INDIGO-Datacloud: foundations and architectural description of a Platform as a Service oriented to scientific computing

D. Salomoni¹, I. Campos², L. Gaido³, G. Donvito⁴, P. Fuhrman⁵, J. Marco², A. López-García², P. Orviz², I. Blanquez⁶, G. Moltò⁶, M. Plociennik⁷, M. Owsiak⁷, M. Urbaniak⁷, M. Hardt⁸, A. Ceccanti¹, B. Wegh⁸, J. Gomes⁹, M. David⁹, C. Aftimiei¹, L. Dutka¹³, S. Fiore¹⁰, G. Aloisio¹⁰, R. Barbera¹¹, R. Bruno¹¹, M. Fargetta¹¹, E. Giorgio¹¹, S. Reynaud¹², and L. Schwarz¹²

¹INFN Division CNAF (Bologna), Italy
²IFCA, Consejo Superior de Investigaciones Científicas-CSIC
³INFN Division of Torino, Italy
⁴INFN Division of Bari, Italy
⁵Deutsches Elektronen Synchrotron (DESY), Germany
⁶Universidad Politecnica de Valencia, Spain
⁷PSNC IBCh PAS, Poland
⁸Karlsruhe Institute of Technology (KIT), Germany
⁹Laboratory of Instrumentation and Particles (LIP), Portugal
¹⁰Fondazione Centro Euro-Mediterraneo sui Cambiamenti Climatici , Lecce, Italy
¹¹INFN Division of Catania, Italy
¹²CC-IN2P3 / CNRS , France
¹³Cyfronet AGH, Poland

April 4, 2016

Abstract

In this paper we describe the architecture of a Platform as a Service (PaaS) oriented to computing and data analysis. In order to clarify the choices we made, we explain the features using practical examples, applied to several known usage patterns in the area of HEP computing. The proposed architecture is devised to provide researchers with a unified view of distributed computing infrastructures, focusing in facilitating seamless access. In this respect the Platform is able to profit from the most recent developments for computing and processing large amounts of data, and to exploit current storage and preservation technologies, with the appropriate mechanisms to ensure security and privacy.
1 Introduction

In the past decade, European research institutions, scientific collaborations and resource providers have been involved in the development of software frameworks that eventually led to the set-up of unprecedented distributed e-infrastructures such as the European Grid Infrastructure (EGI) [1]; their collaboration made it possible to produce, store and analyze Petabytes of research data through hundreds of thousands of compute processors, in a way that has been instrumental for scientific research and discovery worldwide.

This is particularly true in the area of High Energy Physics, where distributed computing has been instrumental as data analysis technology in the framework of the LHC computing Grid [2].

New technological advancements, such as virtualization and cloud computing, pose new important challenges when it comes to exploit scientific computing resources. Services to researchers need to evolve in parallel to maximize effectiveness and efficiency, in order to satisfy different scenarios: from everyday necessities to new complex requirements coming from diverse scientific communities.

In order to expose to researchers the power of current technological capabilities, a key challenge continues to be accessibility. Virtualization techniques have the potential to make available unprecedented amounts of resources to researchers. However, serving scientific computing is much more complex than providing access to a virtualized machine. It implies being able to support in a secure way complex simulations, data transfer & analysis scenarios, which are by definition in constant evolution.

This is particularly the case of the HEP areas, where we can distinguish three paradigmatic usage scenarios:

- Massive data analysis at the LHC and experiments which in general must deal with analysis of large amounts of data in computing farms. The paradigmatic case is the data analysis of the LHC at experiments like CMS and ATLAS. During LHC Run 1, the tools developed by WLCG oriented to Grid computing worked very smoothly. However, both the amount of data involved in Run 2 and beyond, and the evolution of the computing infrastructures implies that new scenarios for accessing resources need to be devised. The experiments CMS and ATLAS have both already analyzed the technical feasibility of using resources in cloud mode (see [3, 4]).

- High Performance Computing in facilities with low latency interconnects dedicated to Monte Carlo Simulations, for example for lattice QCD. Resource requirements for development phases are medium size clusters with up to a few hundred cores; in the production phase, Lattice QCD is served by large HPC farms which can require from a few thousand, up to tens of thousands of cores for ground breaking projects. The storage requirements are in the order of a few Terabytes for development and Petabyte for production phases [5].
• Phenomenological simulation & prediction codes, often including legacy software components and complex library dependencies. The resources requirements can reach about a thousand cores, with storage on the range of the Terabyte. Here one current show stopper for researchers is the possibility of having installed the right environment (in terms of legacy libraries for example) [7].

Making computing and storage resources really exploitable by scientist implies adapting access and usage procedures to the user needs. The challenge that remains is developing such procedures in a manner which is reliable, secure, sustainable, and with the guarantee that results are reproducible.

In order to settle the scenario where our Platform is to be deployed, we must specify what we mean by the term resources. By this we always refer to a large pool of computing and storage resources, in which the users do not need to know (because it is not relevant for the computation) which machine it is actually being used. If the computing requirements would have a strong hardware dependency (optimized for a particular processor), or if the application is so large scale that a significant fraction of the available resources in a given computing center needs to be used, the discussion would be completely different.

This is context of the work of the HORIZON 2020 project INDIGO-DataCloud [3], addressing the challenge of developing advanced software layers, deployable in the form of a data/computing platform, targeted at scientific communities.

The technological developments required to respond to such a challenge need to be addressed in a consistent way at the different layers that compose a scientific computing infrastructure.

