Big Data Management infrastructures: Automatic deployment of a Spark Cluster with DODAS

Daniele Spiga - INFN - (spiga@infn.it)

Corso di Formazione INFN, 11.12.2019 CNAF
Big Data Management infrastructures and Analytics

Organization of the next two sessions

- The objective of the next two sessions is to learn how most of the software applications used so far can be deployed automatically and repeatedly, possibly customizing the underlying stack, on any cloud provider.

- The deployment and setup of Spark cluster will be used as a example

- Today we will concentrate on DODAS general concepts (with focus Spark deployment)
    - Starting from the vision and motivations up to a real case

- Tomorrow there will be the hands-on (done by Diego Ciangottini)

Organization of the next two sessions

The primary objective of the hands-on is to show **how to use DODAS in order to instantiate your own Spark cluster on Openstack**

- There will be no a Spark-specific hands-on. It will be on infrastructure/automation via DODAS

Hands-on Workplan:

- How to interact with DODAS
  - Cluster creation
- How to access and debug the underlying stack
- Verify the instantiated cluster
  - Hello-world session

Outline

- Introduction:
  - What is DODAS and where it come from
  - General architecture and main concept
- Composing BigData platform with DODAS
  - How DODAS fits with BigData?
- DODAS and Spark: walkthrough the internals… and setup details
  -
- A quick overview of other DODAS capabilities
- Summary and future

What is DODAS (in a nutshell)

Dynamic On Demand Analysis Service: DODAS

A INFN solution designed with the goal to enable users **to create and provision infrastructure deployments, automatically and repeatedly**, on “any cloud provider” **with almost zero effort**.

- Implement the **infrastructure as code paradigm**: driven by a templating engine to specify high-level requirements. Declarative approach **allows to describe “What” instead of “How”**
  - Let the underlying system to abstract providers and automatically instantiate and setup the computing system(s)
- Allows to instantiate **on-demand container-based clusters** (Mesos/**Kubernetes**) to execute software applications:
  - E.g. HTCondor batch system, Spark cluster, Data Caches…
  - **But also composition of services e.g. to manage stateless (WLCG-compliant) sites**

… and where it come from

DODAS has been initially prototyped within **INDIGO-DataCloud project (2017)**

- Having in mind a primary use case: to develop a **effective solution for dynamic resource provisioning@CMS** (targeting Opportunistic computing)

Since then it has been **evolved**:

- In term of supported **use cases** (from HTCondor to BigData platforms)
- In term of adopted **technologies** (Mesos/Marathon, Kubernetes)
- In term of supported **communities** (see later)

Currently the project is also supported by **EOSC-hub H2020 EU project as a Thematic Service.**

Still some history

Spring 2017 selected as solution to generate CMS ephemeral site using

- a 20k$ Microsoft Azure Grant

Mid 2017 DODAS in Helix Nebula project

- Extensively used on TSystem IaaS Provider

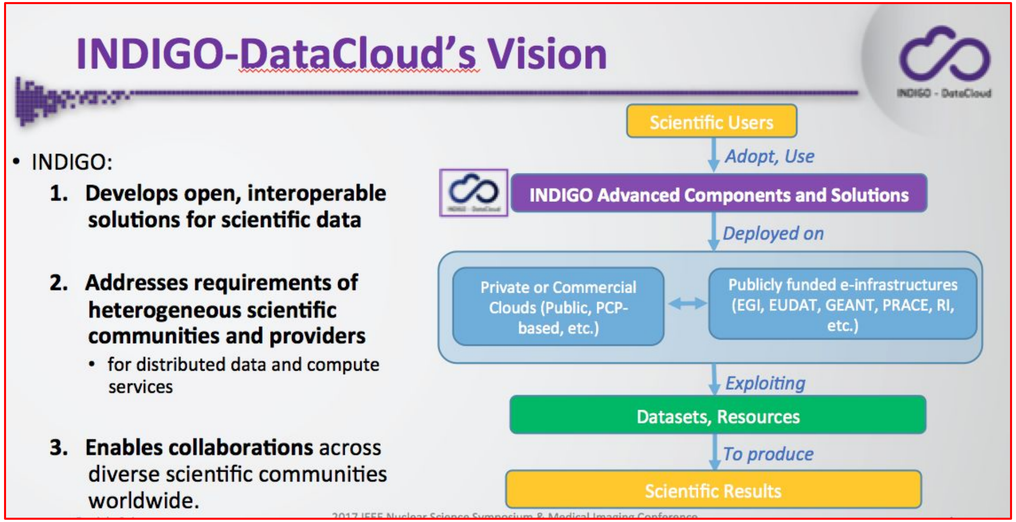
2018 Thematic Services in EOSC-hub project

