A dynamic and extensible web portal enabling the deployment of scientific virtual computational environments on hybrid e-infrastructures

Marica Antonacci
Istituto Nazionale di Fisica Nucleare
Via E. Orabona 4, 70125 Bari, Italy
Email: marica.antonacci@ba.infn.it

Daniele Spiga
Istituto Nazionale di Fisica Nucleare
Via A. Pascoli, 06123 Perugia, Italy
Email: daniele.spiga@pg.infn.it

Davide Salomoni
Istituto Nazionale di Fisica Nucleare
Viale Berti Pichat 6/2, 40127 Bologna, Italy
Email: davide.salomoni@cnaf.infn.it

Diego Ciangottini
Istituto Nazionale di Fisica Nucleare
Via A. Pascoli, 06123 Perugia, Italy

Vincenzo Ciaschini
Istituto Nazionale di Fisica Nucleare
Viale Berti Pichat 6/2, 40127 Bologna, Italy

Alessandro Costantini
Istituto Nazionale di Fisica Nucleare
Viale Berti Pichat 6/2, 40127 Bologna, Italy

Giacinto Donvito
Istituto Nazionale di Fisica Nucleare
Via E. Orabona 4, 70125 Bari, Italy

Doina Cristina Duma
Istituto Nazionale di Fisica Nucleare
Viale Berti Pichat 6/2, 40127 Bologna, Italy

Federica Fanzago
Istituto Nazionale di Fisica Nucleare
Via Marzolo 8, 35131 Padova, Italy

Emidio Giorgio
Istituto Nazionale di Fisica Nucleare
Laboratori Nazionali del Sud
via S.Sofia 62, 95123 Catania, Italy

Alessandro Italiano
Istituto Nazionale di Fisica Nucleare
Via E. Orabona 4, 70125 Bari, Italy

Massimo Sgaravatto
Istituto Nazionale di Fisica Nucleare
Via Marzolo 8, 35131 Padova, Italy

Vincenzo Spinoso
Istituto Nazionale di Fisica Nucleare
Via E. Orabona 4, 70125 Bari, Italy

Stefano Stalio
Istituto Nazionale di Fisica Nucleare
Laboratori Nazionali del Gran Sasso
Via G. Acitelli 22, 67100 Assergi L'Aquila, Italy

Mirco Tracolli
Istituto Nazionale di Fisica Nucleare
Via A. Pascoli, 06123 Perugia, Italy

Marco Verlato
Istituto Nazionale di Fisica Nucleare
Via Marzolo 8, 35131 Padova, Italy

***Abstract*—For the past ten years, INFN has been investing heavily in developing solutions to enable transparent access to a multi-site federated Cloud infrastructure. A main goal is to provide a generic model to allow INFN users fair and simple ways in accessing resources, regardless from the (richness of their) experiment, the proximity to a powerful computing centre, the capability to administer complex resources such as those offering GPUs. The ultimate objective is to reduce the learning curve required to deploy, manage and utilize computing services on a federated cloud system. To this end, a dynamic and extensible web portal has been put in production within the INFN Cloud project. In this paper, a detailed overview of the architecture behind the web portal will be provided. Moreover, the strategy adopted for implementing a re-usable and highly customizable system to describe and deploy any service on cloud will be described. Finally, the foreseen evolution toward a multi-level dashboard will be shown.**

***Keywords*—scientific portal, service composition, automated deployment**

I. Introduction

The Italian National Institute for Nuclear Physics (INFN) is a pioneer in the design and implementation of large-scale computing infrastructures and applications. These were originally developed primarily to meet the needs of the latest generations of high energy physics (HEP) experiments, but are now rapidly extending to other scientific communities. The current INFN production infrastructure consists of 9 medium-sized centers (Tier-2 in LHC Computing Grid parlance) and 1 large Tier-1 center at CNAF (Bologna). During the last decade, INFN invested substantial effort in several R&D activities to develop new systems and platforms for scientific computation, leading and participating to many national and international projects. Among them, particularly significant ones are INDIGO-DataCloud [1], EOSC-hub [2], XDC [4], DEEP [3], and ESCAPE [5]. The INFN computing infrastructure also provides ISO/IEC 27001, 27017 and 27018 certified sites, for the secure processing of data and applications in the cloud

that involve sensitive data (e.g. medical type) [6]. Based on all these activities, as well as on its 10-year experience gained in developing and operating the Grid middleware for scientific communities, at the beginning of 2020 INFN inaugurated a production-level national heterogeneous federated Cloud infrastructure through the "INFN Cloud Project" [7]. INFN Cloud provides:

- a multi-site federated Cloud infrastructure owned by INFN, expandable to other Cloud infrastructures and resources;
- a set of services that can be used through a portal, from a terminal or with a set of APIs;
- a "high-level" mechanism for adapting and evolving the service portfolio, according to the needs and requests of the users;
- a fully distributed intra-INFN organization for the support and management of both the infrastructure and its services;
- a set of rules for regulating access and management to INFN Cloud resources, incorporating INFN policies as well as the more general national and international legal stipulations.

