a cloud can participate in several federations, e.g in Figure 1 cloud **A** participates both in **Fed2** and **Fed3**, and even present different tenants to the same Federation, e.g. cloud **B** participates in **Fed3** with two tenants, for example one for high performance resources and one for less stringent SLAs.

Given a single Federation, the shared resources can be partitioned into the *infrastructure resources* used to host the Federation itself, and the *cloud resources* that can be offered, or acquired, by the individual clouds. Unless differently specified, we use the term *resource* to refer to an infrastructure resource. Analogously, the term *service* refers to an infrastructure service rather than a service offered or acquired by the federation members. In this paper we are not specifically concerned with the implementation details of a Federation Infrastructure, but we just mention the following:

- A Federation Infrastructure is typically realized through a service named Workload Manager that interacts with a centralized registry, for storing and assigning cloud-resources and cloud-offerings.
- Services within the Federation Infrastructure may be relative to networking, monitoring, billing and authorization aspects.
- A Federation Infrastructure may be capable of federating heterogeneously in terms of the particular PaaS solutions used by the individual clouds.
- A Federation Infrastructure may have a (logically) centralized architecture.

Figure 3 shows the UML metamodel for dynamic, service-oriented, network Federation Infrastructures, using a containerized microservice architecture, in which the communication is realized over IP addresses/ports. In particular, a *FedInfrastructure* is associated with a non-empty set of instances of *Service*,

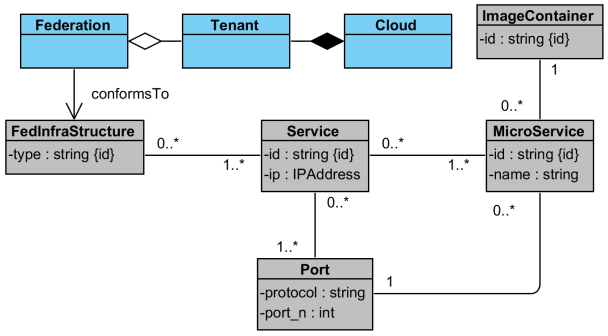

Fig. 3. The metamodel for the federation infrastructure

each described by a static IP address and one or more instances of *Port* to be used by other services. Each service results from the aggregation of a number of instances of *MicroService*, each in turn associated with a single port and with a single *ContainerImage*. The relation is also shown between the infrastructure metamodel and the general model of federations, which need to be conformant to some infrastructure and composed of tenants belonging to clouds.

A particular instance of this model can be translated into a proper set of declarative propositions: together with the declarative propositions provided by **FedUp!** (Section 4), they form the necessary information for a cluster manager to deploy and manage the federation. A Federation Infrastructure deployed by **FedUp!** has the following characteristics:

- A deployed Federation Infrastructure consists in a set of deployed services.
- A deployed service has network visibility among any other deployed service.
- A deployed service is composed of one or more microservices, each one deployed on a different container.

5.1 Creating a Federation

Figure 4 shows a sequence diagram presenting the fundamental steps for creating a federation. An administrator of a tenant *Ten1* in a cloud *Cld1* issues a request to `FedUp.Fed` (namely to its `Create` action) for creating a new Federation (arrow 1). The request contains the desired federation type (among the available Federation Infrastructure solutions) and the credentials usable to interact with *Cld1*'s IaaS Resource Manager. `FedUp.Fed` is capable of interacting with heterogeneous clouds trough a virtual interface exposed by an adapter service provided by **FedUp!** , using plugins associated to different IaaS solutions.

