J Grid Computing (2018) 16:381–408
https://doi.org/10.1007/s10723-018-9453-3

INDIGO-DataCloud: a Platform to Facilitate Seamless Access to E-Infrastructures

D. Salomoni · I. Campos · L. Gaido · J. Marco de Lucas · P. Solagna · J. Gomes · L. Matyska · P. Fuhrman · M. Hardt · G. Donvito · L. Dutka · M. Plociennik · R. Barbera · I. Blanquer · A. Ceccanti · E. Cetinic · M. David · C. Duma · A. López-García · G. Moltó · P. Orviz · Z. Sustr · M. Viljoen · F. Aguilar · L. Alves · M. Antonacci · L. A. Antonelli · S. Bagnasco · A. M. J. J. Bonvin · R. Bruno · Y. Chen · A. Costa · D. Davidovic · B. Ertl · M. Fargetta · S. Fiore · S. Gallozzi · Z. Kurkcuoglu · L. Lloret · J. Martins · A. Nuzzo · P. Nassisi · C. Palazzo · J. Pina · E. Sciacca · D. Spiga · M. Tangaro · M. Urbaniak · S. Vallero · B. Wegh · V. Zaccolo · F. Zambelli · T. Zok

Received: 25 November 2017 / Accepted: 19 July 2018 / Published online: 7 August 2018

Abstract This paper describes the achievements of the H2020 project INDIGO-DataCloud. The project has provided e-infrastructures with tools, applications and cloud framework enhancements to manage the demanding requirements of scientific communities, either locally or through enhanced interfaces. The middleware developed allows to federate hybrid resources, to easily write, port and run scientific applications to the cloud. In particular, we have extended existing PaaS (Platform as a Service) solutions, allowing public and private e-infrastructures, including those provided by EGI, EUDAT, and Helix Nebula, to integrate their existing services and make them available through AAI services compliant with GEANT interfederation policies, thus guaranteeing transparency and trust in the provisioning of such services. Our middleware facilitates the execution of applications using containers on Cloud and Grid based infrastructures, as well as on HPC clusters. Our developments are freely downloadable as open source components, and are already being integrated into many scientific applications.

Keywords Cloud computing · Platform as a service · Containers · Software management · Advanced user interfaces · Authorization and authentication

D. Salomoni · A. Ceccanti · C. Duma
INFN - CNAF, Bologna, Italy

I. Campos (✉) · J. Marco de Lucas · A. López-García · P. Orviz · F. Aguilar · L. Lloret
IFCA, Consejo Superior de Investigaciones Cientificas-CSIC, Santander, Spain
e-mail: isabel.campos@csic.es

L. Gaido · S. Bagnasco · S. Vallero · V. Zaccolo
INFN - Torino, Torino, Italy

G. Donvito · M. Antonacci
INFN - Bari, Bari, Italy

P. Fuhrman
Deutsches Elektronen Synchrotron (DESY), Hamburg, Germany

I. Blanquer · G. Moltó
Institute of Instrumentation for Molecular Imaging - Universitat Politècnica de València, Valencia, Spain

M. Plociennik · M. Urbaniak · T. Zok
PSNC IBCh PAS, Poznań, Poland

M. Hardt · B. Ertl · B. Wegh
Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany

1 Introduction

INDIGO-DataCloud was an European project starting in April 2015, with the purpose of developing a modular architecture and software components to improve how scientific work is supported at the edge of computing services development. Its main goal has been to deliver a Cloud platform addressing the specific needs of scientists in a wide spectrum of disciplines, engaging public institutions and private companies. It aimed at being as inclusive as possible, developing open source software exploiting existing solutions, adopting and enhancing state of the art technologies, connecting with other initiatives and with leading commercial providers.

J. Gomes · M. David · L. Alves · J. Martins · J. Pina
Laboratory of Instrumentation and Experimental Particle Physics (LIP), Lisbon, Portugal

S. Fiore · A. Nuzzo · P. Nassisi · C. Palazzo
Fondazione Centro Euro-Mediterraneo sui Cambiamenti Climatici, Lecce, Italy

R. Barbera · R. Bruno · M. Fargetta
INFN - Catania, Catania, Italy

D. Spiga
INFN - Perugia, Perugia, Italy

L. Dutka
Cyfronet AGH, Krakow, Poland

P. Solagna · M. Viljoen · Y. Chen
EGI Foundation, Amsterdam, Netherlands

L. Matyska · Z. Sustr
CESNET, Prague, Czech Republic

A. M. J. J. Bonvin · Z. Kurkcuoglu
University of Utrecht, Utrecht, The Netherlands

L. A. Antonelli · A. Costa · S. Gallozzi · E. Sciacca
Istituto Nazionale di Astrofisica, Rome, Italy

E. Cetinic · D. Davidovic
Ruder Boskovic Institute, Zagreb, Croatia

M. Tangaro · F. Zambelli
Consiglio Nazionale delle Ricerche, Istituto di Biomembrane, Bioenergetica e Biotecnologie Molecolari, Bari, Italy

F. Zambelli
Department of Biosciences, University of Milano, Milan, Italy

R. Barbera
Department of Physics and Astronomy, University of Catania, Catania, Italy

Since its inception, the project roadmap has been user community driven. Its main focus was on closing the existing technology gaps that hindered an optimal exploitation of Cloud technologies by scientific users. In order to do so, user requirements from several multidisciplinary scientific communities were collected, and systematized into specific technical requirements. This process was carried out across the entire lifetime of the project, which allowed the update of existing requirements as well as the insertion of new ones, thus driving the project architecture definition and the technological developments.

The project also made focus on delivering production-quality software, thus it defined procedures and quality metrics, which were followed by, and automatically checked for, all the INDIGO components. A comprehensive process to package and issue the INDIGO software was also defined. As an outcome of this, INDIGO delivered two main software releases (the first in August 2016, the second in April 2017), each followed by several minor updates. The latest release consists of about 40 open modular components, 50 Docker containers, 170 software packages, all supporting up-to-date open operating systems. This result was accomplished reusing and extending open source software and —whenever applicable— contributing code to upstream projects.

The paper is structured as follows. Section 2 contains a description on how the user requirements were collected and consolidated. From there, the INDIGO architecture is further elaborated from the lower Infrastructure as a Service layer (Section 3) moving towards the Platform layer (Section 4) in order to arrive to the user interfaces (Section 5). The overall software development process is described in Section 6. Section 7 contains a summary of some usage patterns on how to leverage the INDIGO solutions to develop, deploy and support applications in a Cloud framework. The conclusions are laid out in Section 8. The list of upstream contributed software can be found in the Appendix.

1.1 Context and State of the Art

From the collection of user community requests, and its consolidation into technical requirements (see Section 2), we identified a number of technology gaps that today hinder an optimal scientific exploitation of heterogeneous e-infrastructures.

In this Section we will elaborate more on the general strategy to address those requirements, linking our developments with the previous and existing works. The specific enhancements and developments will be further elaborated in the corresponding sections.

Lack of proper federated identity support across several e-Infrastructures is a key issue for the researchers perspective. The provision of an effective distributed authentication and authorization in heterogeneous platforms is fundamental to support access to distributed infrastructures. Several efforts have been made in this context [1–5] but they were focused on specific infrastructures and services. However, although some of these approaches have been used in production in specific e-Infrastructures [6] they are difficult to implement in a broader environment.

In parallel to the development of INDIGO-Datacloud, the *Authentication and Authorisation for Research and Collaboration* project (AARC) defined the AARC Blueprint Architecture [7]. This document describes a set of interoperable architecture building blocks for designing and implementing access management solutions for international research collaborations. Following the AARC recommendations we have developed several key components related with identity and access management, providing a framework compliant with the proposed blueprint architecture, as will be described in Section 3.3.

Facilitating the transparent execution of user applications across different computing infrastructures is also a key issue [8]. Advanced users have nowadays at their disposal tools to implement applications in Clouds provisioned in *Infrastructure as a Service* (IaaS) mode. Examples of such solutions are virtual appliances and contextualization [9] or container technologies [10]).

The situation for non-Cloud resources in scientific facilities is completely different. Here we are referring for instance to local clusters, Grid infrastructures and HPC systems. Such infrastructures are tipically shared among many users with different requirements, therefore it is managerially and technically impossible offering tailored environments to all of them. As a consequence scientific users often need to follow a troublesome process to package and execute their applications.

To address this problem we have applied the technology of Docker containers [11] to facilitate applications execution in multiuser environments. As a result we have provided a flexible user-level solution to give autonomy to users in shared computing facilities [12]. Section 3.1 contains a thorough discussion on the strategy and outcomes.

Adoption of true Platform as a Service (PaaS) Cloud solutions is a common problem for scientific communities. The roots of this problem are on the one hand the non-interoperability of the interfaces [13, 14], and second, the lack of true orchestration mechanisms across federated heterogeneous infrastructures. Both barriers made it difficult for users to adopt Cloud hybrid solutions.

In Section 4 we describe our approach, and how we have tackled this problem by leveraging the OCCI [15–17] and TOSCA [18] open standards. In this regard we have not only supported those standards at the corresponding architectural levels [19, 20], but also we made important contributions to both the standards specifications and implementations. INDIGO has contributed to the networking parts of the OCCI standard, as well as to the improvement of the TOSCA support in the upstream OpenStack components: the Heat Translator and TOSCA parser [21]. Our solution makes the execution of dynamic workflows [22–24] possible, in a more consistent way across hybrid Clouds [25].

In this interoperability context, hybrid Cloud deployments, although possible [14, 26, 27], were complicated from a practical point of view and therefore user adoption has been hindered. By adopting INDIGO solutions users can now express their requirements and deploy them as applications over those hybrid infrastructures [28].

Linked with the previous statements another outstanding gap was the lack of advanced scheduling features in Cloud environments [29]. Common cloud usage scenarios, being industry driven, do not take into account the unique requirements of scientific applications [30], leading to an inefficient utilization of the resources or to non optimal user experience.

Developments in this area can be found in the literature [31–34], where it becomes evident that there are many challenges to be addressed. Within INDIGO we focused (see Section 3.2) in the efficient sharing of resources among users following fair-share approaches (limiting the amount of resources that can be consumed by a user group), proper quota partitioning across different computing frameworks (like HPC and Cloud resources) or new Cloud computing execution models (like preemptible instances) as these are aspects that affect both users and resource providers.

Regarding storage support, INDIGO has performed substantial contributions to storage-related entities and standardization bodies, such as the Research Data Alliance (RDA), where INDIGO has been highly involved the Quality of Service, Data Life cycle and Data Management Plans working group (now renamed to Storage Service Definitions). Moreover, INDIGO has also contributed this work to the SNIA CDMI standard, providing several extensions that have been included in SNIA reference implementations and documents.

2 Analysis of Requirements Coming from Research Communities

In order to guide our developments we performed an analysis of a number of use cases originating in several flagship research communities. In particular coming from the areas of High Energy Physics, Environmental modelling, Bioinformatics, Astrophysics and Social sciences. See Table 1 for the full list.

The deployment of customized computing back-ends, such as batch queues, including automatic elasticity is among the features more demanded by researchers. The automation of the deployment of user-specific software in VMs or containers is also on the top of their wish list. Such automation is a *must* when it is about simplifying the executing applications in heteregeneous infrastructures. For similar reasons, highly specialized applications require also support to hardware accelerators and specialized hardware such as Infiniband, multicore systems, or GP/GPUs.

Often user communities are asking for terminal access to resources, workflow management and data handling, in a way that such access is linked

Table 1 Research Communities and use cases analyzed to extract general requirements

Research community	Application/Use case
LIFEWATCH (Biodiversity)	Monitoring and Modelling Algae Bloom in a Water Reservoir: Support of hydrodynamic and water quality modelling including data input-output management and visualization.
INSTRUCT (Bioinformatics)	Molecular dynamics simulations: Support of Molecular dynamics simulations of macromolecules that need specific hardware (GP/GPUs) using a pipeline of software that combines protocols that automate the step for setup and execution of these simulations.
CTA (Astronomical data)	Astronomical Data Archives: Data analysis and management using different tools such as data discovery, comparison, cross matching, data mining and also workflows. The use case could be described as follows: data production, data reduction, data quality, data handling and workflows, data publication and data link to articles
Climate modelling	Intermodel comparison of data analysis for different climate models using the ENES platform (European Network for Earth System modelling)
EuroBioImaging (Bioinformatics)	Medical Imaging Biobanks: The virtual Biobank integrates medical images from different sources and formats. This case study includes all the steps needed to manage the images, like analysis, storage, processing (pre, post). Privacy is a constraint to take into account for user management
ELIXIR (Bioinformatics)	Galaxy as a Cloud service: Deployment of Galaxy instance that should support all the software/steps needed by the pipeline over, for example, a virtual cluster or cloud instances.
DARIAH (Social sciences)	Transparent access to data catalogues and on-demand data management features.
Mastercode (HEP pheno)	Complex combination of codes including legacy parts to perform combined analysis of data coming from particle detectors, astrophysics experiments, and dark matter detectors. Installation of these codes is in general very complex in multi-user farms. Providing a container based solution would simplify installation across infrastructures.
Lattice QCD (HEP)	Lattice simulations run on large HPC facilities using low latency interconnects, producing large amounts of output. Accessing such facilities in Cloud mode would require implementing MPI parallel processing capabilities.

to a common Authorization and Authentication Infrastructure.

In order to generalize the requirements, we have extracted two generic usage scenarios, which can support a wide range of applications in these areas. The first generic use case is computing oriented, while the second is data analysis oriented. For full details regarding user communities description and detailed usage patterns we refer to the users requirements deliverable of the project available publicly in [35].