The two main architectural pillars of INFN Cloud are: a distributed resource orchestration framework and a consistent, standards-based federated solution for identity access management.

Based on community requirements, INFN Cloud developed an extensible portfolio tailored to multi-disciplinary scientific communities, spanning from traditional IaaS to more elaborate PaaS and SaaS solutions. From a technical perspective, the service portfolio is based on open-source modular software and on *de-jure* or *de facto* standards, following the principle of service composition. The main assumption is that there is no one-size-fits-all solution, but rather workflows that allow to create and deploy the best components integration given a certain problem, using a composition of basic building blocks. The foundation, in this respect, is to develop and provide the proper tools needed to simplify and democratize access to computing resources. This can be achieved on the one hand by reducing the learning curve of the available functionalities, and on the other lowering the adoption barriers that may arise due to complex technicalities that typically represent a significant obstacle, not only to access resources, but also to define and deploy computing models. The "high-level" mechanism provided by INFN Cloud is designed with the goal to enable users to create and provision infrastructure deployments, automatically and in a reproducible way, on "any cloud provider" with almost zero effort. The "any cloud provider" part is essential, since it allows the same services to be created and used on multiple, federated Cloud infrastructures. The core component of the INFN Cloud framework, offering users the handles needed to build their required solutions, is the INDIGO Platform-as-a-Service (PaaS) layer [8]. In fact, the PaaS not only federates and orchestrates heterogeneous compute and storage resources, but also provides the engine on top of which a web portal enabling easy deployment of scientific virtual computational environments has been developed.

The landscape of scientific gateways and Virtual Research Environments (VREs) is rich with the diverse available frameworks and services and each of them have their strengths and peculiarities. For example, WS-PGRADE/gUSE (Web Services Parallel Grid Runtime and Developer. Environment/Grid User Support Environment) provides a workflow-oriented GUI and APIs to create and execute workflows and manage data on grid computing infrastructures [9]. Within the SCI-BUS EU FP7 project [10], the platform was extended in order to allow users to run workflows on cloud systems instead of or in addition to grids through the CloudBroker Platform [11], available in hosted and licensed variants. The Platform is mainly suitable for batch-oriented command line software, both serial or parallel (via MPI or other tools). Another technology relevant to Science Gateways is gCUBE, a software framework designed to build e-Infrastructures supporting Virtual Research Environments [12]. To this end, it offers a comprehensive set of data management commodities on various types of data and a rich array of "mediators" to interface diverse Infrastructures. Via these mediator services, the storage facilities, processing facilities and data resources of the external infrastructures are conceptually unified to become gCube resources. gCube is currently used to govern the infrastructure set up by the European integrated project D4Science [13]. The portal of both WS-PGRADE/gUSE and gCUBE/D4Science is based on Liferay technology [14]. Liferay was designed in early 2000s and became the most used framework to build Science Gateways in the "Grid world". It is a Java-based platform that leverages advanced technologies of JSR 286 portlets [15] and OSGi [16]. New content can be added only in the form of portlets [17] that require Java code development. The web portal we present in this paper has been conceived as a lightweight, highly configurable component that can be easily extended for supporting diverse research domains, writing almost zero code and re-using available building blocks. Therefore, we have opted for modern Web development frameworks such as Flask plus CSS and JavaScript libraries, as detailed in the next sections. Unlike gCUBE/D4Science, the INDIGO PaaS system has been designed as a solution for federating different computing infrastructures (clouds, HPC, container platforms) in a loosely coupled way: no middleware is installed on the external infrastructures and the interactions with them are managed via REST APIs (using standard libraries, such as Apache jclouds [18], to interface different clouds). The adoption of a standard language for describing the topology of the virtual computing environments is another key difference w.r.t. the mentioned frameworks: we provide a high-level portable description of the applications (including nodes and their relationships) and the automation of their deployment following the Infrastructure as Code paradigm.

The remainder of this paper is structured as follows: Section II will detail the architecture of the PaaS Orchestrator, describing both the main high-level steps of a deployment workflow, as well as the technical implementation. It will

also provide the PaaS Dashboard implementation details. In section III, the strategy used to implement the INFN Cloud services will be shown, including the related implementation approach. Section IV will discuss the INFN Cloud service catalogue and its main capabilities, while Section V will cover a concrete example of a service portfolio developed within INFN Cloud. Finally, Section VI will provide an overview of the PaaS Dashboard evolution toward a multi-level approach.

