PaaS Orchestrator

Tech-Talk

Marica Antonacci - INFN

eosc-hub.eu

@EOSC_eu

Dissemination level: Public/Confidential *If confidential, please define:*
Disclosing Party: (those disclosing confidential information)
Recipient Party: (to whom this information is disclosed, default: project consortium)

EOSC-hub receives funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 777536.

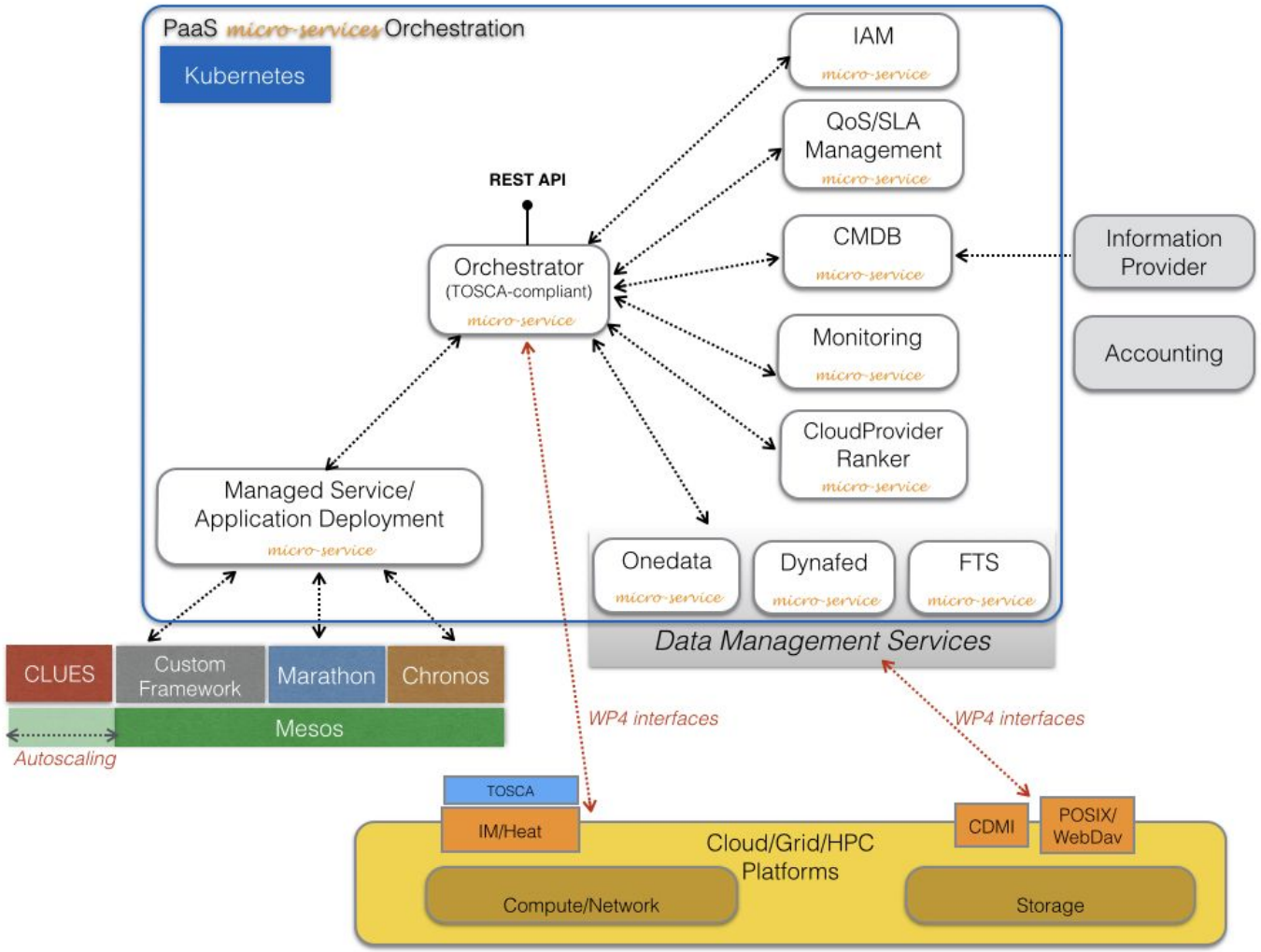
Outline

- PaaS layer
- Deployment workflow
- Orchestrator architecture
- Usage scenarios
- APIs and tools