- In this context has been prototyped the integration with AMS computing workflows and Virgo (work in progress)
- Currently Fermi

The vision (adopted from INDIGO project)

To develop software components and solutions to facilitate (or simply make possible) the exploitation of distributed cloud and storage resources through public or private infrastructures

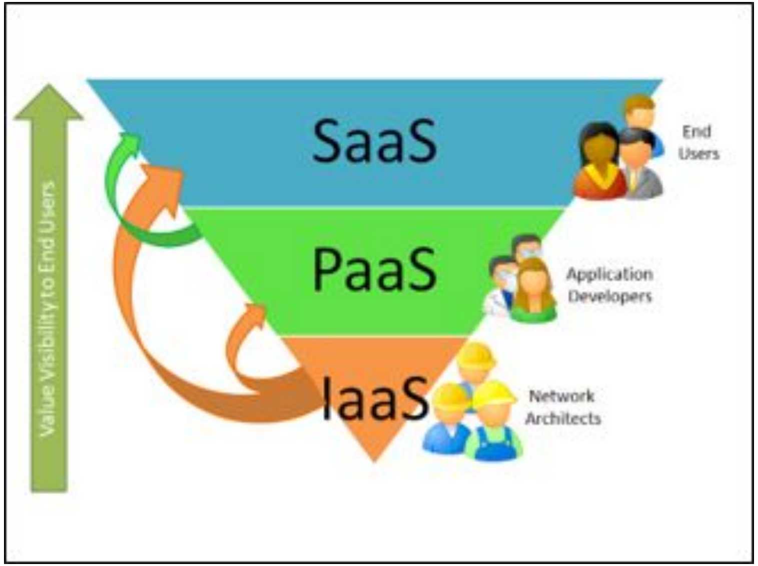
Tailored to science and targeting multi-disciplinary scientific communities



The Applications

The ultimate objective of the development activity is to the to provide technical solutions which allow to build and exploit scientific computing stack with reduced learning curve and operational costs

What matters at the end…
are the applications.



Automation and abstraction

Creating VMs is a rather easy operation… ok and if I need hundreds of them and possibly with several software and services configuration?

Ideally one would like

- To delegate such repetitive (and error prone) operations to a service
- To avoid learning Cloud APIs for any IaaS to exploit
    - As in the case of Hybrid Cloud model
- A key: to provide a common authentication (layer/mechanism)

Also: in order to make “easy” the exploitation of the underlying hardware, even specialized (GPU, SSD etc etc…) fabric level abstraction is a key

The infrastructure

In principle (e.g. currently) each piece of infrastructure added to a site tends to require

- a person located at the site to advocate for setting it up and to manage it
- a 'hand-built' custom installation

However can this labor be reduced:

- Yes. We can have common and abstracted layers which allow to compose sites on demand and based on user requirements.
  - And possibly adding modular services each time..

From the described vision and key concept/objective we defined the architectural pillars mapping to the technological solutions

and we keep evolving/updating the technologies, mostly driven by use cases/requirements