II. INDIGO PaaS Architecture

The main service of the INDIGO PaaS is the Orchestrator [19]. It implements a workflow engine to orchestrate and automate the complex chain of tasks and interactions needed to perform the deployment of services across one of more Cloud infrastructures. These range from the selection of the best provider among a set of federated Clouds, to the complete provisioning and configuration of the required cloud services and resources. Technically, the PaaS layer consists of a set of microservices that interact through lightweight mechanisms based on REST APIs. Figure 1 sketches the main steps of a deployment workflow:

1) a user requests the deployment of a certain service, specifying its requirements through the Dashboard; optionally, he/she can specify the timeout for the deployment to complete;
2) the Dashboard submits the user request to the PaaS Orchestrator through a REST API call;
3) the PaaS Orchestrator contacts a set of auxiliary services to get information useful to schedule the deployment on the best chosen provider. These services are:
   - INDIGO-IAM [20], in charge of validating user identity;
   - the SLA Manager service, providing information about the federated sites where the user is entitled to create resources thanks to a valid Service Level Agreement (SLA);
   - the Configuration Management DB (CMDB), a dynamic information system that publishes information about the federated providers and their services (e.g. the endpoints of the compute and storage services, the list of cloud images, flavors, etc.);
   - the Monitoring service, that collects metrics and monitoring data for each federated provider;
4) the PaaS Orchestrator contacts the Cloud Provider Ranker (CPR) service, providing the information collected above;
5) the CPR calculates the ranking for each site service through a configurable algorithm and provides the PaaS Orchestrator with a sorted list of candidate cloud providers for the required deployment;
6) the PaaS Orchestrator schedules the deployment on the first provider in the list;
7) the PaaS Orchestrator monitors the deployment until it is successful, fails or until the timeout expires;
8) in case of failure or of an expired timeout, the PaaS Orchestrator schedules the deployment on the next provider in the list, if any. If the overall timeout expires, the PaaS Orchestrator flags the deployment as failed and deletes the allocated resources, if any.

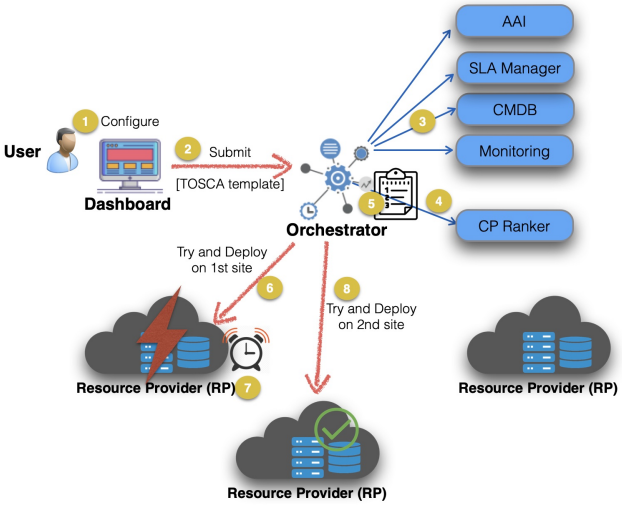

Fig. 1. The INDIGO PaaS high-level architecture.

The CPR is a standalone REST WEB Service which calculates a rank for the cloud provider services depending on the match with specific rules: its core component is a Rule Engine based on the open-source Drools framework [21]. The current ranking algorithm considers the SLA targets (in terms of amount of resources provided to the users) and the monitoring metrics (e.g. health status, response times) as input parameters. The computation of the score is performed through the embedded Rule Engine and can be modified without recompiling the code by changing the Rules files used by the CPR service. The selection of the provider to use is therefore optimized through the ranking algorithm. Nevertheless, failures can happen during the deployment phase that could not be detected in advance by the Monitoring system. In order to handle these cases, the Orchestrator is able to automatically reschedule the deployment to another site and the number of retries is configurable and can be eventually specified by the user.

The federated access to the distributed resources relies on a federated Authentication and Authorization Infrastructure (AAI) and on the ability of each service in the whole stack, from the dashboard to the PaaS and IaaS levels, to manage the OAuth [8] tokens issued by the trusted Identity Provider(s). The PaaS Orchestrator mainly relies on INDIGO-IAM, an Identity and Access Management service first developed in the context of the INDIGO-Datacloud project, and currently maintained and developed by INFN. INDIGO-IAM provides a flexible authentication support (SAML, X.509, OpenID Connect, username/password, etc.), account linking, registration service, group membership management, enforcement of AUP acceptance, easy integration with client applications and services via standard OAuth/OpenID Connect mechanisms.

Finally, the PaaS Orchestrator supports a variety of interfaces in order to cope with distinct needs and requirements. In details, it provides a REST interface, a command-line interface and a graphical user interface (GUI).

A. PaaS REST interface and CLI

The PaaS Orchestrator *lingua franca* is TOSCA (Topology and Orchestration Specification for Cloud Applications) [22]. TOSCA is an OASIS standard language to describe cloud based applications, their components, relationships, dependencies and the processes that manage them. Any service or application to be deployed is therefore described through a TOSCA template. The Orchestrator provides REST APIs for instance to create a deployment starting from a TOSCA template, to get the list of active deployments, or to delete a deployment. A command-line tool is also available, called *orchent* [23], written in Go programming language [24]. It implements a set of commands that match the actions that can be performed using the PaaS Orchestrator REST APIs. The most relevant *orchent* commands are:

- *depcreate*: create a new deployment;
- *depls*: list active deployments;
- *depshow*: show details of a given deployment;
- *deplog*: show the contextualization log of a deployment;
- *depupdate*: update a running deployment;
- *depdel*: delete a deployment.