PaaS Orchestrator Overview

- The PaaS Orchestrator is based on the developments carried out during the **INDIGO-DataCloud** project.
- It allows to coordinate the **provisioning** of *virtualized* compute and storage resources on different Cloud Management Frameworks (like OpenStack, OpenNebula, AWS, etc.) and the **deployment** of dockerized services and jobs on Mesos clusters.
- The PaaS orchestrator features advanced **federation** and **scheduling** capabilities ensuring the transparent access to heterogeneous cloud environments and the selection of the best resource providers based on criteria like user's SLAs, services availability and data location

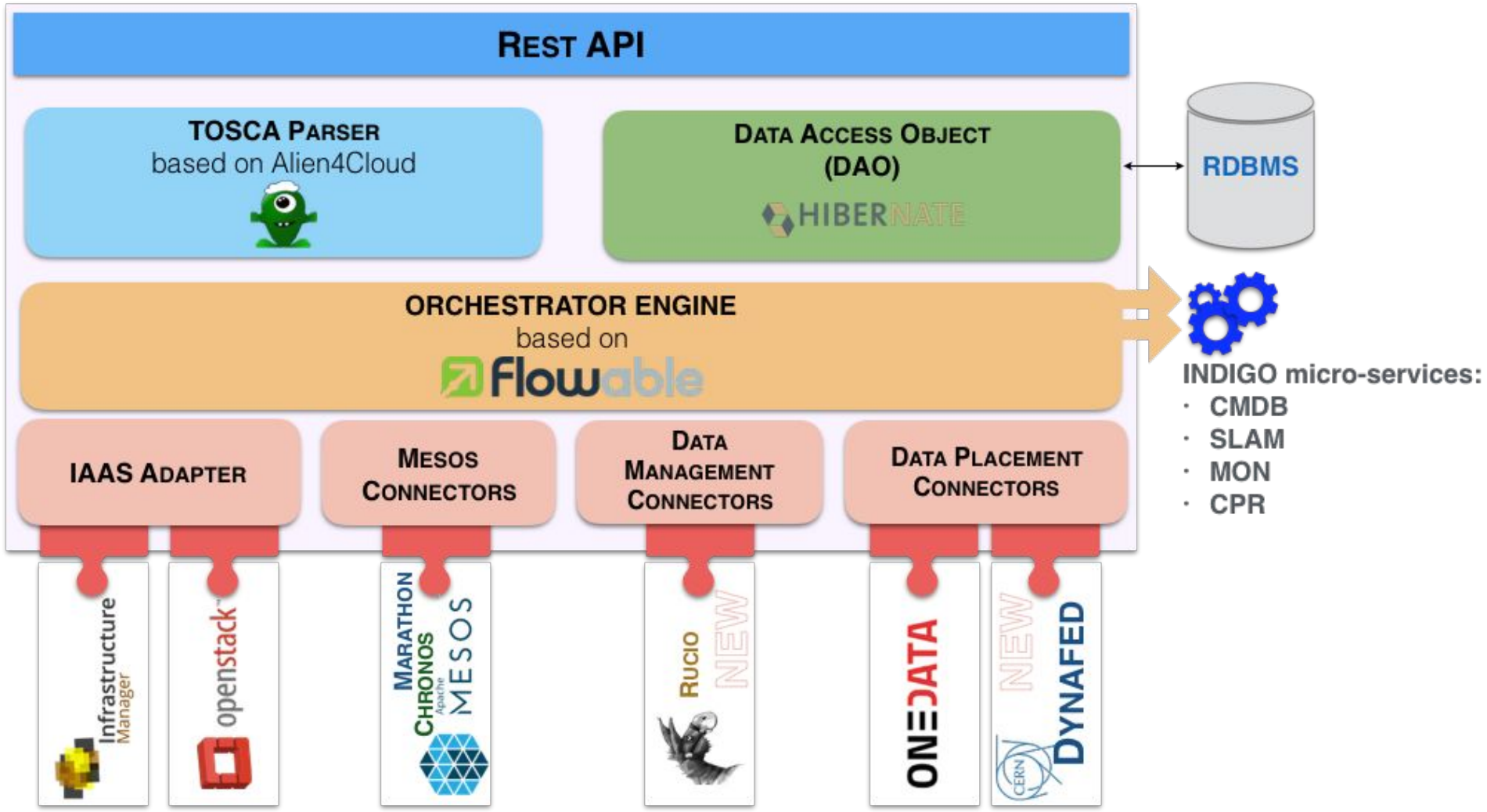
INDIGO Platform as a Service Layer



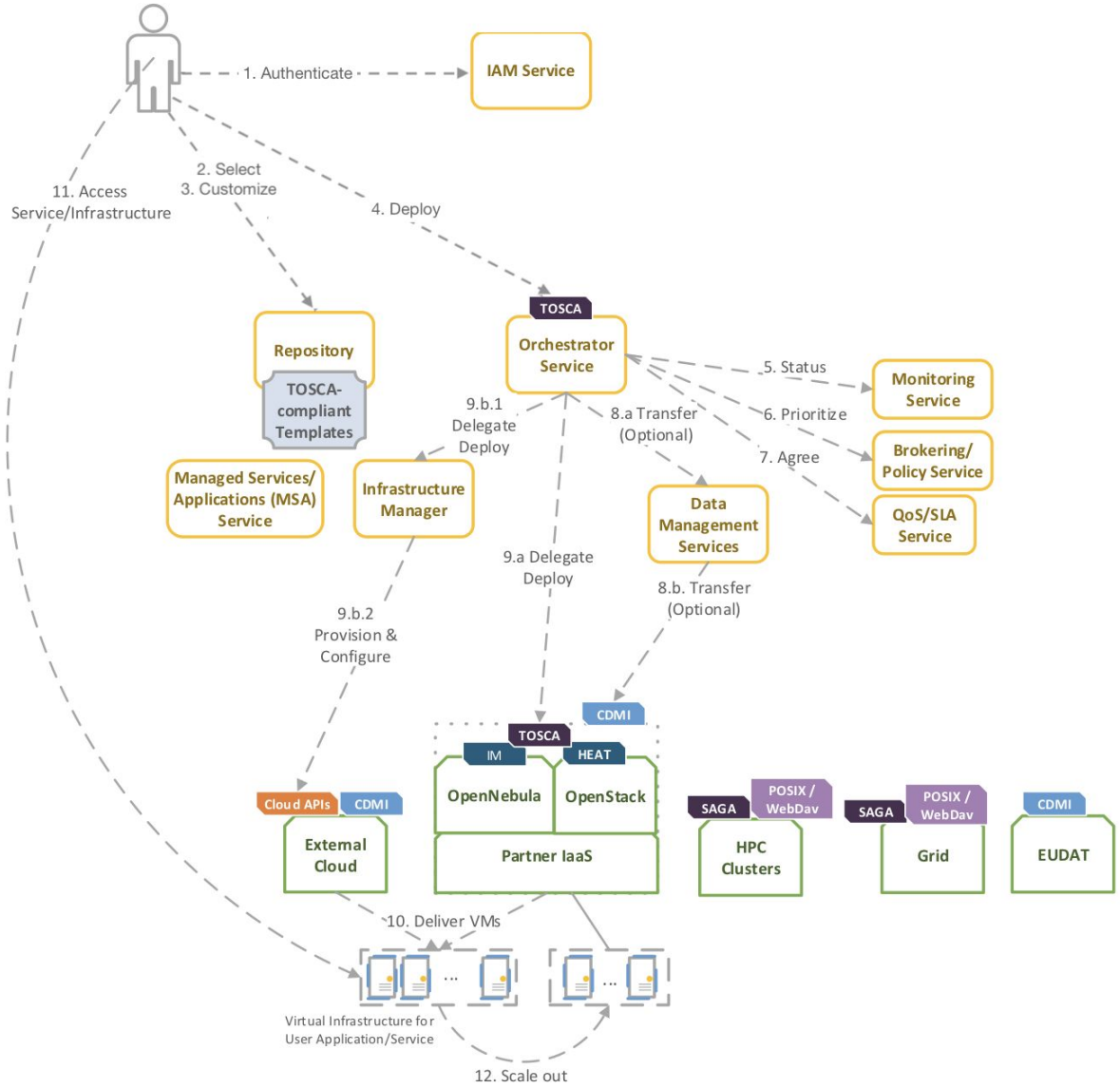
The Deployment Workflow

- The Orchestrator receives the deployment request (TOSCA template)
- The Orchestrator collects all the information needed to deploy the virtual infra/service/job consuming others PaaS µServices APIs:
  - ***SLAM Service***: get the prioritized list of SLAs per user/group;
  - ***Configuration Management DB***: get the the capabilities of the underlying IaaS platforms;
  - ***Data Management Service***: get the status of the data files and storage resources needed by the service/application
  - ***Monitoring Service***: get the IaaS services availability and their metrics;
  - ***CloudProviderRanker Service*** (Rule Engine): sort the list of sites on the basis of rules defined per user/group/use-case;
- The orchestrator delegates the deployment to **IM**, **HEAT** or **Mesos** based on the TOSCA template and the list of sites.
- Cross-site deployments are also possible.