Architectural pillars of DODAS

Resources Abstraction

TOSCA to describe software applications and dependencies

Infastructure Manager as connector with underlying IaaSes

Automation

Ansible for software and application setup

Mesos/Marathon to manage resource and orchestrate

Clues to automate horizontal scalability

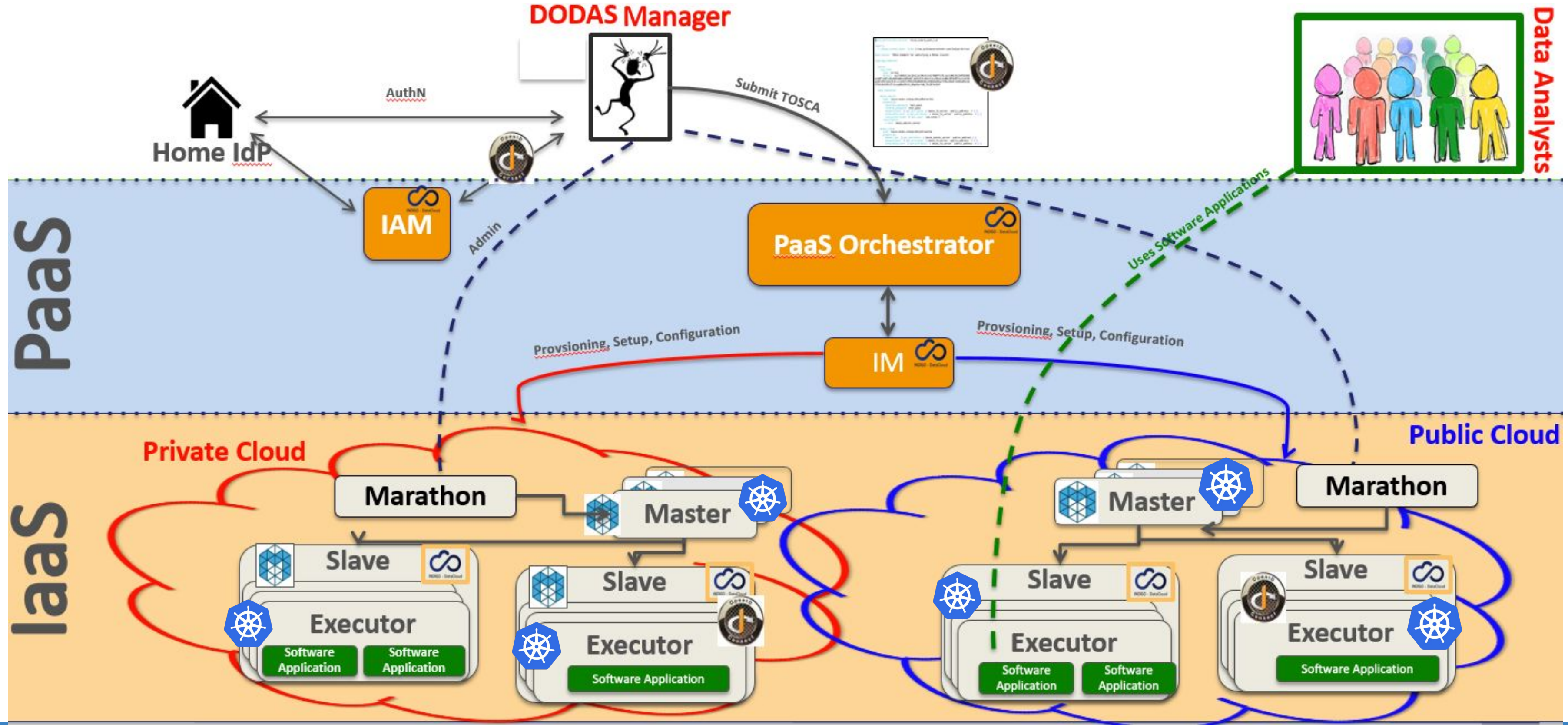
Multi-cloud support

INDIGO-PaaS Orchestrator to deal with multiple heterogeneous Cloud infrastructures

Federated authentication

INDIGO-Identity Access Management to manage JWT, OpenID Connect, SAML2.0, LDAP, Local (Username/Passwd); Identity harmonization etc

Architectural Schema



So, the Strategy based on a Lego Approach

There is a huge set of tools and solutions available, but there is NOT a one-size-fit-all solution

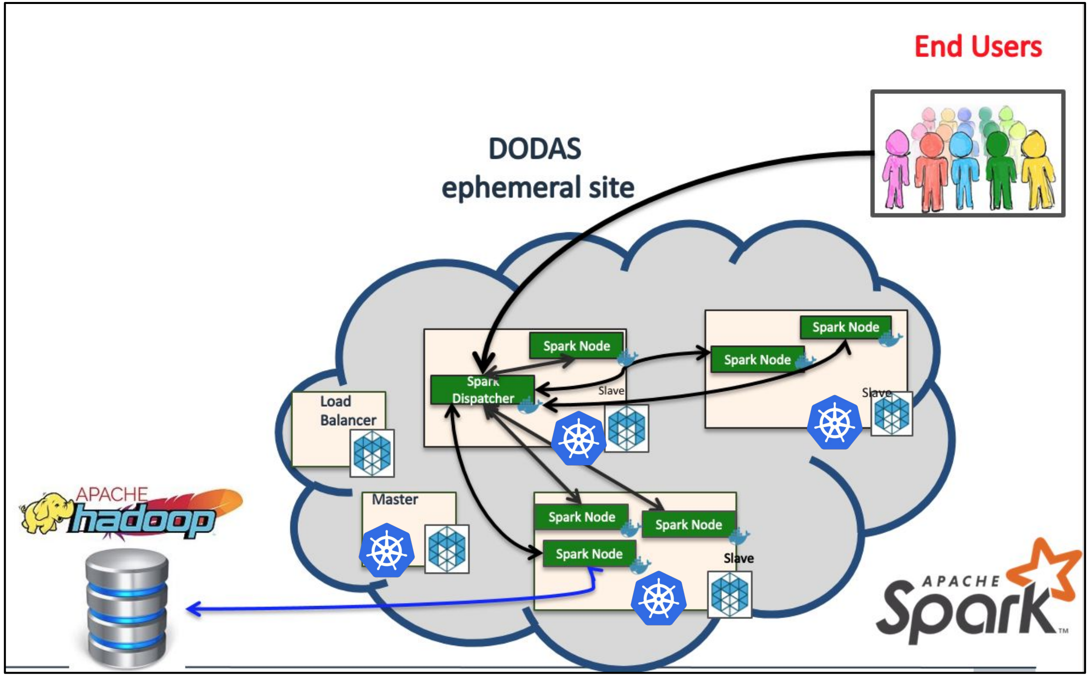
Open, Standard-based, flexible and extensible building blocks

