INDIGO PAAS

Marica Antonacci
on behalf of WP5 team

marica.antonacci@ba.infn.it

INFN-Bari
Outline

- INDIGO PaaS overview
- PaaS components
- Usage scenarios:
  - automated IaaS
  - deployment of Long-Running Service
  - execution of user application in container
INDIGO PaaS Overview
INDIGO will implement an **advanced PaaS layer** allowing scientific communities to exploit, in a powerful and high-level way, several heterogeneous computing and data e-infrastructure such as: **IaaS Cloud, Helix Nebula, EGI Grid, EGI Federated Cloud, PRACE, HPC, EUDAT**, etc.
INDIGO PaaS: Key features

- Improved capabilities in the geographical exploitation of Cloud resources.
- Standard interface to access PaaS services.
  - INDIGO will use the TOSCA standard
- Support for data requirements in Cloud resource allocations.
  - Resources can be allocated where data is stored.
- Integrated use of resources coming from both public and private Cloud infrastructures
INDIGO PaaS: Key features

- **Distributed data federations** supporting legacy applications as well as high level capabilities for distributed QoS and Data Lifecycle Management.
  - This includes for example remote Posix access to data.

- **Transparent client-side import/export of distributed Cloud data.**
  - This supports dropbox-like mechanisms for importing and exporting data from/to the Cloud. That data can then be easily ingested by Cloud applications through the INDIGO unified data tools.

- **Support for distributed data caching mechanisms and integration** with existing storage infrastructures.
INDIGO PaaS: Key features

- Deployment, monitoring and automatic scalability of existing applications.
- Integrated support for high-performance Big Data analytics.
- Support for dynamic and elastic clusters of resources.
  - batch systems on-demand (such as HTCondor or Torque)
  - extensible application platforms (such as Apache Mesos) capable of supporting both application execution and instantiation of long-running services.
INDIGO PaaS Architecture
The PaaS Core is provided as a set of services that expose REST interfaces and interact among them via HTTP.

The PaaS Core Services have to be:
- Deployed
- Managed
- Upgraded
- Monitored
- Scaled
- Self-healed

The platform to provide such functionalities is Kubernetes.
- http://kubernetes.io

An open-source system for managing containerized applications across multiple hosts in a cluster.
The Identity and Access Management (IAM) service provides a layer where identities, enrollment, group membership and other attributes and authorization policies on distributed resources can be managed in an homogeneous way, supporting the federated authentication mechanisms (SAML, OpenID connect) supported by the INDIGO AAI.

The IAM service provides user identity and policy information to services so that consistent authorization decisions can be enforced across distributed services.
IAM Architecture

Main functionalities:

- Authentication
- Session management
- Enrollment
- Attribute and identity management
- User and group provisioning and deprovisioning
- Policy definition, distribution and enforcement
Orchestrator Service

- The Orchestrator **coordinates the deployment process** over the IaaS platforms.
- The Orchestrator collects all the information needed to deploy a service consuming others PaaS µServices APIs:
  - **Monitoring Service**: get the capabilities of the underlying IaaS platforms and their resource availability;
  - **QoS/SLA Service**: get the prioritized list of SLA per user/group
  - **CloudProviderRanker** (Rule Engine) Service: sort the list sites on the basis of rules defined per user/group/use-case;
  - **Data Management Service**: get the status of the data files and storage resources needed by the service/application
- The orchestrator **delegates** the deployment to **IM, HEAT** or **Mesos** based on the TOSCA template and the list of sites.
- **Cross-site deployments** will also be possible.
Orchestrator Service

- Built on **JBPM 6.1** (long-running workflow)
- Exposes **RESTful APIs**
- Supports the **TOSCA Simple Profile in YAML Version 1.0 specification.**

**Other INDIGO components for:**
- SLA
- Cloud Site selection
- Data Management Services
- etc

**Diagram Description:**

- **TOSCA parser** (based on Alien4Cloud)
- **DAO** (Hibernate)
- **jBPM**
- **REST API**
- **RDBMS**
- **Mesos Connectors** (for Chronos, Marathon)
- **Monitoring Connector**
- **TOSCA parser**
- **Native API**
- **Openstack4j**
- **Recipe (HEAT)**
- **Monitoring Pillar**
- **Native API**
- **Openstack**
- **ZABBIX**
Cloud Provider Ranker Service

- Provides information that will be consumed by the Orchestrator in order to properly coordinate the deployment of the required resource on the sites.