B. PaaS Dashboard

Handling TOSCA templates is not a simple task; many technical details should be hidden to the end user, also because typically a researcher is not necessarily interested in learning about them. Similarly, many users might not like using command line interfaces, finding them more difficult to manage than a graphical user interface (GUI). To this end, the PaaS Orchestrator has been equipped with a Dashboard, a user-friendly graphical web portal that allows users to:

1) select the service to deploy from a catalogue of predefined templates;
2) configure and customize the deployment through a simple form;
3) monitor and manage the deployments through dedicated menus and views;
4) get notified as soon as a deployment is complete.

The portal empowers end users with a twofold role: to provide a single point of access to the PaaS components, and to offer a catalogue of deployable assets that cover a wide range of service categories.

C. PaaS Dashboard implementation

The PaaS Dashboard is a Python application, developed using open-source tools, such as the Flask web microframework [25] and Bootstrap [26] to create responsive front-end web pages. The state of the application is saved into a relational database, managed through the well known SQL Python toolkit SQLAlchemy [27]. Moreover, some popular, well-maintained Flask extensions have been used for integrating with external services. The most important one is Flask-Dance [28], a library built on top of OAuthLib, that greatly facilitates the integration of OAuth in the Flask application. We have used Flask-Dance to enable the authentication and authorization of the users through INDIGO-IAM.

The Dashboard is also integrated with the open-source HashiCorp Vault [29] software, to centrally manage secrets such as user ssh keys, public Cloud service credentials (e.g. AWS credentials), disk encryption keys and deployment secrets. This is in general sensitive data (e.g. passwords) needed to configure the deployed services. We have configured Vault to enable the *jwt* login via INDIGO-IAM. Proper policies have then been set, to grant read and/or write permissions to specific paths, depending on the users token claims. In this way, the secrets that refer to a given user are completely isolated from the other users. The interactions with Vault are mediated by the Dashboard, providing simple configuration forms to define the secrets; the Dashboard then takes care of managing them, using the Vault HTTP APIs.

III. Services and high level applications integration strategy

As introduced in Section II-B, one of the main INFN Cloud objectives is to simplify the learning curve needed to build and manage services and applications, as well as to access computing resources. To this extent, the adopted strategy is based on the *Infrastructure as Code* paradigm [30], driven by a templating engine used to specify high-level requirements. This allows users to describe *"What"* is needed, instead of *"How"* it should be implemented. Through the templating strategy, a declarative approach can be adopted. From a technical perspective, the system follows a cloud-native approach both for infrastructure services (i.e. virtual clusters or any interface for data access and processing) and applications. Everything is thus containerized using Docker [32], which represents the main mechanism to manage user-tailored runtime environments. Thanks to the declarative approach, extending and composing services becomes much easier; moreover, the process for adapting and evolving the service portfolio in order to address new user requests is much more efficient and maintainable, thanks to code re-usability and sharing.

A. Technical pillars

The deployment of services and applications, software configurations and package installation, including dependencies and user runtime environments, is managed via a combination of three main pillars:

- TOSCA, used to model the topology of the whole application stack;
- Ansible [31], used to manage the automated configuration of the virtual environments taking care of installing and configuring the needed middleware (e.g. the container management software);
- Docker [32], used to encapsulate the high-level application software and runtime;

These technologies have been chosen and exploited as follows, in order to achieve a high-level of automation, re-usability and easy customization. As mentioned, each service, from the simplest to the more complex one, is described through a TOSCA template that models the topology of the service at different layers: at the infrastructure level, the template specifies the compute and storage resources needed to run the service; on top of that, specific middleware components are installed and configured to execute the service (middleware/application levels). To implement this service integration strategy, we exploit two key features of the TOSCA language: re-usability and extensibility. In fact, TOSCA provides a type system to describe possible building blocks useful to construct a topology template. These TOSCA types are reusable TOSCA entities that can be further customized, implementing new derived TOSCA types. Moreover, in the TOSCA specification, each node can define *interfaces* and *artifacts* for realising its deployment and/or managing its lifecycle: they can be installation scripts or binaries, implementing the application functionality. In the custom types we have been developing, artifacts are mainly in the form of Ansible playbooks, that basically are built around re-usable roles uploaded and shared via Ansible Galaxy [33]. Finally, the Ansible recipes are used to install, configure and manage dockerized applications in different environments, from a standalone docker engine (e.g. using docker compose) to complex orchestration clusters, as described in the next subsection.