2.1 Computing Portal Service

The first generic user scenario is a computing portal service. In such scenario, computing applications are stored by the application developers in repositories as downloadable images (in the form of VMs or containers). Such images can be accessed by users via a portal, and require a back-end for execution; in the most common situation this is typically a batch queue. Support for parallel processing using containers is a requirement that comes up as well from the users.

The application consists of two main parts: the portal / Scientific Gateway and the processing working nodes. The number of nodes available for computing should increase (scale out) and decrease (scale in), according to the workload. The system should also be able to do Cloud-bursting to external infrastructures when the workload demands it. Furthermore, users should be able to access and reference data, and also to provide their local data for the runs. A solution along these lines is shown in Fig. 1.

A solution along these lines has been requested in the user scenarios coming from ELIXIR, WeNMR, INSTRUCT, DARIAH, Climate Change and LIFEWATCH.

2.2 Data Analysis Service

A second generic use case is described by scientific communities that have a coordinated set of data repositories and software services to access, process and inspect them.

Processing is typically interactive, requiring access to a console deployed on the data premises. The application consists of a console / Scientific Gateway that interacts with the data. In Fig. 2 we show a schematic view of such a use case. Examples of such include *R*, *Python* or *Ophidia*. It can be a complementary scenario to the previous one, and it could also expose programmatic services.

The communities related to INSTRUCT, CTA, Climate Change, LIFEWATCH and Lattice QCD have requested related features.

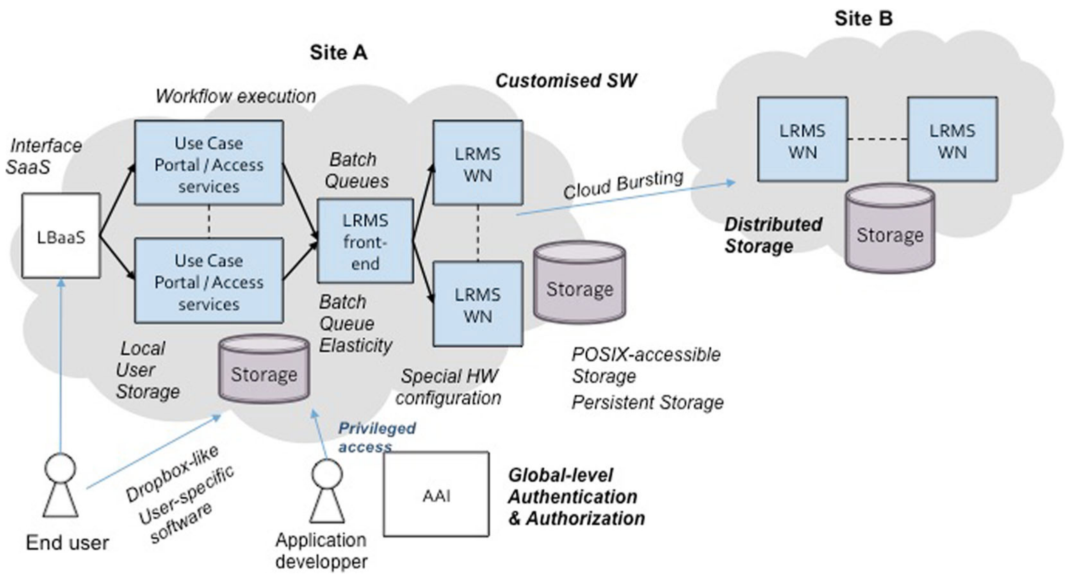


Fig. 1 User community computing portal service

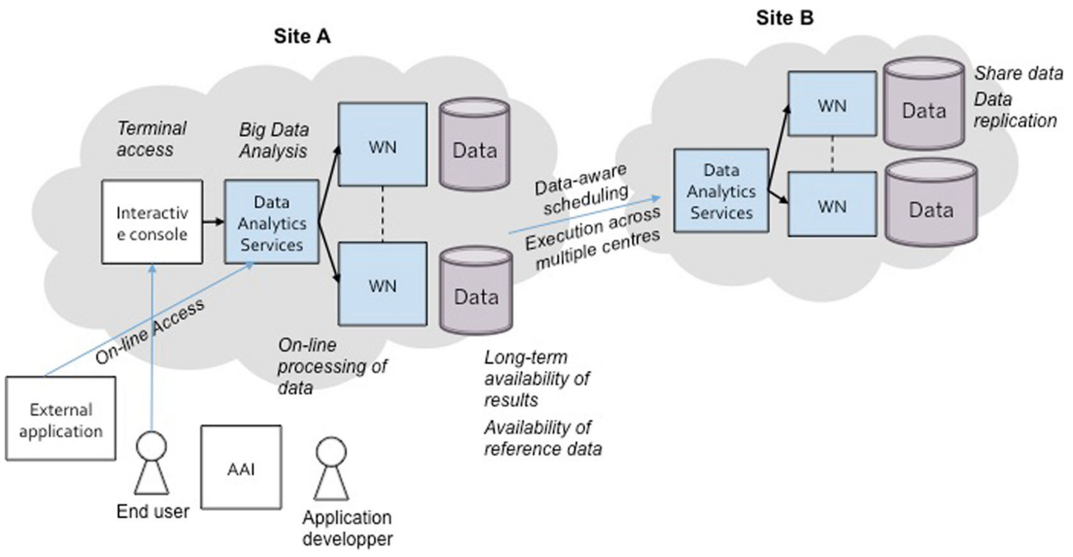


Fig. 2 Data analysis service

3 Developing for the Infrastructure as a Service (IaaS) Layer

INDIGO-DataCloud has provided e-infrastructures with tools, applications and cloud framework enhancements to manage the demanding requirements of modern scientific communities, either locally or through enhanced interfaces enabling those infrastructures to become part of a federation or to connect to federated platform layers (PaaS).

In this section we will describe the highlights of this development work, which was needed to properly interface with the resource centers. This work has focussed on virtualizing local compute, storage and networking resources (IaaS) and on providing those resources in a standardized, reliable and performing way to remote customers or to higher level federated services, building virtualized site independent platforms.

The IaaS resources are provided by large resource centers, typically engaged in well-established European e-infrastructures. The e-infrastructure management bodies, or the resource centers themselves will select the components they operate, and INDIGO will have limited influence on that process.

Therefore, INDIGO has concentrated on a selection of the most prominent existing components and has further developed the appropriate interfaces to high-level services based on standards. We have also developed new components where we felt a full functionality was completely missing.

The contribution of INDIGO to enhance the flexibility to access resources in Cloud and HPC infrastructures will be of paramount importance to enable transparent execution of applications across systems promoting the development of the future European Open Science Cloud (EOSC) ecosystem [36].

As we describe below new components are provided, or already existing components are improved in the areas of computing, storage, networking and Authorization and Authentication Infrastructure (AAI). For almost all components, we succeeded in committing modifications to the corresponding upstream software providers and by that, significantly contributed to the sustainability model of the software.

3.1 Supporting Linux containers

It is unquestionable that Docker is the most widely adopted Linux container technology. Therefore, in order to facilitate application delivery across multiple computational platforms, INDIGO has provided support for container execution, both interactively and through batch systems, in cloud and conventional clusters. This was achieved by developing new tools and extending existing ones.

The key middleware developed for this purpose is **udocker** [12].[1] The udocker novelty consists in enabling to pull and execute Docker containers [11] without using or requiring the installation of the Docker software. By using udocker it is possible to encapsulate applications in Docker containers and execute them in batch or interactive systems where Docker is unavailable. It provides several different execution engines based on PRoot [37], runC [38] and Fakechroot [39]. None of the udocker engines requires root privileges for installation or execution being therefore adequate for deployment and use by end-users without system administrator intervention. In addition, the PRoot and Fakechroot engines execute containers via pathname translation and therefore do not require the use of Linux namespaces.

Since udocker never requires privileges and executes as unprivileged user many of the security concerns associated with the Docker software are avoided. Udocker also supports GPGPU and MPI applications, making it adequate to execute containers in batch systems and HPC clusters. The udocker software suite is meant to be easily deployed by end-users. It only requires the download and execution of a Python script to quickly setup udocker within the user home directory. Udocker empowers end-users to execute Docker containers regardless of the Linux host environment.

Since its first release in June 2016 udocker expanded quickly in the open source community. It has been adopted by a number of projects as a drop-in replacement for Docker. Among them openmole, bioconda, common-work language (cwl) or SCAR - Serverless Container-aware ARchitectures [40].

As an example, udocker is being used with great success to execute code produced by the MasterCode collaboration [41]. The MasterCode collaboration is concerned with the investigation of Supersymmetric models that go beyond the current status of the Standard Model of particle physics. It involves teams from CERN, DESY, Fermilab, SLAC, CSIC, INFN, NIKHEF, Imperial College London, King's College London, the Universities of Amsterdam, Antwerpen, Bristol, Minnesota and ETH Zurich. Examples and documentation can be found at https://github.com/indigo-dc/udocker.

[1]https://github.com/indigo-dc/udocker

INDIGO has also developed **bdocker**,[2] which provides a front-end to execute the Docker software in batch systems under restrictions and limits configurable by the system administrator (resource consumption, access to host directories, list of container images, etc). It has been implemented for the SoGE (Son of Grid Engine) batch system but can be extended to other batch systems. This integrates the benefits of application delivery provided by Docker with the scheduling policies of the batch system. While udocker can be deployed directly by the end-user, bdocker is installed and configured by the system administrator to provide the users with the ability to run limited execution environments provided by Docker. Finally, ONEDock[3] was developed to introduce support to the execution of containers as if they were Virtual Machines in OpenNebula-based on-premises Clouds, by supporting Docker as a hypervisor in this Cloud Management Platform. With this approach, applications and their execution environment packaged as Docker images can be instantiated on-demand through OpenNebula and provide SSH-based access for multiple users, with the advantage for the administrators of reduced overhead (such as low memory footprint) when compared to Virtual Machines.

3.2 Development of Advanced Scheduling Technologies

The end goal of this activity in INDIGO is improving the performance of the cloud management platforms by designing and implementing novel scheduling mechanisms and policies at the IaaS level. Enabling advanced scheduling policies to optimize the usage of the data center will clearly improve the response to the users.

To this end cloud schedulers need to include support for postponing low priority workloads (by killing, preempting or stopping running containers or VMs) in order to allocate higher priority requests.

3.2.1 Preemptible Instances

INDIGO pushed the state of the art in scheduling technologies by implementing preemptible instances on

[2]https://github.com/indigo-dc/bdocker

[3]https://github.com/indigo-dc/onedock

top of the OpenStack[42] cloud management framework, **opie**.[4] Openstack preemptible instances is the materialisation of the preemptible instances extension, serving as a reference implementation.

Preemptible instances differ from regular ones in that they are subject to be terminated by a incoming request for provision of a normal instance. If bidding is in place, this special type of instance could also be stopped by a higher priority preemptible instance (higher bid). Not all the applications are suitable for preemptible execution, only fault-tolerant ones can withstand this type of execution. On the other side they are highly affordable VMs that allow providers to optimize the usage of their available computing resources (i.e. backfilling).

The opie package provides a set of pluggable extensions for OpenStack Compute (nova) making possible to execute preemptible instances using a modified filter scheduler. This solution has gained great interest from the scientific community and commercial partners, and is under discussions to be introduced in the upstream OpenStack scheduler.

3.2.2 Implementing Advanced Scheduling in OpenStack and OpenNebula

In IaaS private clouds the computing and storage resources are statically partitioned among projects. A user typically is member of one project, and each project has its own fixed quota of resources defined by the cloud administrator. A user request is rejected if the project quota has already been reached, even if unused resources allocated to other projects would be available.

This rigid resource allocation model strongly limits the global efficiency of the data centres, which aim to fully utilize their resources for optimizing costs. In the traditional computing clusters the utilization efficiency is maximized through the use of a batch system with sophisticated scheduling algorithms plugged in. However this feature is missing in the most popular cloud middlewares. In the course of INDIGO we have developed support for advanced scheduling policies such as intelligent job allocation based on fair-share algorithms for both OpenStack and OpenNebula cloud frameworks.

[4]https://github.com/indigo-dc/opie

- **Synergy**[5] is an advanced service interoperable with the OpenStack components, which implements a new resource provisioning model based on pluggable scheduling algorithms. It allows to maximize the resource usage, at the same time guaranteeing a fair distribution of resources among users and groups.

  The service also provides a persistent queuing mechanism for handling those user requests exceeding the current overall resource capacity. These requests are processed according to a priority defined by the scheduling algorithm, when the required resources become available.
- The scheduling capabilities of OpenNebula have been enhanced with the development of **one-FaSS** (FairShare Scheduler for OpenNebula).[6] In OpenNebula the scheduler is first-in-first-out (FIFO). One-FaSS grants fair access to dynamic resources priorizing tasks assigned according to an initial weight and to the historical resource usage.

The project has also developed tools to facilitate the management of hybrid data centers, this is, where both batch system based and cloud based services are provided. Physical computing resources can play both roles in a mutual exclusive way. The **Partition Director**[7] takes care of commuting the role of one or more physical machines from *Worker Node* (member of the batch system cluster) to *Compute Node* (member of a cloud instance) and vice versa.

The current release only works with the IBM platform LSF Batch system (version 7.0x or higher) and Openstack Cloud manager instances (Kilo or newer). The main functionalities are switch role of selected physical machines from the LSF cluster to the Openstack one and viceversa, and manage intermediate transition status to ensure consistency.