Each use case can compose and customize

and customize

Let's spoil this talk then

Using DODAS to automatically deploy Spark on a cloud environment



CheckPoint #1

DODAS main concepts

Designed to:

- Support user tailored computing environments
- Automate configuration and deployment of custom services and/or dependencies
- Support declarative approach to define input parameters and to customize the workflow execution

DODAS main concepts (cont)

Provides a highly flexible and modular solution to enable several scenarios:

- Orchestrate and build computing stacks, following a "**all in one**" approach
  - From resources provisioning to application setup and management
    - **TOSCA + Ansible + Helm**
- Build clusters (K8s), possibly customizing the underlying environment
  - custom dependency, or services integration
    - **TOSCA + Ansible**
- Focus just on Application/service orchestration
  - **Helm**

BigData Platforms and analytics

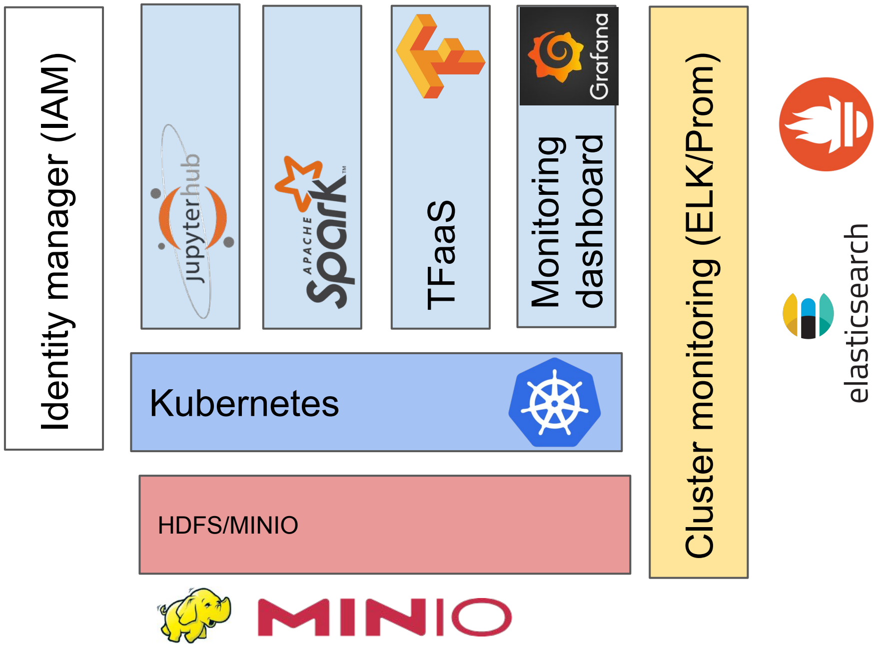
- build your own stack

Big data infrastructure

"big data infrastructure entails the tools and agents that collect data, the software systems and physical storage media that store it, the network that transfers it, the application environments that host the analytics tools that analyze it and the backup or archive infrastructure that backs it up after analysis is complete."