First, at the Infrastructure provider level computing centers offer resources in an Infrastructure as a Service (IaaS) mode. The main challenges to be addressed at this level are:

1. Improved scheduling for allocation of resources by popular open source Cloud platforms, i.e. OpenStack and OpenNebula.

   In particular, both better scheduling algorithms and support for spot-instances are currently much needed. The latter are in particular needed to support allocation mechanisms similar to those available on commercial clouds such as Amazon and Google, while the former are useful to address the fact that typically all computing resources in scientific data centers are always in use.

   For LHC data analysis several attempts at using such spot-instances in the framework of commercial cloud providers have already taken place in the experiment CMS [4] and ATLAS [5]. Having an open source software framework supporting such operation mode at the IaaS level will therefore be very important for the centers aiming to support such data analysis during LHC Run 2 and beyond.

2. Improved Quality of Service (QoS) capabilities of storage resources. The challenge here is to develop a better support for high-level storage require-
ments, such as flexible allocation of disk or tape storage space and support for data life cycle.

The impact of such QoS when applied to storage interfaces such as dCache will be obvious for LHC Data analysis and for the long-term support, preservation and access of experiment data.

3. Improved capabilities for networking support. This is particularly the case when it comes to deploy tailored network configurations in the framework of OpenNebula and OpenStack.

4. Improved and transparent support for Docker containers.
   Containers provide an easy and efficient way to encapsulate and transport applications. Indeed, they represent a higher level of abstraction than the crude concept of a "virtual machine". The benefits of using containers, in terms of easiness in the deployment of specialized software, including contextualization features, eg. for phenomenology applications, are clear. They offer obvious advantages in terms of performance when compared with virtual machines, while also opening the door to exploit specialized hardware such as GPGPUs and low-latency interconnection interfaces (InfiniBand).

The general idea here is to make containers “first-class citizens” in scientific computing infrastructures.

In the next layer we find the PaaS layer. This is a set of services whose objective is to leverage disparate hardware resources coming from the IaaS level (Grid of distributed clusters, public and private clouds, HPC systems) to enhance the user experience.

In this context a PaaS should provide advanced tools for computing and for processing large amounts of data, and to exploit current storage and preservation technologies, with the appropriate mechanisms to ensure security and privacy. The following points describe the most important missing capabilities which today require further developments:

1. Improved capabilities in the geographical exploitation of Cloud resources. End users need not to know where resources are located, because the PaaS layer should be hiding the complexity of both scheduling and brokering.

2. Support for data requirements in Cloud resource allocations. Resources can be allocated where data is stored, therefore facilitating interactive processing of data. The benefits of such an enhancement are clear for software stacks for interactive data processing tools such as ROOT \cite{18} and PROOF \cite{19}.

3. Support for applications requirements in Cloud resource allocations. For example, a given user can request to deploy a cluster with Infiniband interfaces, or with access to specialized hardware such as GPGPUs. Elasticity
in the provisioning of such specialized small size clusters for development purposes would have a great impact in the everyday work of many researchers in the area of Lattice QCD for example.

4. Transparent client-side import/export of distributed Cloud data.

5. Deployment, monitoring and automatic scalability of existing applications, including batch systems on-demand. For example, existing applications such as web front-ends, PROOF clusters or even a complete batch system cluster (with appropriate user interfaces) can be automatically and dynamically deployed in highly-available and scalable configurations.

6. Integrated support for high-performance Big Data analytics and workflow engines such as Taverna [15], Ophidia [16] or Spark [17].

7. Support for dynamic and elastic clusters of resources.

In the next layer we find the user interface, which is responsible to convey all the above mentioned developments to the user. This means in particular that it should provide ready-to-use tools for such capabilities to be exploited, with the smoothest possible learning curve.

Providing such an interface between the user and the infrastructure poses two fundamental challenges:

1. Enabling infrastructure services to accept state of the art user authentication mechanisms (e.g. Open ID connect, SAML) on top of the already existing X.509 technology. For example, distributed authorization policies are badly needed in scientific cloud computing environments, therefore a dedicated development effort is needed in this area. Hence, the Authentication and Authorization Infrastructure (AAI) is a key ingredient to be fed into the architecture.

2. Making available the appropriate libraries, servlets and portlets, implementing the different functionalities of the platform (AAI, data access, job processing, etc.) that are the basis to integrate such services with known user tools, portals and mobile applications.

In this paper we present an architectural design to satisfy the challenges mentioned above, and which we are developing in the framework of the INDIGO project. We highlight the interrelations among the different components involved and outline the reasoning behind the choices that were made.

The remainder of this paper is structured as follows.

In order to motivate our choices we first describe several generic user scenarios in Section 2 with the main functionalities we want to satisfy, offering a global, high-level view of the INDIGO architecture highlighting the interrelations among the different layers. Section 3 describes the architecture of the PaaS layer, together with the description of the user interfaces to be developed. Section secinfrastructure is devoted to the computer center layer of the architecture. In particular, it describes the middleware choices and developments
Section 5 describes how INDIGO interfaces with users. Finally, Section 6 shows the whole workflow and the interactions among all the services composing the INDIGO Architecture, by illustrating a few practical examples.

The paper is concluded by Section 7, drawing some conclusions and highlighting future work.

2 Motivation and High Level view of the Architecture

Our aim is to design an architecture containing the elements needed to provide scientific users with the capability of using heterogeneous infrastructures in a seamless way.