3.3 Development of Authorization and Authentication Infrastructures

INDIGO has provided the necessary components to offer a commonly agreed Authentication and Authorization Infrastructure (AAI). The INDIGO **IAM**[8]

[5]https://github.com/indigo-dc/synergy-service

[6]https://github.com/indigo-dc/one-fass

[7]https://github.com/indigo-dc/dynpart

[8]https://github.com/indigo-dc/iam

(Identity and Access Management service) provides user identity and policy information to services so that consistent authorization decisions can be enforced across distributed services.

IAM has a big impact on the end-user experience. It provides a layer where identities, enrollment, group membership and other attributes and authorization policies on distributed resources can be managed in a homogeneous way, supporting the federated authentication mechanisms supported by the INDIGO AAI.

The INDIGO AAI solution pioneers the usage of OpenID Connect (OIDC) on the SP-IdP proxy. INDIGO has made contributions to the upstream components whenever needed to enable OpenID Connect (namely in OpenStack Keystone and Apache Libcloud).

INDIGO-Datacloud provides a flexible Authentication and Authorization Infrastructure (AAI) whose main components are depicted in Fig. 3.

In order to do authentication and authorization in a consistent way, services rely on the information provided by the central IAM service.

The Login Service component implements brokered authentication: users can authenticate with any of the supported mechanisms (SAML, OpenID Connect, X.509 certificates, local username/password). Identity and authorization information is then exposed to services via standard OAuth/OpenID Connect protocols. This approach simplifies integration with off-the-shelf components and does not overload all services in the infrastructure with the complexity of handling multiple credential types. This approach has a big impact on end-user experience, as it allows users to centralize the management of their credentials and provide a consistent login experience to all services in the system.

The Group Membership Service component provides the tools to manage registration and enrollment flows for the collaboration, organize users into groups and manage user account life cycle.

The Authorization Service component, based on the Argus Authorization Service [43], provides the ability to define fine-grained policies for the collaboration leveraging the flexibility of a XACML policy engine [44]. This component also provides a policy composition and distribution mechanism that is used to ensure consistent authorization across the distributed infrastructure.

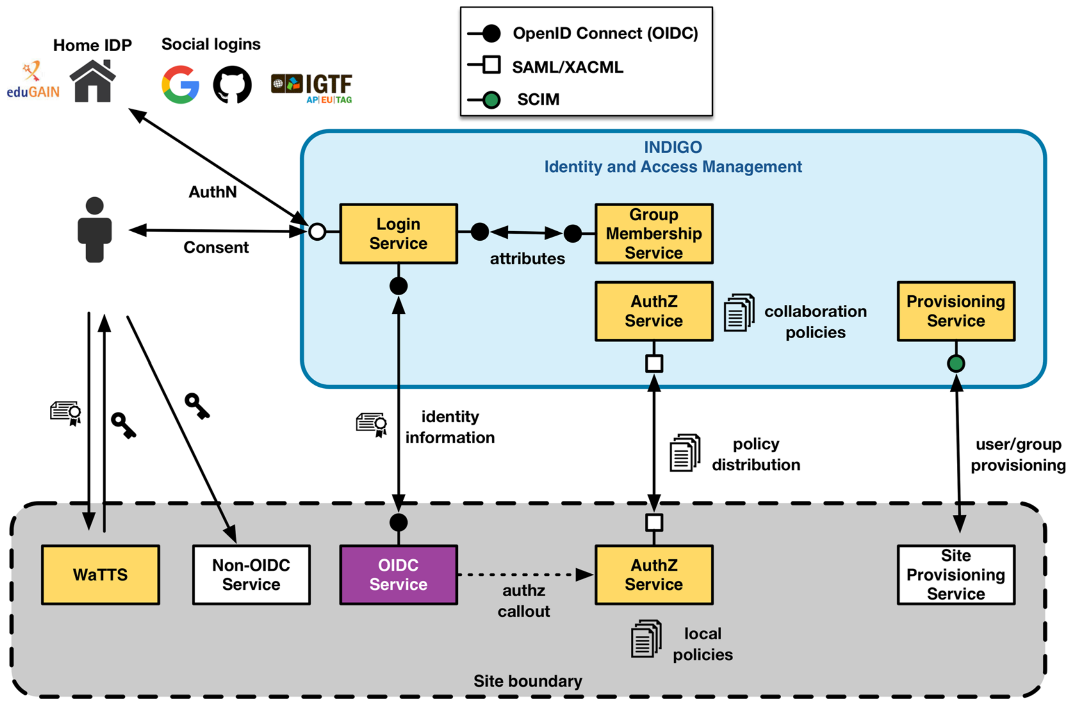


Fig. 3 Architecture of the INDIGO identity and access management service

The Provisioning Service component exposes an SCIM API [45] to provision information about the collaboration users to relying services. This mechanism is useful, for instance, to manage the lifecycle of resources at a site (e.g., local UNIX accounts) depending on the lifecycle of IAM user account information. As an example, accounts could be provisioned automatically across the infrastructure and configured with user SSH public keys as registered in the central IAM at user registration time, and disabled when the user membership at the IAM expires or is suspended due to a security incident.

Finally, to integrate services that do not speak OpenID Connect natively, IAM integrates with WaTTS, the INDIGO Token Translation Service.[9] WaTTS can translate identity and authorization information about a user provided as OpenID Connect tokens to various credential types. This allows the provision of services that do not normally support federated identities to federated users.

IAM is used internally to the PaaS layer in order to deal with the authorization of each user to the services, but also in handling group membership and role management for each user. Users may present themselves with any of the three supported token types (X.509, OpenID Connect, SAML) and are able to access resources that support any of them.

3.4 Virtual Networks

The INDIGO PaaS layer has been developed to exploit a wide range of cloud management frameworks (e.g. OpenStack, OpenNebula, Google Compute Engine, Microsoft Azure, Amazon EC2) and combine resources provided by these frameworks to enable the deployment of complex distributed applications. Each cloud management framework may exhibit a different native API and often these APIs can be configured in different ways. This heterogeneity constitutes a challenge when transparent instantiation of cloud resources across multiple frameworks is required. The INDIGO PaaS layer supports both common native APIs as well as the **Open Cloud Computing Interface**[10] (`OCCI`). The OCCI specification is a standard from the **Open Grid Forum**[11] that provides a flexible and extensible API to access and manage cloud resources. Although OCCI provides a convenient uniform API, its support to manage the network environment is limited in what concerns the setup of public/private network accessibility. Depending on the actual cloud management framework being used the target cloud may need manual network configuration prior to the use of OCCI. To address these problems INDIGO has defined and implemented an OCCI network extension that allows the network environment to be properly setup via the OCCI API regardless of the underlying cloud management framework.

For OpenNebula[46] sites the solution consists in a **Network Orchestrator Wrapper** (NOW)[12] and a corresponding backend in the **rOCCI-server**.[13] NOW enforces site-wide policy and network configuration by making sure that only LANs designated by site administrators are made available to users, and that users cannot reuse LANs assigned to others while they remain reserved. NOW has been released with INDIGO, and the backend has been provided as a contribution to upstream rOCCI-server distribution.

For OpenStack, the OpenStack OCCI Interface (OOI) has been extended with support for advanced networking functions provided by OpenStack's Neutron component such as router, network and subnet setup. The contribution was accepted upstream and is distributed with the OOI implementation.

The networking features of the OCCI gateway for the Amazon's EC2 API were adjusted making sure that the model of setting up and using local virtual networks is in accordance with the model used in the other cloud management frameworks.

In addition a Virtual Router was implemented allowing networks to span across cloud sites, potentially geographically distant, so that a custom networking environment can be setup even if the resources are allocated in different cloud sites. The virtual router is a virtual machine that can be started via OCCI and makes use of **OpenVPN**[14] to implement network tunnels. The Virtual Routers can be instantiated by the PaaS layer to orchestrate the interconnection of virtual machines across cloud providers.

[9]https://github.com/indigo-dc/tts

[10]http://occi-wg.org/

[11]https://www.ogf.org

[12]https://github.com/indigo-dc/now

[13]https://github.com/the-rocci-project/rOCCI-server

[14]https://openvpn.net

4 Architecture of the Platform as a Service

Generally speaking, a Platform as a Service (PaaS) is a software suite, which is able to receive programmatic resource requests from end users, and execute these requests provisioning the resources on some e-infrastructures. We can see already many examples in the industrial sector, in which open source PaaS solutions (eg. OpenShift [47] or Cloud Foundry [48]) are being deployed to support the work of companies in different sectors.

The case of supporting scientific users is more complex in general than supporting commercial activities, because of the heterogeneous nature of the infrastructures at the IaaS level (i.e. the resource centers) and of the inherent complexity of the scientific work requirements. The key point is to find the right agreement to unify interfaces between the PaaS and IaaS levels.

The **Infrastructure Manager** [49] (IM) has been used to address the IaaS level. The IM is able to deploy complex and customized virtual infrastructures on IaaS Cloud deployment(such as AWS, OpenStack, etc.). It automates the deployment, configuration, software installation, monitoring and update of the virtual infrastructure on multiple Cloud back-ends.

In the framework of INDIGO the IM has extended its capabilities. In particular, in the PaaS it is used by the Orchestrator (see below) in order to provision and configure the virtual infrastructure required to support the scientific applications involved in the project.[15]

In order to better adapt to the wide range of use cases provided by the users communities we decided to take a different approach from many of the more used PaaS: our solution is based on the concept of orchestrating complex clusters of services and on the possibility to automatize the actions needed to implement the use cases. This approach was really successful as it gave the possibility to implement also legacy applications and did not depended on the language in which the application is built.

In Figure 4 we show the general interaction between the IaaS and PaaS layers. The Orchestrator provides the entry point to the PaaS layer with its ability to decide the most appropriate site on which to deploy a certain application architecture described in TOSCA Templates. INDIGO-DataCloud fosters local-site orchestration and, therefore, depending on the underlying Cloud Management Framework of the Cloud site, the TOSCA Template is translated into the specific orchestration component of OpenStack (Heat) or it is delegated on the Infrastructure Manager (IM) to execute on OpenNebula-based Cloud sites. Since both Virtual Machines and containers can be provisioned on the underlying Cloud site, Virtual Machine Images available in each are registered in the Information System and container images are pre-staged to the Cloud sites to reduce deployment times.

INDIGO has provided a working PaaS Layer orchestrating heterogeneous computing and storage resources. Using the PaaS Orchestrator together with the IM and TOSCA Templates, the end users are able to exploit computational resources without knowledge about the IaaS details. In the following we describe the main technologies employed to build the PaaS.

4.1 PaaS Layer and Microservices Architecture

The Paas layer should be able to hide complexity and federate resources for both Computing and Storage. For that we have applied the current technologies based on lightweight containers and related virtualization developments using microservices.

Kubernetes [50], an open source platform to orchestrate and manage Docker containers, is used to coordinate the microservices in the PaaS layer. Kubernetes is extremely useful for the monitoring and scaling of services, and to ensure their reliability. The PaaS manages the needed micro-services using Kubernetes, in order, for example, to select the right end-point for the deployment of applications or services. The Kubernetes solution is used in the PaaS layer as is provided by the community.

The microservices that compose the PaaS layer are very heterogeneous in terms of development: some of them are developed ad hoc, some others were already available and used as they are, few others have been significantly modified in order to implement new features within INDIGO.

The language in which the INDIGO PaaS receives end user requests is TOSCA [18]. TOSCA stands for Topology and Orchestration Specification for Cloud Applications. It is an OASIS specification for the interoperable description of applications and infrastructure cloud services, the relationships between parts of these services, and their operational behaviour.

[15]https://github.com/indigo-dc/im

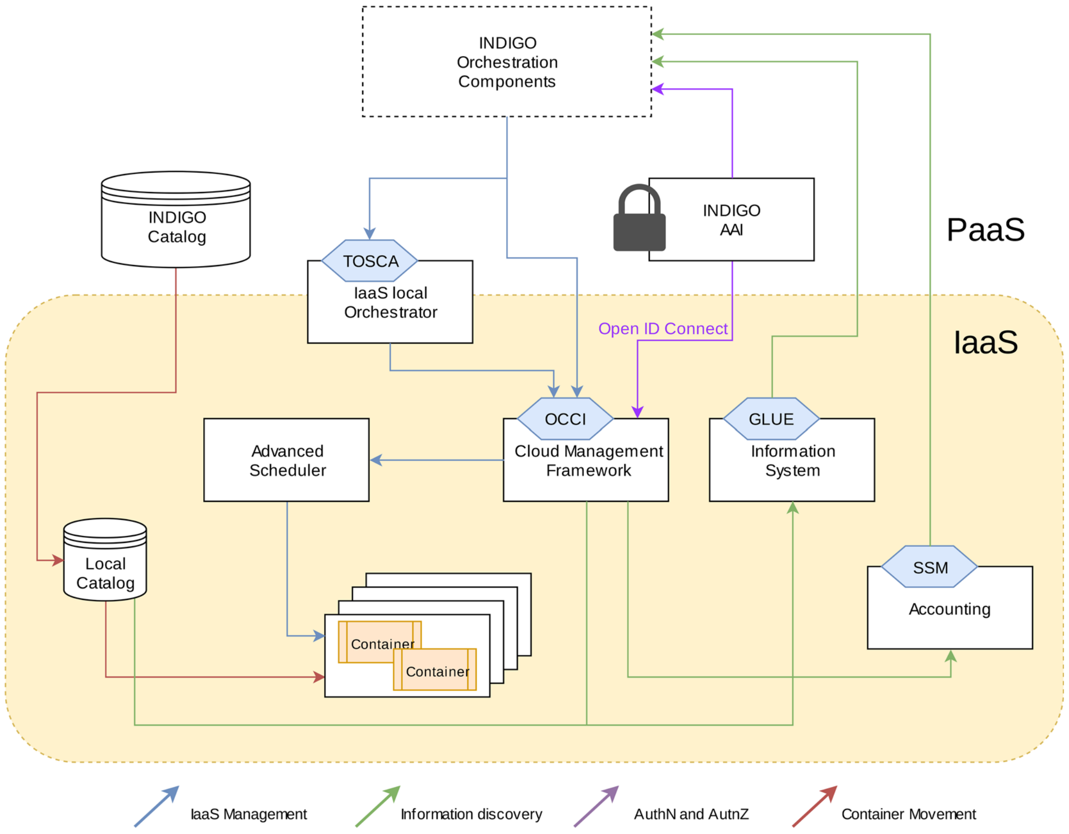


Fig. 4 Interaction between the IaaS and PaaS layers

TOSCA has been selected as the language for describing applications, due to the wide range of adoption of this standard, and since it can be used as the orchestration language for both OpenNebula (through the IM) and OpenStack (through Heat).