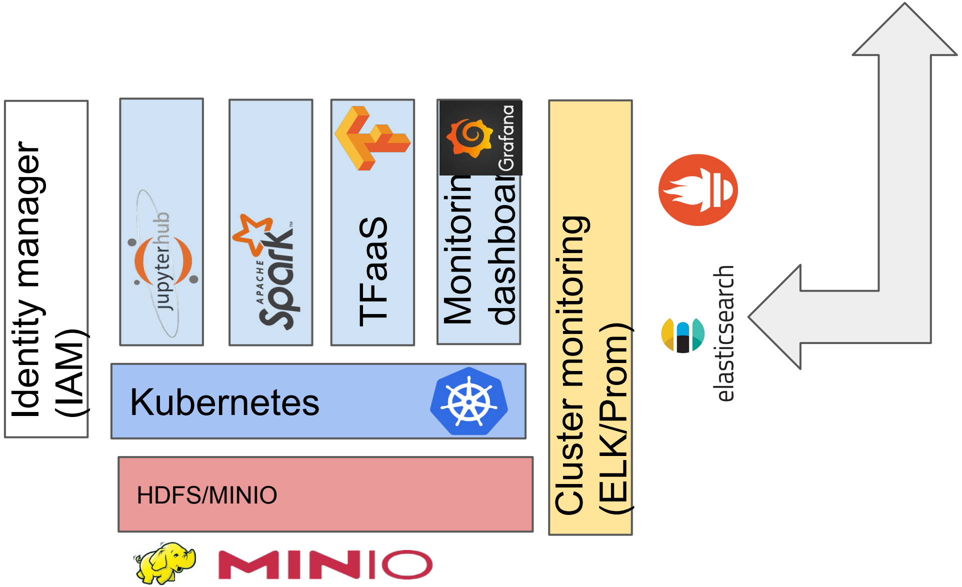
syncsort

As example: something like this?



Most of what has been discussed this week in term of services and software, components
- plus something I will show in the next…

And how DODAS fits into this ?



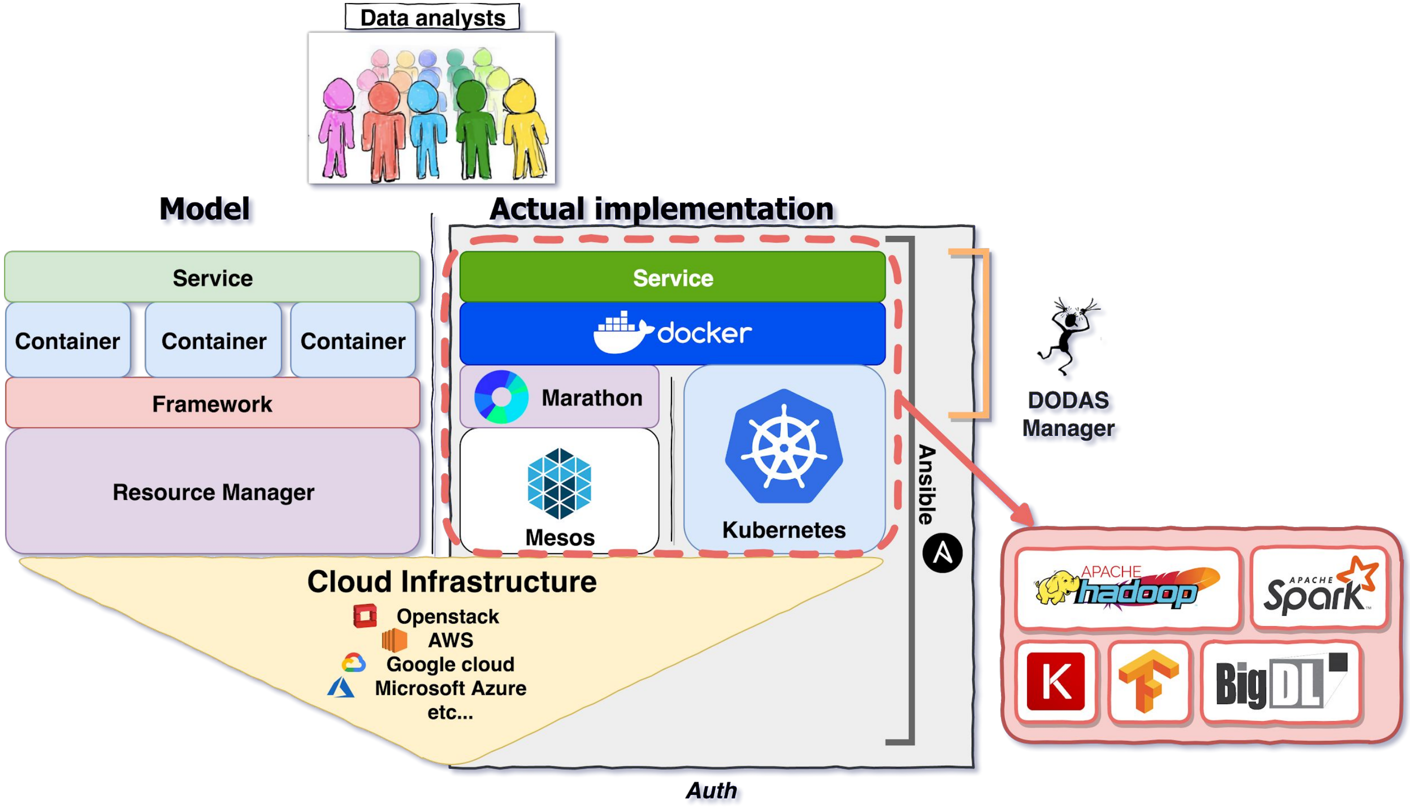
DODAS main concepts (cont)