Therefore, as a first step we have performed a detailed user requirements analysis, whose main conclusions we sketch here in the form of two generic scenarios: the first is computing oriented, while the second is data analysis oriented. For full details containing user communities description and detailed usage patterns we refer to our requirements document [20].

2.1 Pilot user scenarios

Our architecture is based on the analysis of a number of use cases originating from different research communities in the areas of High Energy Physics, Environmental modelling, Bioinformatics, Astrophysics, Social sciences and others. From this requirements analysis we have extracted two generic usage scenarios, which can support a wide range of applications in these areas.

The first generic user scenario is a computing portal service. In such scenario, computing applications are stored by the application developers in repositories
Figure 2: Data Analysis Service

as downloadable images (in the form of VMs or of lightweight 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.

The number of nodes available for computing should scale up and down, 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 Figure 1.

A second generic use case is described by scientific communities that have a coordinated set of data repositories and software services (for example PROOF, or R-studio) to access, process and inspect them. Processing is typically interactive, requiring access to a console deployed on the data premises. In Figure 2 we show a schematic view of such a use case.

As pointed out in the introduction, the current technology based on lightweight containers and related virtualization developments make it possible to design software layers in the form of platforms that support such usage scenarios in a relatively straightforward way.

We can see already many examples in the industrial sector, in which open source PaaS solutions such as OpenShift or CloudFoundry are being deployed to support enterprise work in different sectors [12].

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

2.2 PaaS layers over production scientific e-Infrastructures

For the architecture to go beyond just a theoretical implementation of tools and APIs, we must include the practicalities of the computing centers in the
discussion. The architecture should be capable of supporting the interaction with the resource centers via standard interfaces. Here the word standard is meant in a very wide sense including *de jure* as well as *de facto* standards.

Virtualization of resources is the key word in order to properly address the interface with the resource centers. In other words, the software stack should be able to virtualize local compute, storage and networking IaaS resources, providing those resources in a standardized, reliable and performing way to remote customers or to higher level federated services.

The IaaS layer is normally provided to scientist 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. Therefore, the success of any software layer in this respect is being able to be flexible enough as to interact with the most popular choices of the computer centers, without interfering, or very minimally, in their normal operations.

As a consequence, as a part of the development effort, we have analyzed a selection of the most prominent components to interface computing and storage in the resource centers, and develop the appropriate interfaces to high-level services based on standards. Figure 3 shows a schematic view of the interrelation among those components.

The PaaS core components will be deployed as a suite of small services using the concept of “micro-service” . This term refers to a software architecture style, in which complex applications are composed of small independent processes communicating with each other via lightweight mechanisms like HTTP resource APIs. The modularity of micro-services makes the approach highly desirable for architectural design of complex systems, where many developers are involved.

Kubernetes [13], an open source platform to orchestrate and manage Docker containers, will be used to coordinate the micro-services in the PaaS. Kubernetes is extremely useful for the monitoring and scaling of the services, and will ensure the reliability of all of them. In Figure 4 we show the high-level view of the PaaS in which the interrelations among services are also indicated with arrows.

The following list briefly describes the key components of the INDIGO PaaS:

- the Orchestrator: this is the core component of the PaaS layer. It receives high-level deployment requests from the user interface software layer, and coordinates the deployment process over the IaaS platforms;

- the Identity and Access Management (IAM) Service: it provides a layer where identities, enrolment, group membership, attributes and policies to access distributed resources and services can be managed in a homogeneous and interoperable way;

- the Monitoring Service: this component is in charge of collecting monitoring data from the targeted clouds, analysing and transforming them into information to be consumed by the Orchestrator;
• the Brokering/Policy Service: this is a rule-based engine that allows to manage the ranking among the resources that are available to fulfil the requested services. The Orchestrator will provide the list of IaaS instances and their properties to the Rule Engine. The Rule Engine will then be able to use these properties in order to choose the best site that could support the users' requirements. The Rule Engine can be configured with different rules in order to customize the ranking;

• the QoS/SLA Management Service: it allows the handshake between a user and a site on a given SLA; moreover, it describes the QoS that a specific user/group has, both over a given site or generally in the PaaS as a whole. This includes a priority for a given user, i.e. the capability to access different levels of QoS at each site (e.g., Gold, Silver, Bronze services);

• the Managed Service/Application (MSA) Deployment Service: it is in charge of scheduling, spawning, executing and monitoring applications and services on a distributed infrastructure; the core of this component consists of an elastic Mesos cluster with slave nodes dynamically provisioned and distributed on the IaaS sites;

• the Infrastructure Manager (IM): it deploys complex and customized virtual infrastructures on IaaS Cloud deployment providing an abstraction layer to define and provision resources in different clouds and virtualization platforms;

• the Data Management Services: this is a collection of services that provide an abstraction layer for accessing data storage in a unified and federated way. These services will also provide the capabilities of importing data, schedule transfers of data, provide a unified view on QoS and distributed Data Life Cycle Management.
Figure 4: Key components of the PaaS and their high-level interrelations
3 Architecture of the PaaS Layer

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.

In the INDIGO approach, the PaaS will deal with the instantiation of services and with application execution upon user requests relying on the concept of micro-services. In turn, the micro-services will be managed using Kubernetes, in order, for example, to select the right end-point for the deployment of applications or services. Cross-site deployments will also be possible.

The language in which the PaaS is going to receive end user requests is TOSCA [21] (Topology and Orchestration Specification for Cloud Applications). It is an OASIS specification for the interoperable description of application and infrastructure cloud services, the relationships between parts of these services, and their operational behaviour.