The released INDIGO PaaS layer (see Fig. 5) is able to provide automatic distribution of the applications and/or services over a hybrid and heterogeneous set of IaaS infrastructures, on both private and public clouds.

The PaaS layer is able to accept a description of a complex set, or cluster, of services/applications by mean of TOSCA templates, and is able to provide the needed brokering features in order to find the best fitting resources. During this process, the PaaS layer is also able to evaluate data distribution, so that the resources requested by the users are chosen by the closeness to the storage services hosting the data requested by those specific applications/services.

4.1.1 The Orchestrator Engine

The INDIGO PaaS **Orchestrator**[16] is a core component of the PaaS layer: it orchestrates the provisioning of virtualized compute and storage resources on Cloud Management Frameworks (like OpenStack and OpenNebula) and on Mesos clusters.

It receives the deployment requests, expressed through templates written in TOSCA, and deploys them on the best available cloud site. In order to select the best site, the Orchestrator implements a complex workflow: it gathers information about the SLAs signed by the providers and monitoring data about the availability of the compute and storage resources. Finally the Orchestrator asks the **Cloud Provider Ranker**[17] to provide a ranked list of best cloud

[16]https://github.com/indigo-dc/orchestrator
[17]https://github.com/indigo-dc/cloudproviderranker

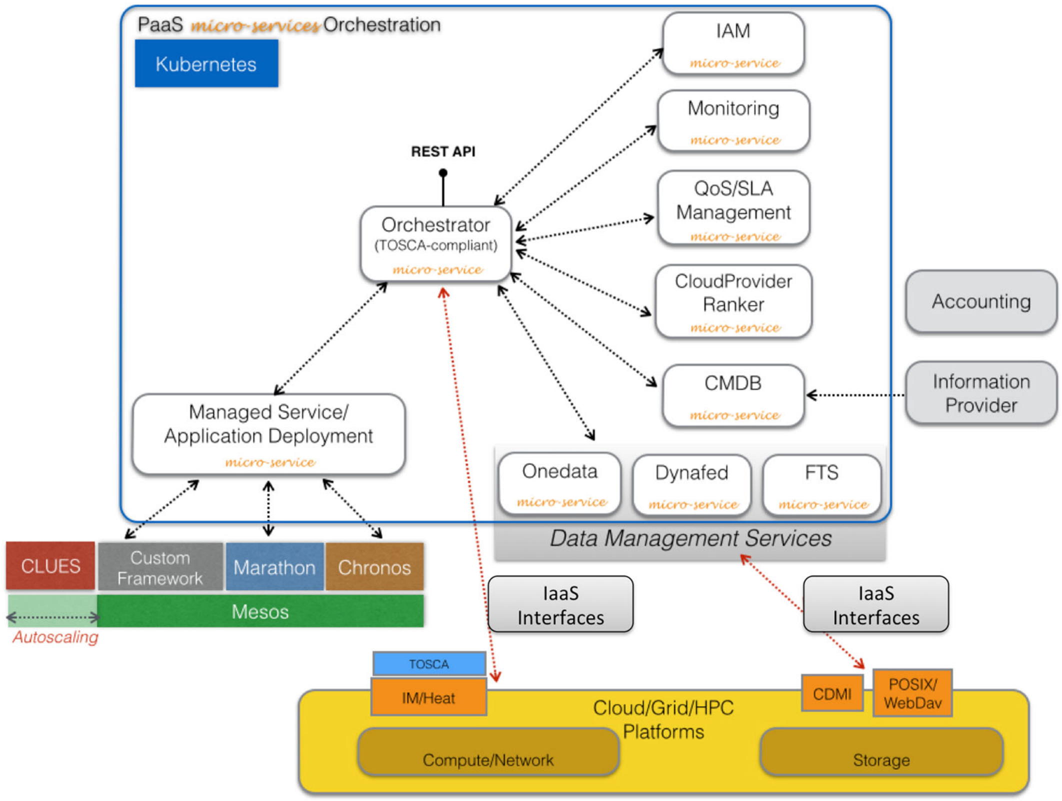


Fig. 5 Architecture of the INDIGO platform as a service layer

providers according an algorithm described below. Therefore the Orchestrator mission is to coordinate the deployment process over the IaaS platforms. See Fig. 6 for an overview of the Orchestrator architecture.

The Orchestrator is based on developments already done in other publicly funded projects, such as the Italian PONs PRISMA [51], and Open City Platform [52]. During the INDIGO project, this component has been extended and enhanced to support the specific microservices building the INDIGO PaaS Layer. It delegates the actual deployment of resources to IM, OpenStack Heat or Mesos frameworks based on TOSCA templates and the ranked list of sites.

A very innovative component is the Cloud Provider Ranker. This is a standalone REST web service, which ranks cloud providers on the basis of rules defined per user/group/use case, with the aim of fully decoupling the ranking logic from the business logic of the Orchestrator.

It allows the consumers of the service (one or more orchestrators) to specify preferences on cloud providers. If some preferences have been specified for some providers, then they have absolute priority over any other provider information (like monitoring data). On the other hand, when preferences are not specified, for each provider the rank is calculated, by default, as the sum of SLA ranks and a combination of monitoring data, conveniently normalized with weights specified in the Ranker configuration file. Moreover, the ranking algorithm can be customized to the specific needs.

This is a completely new service, fully implemented within the INDIGO project; it is based on an open source tool, Drools,[18] in order to reduce the needed development effort, and to simplify the long-term support.

As a summary, one of the main achievements of INDIGO in this respect is the implementation of TOSCA Templates on IaaS that do not support natively TOSCA, like Standard OpenStack, OpenNebula, or Public clouds (like Microsoft Azure, AWS, OTC,...) using the Infrastructure Manager.

[18]https://www.drools.org

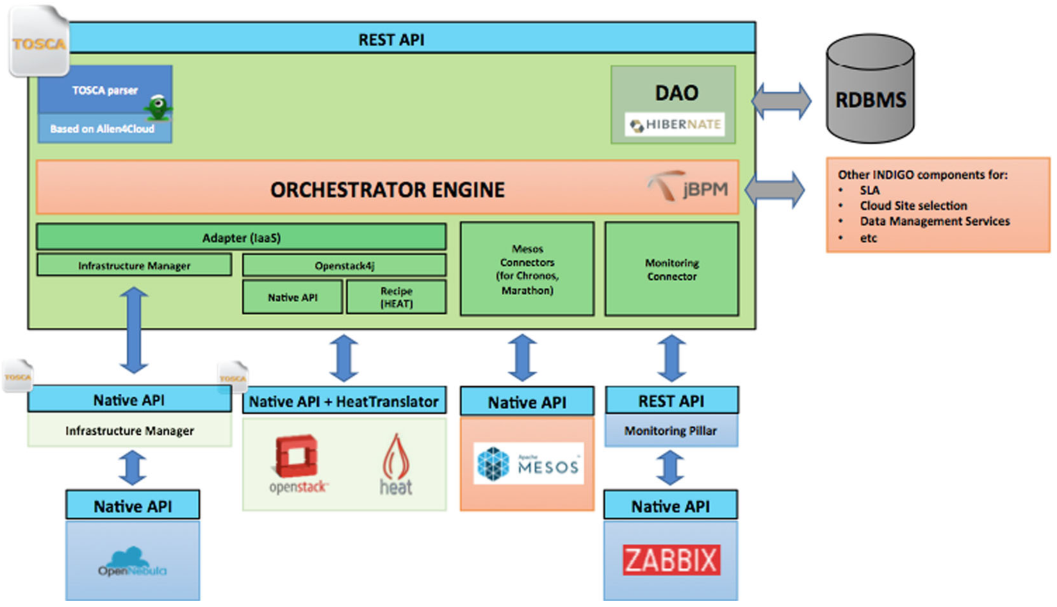


Fig. 6 Architecture of the orchestrator within the PaaS layer

In particular, using the PaaS Orchestrator and the TOSCA templates, the end user can exploit computational resources without knowledge about the IaaS details: indeed the TOSCA standard language ensures that the same template can be used to describe a virtual cluster on different cloud sites; then the Infrastructure Manager implements the TOSCA runtime for contacting the different cloud sites through their native APIs. The provisioning and configuration of the IaaS resources is therefore accomplished in a completely transparent way for the end user. The same approach is used also for submitting dockerized applications and services to a Mesos cluster (and its frameworks Marathon and Chronos): the user can describe his request in a TOSCA template and the Orchestrator provides the TOSCA runtime for contacting the Mesos master node, submitting the request and monitoring its status on behalf of the user as detailed in the next section.

4.1.2 High-Level Geographical Applications/Service Deployment

INDIGO has developed the tools and services to provide a solution for orchestrating Docker containers for both applications (job-like execution) and long running services. **Mesos/Marathon/Chronos**[19] is used to manage the deployment of services and applications (MSA service).

From a resource perspective Mesos is a cluster management tool: it pools several resource centers to be centrally managed as single unit; from an application perspective, Mesos is a scheduler: it dispatches workloads to consume pooled resources, scaling up to thousands of nodes.

Mesos is fault tolerant, as it is possible to replicate the master process. The INDIGO PaaS uses also **Marathon** and **Chronos**. Marathon is used to deploy, monitor and scale Long-running services, ensuring that they are always up and running. Chronos is used to run user applications (jobs) taking care of fetching input data, handling dependencies among jobs or rescheduling failed jobs.

Therefore the submission of jobs uses an approach very similar to a batch system, exploiting resources where they are, without even knowing about the details. This includes deployment on multiple IaaS, both private and public, hiding to the end users the complexity of the distributed resources.

[19]https://github.com/indigo-dc/mesos-cluster

It is also possible using the **CLUES**[20] (Cluster Energy Saving) service to auto-scale (up & down) depending on the load, on several types of clusters: Mesos, SLURM, PBS, HTCondor, etc. CLUES is an elasticity manager system for HPC clusters and Cloud infrastructures that features the ability to power on/deploy working nodes as needed (depending on the job workload of the cluster) and to power off/terminate them when they are no longer needed.

The sustainability of our PaaS layer relies heavily on the fact that we have used Open Source frameworks when already available.

4.1.3 Data Management Services

The goal of the data management developments in INDIGO has been pushing forward the state of the art concerning unified data access over heterogeneous infrastructures. Among the features demanded by the use cases there are High-performance data access, migration and replica management. Supporting such features requires at the user level a flexible security framework based on tokens and Access Control Lists (ACLs).

Data management services developed in INDIGO are based on three open source components: Onedata [53], DynaFed [54] and FTS3 [55].

INDIGO has invested a substantial effort in the development of **Onedata**,[21] which is a global data management system aiming to provide easy access to distributed storage resources. The main goal is supporting a wide range of use cases from personal data management to data-intensive scientific computations.

Support for federation in Onedata can be achieved by the possibility of establishing a distributed provider registry, where various infrastructures can setup their own provider registry and build trust relationships between these instances, allowing users from various platforms to share their data transparently.

Onedata provides an easy to use Graphical User Interface for managing storage Spaces, with customizable access control rights on entire data sets or single files to particular users or groups.

The INDIGO PaaS Orchestrator integrates a plugin for interacting with the Onedata services providing advanced capabilities of data location aware scheduling. Combining the information about the distribution of the compute resources and the data providers with the data requirements specified by the user, the Orchestrator is able to schedule the processing jobs to the computing center nearest to the data. A prototype based on Onedata has been implemented and demonstrated for some use-cases, e.g. the LifeWatch AlgaeBloom (See Table 1).

Using Onedata is possible to integrate already available storage services in the INDIGO Platform exploiting the data stored in external infrastructures. This is the case of the data stored in WLCG by the CMS experiment from LHC. In this case the INDIGO PaaS provided the services needed in order to deal with authentication and autorization (Token Translation)

5 Interfacing with the Users

Users typically do not access the PaaS core components directly. Instead, they often Graphical User Interfaces or simpler APIs. A user authenticated on the INDIGO Platform can access and customize a rich set of TOSCA-compliant templates through a GUI-based portlet.

The INDIGO repository provides a catalogue of pre-configured TOSCA templates to be used for the deployment of a wide range of applications and services, customizable with different requirements of scalability, reliability and performance. In these templates a user can choose between two different examples of generic scenarios:

1. Deploy a customized virtual infrastructure starting from a TOSCA template that has been imported, or built from scratch: the user will be able to access the deployed customized virtual infrastructure and run, administer and manage applications running on it.
2. Deploy a service/application whose lifecycle will be directly managed by the PaaS platform: in this case the user will be returned with list of endpoints to access the deployed services.