`FedUp.Fed` uses the received credentials to instantiate a Virtual Machine inside *Ten1* (arrow 2). This VM, named *Federation Configurator*, is a SaltStack slave of the `FedUp.ConfiguratorMaster`, located on the **FedUp!** side. The `FedUp.ConfiguratorMaster` remotely configures the *Federation Configurator* (arrow 3) so that it can in turn instantiate and configure inside *Ten1* a set of

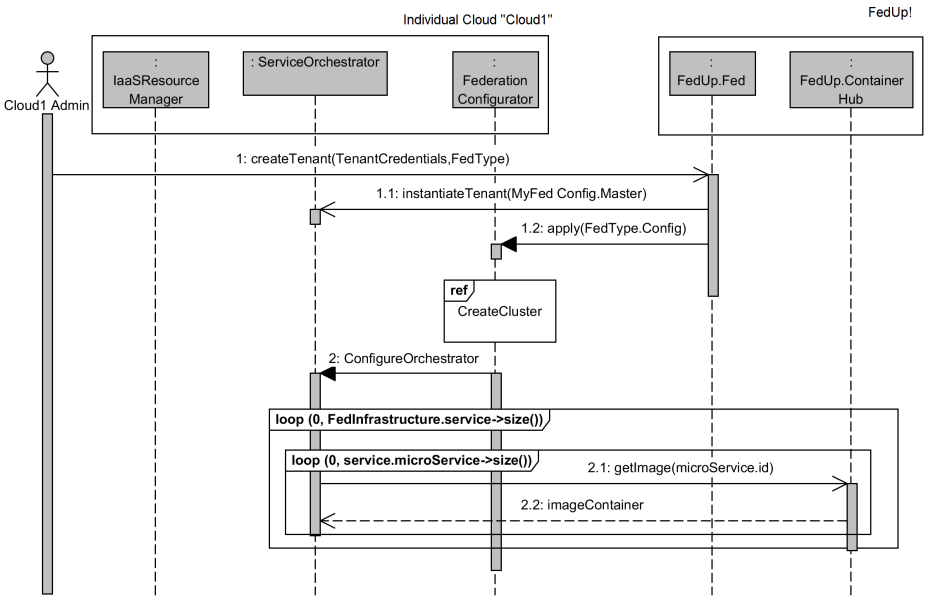

Fig. 4. The sequence diagram for creating a federation

virtual machines dedicated to form a Kubernetes cluster (through the operation `Create Cluster`, not further detailed here). In particular, a *Service Orchestrator* VM is dedicated to host a Kubernetes Master, while the other ones are set up to become Kubernetes nodes. The *Federation Configurator* is at the same time a SaltStack slave of `FedUp.ConfiguratorMaster` and a SaltStack master of the machines composing the cluster.

Figure 5 shows the considered SaltStack dependencies: a black triangle represents a slave relation with respect to a SaltStack master connected to the line entering to the triangle, while a star represents a virtual machine which is part of a Kubernetes Cluster. After the Kubernetes cluster is created, the *Service Orchestrator* is remotely configured (arrow 4 of Figure 4) to retrieve from **FedUp!** the necessary files to deploy the Infrastructure Services. In particular, it is shown how the `FedUp.ContainerHub` provides the containers relative to the microservices composing the Infrastructure Services. Similarly, the *Service Orchestrator* retrieves the configuration files for the pods, containers and services (not shown) from the `FedUp.ServiceRegistry`.

5.2 Joining a federation

Figure 6 presents the fundamental steps for joining a federation, from the perspective of a new member. The administrator of a Tenant *Ten2* of cloud *Cld2* (not federated yet) communicates the availability to join a federation, also providing a set of tags, describing its possible contributions. An ID `id_offer_x` is