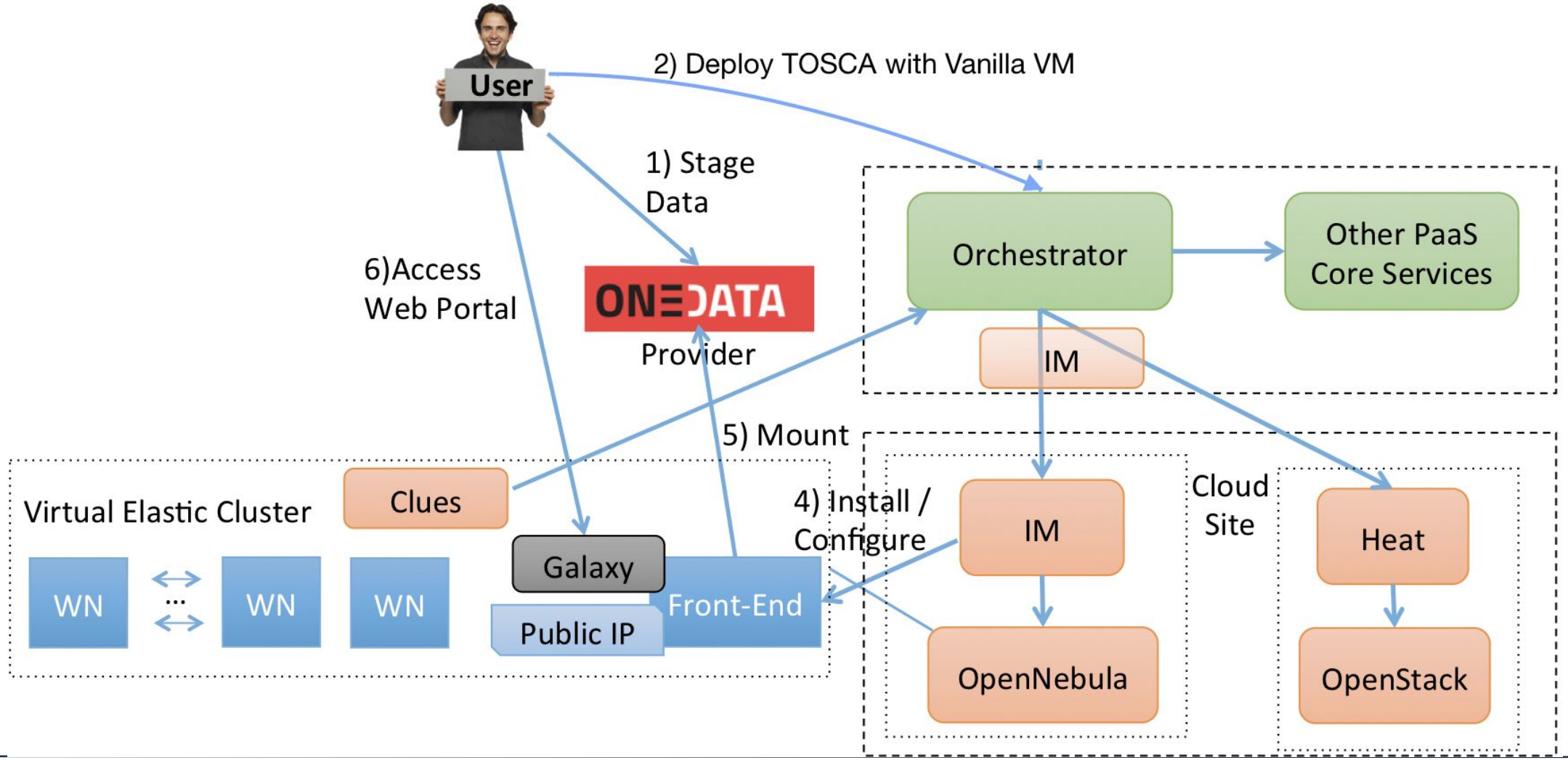
PaaS Orchestrator Architecture



Scenario I: Deployment of Virtual