APIs for accessing the INDIGO PaaS layer are available, they allow for an easy integration of the PaaS features inside Portals, Desktop Applications

[20] https://github.com/indigo-dc/clues-indigo

[21] https://github.com/indigo-dc/onedata

and Mobile Apps. The final release of INDIGO-DataCloud software includes a large set of components to facilitate the development of Science Gateways and desktop/mobile applications, big data analytics and scientific workflows. The components directly related to the end-user interfaces are integrated in the INDIGO Future Gateway (FG) framework. The FG can be used to build powerful, customized, easy to use science gateways environments and front-ends, on top of the INDIGO-DataCloud PaaS layer and integrated with data management services. The FG provides capabilities including:

- The FG API server, used to integrate third-party science gateways; the FG Liferay Portal, containing base portlets for the authentication, authorization and administration of running applications and deployments;
- Customizable Application Portlets, for user-friendly specification of the parameters used by TOSCA templates;
- A workflows monitoring portlet, used for monitoring task execution via integrated workflow systems, described below.
- An Open Mobile Toolkit as well as application templates for Android and iOS, simplifying the creation of mobile apps making use of the FG API Server.
- Support for scientific workflows, where the INDIGO components:
  - Provide dynamic scalable services in a Workflows as a Service model;
  - Implement modules and components enabling the usage of the PaaS layer (via FG API Server) for the main scientific workflow engines deployed by user communities (such as Kepler, Ophidia, Taverna,Pegasus);
  - Support a two-level (coarse and fine grained) workflow orchestration, essential for complex, distributed experiments involving (among others) parallel data analysis tasks on large volumes of scientific data.
- Key extensions to the Ophidia big data analytics framework (allowing to process, transform and manipulate array-based data in scientific contexts), providing many new functionalities, including a set of new operators related to data import regarding heterogeneous data formats (e.g. SAC, FITS), a new OpenIDConnect interface and new workflow interface extensions.
- Enhancements of the jSAGA library through a ”Resource Management API”, complementing the standard Job/Data Management API. This allows to acquire and manage resources (compute, storage, network) and enables the wrapping of underlying technologies (cloud, pilot jobs, grid, etc.) by means of a single API, supporting asynchronous mode (task), timeout management, notification (metrics) and security context forwarding.
- Command-line clients for the PaaS layer to provide an easy way for users to interact with the Orchestrator or with WATTS:
  - **Orchent**[22]: a command-line application to manage deployments and their resources on the INDIGO-DataCloud Orchestrator;
  - **Wattson**[23]: a command-line client for the INDIGO Token Translation Service.

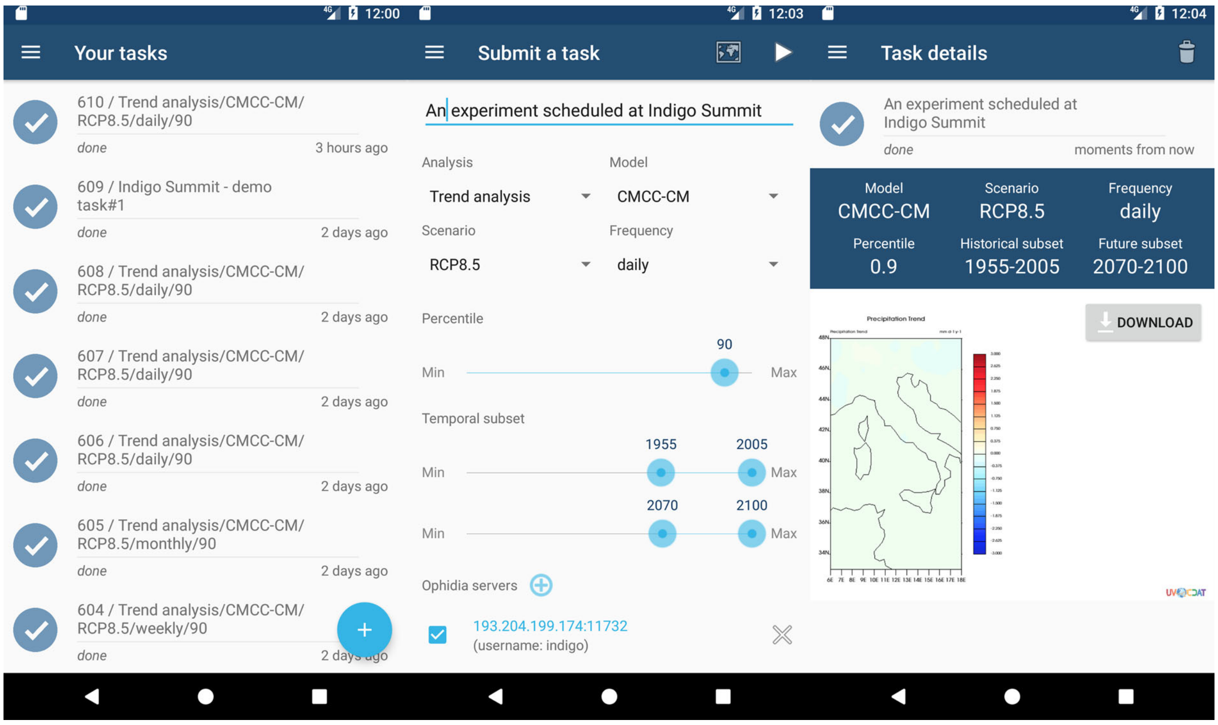
INDIGO has provided the tools for a simple and effective end user experience, both for software developers and for researchers running the applications. In Fig. 7 we show how to launch an application using a mobile platform developed in the project for the climate change application ENES.

The ENES end-user starts the app and needs to authenticate and authorize using IAM service. The apps request to have an access to id, e-mail and offline access, which will be required to refresh the existing tokens during the mobile app lifecycle. After the user gives the permissions and logs in, the user will see the list with scheduled analysis if available.

The mobile app is a handy interface for scheduling new tasks as well. The submitting form requires that the user selects model, scenario, frequency, percentile and nodes to run the analysis. After the user provides the necessary inputs, the app sends the request to the FutureGateway server using its API. The user is then able to monitor the status of the analysis. If the task is done, the user is able to see and download results as PNG files illustrates predicted climate changes on

[22]https://github.com/indigo-dc/orchent

[23]https://github.com/indigo-dc/wattson


12:00
Your tasks

610 / Trend analysis/CMCC-CM/RCP8.5/daily/90
done 3 hours ago

609 / Indigo Summit - demo task#1
done 2 days ago

608 / Trend analysis/CMCC-CM/RCP8.5/daily/90
done 2 days ago

607 / Trend analysis/CMCC-CM/RCP8.5/daily/90
done 2 days ago

606 / Trend analysis/CMCC-CM/RCP8.5/daily/90
done 2 days ago

605 / Trend analysis/CMCC-CM/RCP8.5/monthly/90
done 2 days ago

604 / Trend analysis/CMCC-CM/RCP8.5/weekly/90
done 2 days ago

12:03
Submit a task

An experiment scheduled at Indigo Summit

Analysis: Trend analysis
Model: CMCC-CM
Scenario: RCP8.5
Frequency: daily

Percentile
Min 90 Max

Temporal subset
Min 1955 2005 Max
Min 2070 2100 Max

Ophidia servers
193.204.199.174:11732
(username: indigo)

12:04
Task details

An experiment scheduled at Indigo Summit
done moments from now

Model	Scenario	Frequency
CMCC-CM	RCP8.5	daily
Percentile	**Historical subset**	**Future subset**
0.9	1955-2005	2070-2100

Fig. 7 Task submission and visualization of results on the mobile platform for ENES

Earth. Finally the user can remove useless or aborted tasks.

6 Software Lifecycle Management

The software lifecycle process in INDIGO has been supported by a continuous improvement process that encompassed the software quality assurance (SQA), the software release and maintenance, the deployment of pilot infrastructures for software integration and testing, and, lastly, the exploitation activities and support services. In Fig. 8 we depict the interdependencies between the different processes, together with the services involved at each stage. Appendix B describes the tools and services that were required for the implementation of the software lifecycle process.

The quality requirements [56], that drive the software lifecycle process, define the minimum set of criteria that the software developed in INDIGO has to comply with. The requirements are met for each change in the codebase, thus the production version of a given software component is permanently in a workable status, protected from incoming changes that do not adhere to the SQA criteria. The continuous evaluation of the SQA requirements is only possible through the aid of automation, achieved in INDIGO through the progressive adoption of DevOps practices.

In the next sections we will describe the DevOps approaches being adopted and the upstream contributions included in the official distributions of external open source projects.

6.1 DevOps Approach in INDIGO

Progressive levels of automation were being adopted throughout the different phases of the INDIGO-DataCloud project software development and delivery processes. This evolution was intentionally marked by the commitment to DevOps principles [57]. Starting with a continuous integration (CI) approach, achieved already in the early stages of the project, the second part of the project was devoted to the establishment of the next natural step in the DevOps practices: the continuous delivery (CD).

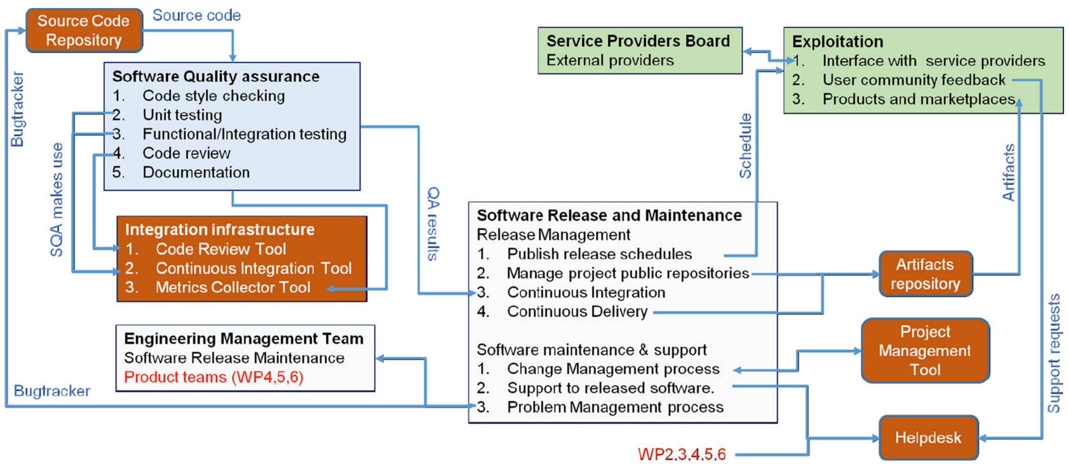

Source Code Repository
Source code
Software Quality assurance
1. Code style checking
2. Unit testing
3. Functional/Integration testing
4. Code review
5. Documentation
Integration infrastructure
1. Code Review Tool
2. Continuous Integration Tool
3. Metrics Collector Tool
Engineering Management Team
Software Release Maintenance
Product teams (WP4,5,6)
Bugtracker
SQA makes use
QA results
Service Providers Board
External providers
Exploitation
1. Interface with service providers
2. User community feedback
3. Products and marketplaces
Schedule
Artifacts
Support requests
Software Release and Maintenance
Release Management
1. Publish release schedules
2. Manage project public repositories
3. Continuous Integration
4. Continuous Delivery
Software maintenance & support
1. Change Management process
2. Support to released software.
3. Problem Management process
Artifacts repository
Project Management Tool
Helpdesk
WP2,3,4,5,6

Fig. 8 Software lifecycle, release, maintenance and exploitation interdependencies

6.1.1 Services for Continuous Integration and SQA

The INDIGO-DataCloud CI process is schematically shown in Fig. 9 is explained below. The process, in its different steps, reflects some of the main and important achievements of the software integration team.

- New features are developed independently from the production version in *feature branches*. In order to test and review the new change, a pull request (PR) is created in GitHub. The PR creation marks the start of the automated validation process through the execution of the SQA jobs in the CI infrastructure (Jenkins).
- The SQA jobs perform the adherence of the code to a style standard and calculate unit and functional test coverage. Other checks are executed at this stage for security static analysis and metrics gathering.