TOSCA has been selected as the language for describing applications, due to the wide-ranging adoption of this standard, and since it can be used as the orchestration language for both OpenNebula [22] (through the IM [34]) and OpenStack [23] (through Heat [24]).

The PaaS Core provides an entry point to its functionality via the Orchestrator service, which will feature a RESTful API that receives a TOSCA-compliant description of the application architecture to be deployed. Providing such TOSCA-compliance will enhance interoperability with existing and prospective software.

Users can choose between accessing the PaaS core directly or using a Graphical User Interface or simple APIs. A user authenticated on the INDIGO Platform will be able to access and customize a rich set of TOSCA-compliant templates through a GUI-based portlet.

The INDIGO repository will provide 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:

- Scenario A. Deploy a customized virtual infrastructure starting from a TOSCA template that has been imported, or built from scratch (see Figure 5). The user will be able to access the deployed customized virtual infrastructure and run/administer/manage applications running on it.

- Scenario B. Deploy a service/application whose life-cycle will be directly managed by the PaaS platform (see Figure 6). The user will be returned the list of endpoints to access the deployed services.

In both cases the selected template will be submitted to the PaaS Orchestrator using its REST API endpoint. Then, the Orchestrator will collect all the information needed to generate the deployment workflow:
• Health status and capabilities of the underlying IaaS platforms and their resource availability from the Monitoring Service;

• Priority list of sites sorted by the Brokering/Policy Service on the basis of rules defined per user/group/use-case;

• QoS/SLA constraints from the SLA Management System;

• The status of the data files and storage resources needed by the service/application and managed by the Data Management Service.

This information will be used to perform the matchmaking process and to decide where to deploy each service. Note that the Orchestrator will be able to trigger the data migration function provided by the Data Management Service component if the data location does not meet the application deployment requirements.

As pointed out before, we rely on the Mesos architecture to manage the distributed set of IaaS resources. The architecture of a Mesos cluster \[26\] consists of one or more master nodes, and of slave nodes that register with the master and offer resources from the IaaS nodes. The master node is aware of the state of the whole IaaS resources, and can share and assign them to the different applications (called frameworks in the Mesos terminology) according to a pluggable scheduling policy (fair share, strict priority, etc.).

The Automatic Scaling Service, based on EC3/CLUES \[27\], will ensure the elasticity and scalability of the Mesos cluster by monitoring its status. When additional computing resources (worker nodes) are needed, the Orchestrator will be requested to deploy them on the underlying IaaS matching the QoS/SLA, health and user/group/use-case policies selected by the Broker.

In the case of Long Running Services, the Management Service/Application (MSA) deployment Service will use Marathon (an already available Mesos framework) to ensure that the services are always up and running. Marathon is able to restart the services, migrate them if problems occur, handle their mutual dependencies and load-balancing, etc.

The MSA Deployment Service will also use Chronos (another already available Mesos framework) to execute applications having input/output requirements or dependencies. It may also handle the rescheduling of failed applications, or simple workflows composed by different applications.

By leveraging the Mesos plugin-based architecture, new frameworks can be developed, such as one able to deploy a batch cluster (e.g. HTCondor) on demand, in order to meet specific use-cases. For example, batch execution of LHC data analysis is often using HTCondor to manage the job scheduling \[8\].

With respect to Data Management Services, some interfaces will be provided to advanced users for specific data management tasks.

First of all, the OneData \[28\] component will provide several features: a web-based interface for managing user spaces (virtual folders) and controlling access rights to files on a fine-grained level, a Posix interface to a unified file system namespace addressing both local and remote data, caching capabilities and
Deployment of Customized Virtual Infrastructures using INDIGO-DataCloud Orchestrator Service

Brokering/Policy Service

QoS/SLA Service

Managed Services/Applications (MSA) Service

Service Monitoring Service

Data Management Services

Infrastructure Manager

TOSCA-compliant Templates

Repository

IAM Service

Figure 5: Deployment of a customized virtual infrastructure: When a customized virtual infrastructure deployment is requested (scenario A), the Orchestrator manages the instantiation and configuration of the required resources (e.g., virtual machines) on the selected IaaS infrastructure using the REST APIs exposed by the IaaS orchestrator (i.e., Heat or IM) of the INDIGO sites or delegating the interaction with external clouds to a dedicated instance of the IM.
Figure 6: Deployment of a managed service/application: When a managed PaaS service deployment is requested (scenario B), the Orchestrator interacts with the Managed Service/Application (MSA) Deployment Service in order to supervise its deployment on the elastic Mesos cluster that will host the user application/service.
the possibility to map object stores such as S3 \[35\] to Posix filesystems. Additionally, the FTS-3 \[24\] service will provide a web-based interface for monitoring and scheduling large data transfers.

Furthermore, all the standard interfaces exposed by the data management components will be accessible to users’ applications as well through standard protocols such as CDMI \[30\] and WebDAV \[31\].

4 Architectural impact on the Infrastructure

The impact of the implementation of INDIGO software developments at the level of infrastructure resource providers is a key discussion to guarantee the adoption of the solutions being developed.

A successful architecture should be able to provide the means to unify the interfaces between the PaaS layer and the core services. This is necessary, as resources sites have already their own administration software installed. A PaaS, like any software layer dealing with resource management, needs to be totally customizable to guarantee a good level of adoption by infrastructure providers.