Infrastructures



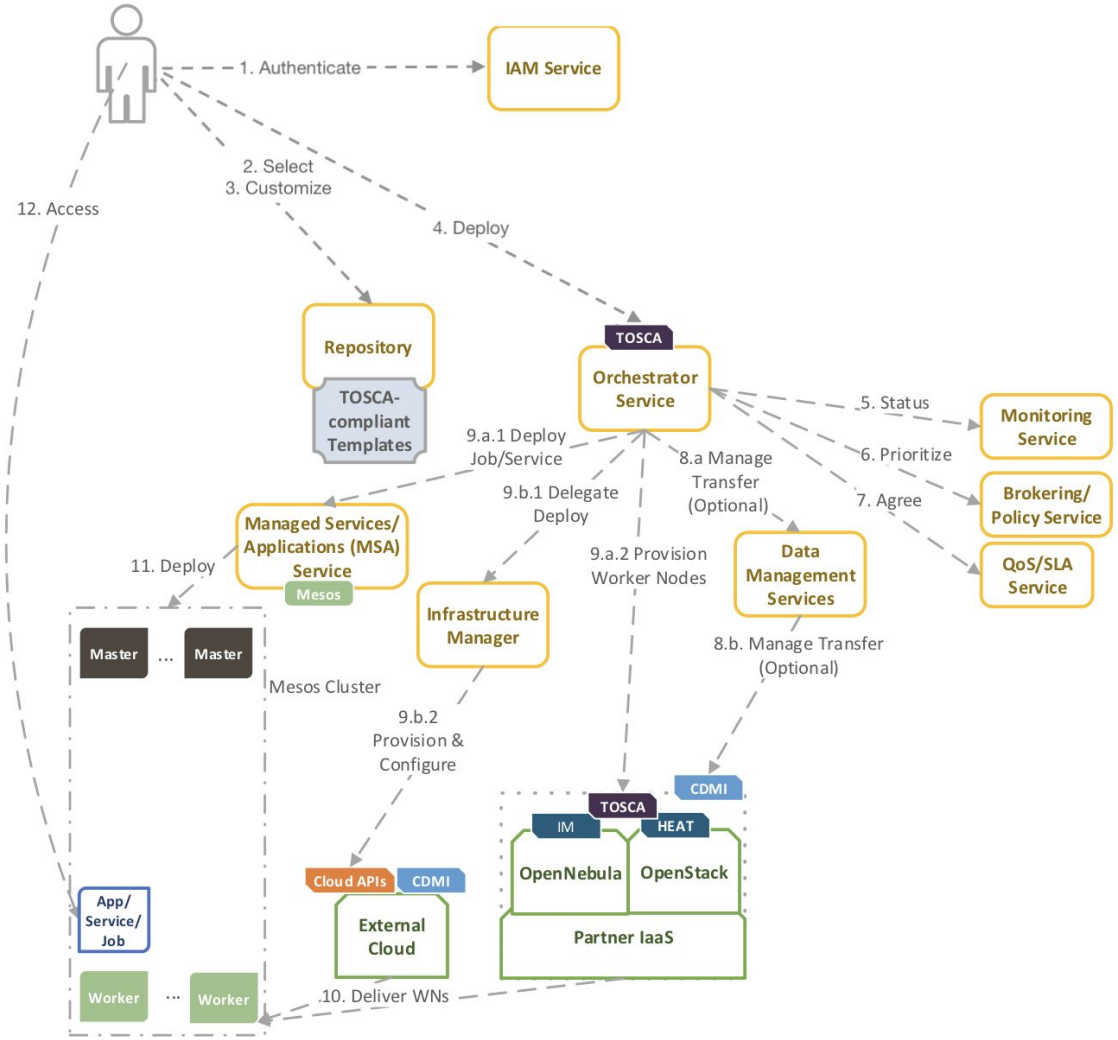
Use case: frontend + elastic batch system