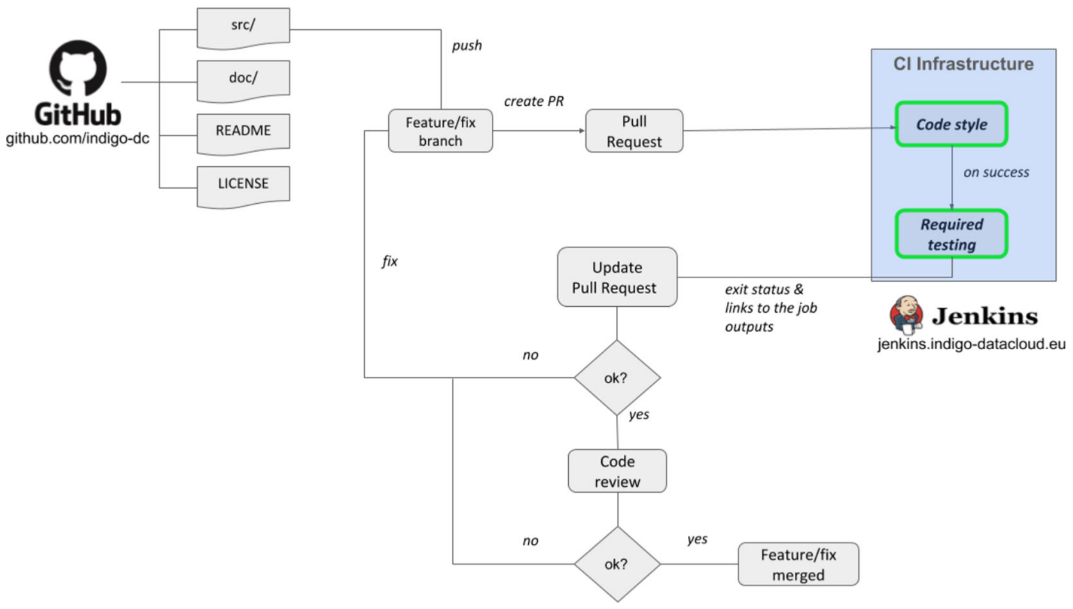


Fig. 9 Continuous Integration workflow followed by new feature additions in the production codebase

- The results of the several SQA jobs automatically executed in Jenkins are notified back to GitHub, updating the PR with the exit status and the links to the output logs.
- On successful completion of the SQA tests, the code review is the last step before the source code is merged in the production version. The GitHub PR provides a place for discussion, open to collaboration, where the developers and/or external experts analyze the results of the SQA jobs and discuss any relevant aspect of the change (internal to the code or in terms of the goals or applicability).
- Once peer-reviewed, the change is merged and becomes ready for integration and later release.

6.1.2 Continuous Delivery

Continuous delivery adds, on top of the CI approach described above, a seamless manufacturing of software packages ready to be deployed into production.

In the INDIGO-DataCloud scenario, the continuous delivery adoption translates into the definition of pipelines. A pipeline is a serial execution of tasks that encompasses in the first place the SQA jobs (CI phase) and adds as the second part (CD phase) the building and deployment testing of the software packages created. The pipeline only succeeds if each task is run to completion, otherwise the process is stopped and set as a build failure.

6.1.3 DevOps Adoption from User Communities

The experience gathered throughout the project with regards to the adoption of different DevOps practices is not only useful and suitable for the software related to the core services in the INDIGO-DataCloud solution, but also applicable to the development and distribution of the applications coming from the user communities.

The novelty introduced, showcased in Appendix C, is the validation of the user application by comparing the execution results with a set of reference outputs. Thus this pipeline implementation goes a step forward, with respect to the former DevOps approaches, as the application execution is tested before the new version is released.

6.2 INDIGO Upstream Software Contributions

The INDIGO software solution encompasses not only products implemented from scratch within the project but also external services adopted from open source initiatives. These latter set of products were actively developed to enhance their functionality to match the INDIGO project's objectives, but at the same time, aiming to be considered as upstream contributions. Thus, multiple contributions developed by INDIGO have been pushed and accepted in the official distributions of major open source projects such as OpenStack, OpenNebula and OpenID Connect. Appendix A lists the software projects and products being contributed by INDIGO-DataCloud.

7 Examples of Implementation Towards Research Communities

In what follows we try to provide some basic information that may be useful for promoting the use of INDIGO solutions towards the Research Communities. Based on the described architecture we will introduce the basic ideas on how to develop, deploy and support applications in the Cloud framework, exploiting the different service layers, and introducing generic examples that may make easier the use of INDIGO solutions.

7.1 Understanding the Services of the Cloud Computing Framework

Figure 10 provides the description of how an application can be built using a service oriented architecture in the Cloud, using INDIGO solutions.

This layered scheme includes different elements, that are managed by different actors, that must be minimally understood in order to design, develop, test, deploy and put in production an application.

The lowest layer, Infrastructure as a Service, provides a way to access to the basic resources that the application will use: computing, storage, network, etc. These resources are physically in a site, typically a computing center, either in a research centre, or in a cloud provider (for example commercial cloud services), and are handled by the system managers at

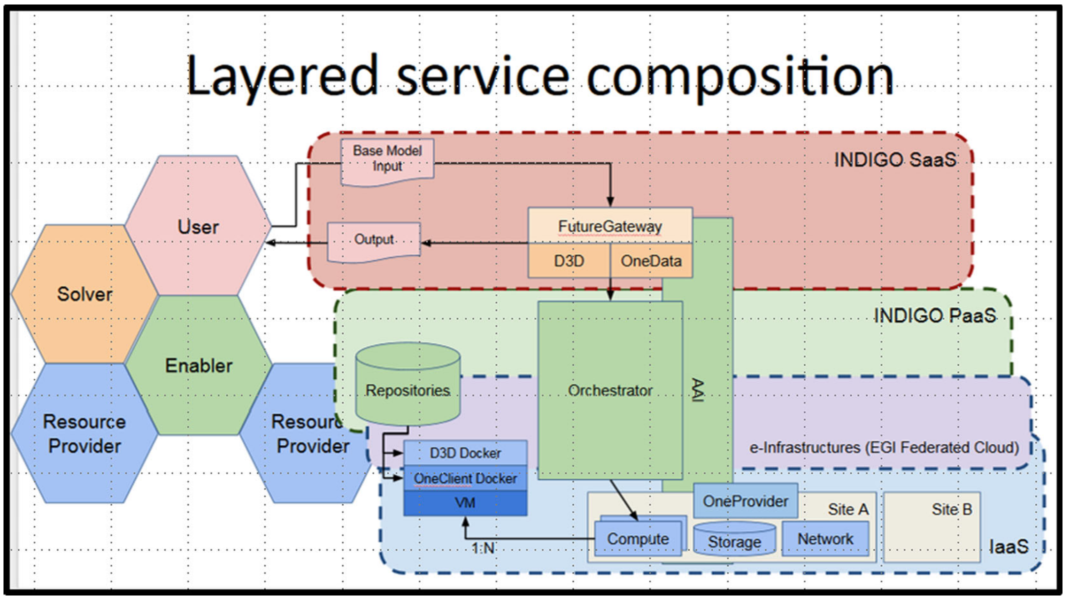


Fig. 10 Service composition in the Service Oriented Architecture

those sites, that install a IaaS solution compatible with the INDIGO software stack (e.g. OpenStack, OpenNebula, Google Compute Engine, Microsoft Azure, Amazon EC2).

By accessing through a web interface such as Horizon for OpenStack, a user that is granted access to a pool of resources at a site can launch a virtual machine, for example a server with 2 cores, 4GB RAM, 100GB of storage and with a certain Linux flavour installed.

The user can then get access in console mode this machine, using for instance ssh protocol. Once logged in, the user can execute a simple script, or can install a web server, etc. When the work is finished the machine can be stopped by the user, liberating so the resources. This is a very basic mode of accessing Cloud services, which shares analogies with the usual access to classical computing services, like for example any remote server or a cluster.

A different way to interact with IaaS services is to use the existing APIs to manage the resources using the web services protocol. Such invocation of services can be made from any program or application, for example from a python script, and even through a web interface.

Many applications require the setup, launching and interconnection of several (IaaS) services implemented in different virtual machines, and managed under a single control, as a Platform. The Platform as a Service (PaaS) layer enables this orchestration of IaaS services, and in the case of a Federated Cloud they might even be located in different sites.

For example, an Apache Web Server may be launched in site 1, with the purpose of displaying the output of a simulation running in a cluster launched on demand in site 2. The Apache Web server may be better supported with a pool of resources using another cloud-oriented solution such as Marathon/Mesos. Launching an application in this context requires identifying the available resources and launching them via the IaaS services. INDIGO is supporting the TOSCA standard to prepare a template that can be used to automatize this selection and orchestration of services.

7.2 Building and Executing Applications Using INDIGO Solutions

In what follows we present below several simple examples of basic, but generic, applications exploiting INDIGO solutions.

7.2.1 Executing Containers on HPC Systems

The first generic example is how to build an application encapsulated as a container and how to executed it in an HPC system. This basic example of using INDIGO solutions is shown in Fig. 11. A user can create a container using a conventional Dockerfile which describes the steps required to create the Docker image. The process can be fully automatized using GitHub and Docker Hub in such a way that a change in the Dockerfile, immediately triggers a rebuilt of the application container.

INDIGO provides the udocker tool to enable execution of application containers in batch systems. The end-user can download the udocker Python script from GitHub or can send it with the batch job. Once executed for the first time it setups itself in the user home directory. udocker provides a Docker like command line interface with which the user can pull, import or load Docker containers and then execute them using a chroot-like environment. The software within the container must not require privileges during execution as it will be executed under the user that invokes udocker.

This is also a good solution for research communities that want to migrate towards a cloud-based framework using containers, but keep exploiting resources like grid-enabled clusters or even supercomputers.

udocker is used by the Case Studies on Structural Biology (Powerfit and Disvis) exploiting grid resources, on Phenomenology in Particle Physics, and recently for Lattice QCD on supercomputers. Also, the TRUFA genomic pipeline exploits this solution, and it is being extended to similar applications in the area that require the integration of legacy libraries.

7.2.2 Executing Containers on the Cloud

The second example is how to build an application encapsulated as a container and launch it in the Cloud, from a web interface, using the INDIGO solutions FutureGateway and PaaS Orchestrator

This second example, compared to the first one, shows the evolution required to move an application to the Cloud arena: the application must be encapsulated into a container, as before, but to launch this container the cloud resources must be allocated, the user must authenticate and get the access granted. If different services are required, they must be orchestrated.

The way to express these requirements, using an open standard, is a TOSCA template. FutureGateway offers a user-friendly web-based interface to customize the TOSCA template, authenticate the user, select the container to be executed, interact with the Orchestrator to allocate the required cloud resources and launch the application. As in the first example the container can be created using a dockerfile. Automation in this step can be achieved using GitHub and DockerHub.

Using the Future Gateway portal or the command line tool Orchent, the user can submit a TOSCA template to the Orchestrator, which in turn will request and allocate the resources at the IaaS level by asking the Infrastructure Manager to do so.

The user may wish to connect to the container that has been launched via the orchestrator using the `ssh` command. Once in the container it is posible to mount

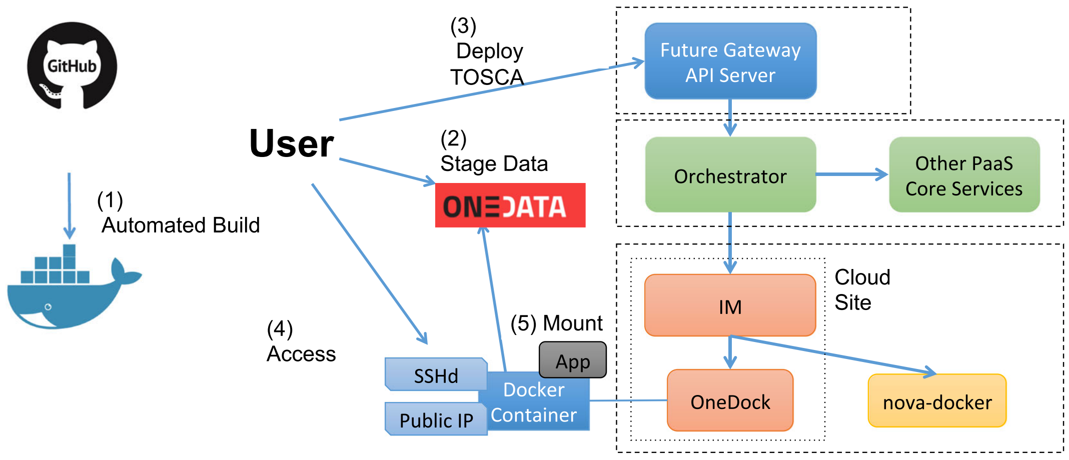


Fig. 11 Basic execution of containers using INDIGO solutions

remote data repositories or stage the output data using Onedata, available at the IaaS layer.

7.3 Building Advanced Applications Using INDIGO Solutions

In this section we present several simplified schemes corresponding to different applications already implemented, with the idea that they can be more easily used as a guide to configure new applications.

The key, as stated before, is the composition of the template, written using the TOSCA language. The template should specify the image of the application to be used, as a container, using docker technology. We also need to specify the resources (CPUs, storage, memory, network ports) required to support the execution. The parameters required to configure INDIGO services used like, for example, Onedata, Mesos/Marathon or other additional cloud services need to be specified as well.

Examples of TOSCA templates can be found at https://github.com/indigo-dc/tosca-templates. FutureGateway offers a friendly way to handle the TOSCA templates to launch the applications.

7.3.1 Deployment of a Digital Repository

A first example is the deployment, as a SaaS solution, of a digital repository. The scheme is presented in the Fig. 12 below. The specific template for this application is available for reuse in the github repository of the project.

The repository manager, controlling the application, uses the FutureGateway to configure the application, based on the ZENODO software, that can be automatically scaled up and ensure its high availability using Cloud resources as needed, and also enabling the authentication and authorization mechanism for their research community, DARIAH, based on the INDIGO solution IAM.

All these details are transparent to the final user, who accesses the repository directly through its web interface, and benefits of the enhanced scalability and availability.

7.3.2 Launching a Virtual Elastic Cluster for Data Intensive Applications

A second example is the launch of a Virtual Elastic Cluster to support a data intensive system.The scheme is presented in Fig. 13 below.

The specific template for this advanced application is available for reuse in the github repository of the project.

Galaxy is an open source, web-based platform for data intensive biomedical research. This application deploys a Galaxy instance provider platform, allowing to fully customize each virtual instance through a user-friendly web interface, ready to be used by life scientists and bioinformaticians.