Therefore our strategy is to focus on the most popular standards and provide well-documented common interfaces to these. Examples are OCCI\[9\], TOSCA or a consistent container support for OpenStack and OpenNebula. An example in the data area is the support of CDMI for the various storage systems like dCache, StoRM, HPSS and GPFS.

A closely related goal, often being the result of the topic discussed previously, is the functional unification between different software systems. For instance, an option for providing TOSCA for OpenNebula is to port the OpenStack Heat TOSCA translator to the chosen OpenNebula orchestration tool.

Another area of development which is currently demanded by scientific communities is the introduction of “Quality of Service” and “Data Life-cycle Policies” in the data area. This is the result of the various ”Data Management Plans, DMP” provided by data intensive communities and also required by the EU when submitting proposals. One important aspect of the DMPs is the handling of quality of service and access control of precious and irreproducible data over time, resulting in support and manageability of those attributes at the site or storage system level.

Although the different types of resources are closely interlinked, we distinguish between Computing, Storage and Network resources for organizational reasons.

In the computing area the provision of standard APIs is covered by supporting OCCI at the lowest resource management level and TOSCA at the infrastructure orchestration level.

Within the storage area, common access and control mechanisms are evaluated for negotiating data quality properties, e.g. access latency and retention policies, as well as the orchestration of data life cycles for archival. Together with established standardization bodies, like RDA\[11\] and OGF\[10\], we envision to extend the SNIA CDMI protocol\[30\] for our purposes.
Similarly for Networking, we need to evaluate commonalities in the use of Software Defined Networks (SDN) between different vendors of network appliances.

One notably attractive concept is that all features developed at the IaaS level will not only be available through the INDIGO PaaS layer, but can be utilized by users accessing the IaaS layer directly.

Similarly, tracking of user identities is available throughout the entire execution stack. Consequently, users can be monitored down to the IaaS layer with the original identities they provided to portals or workflow engines when logged via the PaaS INDIGO layer.

4.1 Software to interface the Resource Centers with the PaaS

Based on the scientific use cases we have considered (see [20]), we identified a set of features that, if supported, have the potential to impact in a positive way the usability and easy access to the Infrastructure layers.

In the computing area, these features are enhanced support for containers, integration of batch systems, including access to hardware specific features like InfiniBand and General Purpose GPUs, support for trusted container repositories, introduction of spot instances and fair-share scheduling for selected Cloud Management Frameworks (CMF), as well as orchestration capabilities common to INDIGO selected CMFs using TOSCA. See Figure 7 for a graphical representation.

In certain applications, the use of ‘Containers’ as a lightweight alternative to hypervisor-based virtualization is becoming extremely popular, due to their significantly lower overhead. However, support in major Cloud Management Frameworks (CMFs) is still under development or does not exist at all.

For OpenStack and OpenNebula, the top two CMF’s on the market, INDIGO, in collaboration with the corresponding Open Source communities, is spending significant efforts to make containers first-class citizens and, concerning APIs and management, indistinguishable from traditional VMs.

While in OpenStack only adjustments are necessary, for OpenNebula more work is required. Although both CMFs offer highly optimized proprietary interfaces, INDIGO will make as much functionality as possible available through the OCCI standard, allowing uniform access from higher level services to both systems.

Although cloud-like access to resources is becoming popular and cloud middleware is being widely deployed, traditional scientific data centers still provide their computational power by means of Batch Systems for HTC and HPC. Consequently, it is interesting to facilitate the integration of containers in batch systems, providing users with the ability to execute large workloads embedded inside a container. In the same context, we also need to explore access to specialized hardware, deployed in high-end data centers, such as InfiniBand interconnects and GPGPUs.
With the pressure of optimizing computer center resources but at the same time providing fair, traceable and legally reproducible services to customers, available cloud schedulers need to be improved. Therefore, we are focusing on the support of spot-instances allowing brokering resources based on SLAs and prices. Technically this feature requires the CMF to be able to preempt active instances based on priorities.

On the other hand, to guarantee an agreed usage of compute cycles integrated over a time interval, we need to invest in the evaluation and development of fair-share schedulers integrated in CMFs. This requires a precise recording of already used cycles and the corresponding readjustment of permitted current and future usage per individual or group. The combination of both features allows resource providers to partition their resources in a dynamic way, ensuring an optimized utilization of their infrastructures.

The middleware will also provide local site orchestration features by adopting the TOSCA standard in both OpenStack and OpenNebula, with similar and comparable functionalities.

Finally, resource orchestration is also covered within this architecture. Although this area can be managed at a higher level, we will provide compute, network and storage resource orchestration by means of the TOSCA language standard at the IaaS level as well. As a result, both, the upper platform layer and the infrastructure user may deploy and manage complex configurations of resources more easily.

While in the cloud computing area, the specification of service qualities,
e.g. number and power of CPU’s, the amount of RAM and the performance of network interfaces, is already common sense, negotiating fine grained quality of service in the storage area, in a uniquely defined way, is not offered yet.

Therefore, the high level objective of the storage area is to establish a standardized interface for the management of Quality of Services (QoS) and Data Life Cycle in Storage (DLC). Users of e-infrastructures will be enabled to query and control properties of storage areas, like access latency, retention policy and migration policies with one standardized interface. A graphical representation of the components is shown in Figure 8.

To engage scientific communities into this endeavour as early as possible, INDIGO initiated a working group within the framework of the Data Research Alliance [11] (RDA), and will incorporate ideas and suggestions of that group at any stage of the project into the development of the system.