B. Enhance the service automation

As introduced above, a key objective is to implement a high level of automation. This is a common motif of the INFN Cloud project that propagates up to the upper layer, i.e. where end user applications are eventually managed. In order to enhance the automation and self-healing of the deployed services, INFN Cloud has been integrating Kubernetes (k8s) [35] as container orchestrator, also building on the experience gained with the DODAS project [34]. Nowadays, k8s is a *de facto* standard, widely adopted also in many scientific domains. A few key characteristics of interest in this context are:

- failure recovering: only healthy resources receive traffic. Those that fail health checks for too long will be restarted (nodes and containers);
- automatic scaling, based on resource requests. This is extensible to support a scaling logic based on custom metrics;
- declarative model: the user declares the desired state of an application in a YAML manifest. k8s then does what is needed to maintain that required state;
- comprehensive API set, that for instance allow a CI/CD system to communicate with k8s via REST APIs and CLI. For example, one can automatically deploy new releases defining the proper rollout/rollback strategies.

C. From HELM to TOSCA

From a technical point of view, the k8s based service deployment extends the templating and encapsulation logic explained above. On the one hand, this shows the power and flexibility of the originally developed model; on the other, it keeps the declarative approach valid, given the adoption of Helm Charts [36] to define, install, and upgrade k8s applications. Concretely, we can distinguish two main cases: a plain k8s cluster deployment, and the deployment of an application on top of a k8s cluster. The creation of a plain k8s cluster happens through a configurable Ansible role that takes care of all the needed steps: from the cluster initialization through the "kubeadm" tool, to the deployment of a set of utility services, such as the ingress service based on NGINX, the automatic certificate manager "Cert-Manager", the k8s dashboard, and the metric server. Other optional services can also be activated, based on the actual use case requirement: for instance, it is possible to leverage a shared filesystem, choosing one of the following solutions: Longhorn (the default), NFS and Cephfs via ROOK operator. Also, an integrated monitoring system based on Prometheus [38] and Grafana [39] is provided out of the box [37]. Regarding the deployment of applications on top of a k8s cluster, the presented model adopts the Helm templating technology. This is a key aspect, enabling an efficient and portable way to deploy applications via a common language: in fact, the same Helm chart can be used to deploy the application on top of the k8s cluster deployed via TOSCA, as well as on top of any pre-existing k8s cluster hosted on any other cloud provider (private or public). The applications are structured in such a way that, through the very same base template structure, different flavors of the same cluster can be deployed. For instance, one can activate a certain type of shared filesystem to be used, by putting a flag at the Helm configuration level (in the so called "Helm values"). In addition, multiple applications can be combined as needed using the Helm dependency system, where a child application will wait for the parent to be completely deployed, before starting its own installation [37]. The Helm charts integration in the TOSCA template has been possible thanks to the usage of Ansible roles, which take care of compiling Helm values only when the cluster has been automatically created, and thus all the parametrized information are known. All the produced charts are documented following the current Helm best practices, promoting contributions from anyone interested to fix or add features to the existing charts. Moreover, it allows automatic linting and testing for any external proposed change [40], [41].

IV. Service catalogue

The catalogue available through the PaaS Dashboard (Figure 2) is a graphical representation of the TOSCA templates repository that we have been developing by extending the INDIGO-DC custom types [42]. Each card in the catalogue is associated to one or more templates that have been implemented using a "lego-like" approach, building on top of reusable components and exploiting the TOSCA service

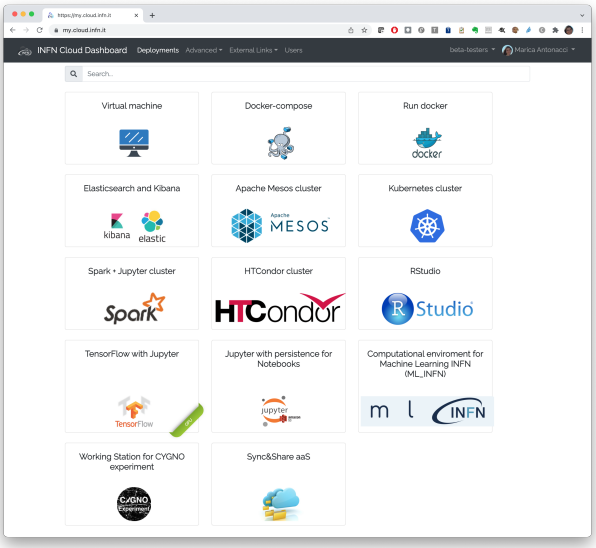

Fig. 2. Service Catalogue

composition pattern. The Dashboard is configured to load the templates from a local folder and is able to dynamically create an HTML form starting from the inputs section of each TOSCA template. An example is shown in Figure 3: when a user selects the "Virtual machine" service from the catalogue, the inputs defined in the corresponding TOSCA template (Figure 4 are automatically interpreted by the Dashboard that creates the web form. Once the user has filled in the input values and submitted the form, the Dashboard sends the template and the input values to the PaaS Orchestrator, calling its REST API. When a user logs in to the Dashboard, the main page will show a customized view of the available services depending on the specific INDIGO-IAM group membership of the user. This allows to simply define customized views (per-user, or per-group) of the Service Catalogue.