Provides a highly flexible and modular solution to enable several scenarios:

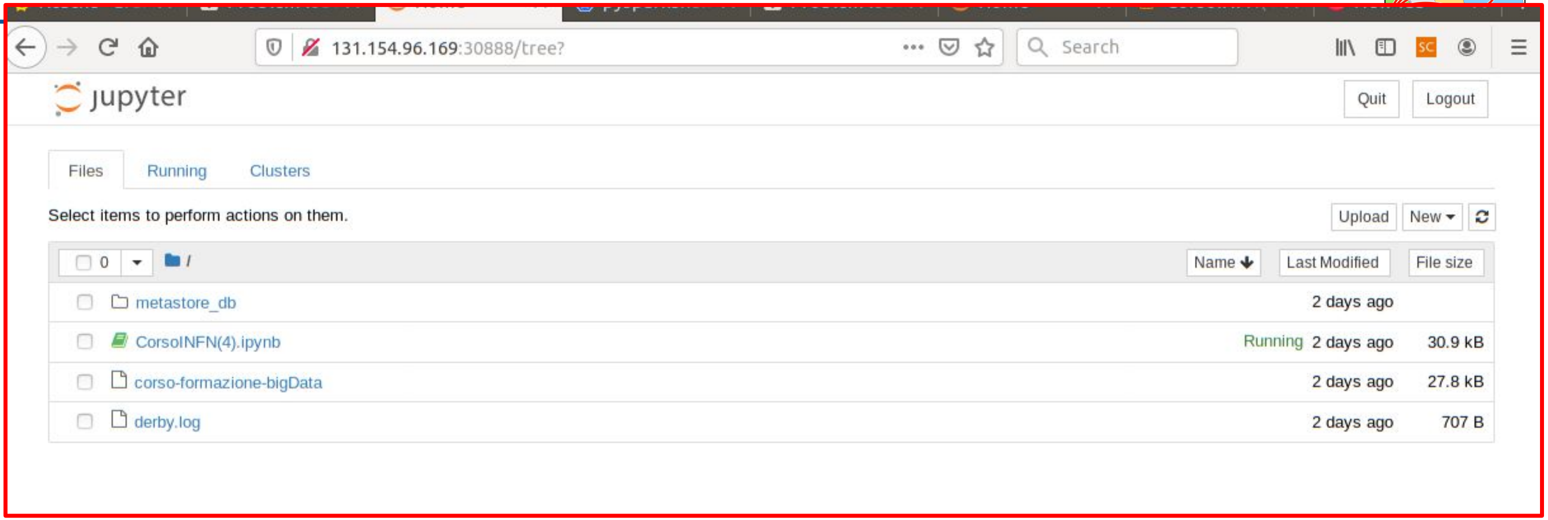
- Orchestrate and build computing stacks, following a “**all in one**” approach
  - From resources provisioning to application setup and management
    - **TOSCA** + **Ansible** + **Helm**

- Implement the **infrastructure as code paradigm**: driven by a templating engine to specify high-level requirements. Declarative approach **allows to describe “What” instead of “How”**
  - Let the underlying system to abstract providers and automatically instantiate and setup the computing system(s)

Let's start connecting some dots…



From user perspectives



Accessed by

Runs on

CheckPoint #2

DODAS and BigData

DODAS is a deployer manager which allows user **to create and provision infrastructure deployments, automatically and repeatedly**, on “any cloud provider” **with almost zero effort**.