Ready-to-use Templates

- **Elastic Galaxy Cluster**
  - a Galaxy portal is automatically deployed from TOSCA and configured to use a SLURM elastic cluster
- **Elastic Mesos Cluster**
  - a complete HA Mesos cluster with Chronos/Marathon framework is automatically deployed from a TOSCA template
- **Jupyter with K8s Cluster**
- **HTCondor cluster on Mesos (DODAS)**
- **Big-data Analysis Cluster (Spark on Mesos)**
- **…**

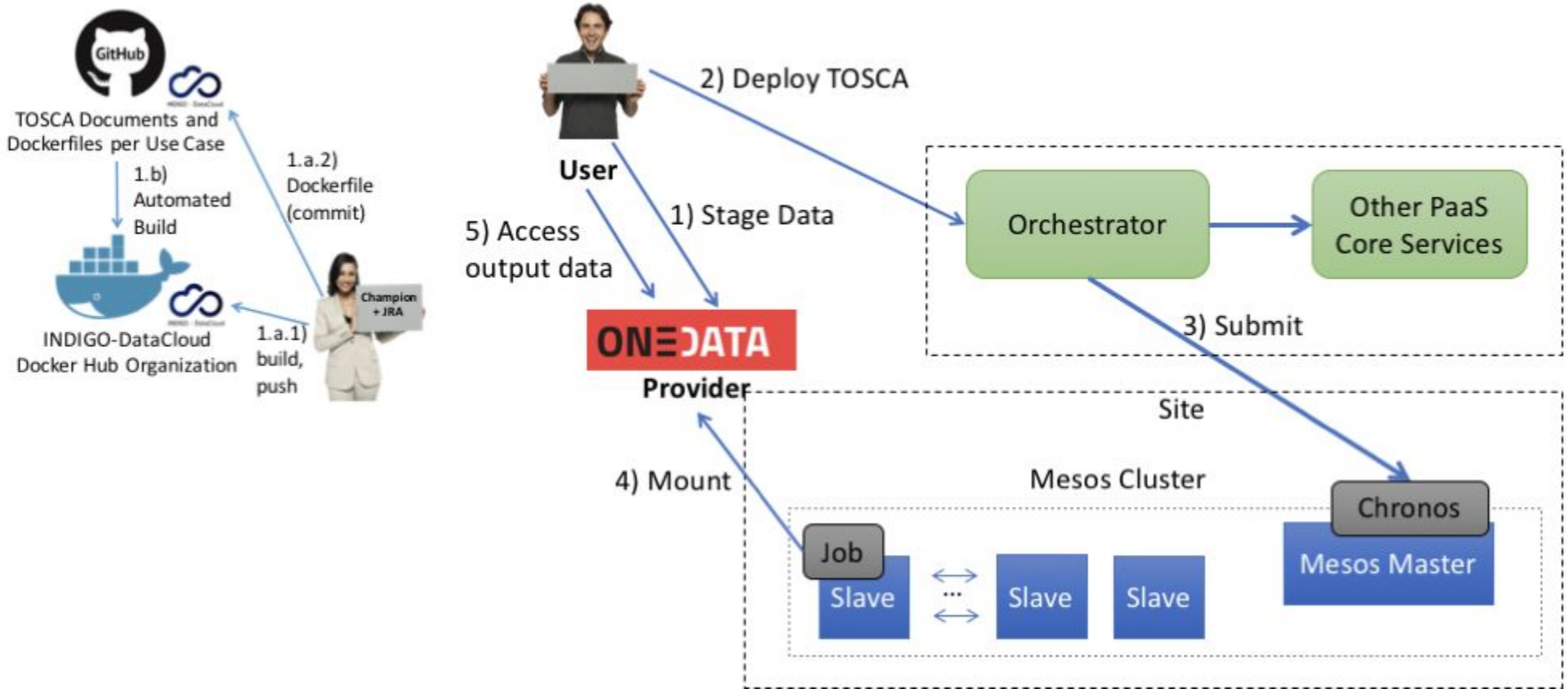
Scenario II: Deployment of managed services/jobs



- PaaS orchestrator interacts with:
  - 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.