For each deployment, the Dashboard allows also to set some advanced configuration options, as shown in Figure 5. In particular, it is possible to bypass the automatic scheduling implemented by the PaaS Orchestrator: in this case, a user can select a specific provider to send the deployment request to. Under the hood, the drop-down menu for selecting a provider is automatically created by the Dashboard, interacting with the SLA Manager Service to get the list of sites where the user owns some resource quota. Before submitting the request to the PaaS Orchestrator, the Dashboard completes the TOSCA template including the proper SLA placement policy (Figure 6).

V. A PRODUCTION-READY EXAMPLE: THE INFN CLOUD SERVICE PORTFOLIO

At the time of writing, there are several categories of services available: 1) general purpose services, 2) experiment specific services, and 3) k8s-based services.

The general purpose services allow the automatic deployment of:

- Virtual Machine with or without external block storage, eventually equipped with docker engine and docker-compose, on top of which dockerized services can be automatically started;
- data analytics and visualization environments based on Elasticsearch [43] and Kibana [44];
- file sync & share solution based on OwnCloud [45] with 1) replicated backend storage on the S3-compliant Object Storage provided by the INFN Cloud infrastructure; 2) automatic configuration for enabling INDIGO IAM OpenID Connect authentication; 3) pre-installed and configured backup cron jobs for safely storing configuration and data on the Object Storage for future restore in case of disaster; 4) integrated application and backup monitoring based on Nagios [46];
- web-based multi-user interactive development environment for notebooks, code and data built on JupyterLab [47] and enahnced with 1) persistent storage areas for storing results and notebooks for future re-use; 2) a monitoring system based on Prometheus [38] and Grafana [39] for collecting relevant metrics;

Moreover, the INFN Cloud Service Portfolio already contains several examples of experiment-specific services. The more advanced ones are developed for the CYGNO [48] experiment, studying Dark Matter and Neutrinos, and for ML_INFN [49], a INFN-funded project aiming at lowering the potential barriers for accessing specialized hardware for the exploitation of Machine Learning techniques. Finally, as

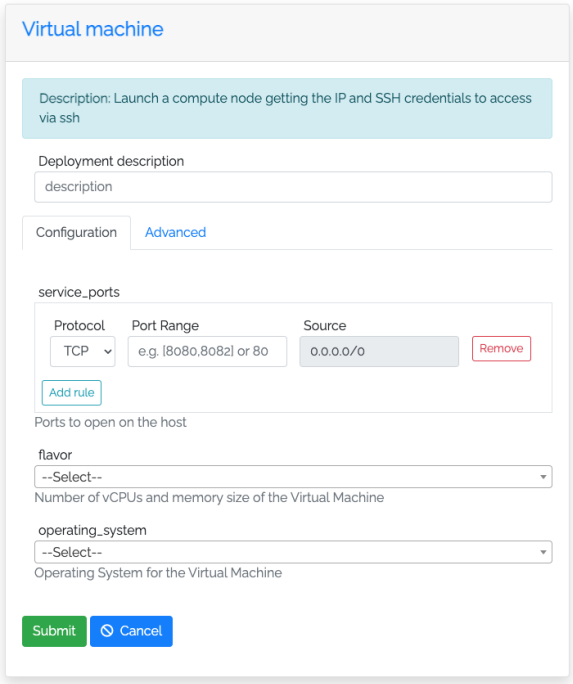

Fig. 3. Service Configuration form

anticipated above, INFN Cloud has developed a set of services built on top of k8s. A few examples in this category include: HTCondor on-demand clusters; Spark clusters, integrated with Jupyter; a centrally managed multi-user JupyterHub offered as a service.

VI. Moving towards a multi-level dashboard

In order to further enhance the flexibility to deploy and integrate applications, as well as to reduce the operational costs for users to deploy and maintain the services, we are moving a step forward by developing the support for a multi-level dashboard. The latter should be seen as a nested dashboard with respect to the one shown in Figure 2.

The idea behind the multi-level dashboard is to extend the concept of service composition to the very last layer of the infrastructure stack. The main objectives of a multi-level dashboard are to allow to manage (i.e. upgrade) and compose services with no need to re-define and re-instantiate the underlying virtual hardware and middleware when they can be provided by some other well-established methods, and to share resources on an existing cluster across different applications at different points in time. In addition, this should permit a faster turnaround in the integration and development of new services. From another perspective, we expect this approach to be key to implement a model where system administration responsibilities can be delegated to a sub-group of computing expert users, to whom access to the main dashboard is granted. The main dashboard would then be used by these users to deploy the actual base services, while researchers and data analysts would be provided with the inner dashboard, that

```
inputs:

  num_cpus:
    type: integer
    description: Number of virtual cpus for the VM
    default: 1

  mem_size:
    type: scalar-unit.size
    description: Amount of memory for the VM
    default: 2 GB

  os_distribution:
    type: string
    default: ubuntu
    description: Operating System distro
    constraints:
       - valid_values: [ "ubuntu", "centos" ]

  os_version:
    type: version
    default: 20.04
    description: Operating System distribution version
    constraints:
       - valid_values: [ 18.04, 20.04, 7 ]

  service_ports:
    type: map
    required: false
    default: { "ssh": { "protocol": "tcp", "source": 22 } }
    constraints:
      - min_length: 0
    entry_schema:
      type: tosca.datatypes.network.PortSpec
    description: Ports to open on the host
```