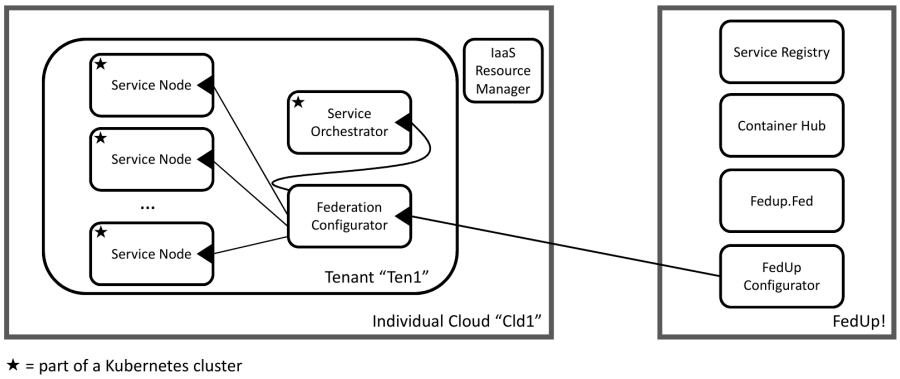

Fig. 5. Visualization of the SaltStack dependencies: a black triangle represents a slave relation with respect to a SaltStack master connected to the line entering to the triangle. A star identifies an element of a Kubernetes cluster.

sent back to the administrator, to be used in a polling request for the state of its offering. Assuming that at a certain moment an existing Federation *FedY* accepts this join offering (either as a consequence of an administrator explicit action, or of a standing search for matching offerings) and acquires the originator tenant, the ID of *FedY* will be returned to *Ten2* on the subsequent request.

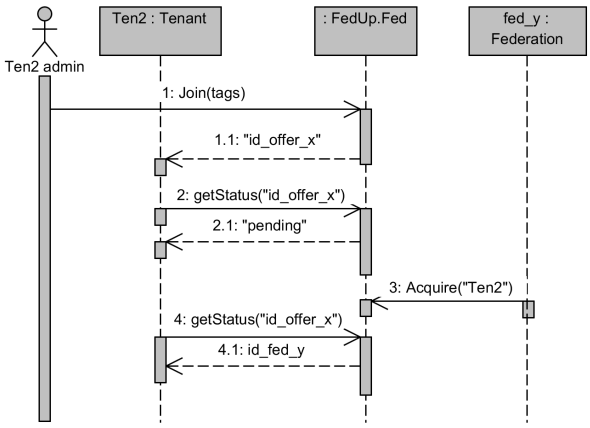

Fig. 6. The sequence diagram for a tenant joining a federation

Figure 7 shows the process for a *Federation Y*, identified by `id_offer_y`, to acquire a cloud *Cld2*. *Federation Y* looks up the `FedUp.Registry` for a set of acquirable tenants, based on a set of tags ("filter"). The set is populated based on an interaction between the registry and `FedUp.Fed`. In particular a subset of

the available offerings, arranged into a list of IDs, (`[...,id_offer_x,...]`) is returned to the Federation. *Federation Y* can communicate its intention to acquire the offering identified by `id_offer_x`, which had been published by *Cld2*, to the `Registry`, which forwards the request to `FedUp.Fed`, that will complete the join process. An acknowledgement is sent back to *Cld2*, communicating `id_offer_y`.

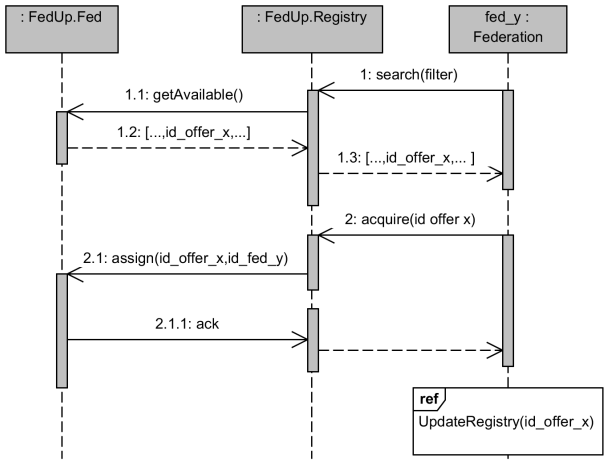

Fig. 7. The sequence diagram for a federation acquiring a tenant

6 Conclusions and Future Work

FedUp! provides a lean PaaS approach to the flexible deployment and management of federations of clouds, opening the way to the notion of Federation as a Service, whereby a central platform will maintain images of predefined configurations that an administrator can decide to install and activate on a managed portion of a cloud. In principle, any cloud user could exploit the resources obtained by joining a federation to federate them in turn within existing federations, or to start a new federation.