  - A WEB Service providing REST APIs to rank Cloud Providers described by a JSON blob containing:
    - Total VirtualCPUs, total VirtualRAM
    - Total Virtual Ephemeral Disk (space for instances)
    - Total Virtual Disk (block storage, e.g. Cinder)
    - In use VCPU, in use VRAM, in use VDISK, in use VEphDisk

- Ranking algorithm implemented using the largely diffused Drools Rule Engine runtime framework
The Monitoring service provides a comprehensive REST API to gather information about

- the **PaaS Core Services**
  - Using *Heapster* in the *Kubernetes* cluster.
- the **customized virtual infrastructures**
  - Using *Zabbix agents* deployed inside VMs/containers
- the **state of the sites** (leveraging EGI FedCloud approach)

Monitored information has to be exposed via a REST API to be consumed by other services
QoS/SLA Service

- Allows the **handshake** between a **user** and a **site** on a given **SLA**

- Provides the Orchestrator/Ranker with the useful information for taking the decision on tasks scheduling according to the agreed and valid SLAs

- 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 of a given users, the capability to access to different QoS at each site (Gold, Silver, Bronze services)
Accounting Service

- Accounts for resource usage on the INDIGO PaaS and provides that data to other INDIGO-DataCloud services
  - QoS/SLA service will use information gathered by the Accounting service (and the monitoring pillar) to monitor SLA violation
- Usage data is extracted from the system where the resources are used and sent to a central repository
- The repository aggregates the data from across the infrastructure to produce totals based on a number of fields - such as user, site, month, year, etc.
CMDB Service

- Indigo Configuration Management DB (CMDB)
  - Need to know what are INDIGO providers and services
  - Need to register specific data (not covered by GocDB)
- Idea comes from good practices of IT Service Management (ITIL, FitSM)
The IM is a service for the whole orchestration of virtual infrastructures and applications deployed on them, including resource provisioning, deployment, configuration, re-configuration and termination.

A configuration manager based on Ansible configures the VMs deployed by the cloud connector and installs the necessary software.

IM supports APIs from a large number of virtual platforms, making user applications cloud-agnostic.

IM has been extended in INDIGO to support the TOSCA Simple Profile in YAML Version 1.0 for infrastructure description.

Documentation: [http://www.grycap.upv.es/im](http://www.grycap.upv.es/im)
Automatic Scaling Service

- Extends EC3 CLUES adding the interfaces required to interact with the INDIGO Orchestrator

- Documentation: http://www.grycap.upv.es/ec3

- Implements the elasticity rules considering the state of the virtual cluster.

- The virtual cluster will deploy additional worker nodes as required, and integrate them on the LRMS without user intervention, in order to cope with increased workload of jobs. Worker nodes will be terminated when they are no longer required.

- Plugins are available for: SLURM, Torque/PBS, HTCondor, Mesos
This service 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 Apache **Mesos** cluster with slave nodes dynamically provisioned and distributed on the IaaS sites.

Apache Mesos provides efficient resource isolation and sharing across distributed applications (frameworks).
Data Services: Unified data access

- Unified vision of geographically distributed data set
- Data affinity
- Computation jobs started on resources close to data.
- Federated data access
  - Interoperability and OpenData
- Optimization and Data on the fly
  - when data is not staged
INDIGO PaaS Usage Scenarios
Scenario I: Deployment of Virtual Infrastructures

Deployment of Customized Virtual Infrastructures using INDIGO-DataCloud Orchestrator Service
Brokering/Policy Service
QoS/SLA Service
Managed Services/Applications (MSA) Service
Infrastructure Manager
TOSCA-compliant Templates
Repository
GUI-based Portlets

1. Access
2. Authenticate
3. Select / Customize
4. Deploy
5. Status
6. Prioritize
7. Agree
8. Transfer (Optional)
9. Delegate Deploy
10. Deliver VMs
11. Access Service/Infrastructure
12. Scale out

Virtual Infrastructure for User Application/Service

1. Access
2. Authenticate
3. Select / Customize
4. Deploy
5. Status
6. Prioritize
7. Agree
8. Transfer (Optional)
9. Delegate Deploy
10. Deliver VMs
11. Access Service/Infrastructure
12. Scale out

External Cloud

OpenNebula
OpenStack
Partner IaaS
HPC Clusters
Grid
EUDAT

Cloud APIs
CDMI
TOSCA
HEAT
POSIX/WebDav
POSIX/WebDav
CDMI

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.
Use-case: Interactive usage of a Docker container with ssh

- A Docker container is instantiated automatically after a simple request on the web portal from an end-user.
  - This will exploit a TOSCA Template through the WP5 orchestrator

- The container has a public IP address and the user (or the portal) can get access directly to it.

- Users could mount (at least interactively) a local or remote posix file-system through Onedata

- The application in the Docker container is able to simply read the files provided via web browser by the end user and to write posix files that are available to users via web browsers.