Fig. 4. TOSCA inputs for configuring a virtual host in cloud. These fields are automatically rendered in the HTML form shown in Figure 3

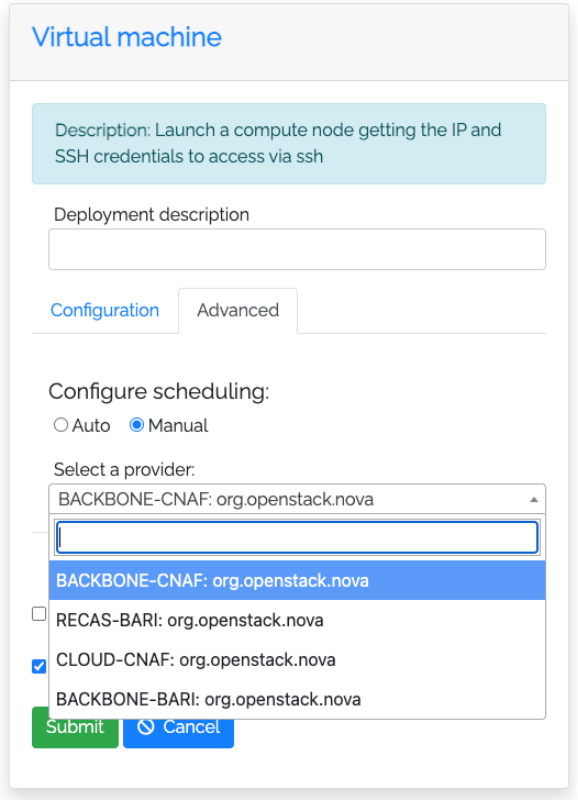

Fig. 5. Advanced configuration options

```
policies:
  - deploy_on_specific_site:
      type: tosca.policies.indigo.SlaPlacement
      properties:
        sla_id: 7868ea9d-16d1-4aa7-81ca-613fde477a08
```

Fig. 6. Example of SLA placement policy

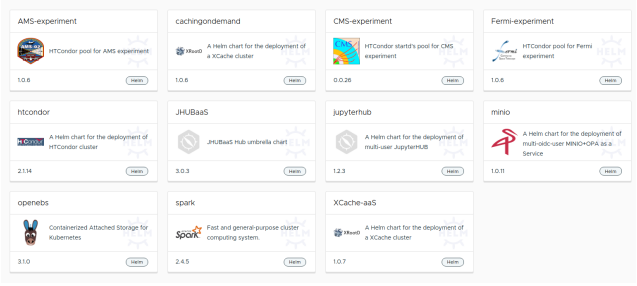

Fig. 7. Kubeapps custom dashboard

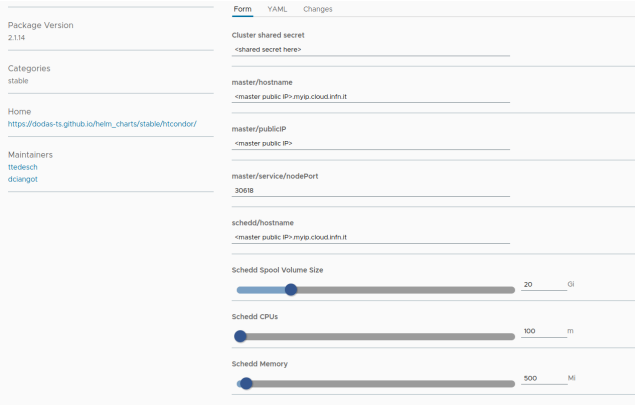

Fig. 8. Kubeapp service configuration example

does not require system admin responsibility at all. Access to the inner dashboard can then be limited in scope, i.e. granting access only to a specific service, application or set of applications/services. In particular, the inner dashboard will not provide any system admin capabilities, but only allows to manage applications. Early prototypes of this multi-level dashboard are already available, based on the adoption of the *Kubeapps* [50] technology. Adopting Helm as a standard for our application packaging allowed us to leverage Kubeapps to instantiate such a inner web UI, as shown in Figure 7. In this framework, users can log in as usual via OIDC authentication, and compile Helm values via simple forms, before deploying the application with a simple click; see Figure 8. As a side note, thanks to the modular approach described above, any Helm-based service can be deployed, reusing the very same YAML files both via the main dashboard (through a dedicated TOSCA template) and directly within the inner dashboard (the one using Kubeapps).