- **Marathon** and **Chronos** are two powerful frameworks that can be deployed on top of a Mesos Cluster.
- **Mesos** is able to manage cluster resources (cpu, mem) providing *isolation* and *sharing* across distributed applications (frameworks)

Use-case: execution of batch-like jobs



Features

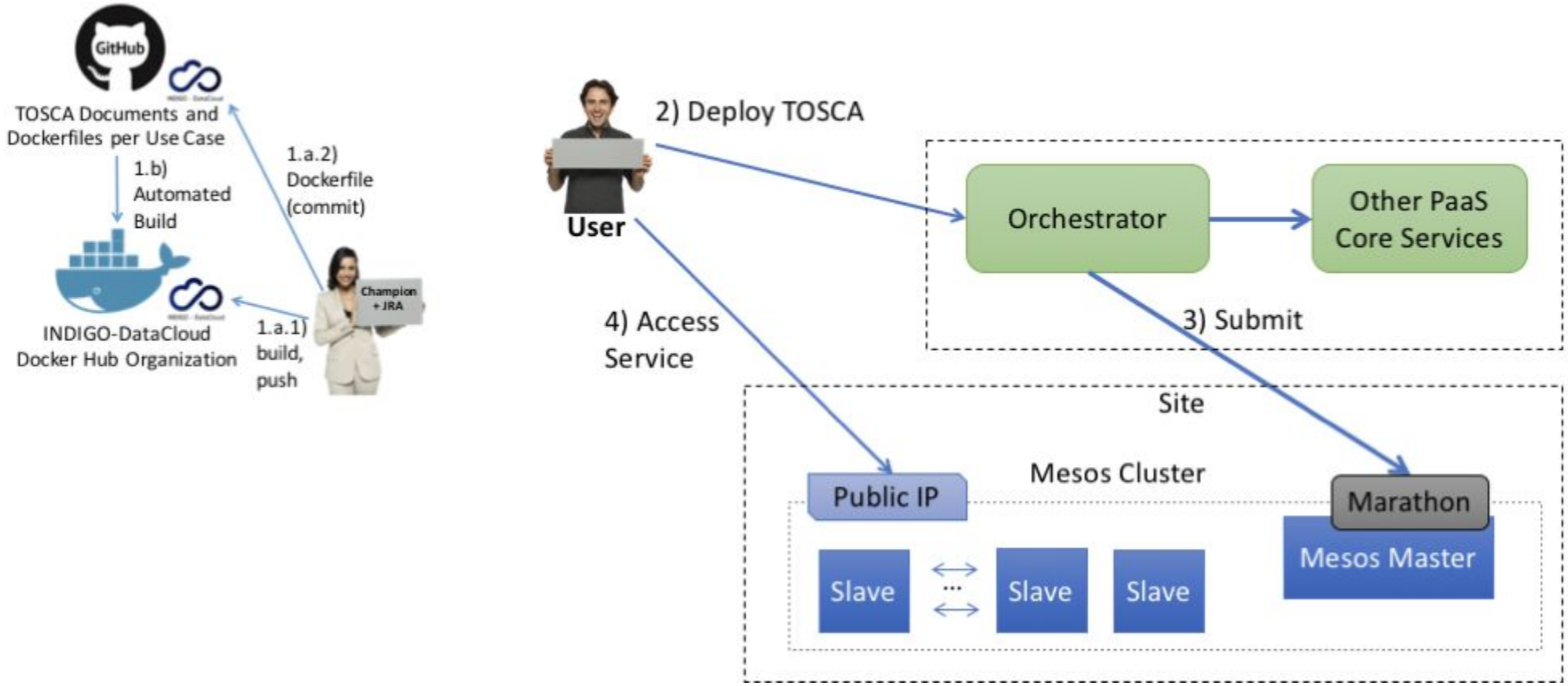
- **Parameter sweep**
  - the TOSCA template can describe multiple jobs
    - Each job is run with a specific set of input parameters
    - The jobs are run in parallel on the Mesos cluster
      - the scaling service ensures that new slave nodes are added to the cluster, if needed
- **Retries**
  - each job is automatically re-submitted with a configurable number of retries before being marked as failed
- **Job dependencies**
  - dependency among jobs can be defined in the TOSCA template and managed automatically
- **Data-aware Scheduling**
  - The Orchestrator is able to select automatically the best compute cluster based on the data location