As with all infrastructure services, the interface is supposed to be used by either the PaaS storage federation layer or by user applications utilizing the infrastructure directly.

This will be pursued in a component-wise approach. Development will focus on QoS and interfaces for existing storage components and transfer protocols that are available at the computer centers. Ideally, the Storage QoS component can be integrated just like another additional component into existing infrastructures.

Besides providing the translation layer between the management interface and the underlying storage technologies, the software stack needs to be integrated into an existing local infrastructure.

One of the major issues here is to map different identities to the same individual or functional account, i.e. an Authentication and Authorization Infrastructure needs to be supported at the middleware level. Such Identity Harmonization is described in our AAI Architecture Document.

As principle technology for authentication and authorization among virtualized resources a Macaroon [35] based service infrastructure will be developed, incorporating Identity Providers (IdPs) and Identity Access Management (IAM) systems supporting multiple identities (SAML, X.509) per user. More details on state-of-the-art concepts and technologies can be found here.

Macaroons are flexible authorization credentials for Cloud services introduced by Google [35] that support decentralized delegation between principals. Macaroons are based on a construction that uses nested, chained Message Authentication Codes (MAC) involving a cryptographic hash function (they are hence called HMACs) in a manner that is highly efficient, easy to deploy, and widely applicable.

Although macaroons are bearer credentials, like Web cookies, macaroons embed caveats that attenuate and contextually confine when, where, by whom, and for what purpose a target service should authorize requests.

The high level objective of the network area is to provide mechanisms for

---

1 See https://owncloud.indigo-datacloud.eu/index.php/s/Ur70RPKM8IF0adQ
2 See https://owncloud.indigo-datacloud.eu/index.php/s/sUTRpymjANAX0Hd
Figure 8: Storage services from the INDIGO architecture perspective
orchestrating local and federated network topologies. To unify the orchestration management, we will ensure that the network part of the OCCI open standard can be used in INDIGO supported CMFs: OpenStack and OpenNebula.

5 Interfacing with the user

The upper layer of the INDIGO architecture addresses the challenge of guaranteeing a simple and effective final usage, both for software developers and application running.

We will provide the tools needed for the development of APIs to access the INDIGO PaaS framework. It is via such APIs that the PaaS features can be exploited via Portals, Desktop Applications or Mobile Apps. Therefore, the main goals of such endeavour from a high level perspective are to:

- Provide User Friendly front ends demonstrating the usability of the PaaS services;
- Manage the execution of complex workflows using PaaS services;
- Develop Toolkits (libraries) that will allow the exploitation of PaaS services at the level of Scientific Gateways, desktop and mobile applications;
- Develop an Open Source Mobile Application Toolkit that will be the base for development of Mobile Apps (applied, for example, to a use case provided by the Climate Change community).

The architectural elements of the user interface (see Figure 9) can be described as follows:

- FutureGateway Portal: it provides the main web front-end, enabling most of the operations on the e-infrastructure. A general-purpose instance of the Portal will be available to all users.

Advanced features that can be included via portlets are:

- Big Data Portlets for Analytics Workflows - aiming at supporting a key set of functionalities regarding big data and metadata management, as well as support for both interactive and batch analytics workflows.
- Admin Portlet - to provide the web portal administrator with a convenient set of tools to manage access of users to resources.
- Workflow Portlets - to show the status of the workflows being run on the infrastructure, and provide the basic operations.

- FutureGateway Engine: it is a service intermediating the communication between e-Infrastructures and the other user services developed. It incorporates many of the functionalities provided by the Catania Science
Gateway Framework [32], extended by others specific to INDIGO. It exposes a simple RESTful API for developers building portals, mobile and desktop applications. Full details can be found in [37].

- Scientific Workflows Systems: these are the scientific workflow management systems orchestrating data and job flow. We have selected Ophidia, Galaxy, LONI and Kepler as the ones more demanded by the user communities.

- Wfms plug-ins – these are plug-ins for the Scientific Workflow Systems that will make use of the FutureGateway Engine REST API, and will provide the most common set of the functionalities. These plug-ins will be called differently depending on the Scientific Workflow system (modules, plug-ins, actors, components).

- Open Mobile Toolkit: these are libraries that make use of the FutureGateway Engine REST API, providing the most common set of the functionalities that can be used by multiple domain-specific mobile applications running on different platforms. Foreseen libraries include support for iOS and Android and, if required, for WindowsPhone implementations.

- INDIGO Token Translation Service Client - The Token Translation Service client enables clients that do not support the INDIGO-token to use the INDIGO AAI architecture.

A key component, with a big impact on the end-user experience, is the IAM service. This service provides user identity so that consistent authorization decisions can be enforced across distributed services. The IAM will be used internally to the PaaS layer in order to deal with the authorization of each
user to the services, but also in order to deal with group membership and role management for each user. In particular it provides a layer where identities, enrolment, group membership and other attributes, and authorization policies on distributed resources can be managed in a homogeneous way across the supported federated authentication mechanisms.

Users may present themselves with any of the three supported token types (X.509, OpenID Connect, SAML) and will be able to access resources that support any of them. All the interaction across all the services will require an authentication based on a well-scoped token. The IAM service will also provide the basic for building the Token Translation Service.

6 Interactions among services

In order to further clarify the architectural design, in this section we provide examples describing the interaction among the services in the INDIGO platform.

Starting from the user interface layer the most typical example of interaction could be represented as follows (see Figure 10).