VII. Conclusions and Future Work

The paper describes an open-source scientific gateway built on top of the INDIGO PaaS orchestration system. This gateway allows users to easily access federated distributed compute and storage resources, on top of which both simple and complex scientific virtual computational environments can be automatically deployed and configured. Besides describing the portal and the PaaS orchestration framework, an efficient and sustainable strategy for developing integrated high-level services is also described. A production-ready implementation of the gateway is shown in the context of the INFN Cloud project. It supports a dynamic and extensible portfolio of services including both general purpose services and applications tailored to specific scientific experiments and collaborations. Our vision on how to simplify access and management of federated cloud resources and services is detailed. Moving forward, we can optionally identify two classes of users: those who are responsible for deploying the services and who take care of their operations and maintenance (expert users); and those who access the services that have been deployed by someone else. Technically, this distinction is reflected in the approach taken with the design of our multi-level dashboard, which is currently in the prototype stage. A further and final consideration is that we observe a constant increase in the number of users that are adopting the INFN Cloud modular solutions. This means that providing service templates that are frequently updated and include security patches is of crucial importance. To this end, we are implementing an automatic testing system based on Jenkins that will allow to replace the current semi-manual template testing with pre-defined, fully automated job pipelines. These pipelines perform automatic checks for each of the services available in the catalogue, including scans related to security vulnerabilities. A comment is in order. Due to the large adoption of Machine Learning techniques (ML), one could think about the implementation of such procedures in our context. This would have a twofold advantage: on one hand, the adoption of ML algorithms could offer further benefits in testing identified patterns on the new templates to help predict and avoid future failures; on the other hand, due to the increasing number of users, it would help to address future issues in terms of the maintenance burden. We expect that this will further enhance the overall usability, sustainability and scalability of the approach adopted by INFN Cloud.

References

[1] INDIGO-DataCloud Project. URL https://www.indigo- datacloud.eu/
[2] EOSC-hub Project. URL https://www.eosc-hub.eu
[3] DEEP Hybrid DataCloud Project. URL https://deep-hybrid-datacloud.eu
[4] eXtreme-DataCloud (XDC) Project. URL http://www.extreme-datacloud.eu
[5] ESCAPE Project. URL https://projectescape.eu
[6] A. Chierici, B. Martelli et al; SGSI project at CNAF. EPJ Web of Conferences 214, 08017 (2019) https://doi.org/10.1051/epjconf/201921408017
[7] INFN Cloud - Cloud resources for research. URL https://www.cloud.infn.it/
[8] Salomoni, D., Campos, I., Gaido, L. *et al*. INDIGO-DataCloud: a Platform to Facilitate Seamless Access to E-Infrastructures. *J Grid Computing* 16, 381–408 (2018). https://doi.org/10.1007/s10723-018-9453-3
[9] Kacsuk, P., Farkas, Z., Kozlovszky, M. *et al.* WS-PGRADE/gUSE Generic DCI Gateway Framework for a Large Variety of User Communities. *J Grid Computing* 10, 601–630 (2012). https://doi.org/10.1007/s10723-012-9240-5
[10] The sci-bus project: http://www.sci-bus.eu/
[11] CloudBroker Platform: http://cloudbroker.com/platform/
[12] Massimiliano Assante, Leonardo Candela, Donatella Castelli, Roberto Cirillo, Gianpaolo Coro, Luca Frosini, Lucio Lelii, Francesco Mangiacrapa, Valentina Marioli, Pasquale Pagano, Giancarlo Panichi, Costantino Perciante, Fabio Sinibaldi, The gCube system: Delivering Virtual Research Environments as-a-Service, *Future Generation Computer Systems*, Volume 95, 2019, Pages 445-453, ISSN 0167-739X, https://doi.org/10.1016/j.future.2018.10.035
[13] D4Science Infrastructure: https://www.d4science.org/
[14] Liferay: Digital Experience Software Tailored to Your Needs. https://www.liferay.com/
[15] JSR 286: Portlet Specification 2.0. https://jcp.org/en/jsr/detail?id=286
[16] OSGi, The Dynamic Module System for Java. https://www.osgi.org/resources/what-is-osgi/
[17] Writing Your First Liferay Application. https://help.liferay.com/hc/en-us/articles/360018176951-Writing-Your-First-Liferay-Application
[18] Apache jclouds: https://jclouds.apache.org/
[19] INDIGO-DataCloud PaaS Orchestrator. Documentation: https://indigo-dc.gitbook.io/indigo-paas-orchestrator/. Github: https://github.com/indigo-dc/orchestrator
[20] Andrea Ceccanti, Enrico Vianello, & Marco Caberletti. (2018, May 18). INDIGO Identity and Access Management (IAM) (Version v1.4.0). Zenodo. http://doi.org/10.5281/zenodo.1874791