Example TOSCA Templates

- **Scenario I**: the job data are staged in/out using the user's onedata spaces (providing proper access token(s))
  - example template: https://github.com/indigo-dc/tosca-types/blob/master/examples/indigo_job_onedata.yaml

- **Scenario II**: the job data are fetched from public URLs and uploaded to a repository (web, swift/S3, etc.) using the credentials specified by the user
  - example template: https://github.com/indigo-dc/tosca-types/blob/master/examples/indigo_job_output_upload_swift.yaml

Use-case: deployment of a long-running service



Long-running service: generic template

```
tosca_definitions_version: tosca_simple_yaml_1_0

imports:
  - indigo_custom_types: https://raw.githubusercontent.com/indigo-dc/tosca-types/master/custom_types.yaml

description: >
  TOSCA examples for specifying Marathon applications to enable the
  specification of long-running services in INDIGO.

topology_template:

  inputs:

    cpus:
      type: float
      description: Amount of CPUs for this service
      required: yes
      default: 1.0

    mem:
      type: scalar-unit.size
      description: Amount of Memory for this service
      required: yes
      default: 1 GB

    docker_image:
      type: string
      description: Docker image to be used to run the container application
      required: yes
      default: ""

    port:
      type: integer
      description: service port (exposed by the docker container)
      required: yes
      default: 8080

    data_path:
      type: string
      description: container path for persistent data
      required: no
      default: "/data"
```

```
  outputs:
    endpoint:
      value:  { concat: [ { get_attribute : [ marathon-app, load_balancer_ips, 0 ] }, ':', { get_attribute : [ docker_runtime, host, publish_ports, 0, target ] } ] }
```

New Features and enhancements

- The PaaS Orchestrator is being extended in order to:

-Support the transparent access to specialised computing hardware (GPUs, Infiniband, etc.) and HPC resources

-Improve the workflow for hybrid deployments

-Integrate a data management policy engine (QoS and Data Life Cycle)

-Support workflows for data pre-processing at ingestion

Orchestrator APIs

- **Create a deployment**:
  - POST request to /deployments - parameters:
    - template: string containing a TOSCA YAML-formatted template
    - parameters: the input parameters of the deployment (map of strings)
- **Get deployment details**:
  - GET request to /deployments:
    - curl 'http://localhost:8080/deployments/<uuid>'
- **Delete deployment**:
  - DELETE request
    - curl 'http://localhost:8080/deployments/<uuid>'
- **Documentation**: http://indigo-dc.github.io/orchestrator/restdocs/#overview

Orchent: The Orchestrator CLI

```
export ORCHENT_TOKEN=<your access token>
export ORCHENT_URL=<orchestrator_url>

usage: orchent <command> [<args> ...]

Commands:
  help [<command>...]
    Show help.

  depls
    list all deployments

  depshow <uuid>
    show a specific deployment

  depcreate [<flags>] <template> <parameter>
    create a new deployment

  depupdate [<flags>] <uuid> <template> <parameter>
    update the given deployment

  deptemplate <uuid>
    show the template of the given deployment

  depdel <uuid>
    delete a given deployment

  resls <depployment uuid>
    list the resources of a given deployment
```

Installation guide:
https://indigo-dc.gitbooks.io/orchent/content/admin.html
User guide:
https://indigo-dc.gitbooks.io/orchent/content/user.html

Resources

- **TOSCA Templates**
  - Use-cases templates: https://github.com/indigo-dc/tosca-templates
  - Example templates: https://github.com/indigo-dc/tosca-types/tree/master/examples
- **Ansible Roles**
  - Ansible Galaxy: https://galaxy.ansible.com/indigo-dc/
- **Docker images**
  - Docker hub: https://hub.docker.com/u/indigodatacloudapps/dashboard/

DEMO

https://www.youtube.com/watch?v=rVFpsRaxbaU

Thank you for your attention!

Questions?

Contact

Marica Antonacci: marica.antonacci@ba.infn.it

EOSC-hub

eosc-hub.eu @EOSC_eu