A user interacts with the FutureGateway portal. Authentication is achieved via the IAM service, which generates a redirection to an institutional Identity Provider (for example, the user’s home University). This method is very convenient because the end user can exploit his own home credentials, without the need to apply for new accounts or certificates.

After this process, the user will be provided with an INDIGO Token, which can be used in order to transport all the needed information about him, and to login on all the INDIGO services.

In order to start a service/application request, the end user should describe the use case providing the needed customization parameters as a TOSCA template. A number of basic templates will be made available to describe basic use cases. The template describes in detail which computational/storage/network resources are needed to instantiate the requested service/application, with information about possible dependencies among them.

At this point, the user interface layer sends the TOSCA Template to the PaaS Orchestrator.

The PaaS Orchestrator will take care of the request, updating the user layer with the detailed status of the deployment. It will also analyze the content of the TOSCA Template understanding the requirements in terms of services and resources. The PaaS orchestrator will interact with the micro-services of the PaaS Layer in order to find the IaaS resources that are able to fulfil the user request (see Figure 11).

This interaction among the PaaS Orchestrator and the other micro-services is done via REST APIs, with the approach that each service is providing a REST web service, and the Orchestrator able to interact with each service as a client.

In particular:
Figure 10: High-level representation of the support to basic user interactions

Figure 11: Interaction between the PaaS orchestrator and the micro-services
1. The monitoring micro-service is responsible for providing aggregated information about the status of IaaS resources and the information about the supported features by each of the endpoints.

2. The QoS/SLA micro-service is responsible for mapping SLA requests between the user and the resource provider. These services will be able to provide this information to the orchestrator. The QoS/SLA micro-service will also be able to monitor an SLA violation, taking into account information gathered by the Accounting Service and the monitoring infrastructure.

3. The Data Services will be used by the PaaS Orchestrator in order to define where the application should be deployed taking into account data location; alternatively, the Orchestrator can ask for data import/movement. More precisely, there will be three different data services (Onedata, FTS and Dynafed), responsible for:
   - Providing information about data location (Onedata, DynaFed).
   - Provide data import features (FTS).
   - Remote data access facilities (Onedata, DynaFed).
   - Data movement (FTS, Onedata).

4. The Policy Management micro-service is responsible for ranking the IaaS that are able to fulfill the users’ requirements. In this way, each time there is more than one IaaS available to deploy the requested service/application, the orchestrator could be provided with a rank of the resources fulfilling the rules configured in this service.

   If the selected IaaS is able to deal directly with a TOSCA Template, the Orchestrator can request the execution of such a template directly to the IaaS Heat/IM endpoint.

   Whenever this is not possible, the orchestrator will send the TOSCA Template to the Infrastructure Manager (IM) at the level of PaaS layer; the IM will then be able to “translate” it into the cloud native APIs (see Figure 11).

   In this way it will be possible to exploit also legacy cloud infrastructures that will be enabled with a TOSCA compliant orchestrator.

6.1 Data handling and user interaction

Depending on how data are stored/accessible, they will be made available through different services (see Figure 12).

   If data are available only via a Webdav gateway, they can be aggregated using DynaFed. Users will be able to read them via a federation layer regardless of where the data are really stored.

   If data are stored on a POSIX-compliant filesystem and the site admin is willing to install the Onedata gateway, data can be accessed with a more powerful interface, including:
Figure 12: Data access high-level description and involved services

- ACLs and QoS management.
- Posix access (also remotely).
- Web or webdav access.
- Simple metadata management (based on the concept of key-value pair).
- Data movement, replication and caching across the site of the federation.
- Transparent local caching of data.
- Transparent translation of object storage resources (for example available through an S3 interface) to a Posix filesystem.

The FTS service will be exploited for the capability of managing third-party transfers among griftp servers. It will be used in order to import data from external gridftp servers, globus-online compatible services, etc.

In the TOSCA Template the user will provide information about the data needed to execute the desired service/application, and also how data should be accessed.

Given the data requirements described in the TOSCA Template, the orchestrator will be able to understand if it has to request either FTS or Onedata to schedule a data import/movement, or if instead it is better to move the application “close” to the data.

At the end of this cycle the data will be available to the end user service exploiting Onedata or DynaFed. This will allow also legacy application/services to be supported with their native data access approach (Webdav or Posix).

In order to access and manage data, we will exploit the interfaces provided by the infrastructure layer:
Figure 13: Data handling possibilities from the user viewpoint

- Posix and Webdav for data access.
- GridFTP for data transfer.
- CDMI for the Metadata Management.
- REST APIs to expose QoS features of the underline storage.

In summary (see Figure 13), an end user will have to possibility to handle data in several ways:

- Asking for an import action using a TOSCA Template exploiting FTS and Gridftp.
- Uploading files or directories using a web interface.
- Importing data from his desktop via a Dropbox-like tool.

6.2 Enabling PaaS Services at user level

The INDIGO Architecture will provide the end-user also the capabilities of enabling typical PaaS Services. In this case, the platform will take care of monitoring the service, restarting o migrating it in case of failure. This feature will be implemented by means of a PaaS layer based on Mesos, enhanced with the usage of Marathon and Chronos (see Figure 14).

The platform will take care of automatic resource provisioning for the Mesos cluster. A component named Clues [27] will monitor the load of the cluster and request new resources to the PaaS Orchestrator, that will then provide them through the usual workflow. This will let Mesos to automatically scale the size of the cluster in order to fit users’ requirements.