The platform relies on notions of containers, microservices, configurations and clusters made popular by technologies such as Docker, Kubernetes,and Saltstack, which are adopted in its current specification, but is in principle not tied to them, as it only requires that the tenant designated to start a federation will allow the installation of a virtual machine with an initial configurator, which will then execute the needed operations on the tenant. The current working prototype will be subject to extensive evaluation.

In this line, one can envision a marketplace of federation services, where also specific services that a single federation wants to offer can be defined and offered for configurations. For example, one could offer specific forms of workload management, or of data security across the different members of the federation.

In this paper we described how instances of a metamodel for Federation Infrastructures to be generated are memorized by **FedUp!**. Moreover, we described how tags can be used to facilitate a targeted encounter between individual clouds and Federations. It is possible to further develop these concepts. One can think of supporting the sharing of the definition of services among different types of Federation Infrastructure. In this context, tags could be associated with specific definitions of services, for example to assess compatibility with other services, or with different configurations of the same service to be used in different scenarios. In this manner, a customizable Federation Infrastructures could be deployed by blending and re-using (partially or integrally) services memorized by **FedUp!** with a modular approach. The same concept could even apply to microservices.

Finally, **FedUp!** could provide the possibility to a tenant administrator to specify parameters to be used in the process of creating a Federation. For example, by allowing to choose the desired number of replicas for services, the possibility would be offered to provide a parametric tuning of the robustness, in terms of availability, of the deployed services.

Acknowledgments

This work has been supported by the EU H2020 Programme under the SUNFISH project, grant agreement N.644666.

References

1. O. Agmon Ben-Yehuda, M. Ben-Yehuda, A. Schuster, and D. Tsafrir. The rise of RaaS: the resource-as-a-service cloud. *Commun. ACM*, 57(7):76–84, 2014.
2. D. Bermbach, T. Kurze, and S. Tai. Cloud federation: Effects of federated compute resources on quality of service and cost. In *2Proc. IC2E 2013*, pages 31–37, 2013.
3. A. Kertesz. Characterizing cloud federation approaches. In *Cloud Computing: Challenges, Limitations and R&D Solutions*, pages 277–296. Springer, 2014.
4. A. Marinos and G. Briscoe. Community cloud computing. In M. G. Jaatun, G. Zhao, and C. Rong, editors, *Proc. CloudCom 2009*, pages 472–484. Springer, 2009.
5. S. Newman. *Building Microservices. Designing Fine-Grained Systems.* O'Reilly Media, 2015.
6. H. Nicanfar, Q. Liu, P. TalebiFard, W. Cai, and V. C. M. Leung. Community cloud: Concept, model, attacks and solution. In *Proc. CloudCom 2013*, volume 2, pages 126–131, Dec 2013.
7. B. Rochwerger, D. Breitgand, E. Levy, A. Galis, K. Nagin, I. M. Llorente, R. S. Montero, Y. Wolfsthal, E. Elmroth, J. A. Cáceres, M. Ben-Yehuda, W. Emmerich, and F. Galán. The Reservoir model and architecture for open federated cloud computing. *IBM Journal of Research and Development*, 53(4), 2009.
8. C. Sebenik and T. Hatch. *Salt Essentials. Fast, Scalable, and Flexible Automation.* O'Reilly Media, 2015.
9. D. Villegas, N. Bobroff, I. Rodero, J. Delgado, Y. Liu, A. Devarakonda, L. Fong, S. M. Sadjadi, and M. Parashar. Cloud federation in a layered service model. *Journal of Computer and System Sciences*, 78(5):1330 – 1344, 2012.