- As such allows to build platforms for BigData processing and analytics

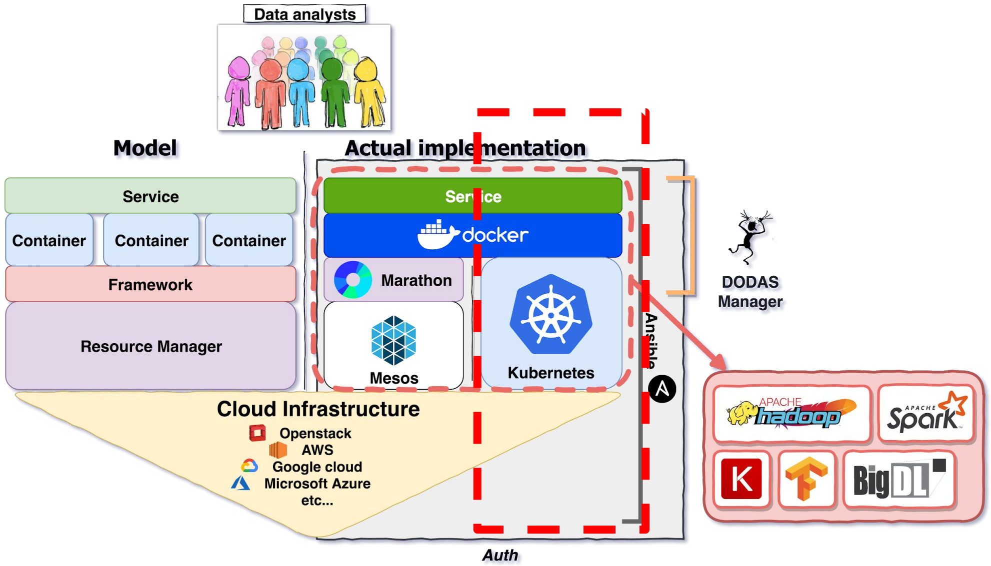
It doesn’t provide you with a solution/the solution… the other way around: there is a set of tools and solutions available, but there is NOT a one-size-fit-all

- Compose your own, reuse code/configurations (see later), ad modules extend building blocks

Let’s now dig a bit into the declarative approach…

A disclaimer

As anticipated the system keep evolving since the initial implementation towards several dimension...



We will focus today on K8s based implementation

And we will run Spark on K8s

- I will sketch also additional solutions

DODAS and Kubernetes

TOSCA

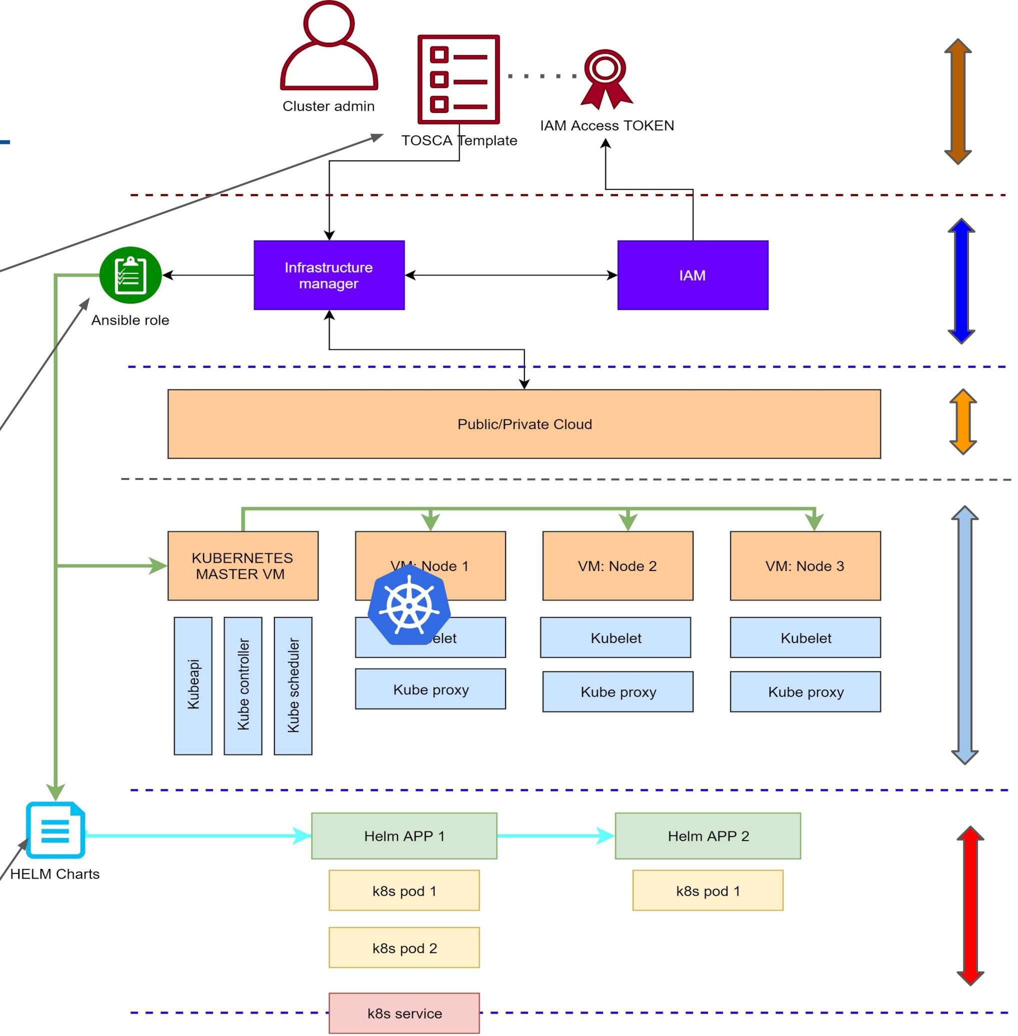
- Define the infrastructure (the HW)
- Define services (k8s) & Applications to setup (through Ansible)
- Declare ("any") input parameters