- The same Docker container could be used to execute a large list of applications in a batch-like behaviour.
Interactive usage of a Docker container with ssh

1.a.1) build, push
1.a.2) Dockerfile (commit)

1) Build
2) Stage Data
3) Deploy TOSCA
4) Access
5) Mount

Docker Container

App

SSHd

Public IP

OneDock

nova-docker

IM

Other PaaS Core Services

Orchestrator

Future Gateway API Server

WP6

WP5

WP4

Cloud Site

INDIGO-DataCloud Docker Hub Organization

TOSCA Documents and Dockerfiles per Use Case

1.b) Automated Build

INDIGO-DataCloud Champion + JRA

User

GitHub

3)

Deploy

TOSCA

2)

Stage Data

4)

Access

5)

Mount

Interactive usage of a Docker container with ssh

1.a.1) build, push
1.a.2) Dockerfile (commit)
Use-case: Interactive usage of a Portal within a Virtual Machine

- A stand-alone Galaxy server is installed automatically in a virtual machine.
- A block device used for hosting the reference data is instantiated.
- All the steps (installing Galaxy, installing applications, downloading reference data) are automatically executed by parametric scripts.
- All the needed IaaS resources are orchestrated by TOSCA Templates.

- NOTE: The procedure would be similar for containers.
Interactive usage of a Portal within a Virtual Machine

1) Create Artifacts to deploy software (Ansible roles)

2) Stage Data

3) Deploy TOSCA with Vanilla VM

4) Install / Configure

5) Mount

6) Access Web Portal

User

Orchestrator

Future Gateway API Server

Other PaaS Core Services

IM

Galaxy

Public IP

VM

OpenNebula

OpenStack

Heat

Cloud Site

Champion + JRA

TOSCA Documents Referencing Artifacts per Use Case

GitHub

ANSIBLE GALAXY

Future Gateway

Orchestrator

IM

OpenNebula

Heat

OpenStack
The web portal is instantiated, installed and configured automatically exploiting scripts and Templates.

A remote posix share is automatically mounted on the web portal

The same posix share is automatically mounted also on worker nodes

End-users can see and access the same files via simple web browsers or similar.

A **batch system** is dynamically and automatically configured via TOSCA Templates

The portal is automatically configured in order to execute job on the batch cluster

The batch cluster is **automatically scaled up & down** looking at the job load on the batch system.
A web portal that exploits a batch system to run applications
Galaxy: elastic cluster

1. Deploy Galaxy TOSCA Template
2. Provision Front-End
3. Deploy VM & Configure
4. Access Galaxy portal
5. Submit jobs to LRMS
6. Spawn additional VMs (horizontal elasticity)
7. Execute jobs
8. Terminate VMs when unused

Elastic Virtual Cluster

Cloud Site

TOSCA Templates

PaaS Orchestrator
Scenario II: Deployment of a Managed Service/application

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.
Managed Services Deployment and Applications Execution through Mesos

• **Mesos** is able to manage cluster resources (cpu, mem) providing *isolation* and *sharing* across distributed applications (frameworks)

• **Marathon** and **Chronos** are two powerful frameworks that can be deployed on top of a Mesos Cluster.

• INDIGO PaaS uses:
  • **Marathon** to deploy, monitor and scale **Long-Running services**, ensuring that they are always up and running.
  • **Chronos** to run user **applications** (jobs), taking care of fetching input data, handling dependencies among jobs, rescheduling failed jobs.
Mesos in INDIGO

- Automatic deployment through **Ansible** recipes embedded in TOSCA and HOT templates
  - All the services run in docker containers;

- **High-availability** of the cluster components:
  - Leader election among master nodes managed by Zookeeper;
  - HA Load-balancing;

- **Service discovery** through **Consul** that provides also DNS functionality and health checks;
  - services are automatically registered in Consul as soon as they are deployed on the cluster

- The external access to the deployed services is ensured through **load-balancers** in HA (unique entrypoint: cluster Virtual IP)

- Cluster **elasticity** and application **auto-scaling** through CLUES plugin
INDIGO Mesos Cluster

Elastic Mesos Cluster

- Marathon
- Mesos Master
- Mesos Worker
- Chronos

Load Balancer
- HAProxy Keepalived
- Consul-template

Load Balancer
- HAProxy Keepalived
- Consul-template

VIP (Virtual IP)

Access Application VIP:33306

Submit Application Deployment

http://master[1-3]:8080/v2/apps

network: BRIDGE
containerPort:3306
servicePort:33306

VIP (Virtual IP)

Mesos Slave
- Consul Agent
- App (docker container)
- marathon-consult

Mesos Slave
- Consul Agent
- App (docker container)
- marathon-consult

Mesos Slave
- Consul Agent
- App (docker container)
- marathon-consult

Mesos Slave
- Consul Agent
- App (docker container)
- marathon-consult
INDIGO PAAS Tutorial

- Introductory Concepts
  - TOSCA
  - Ansible
  - Docker
  - Orchestrator APIs
  - INDIGO TOSCA custom types and templates

- Demos