The front-end that will be in charge of managing the cluster elasticity can use a specified LRMS (selected among torque, sge, slurm and condor) workload.

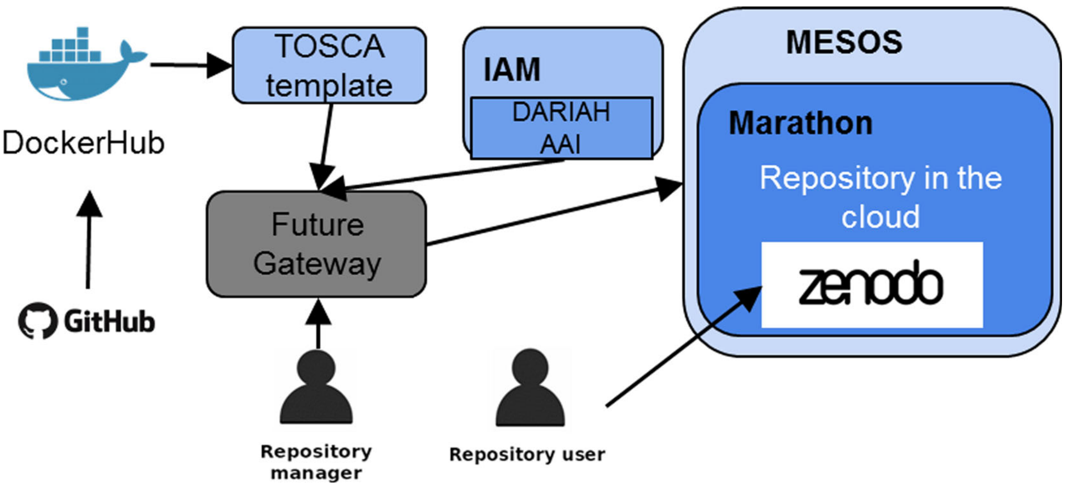


Fig. 12 Deployment of a digital repository using INDIGO solutions

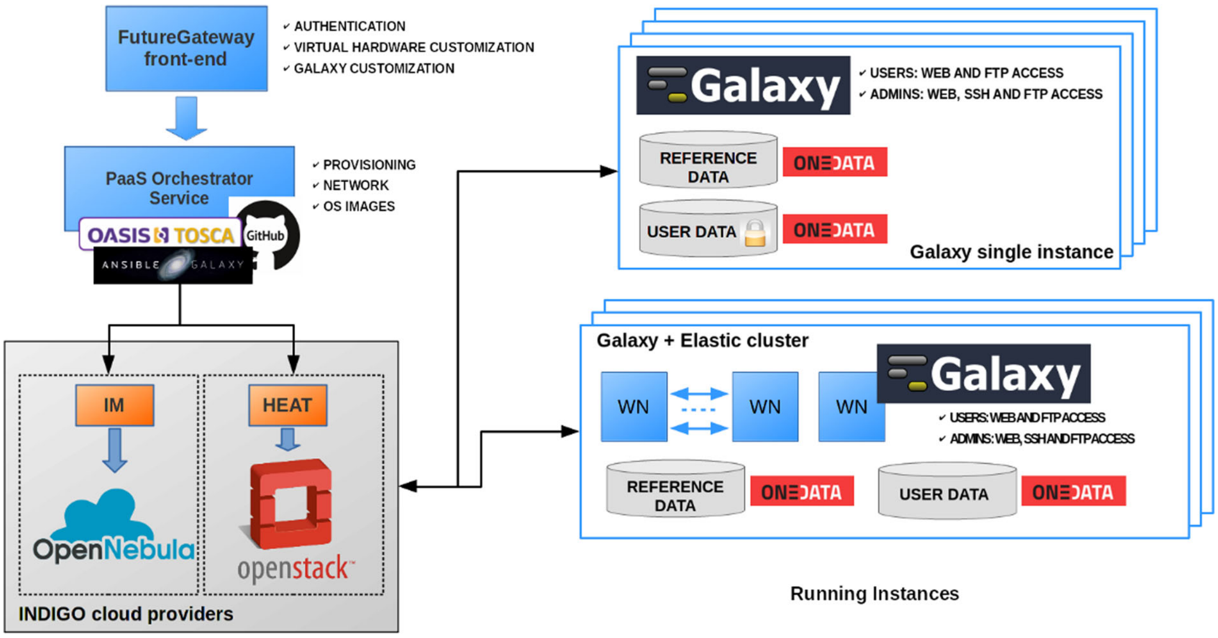


Fig. 13 Launching of a Virtual Elastic Cluster using INDIGO solutions

All these details are transparent to the final user, the researcher, who accesses the Galaxy instance directly through its web interface, and benefits of the enhanced scalability and availability.

This complex template includes the configuration of the distributed storage based in Onedata, the use of the encrypted files via LUKS, the deployment of elastic clusters using another INDIGO solution, CLUES, and the integration of the Authentication and Authorization mechanism, very relevant for this application area, using IAM.

8 Conclusions

Thanks to the new common solutions developed by the INDIGO project, teams of first-line researchers in Europe are using public and private Cloud resources to get new results in Physics, Biology, Astronomy, Medicine, Humanities and other disciplines.

INDIGO-developed solutions that have for instance enabled new advances in understanding how the basic blocks of matter (quarks) interact, using supercomputers, how new molecules involved in life work, using GPUs, or how complex new repositories to preserve and consult digital heritage can be easily built. The variety of the requirements coming from these so diverse user communities proves that the modular INDIGO platform, consisting of several state-of-the-art, production-level services, is flexible and general enough to be applied to all of them with the same ease of use and efficiency.

These services allow to federate hybrid resources, to easily write, port and run scientific applications to the cloud. They are all freely downloadable as open source components, and are already being integrated into many scientific applications, namely:

- High-energy physics: the creation of complex clusters deployed on several Cloud infrastructures is automated, in order to perform simulation and analysis of physics data for large experiments.
- Lifewatch: parameters from a water quality model in a reservoir are calibrated, using automated multiple simulations.
- Digital libraries: multiple libraries can easily access a cloud environment under central coordination but uploading and managing their own collections of digital objects. This allows them to consistently keep control of their collections and to certify their quality.
- Elixir: Galaxy, a tool often used in many life science research environments, is automatically configured, deployed on the Cloud and used to process data through a user-friendly interface.

- Theoretical HEP physics: the MasterCode software, used in theoretical physics, adopts INDIGO tools to run applications on Grids, Clouds and on HPC systems with an efficient, simple-to-use, consistent interface.
- In DARIAH, a pan-european social and technical infrastructure for arts and humanities, the deployment of a self-managed, auto-scalable Zenodo-based repository in the cloud is automated.
- Climate change: distributed, parallel data analysis in the context of the Earth System Grid Federation (ESGF) infrastructure is performed through software deployed on HPC and cloud environments in Europe and in the US.
- Image analysis: in the context of EuroBioImaging, a distributed infrastructure for microscopy, molecular and medical imaging, INDIGO components are used to perform automatic and scalable analysis of bone density.
- Astronomical data archives: big data consisting of images collected by telescopes are automatically distributed and accessed via INDIGO tools.

The same solutions are also being explored by industry, to provide innovative services to EU companies: for example, modelling water reservoirs integrating satellite information, improving security in cyberspace, or assisting doctors in diagnostics through medical images. INDIGO solutions are also being intensively tested in other projects, such as HelixNebula ScienceCloud.

INDIGO services are fundamental for the implementation of the EOSC. In particular, many INDIGO components are included in the unified service catalogue provided by the project EOSC-hub [58], that will put in place the basic layout for the European Open Science Cloud. Two additional Horizon 2020 projects were also approved (`DEEP Hybrid DataCloud` and `eXtreme DataCloud`), that will continue to develop and enhance INDIGO components.

The outcomes of INDIGO-DataCloud will persist, and also be extended, after the end of the project in the framework of the *INDIGO Software Collaboration agreement*. This Collaboration shall be continued without financial support from the European Union. It is open to new initiatives and partners willing to contribute, extend or maintain the INDIGO-DataCloud software components.

Acknowledgments INDIGO-Datacloud has been funded by the European Commision H2020 research and innovation program under grant agreement RIA 653549.

Appendix A: Contribution to Open Source Software Projects

Here follows the list of software developed in the framework of INDIGO-Datacloud that has been contributed upstream to the Open Source community.

- OpenStack (https://www.openstack.org)
    - Changes/contribution done already merged upstream
        * Nova Docker
        * Heat Translator (INDIGO-Data Cloud is 3rd overall contributor and core developer)
        * TOSCA parser (INDIGO-Data Cloud is 2nd overall contributor and core developer)
        * OpenID Connect CLI support
        * OOI: OCCI implementation for OpenStack
    - Changes/contribution under discussion to be merged upstream OpenStack Preemptible Instances support (extensions)
- OpenNebula
    - Changes/contribution done already merged upstream
        * ONEDock
- Changes/contribution done already merged upstream for:
    - Infrastructure Manager (http://www.grycap.upv.es/im/index.php)
    - CLUES (http://www.grycap.upv.es/clues/eng/index.php)
    - Onedata (https://onedata.org)
    - Apache Libcloud (https://github.com/apache/libcloud)

- Kepler Workflow Manager (https://kepler-project.org/)
- TOSCA adaptor for JSAGA (http://software.in2p3.fr/jsaga/dev/)
- CDMI and QoS extensions for dCache (https://www.dcache.org)
- Workflow interface extensions for Ophidia (http://ophidia.cmcc.it)
- OpenID Connect Java implementation for dCache (https://www.dcache.org)
- MitreID (https://mitreid.org/) and OpenID Connect (http://openid.net/connect/) libraries
- FutureGateway (https://www.catania-science-gateways.it/)

Appendix B: Tools and Services Involved in the Software Lifecycle

Figure 14 showcases the tools and services used for the development and distribution of the INDIGO-DataCloud software:

- Project management service using **openproject.org**: It provides tools such as an issue tracker, wiki, a placeholder for documents and a project management timeline.
- Source code is publicly available, housed externally in GitHub repositories, increasing so the visibility and simplifying the path to exploitation beyond the project lifetime. The INDIGO-DataCloud software is released under the Apache 2.0 software license [59].
- Continuous Integration service using **Jenkins**: Service to automate the building, testing and packaging, where applicable. Testing includes the style compliance and estimation of the unit and functional test coverage of the software components.
- Artifact repositories for RedHat and Debian packages [60] and virtual – Docker – images [61].
- Code review service using GitHub: Source code review is one integral part of the SQA as it appears as the last step in the change verification process. This service facilitates the code review process, recording the comments and allowing the reviewer to verify the candidate change before being merged into the production version.
- Issue tracking using GitHub Issues: Service to track issues, new features and bugs of INDIGO-DataCloud software components.
- Release notes, installation and configuration guides, user and development manuals are made available on **GitBook** [62].
- Code metrics services using **Grimoire**: To collect and visualize several metrics about the software components.

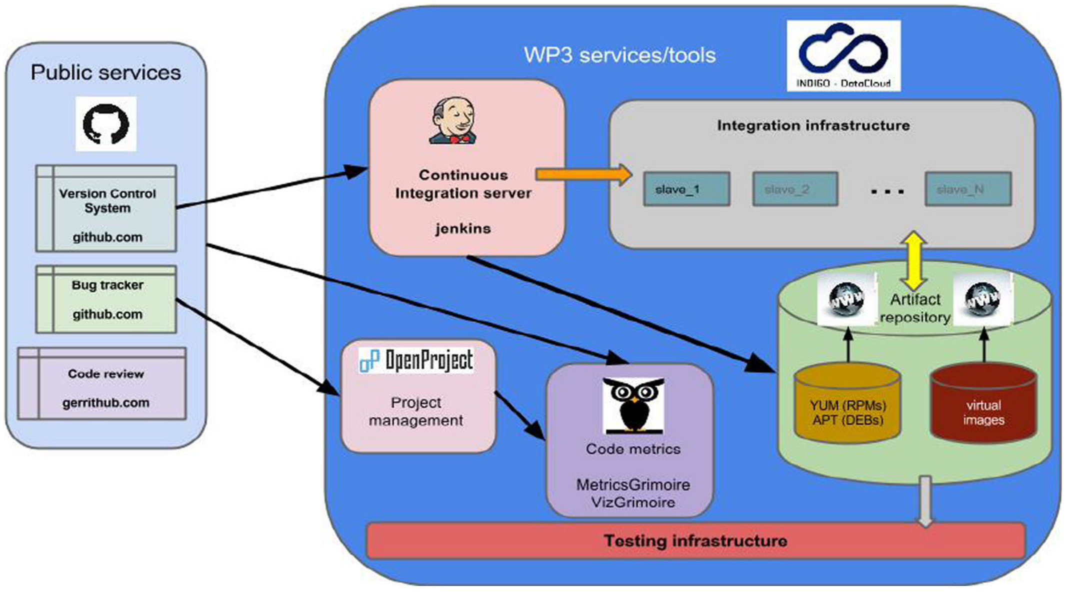


Fig. 14 Tools and services used to support the software lifecycle process

- Integration infrastructure: this infrastructure is composed of computing resources to support directly the CI service.
- Testing infrastructure: this infrastructure aims to provide a stable environment for users where they can preview the software and services developed by INDIGO-DataCloud, prior to its public release.
- Preview infrastructure: where the released artifacts are deployed and made available for testing and validation by the use-cases.