In the first phase of the project the Mesos cluster will be instantiated on a given IaaS, while in the second release of the INDIGO Platform, we foresee to
have the ability to span a Mesos cluster across several IaaS instances, dealing with the complexity coming from the need of an overlay network crossing several sites.

In both cases of automated IaaS deployed services and of PaaS services deployed through Mesos, an end user could choose between Virtual Machines or Docker containers, in order to instantiate the services or execute applications. Users will be able to push their Docker containers into a trusted public repository (Docker-hub) and the platform will allow the execution of the container in an easy and transparent way.

7 Conclusions and future work

The INDIGO Architecture, based on a three level approach (User Interface, PaaS layer and Infrastructure Layer), is able to fulfil the requirements described in the introduction.

Throughout this document we have summarized the work being performed at the infrastructure, Platform and user interface level, which also includes the description and implementation of some typical use case scenarios, to provide more clarity as to what we want to support with this architectural construction.

The plan is to deliver a first release of the platform by July 2016, implementing the most important features to let users deploy their services and applications across a number of testbed provided by the INDIGO partners, and provide developers with an initial feedback.

In the automated IaaS services scenario, users have the capability of manage the full stack of the services, with the simplification of:

- finding the right IaaS to host a service.
- automatize the instantiation and configuration of a service.
• describe dependency among each service, with the possibility to create clusters.

On the other hand, the PaaS scenario has the advantage that the management of the application/services is not managed by users only. In particular, the INDIGO platform has advanced capabilities related with service resilience that make it very attractive for users.

For example, in the case of failure of one the services or of an application, the platform itself will take care of restarting the service or re-executing the application.

In the second INDIGO release, scheduled by March 2017, we plan to also integrate advanced support for features such as moving applications to Cloud infrastructures, addressing Cloud bursting, enhancing data services and providing additional sample templates to support use cases as they are presented by scientific communities. The release cycle, roadmap and key procedures of the INDIGO software are described here [39].

References

[1] For more details on the European Grid Infrastructure and software context around it see http://www.egi.eu
[2] See details in http://wlcg.web.cern.ch
[3] Project INDIGO-Datacloud
Integrating Distributed Data Infrastructures for Global Exploitation.
HORIZON 2020, Contract number RIA 653549
http://www.indigo-datacloud.eu

[4] “The CMS Tier0 goes Cloud and Grid for LHC Run 2”
D. Hufnagel et al (CMS Collaboration)
Proceedings of CHEP2015
Journal of Physics: Conference series 664 (2015) 032014

[5] “The evolution of Cloud Computing in ATLAS”
R. Taylor et al. (ATLAS Collaboration)
Proceedings of CHEP2015
Journal of Physics: Conference series 664 (2015) 022038

[6] “Computational Strategies in Lattice QCD”
M. Lüscher, Summer School on ”Modern perspectives in lattice QCD”, Les Houches, August 3-28, 2009
arxiv link: http://arxiv.org/abs/1002.4232v2 and references therein.

[7] “Phenomenology Tools on Cloud Infrastructures using OpenStack”
I. Campos, E. Fernández-del-Castillo, S. Heinemeyer, A. López, F. v.d. Pahlen
European Physical Journal C (2013) 73:2375
“PANDA analysis on IaaS clouds with Condor + Cloud Scheduler”
A. Agarwald et. al
See http://heprc.phys.uvic.ca/sites/heprc.phys.uvic.ca/files/UVic-ATLAS-ADC-2012-02-29.pdf

For the specification of the Open Cloud Computing Interface see http://occi-wg.org/about/specification/

The Open Grid Forum. See http://www.ogf.org

Research Data Alliance (see https://rd-alliance.org)

Example of OpenShift as Commercial PaaS www.openshift.com.

See http://www.infoq.com/articles/scaling-docker-with-kubernetes

See https://www.docker.com/

See http://www.taverna.org.uk

See http://ophidia.cmcc.it

See http://spark.apache.org

See https://root.cern.ch

See https://root.cern.ch/proof

See the INDIGO official deliverable in: https://www.indigo-datacloud.eu/documents/initial-requirements-research-communities-d21

See http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.html

See http://opennebula.org/

See https://www.openstack.org/

See https://wiki.openstack.org/wiki/Heat

See https://open.mesosphere.com/

For an architectural view see: https://mesos.apache.org/documentation/latest/architecture

See http://servproject.i3m.upv.es/ec3/

See https://onedata.org/

See https://svnweb.cern.ch/trac/fts3
[30] See http://www.snia.org/cdmi

[31] See https://en.wikipedia.org/wiki/WebDAV

[32] Catania Science Gateway: http://www.catania-science-gateways.it/

[33] See https://www.dcache.org/

[34] See http://www.grycap.upv.es/im/index.php

[35] “Macaroons: Cookies with Contextual Caveats for Decentralized Authorization on the cloud”
   A. Birgisson, J. Gibbs, U. Erlingsson and A. Taly
   Proceedings of the NDSS Symposium 2014
   (see http://theory.standford.edu/~ataly/Papers/macaroons.pdf)

[36] See the official INDIGO document at:
    https://www.indigo-datacloud.eu/documents/state-art-technologies-base-paas-architecture-d51

[37] See the official INDIGO document at:
    https://www.indigo-datacloud.eu/documents/software-architecture-and-work-plan-wp6-d61

[38] See https://aws.amazon.com/s3/

[39] See https://www.indigo-datacloud.eu/indigo-roadmap