Appendix C: DevOps Adoption from User Communities

DisVis [63] and PowerFit [64] applications were integrated into a CI/CD pipeline described above. As it can be seen in the Fig. 15, with this pipeline in place the application developers were provided with both a means to validate the source code before merging and the creation of a new versioned Docker image, automatically available in the INDIGO-DataClouds catalogue for applications i.e. DockerHub???s `indigodatacloudapps` repository.

Once the application is deployed as a Docker container, and subsequently uploaded to `indigodatacloudapps` repository, it is instantiated in a new container to be validated. The application is then executed and the results compared with a set of reference outputs. Thus this pipeline implementation goes a step forward by testing the application execution for the last available Docker image in the catalogue.

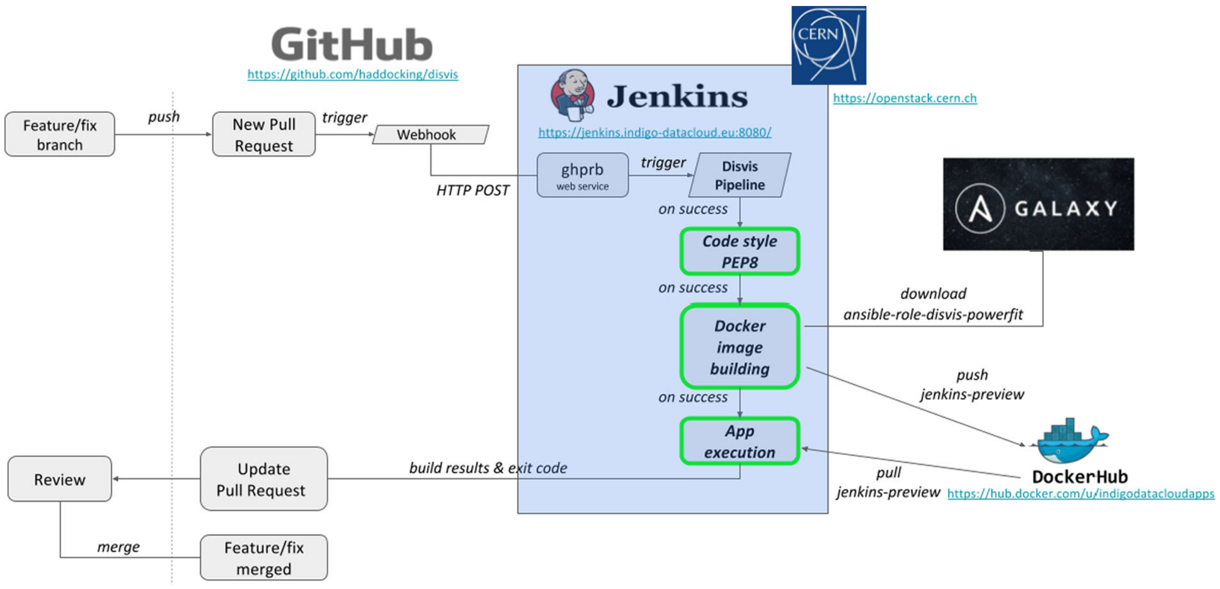


Fig. 15 DisVis development workflow using a CI/CD approach

References

1. García, A.L., Castillo, E.F.-d., Puel, M.: Identity federation with VOMS in cloud infrastructures. In: 2013 IEEE 5Th International Conference on Cloud Computing Technology and Science, pp 42–48 (2013)
2. Chadwick, D.W., Siu, K., Lee, C., Fouillat, Y., Germonville, D.: Adding federated identity management to OpenStack. Journal of Grid Computing **12**(1), 3–27 (2014)
3. Craig, A.L.: A design space review for general federation management using keystone. In: Proceedings of the 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing, pp 720–725. IEEE Computer Society (2014)
4. Pustchi, N., Krishnan, R., Sandhu, R.: Authorization federation in iaas multi cloud. In: Proceedings of the 3rd International Workshop on Security in Cloud Computing, pp 63–71. ACM (2015)
5. Lee, C.A., Desai, N., Brethorst, A.: A Keystone-Based Virtual Organization Management System. In: 2014 IEEE 6Th International Conference On Cloud Computing Technology and Science (Cloudcom), pp 727–730. IEEE (2014)
6. Castillo, E.F.-d., Scardaci, D., García, A.L.: The EGI Federated Cloud e-Infrastructure. Procedia Computer Science **68**, 196–205 (2015)
7. AARC project: AARC Blueprint Architecture, see https://aarc-project.eu/architecture. Technical report (2016)
8. Oesterle, F., Ostermann, S., Prodan, R., Mayr, G.J.: Experiences with distributed computing for meteorological applications: grid computing and cloud computing. Geosci. Model Dev. **8**(7), 2067–2078 (2015)
9. Plasencia, I.C., Castillo, E.F.-d., Heinemeyer, S., García, A.L., Pahlen, F., Borges, G.: Phenomenology tools on cloud infrastructures using OpenStack. The European Physical Journal C **73**(4), 2375 (2013)
10. Boettiger, C.: An introduction to docker for reproducible research. ACM SIGOPS Operating Systems Review **49**(1), 71–79 (2015)
11. Docker: http://www.docker.com (2013)
12. Gomes, J., Campos, I., Bagnaschi, E., David, M., Alves, L., Martins, J., Pina, J., Alvaro, L.-G., Orviz, P.: Enabling rootless linux containers in multi-user environments: the udocker tool. Computing Physics Communications. https://doi.org/10.1016/j.cpc.2018.05.021 (2018)
13. Zhang, Z., Chuan, W., Cheung, D.W.L.: A survey on cloud interoperability taxonomies, standards, and practice. SIGMETRICS perform. Eval. Rev. **40**(4), 13–22 (2013)
14. Lorido-Botran, T., Miguel-Alonso, J., Lozano, J.A.: A Review of Auto-scaling Techniques for Elastic Applications in Cloud Environments. Journal of Grid Computing **12**(4), 559–592 (2014)
15. Nyrén, R., Metsch, T., Edmonds, A., Papaspyrou, A.: Open Cloud Computing Interface–Core. Technical report, Open Grid Forum (2010)
16. Metsch, T., Edmonds, A.: Open Cloud Computing Interface-Infrastructure. Technical report, Open Grid Forum (2010)
17. Metsch, T., Edmonds, A.: Open Cloud Computing Interface-RESTful HTTP Rendering. Technical report, Open Grid Forum (2011)
18. (Ca Technologies) Lipton, P., (Ibm) Moser, S., (Vnomic) Palma, D., (Ibm) Spatzier, T.: Topology and Orchestration Specification for Cloud Applications. Technical report, OASIS Standard (2013)
19. Teckelmann, R., Reich, C., Sulistio, A.: Mapping of cloud standards to the taxonomy of interoperability in IaaS. In: Proceedings - 2011 3rd IEEE International Conference on Cloud Computing Technology and Science, CloudCom 2011, pp. 522–526 (2011)
20. García, A.L., Castillo, E.F.-d., Fernández, P.O.: Standards for enabling heterogeneous IaaS cloud federations. Computer Standards & Interfaces **47**, 19–23 (2016)
21. Caballer, M., Zala, S., García, A.L., Montó, G., Fernández, P.O., Velten, M.: Orchestrating complex application architectures in heterogeneous clouds. Journal of Grid Computing **16**(1), 3–18 (2018)
22. Hardt, M., Jejkal, T., Plasencia, I.C., Castillo, E.F.-d., Jackson, A., Weiland, M., Palak, B., Plociennik, M., Nielsson, D.: Transparent Access to Scientific and Commercial Clouds from the Kepler Workflow Engine. Computing and Informatics **31**(1), 119 (2012)
23. Fakhfakh, F., Kacem, H.H., Kacem, A.H.: Workflow Scheduling in Cloud Computing a Survey. In: IEEE 18Th International Enterprise Distributed Object Computing Conference Workshops and Demonstrations (EDOCW), 2014, Vol. 71, pp. 372–378. Springer, New York (2014)
24. Stockton, D.B., Santamaria, F.: Automating NEURON simulation deployment in cloud resources. Neuroinformatics **15**(1), 51–70 (2017)
25. Plóciennik, M., Fiore, S., Donvito, G., Owsiak, M., Fargetta, M., Barbera, R., Bruno, R., Giorgio, E., Williams, D.N., Aloisio, G.: Two-level Dynamic Workflow Orchestration in the INDIGO DataCloud for Large-scale, Climate Change Data Analytics Experiments. Procedia Computer Science **80**, 722–733 (2016)
26. Moreno-Vozmediano, R., Montero, R.S., Llorente, I.M.: Multicloud deployment of computing clusters for loosely coupled mtc applications. IEEE transactions on parallel and distributed systems **22**(6), 924–930 (2011)
27. Katsaros, G., Menzel, M., Lenk, A.: Cloud Service Orchestration with TOSCA, Chef and Openstack. In: Ic2e (2014)
28. Garcia, A.L., Zangrando, L., Sgaravatto, M., Llorens, V., Vallero, S., Zaccolo, V., Bagnasco, S., Taneja, S., Dal Pra, S., Salomoni, D., Donvito, G.: Improved Cloud resource allocation: how INDIGO-DataCloud is overcoming the current limitations in Cloud schedulers. J. Phys. Conf. Ser. **898**(9), 92010 (2017)
29. Singh, S., Chana, I.: A survey on resource scheduling in cloud computing issues and challenges. Journal of Grid Computing, pp. 1–48 (2016)
30. García, A.L., Castillo, E.F.-d., Fernández, P.O., Plasencia, I.C., de Lucas, J.M.: Resource provisioning in Science Clouds: Requirements and challenges. Software: Practice and Experience **48**(3), 486–498 (2018)

31. Chauhan, M.A., Babar, M.A., Benatallah, B.: Architecting cloud-enabled systems: a systematic survey of challenges and solutions. Software - Practice and Experience **47**(4), 599–644 (2017)
32. Somasundaram, T.S., Govindarajan, K.: CLOUDRB A Framework for scheduling and managing High-Performance Computing (HPC) applications in science cloud. Futur. Gener. Comput. Syst. **34**, 47–65 (2014)
33. Sotomayor, B., Keahey, K., Foster, I.: Overhead Matters: A Model for Virtual Resource Management. In: Proceedings of the 2nd International Workshop on Virtualization Technology in Distributed Computing SE - VTDC '06, p. 5. IEEE Computer Society, Washington (2006)
34. SS, S.S., Shyam, G.K., Shyam, G.K.: Resource management for Infrastructure as a Service (IaaS) in cloud computing SS Manvi A survey. J. Netw. Comput. Appl. **41**, 424–440 (2014)
35. INDIGO-DataCloud consortium: Initial requirements from research communities - d2.1, see https://www.indigo-datacloud.eu/documents/initial-requirements-research-communities-d21. Technical report (2015)
36. Europen open science cloud: https://ec.europa.eu/research/openscience (2015)
37. Proot: https://proot-me.github.io/ (2014)
38. Runc: https://github.com/opencontainers/runc (2016)
39. Fakechroot: https://github.com/dex4er/fakechroot (2015)
40. Pérez, A., Moltó, G., Caballer, M., Calatrava, A.: Serverless computing for container-based architectures Future Generation Computer Systems (2018)
41. de Vries, K.J.: Global fits of supersymmetric models after LHC run 1. Phd thesis Imperial College London (2015)
42. Openstack: https://www.openstack.org/ (2015)
43. See http://argus-documentation.readthedocs.io/en/stable/argus_introduction.html (2017)
44. See https://en.wikipedia.org/wiki/xacml (2013)
45. See http://www.simplecloud.info (2014)
46. Opennebula: http://opennebula.org/ (2018)
47. Redhat openshift: http://www.opencityplatform.eu (2011)
48. The cloud foundry foundation: https://www.cloudfoundry.org/ (2015)
49. Caballer, M., Blanquer, I., Moltó, G., de Alfonso, C.: Dynamic management of virtual infrastructures. Journal of Grid Computing **13**(1), 53–70 (2015)
50. See http://www.infoq.com/articles/scaling-docker-with-kubernetes (2014)
51. Prisma project: http://www.ponsmartcities-prisma.it/ (2010)
52. Opencitiy platform: http://www.opencityplatform.eu (2014)
53. Onedata: https://onedata.org/ (2018)
54. Dynafed: http://lcgdm.web.cern.ch/dynafed-dynamic-federation-project (2011)
55. Fts3: https://svnweb.cern.ch/trac/fts3 (2011)
56. Fernández, P.O., García, A.L., Duma, D.C., Donvito, G., David, M., Gomes, J.: A set of common software quality assurance baseline criteria for research projects, see http://hdl.handle.net/10261/160086. Technical report
57. Httermann, M.: Devops for developers Apress (2012)
58. EOSC-Hub: "Integrating and managing services for the European Open Science Cloud" Funded by H2020 research and innovation pr ogramme under grant agreement No. 777536. See http://eosc-hub.eu (2018)
59. Apache License: author = https://www.apache.org/licenses/LICENSE-2.0 (2004)
60. INDIGO Package Repo: http://repo.indigo-datacloud.eu/ (2017)
61. INDIGO DockerHub: https://hub.docker.com/u/indigodatacloud/ (2015)
62. Indigo gitbook: https://indigo-dc.gitbooks.io/indigo-datacloud-releases (2017)
63. Van Zundert, G.C., Bonvin, A.M.: Disvis: quantifying and visualizing the accessible interaction space of distance restrained biomolecular complexes. Bioinformatics **31**(19), 3222–3224 (2015)
64. Van Zundert, G.C., Bonvin, A.M.: Fast and sensitive rigid–body fitting into cryo–em density maps with powerfit. AIMS Biophys. **2**(0273), 73–87 (2015)