Ansible based installation using:

- Kubeadm (initialization)
- Flannel (default but others available)
- nginx ingress (optional)
- k8s dashboard (optional)

Helm (Applications layer)

- Dynamically load and compile values (from tosca through ansible)
- Install applications



TOSCA

```yaml
inputs:

  number_of_masters:
    type: integer
    default: 1

  num_cpus_master:
    type: integer
    default: 2

  mem_size_master:
    type: string
    default: "4 GB"

  number_of_slaves:
    type: integer
    default: 1

  num_cpus_slave:
    type: integer
    default: 2

  mem_size_slave:
    type: string
    default: "4 GB"

  server_image_slave:
```

```yaml
    k8s_master:
      type: tosca.nodes.indigo.LRMS.FrontEnd.Kubernetes
      properties:
        admin_token: testme
        kube_version: 1.14.0
        kube_front_end_ip: { get_attribute: [ k8s_master_server, private_address, 0 ] }
      requirements:
        - host: k8s_master_server
```

```yaml
    k8s_wn:
      type: tosca.nodes.indigo.LRMS.WorkerNode.Kubernetes
      properties:
        front_end_ip: { get_attribute: [ k8s_master_server, private_address, 0 ] }
        kube_version: 1.14.0
        nfs_master_ip: { get_attribute: [ k8s_master_server, private_address, 0 ] }
      requirements:
        - host: k8s_slave_server
```

TOSCA (cont)

```yaml
k8s_master_server:
  type: tosca.nodes.indigo.Compute
  capabilities:
    endpoint:
      properties:
        network_name: PUBLIC
        ports:
          kube_port:
            protocol: tcp
            source: 6443
          dashboard_port:
            protocol: tcp
            source: 30443
          web_ui:
            protocol: tcp
            source: 30808
          jupyter:
            protocol: tcp
            source: 30888
    scalable:
      properties:
        count: { get_input: number_of_masters }
    host:
      properties:
        num_cpus: { get_input: num_cpus_master }
        mem_size: { get_input: mem_size_master }
    os:
      properties:
        image: { get_input: server_image }
```

```yaml
k8s_slave_server:
  type: tosca.nodes.indigo.Compute
  capabilities:
    endpoint:
      properties:
        network_name: PRIVATE
    scalable:
      properties:
        count: { get_input: number_of_slaves }
    host:
      properties:
        num_cpus: { get_input: num_cpus_slave }
        mem_size: { get_input: mem_size_slave }
    os:
      properties:
        image: { get_input: server_image_slave }
```

TOSCA but finally Spark

```
      type: tosca.nodes.indigo.HelmInstall
      properties:
        externalIP: { get_attribute: [ k8s_master_server, public_address,
        name: "spark"
        chart: "cloudpg/spark"
        repos:
            - { name: cloudpg, url: "https://cloud-pg.github.io/charts/" }
        values_file: { get_input: helm_values }
      requirements:
        - host: k8s_master
        - dependency: k8s_wn
```

Compiling vaules at runtime and install

Who is doing this?

That's the last step..
Installing Spark on top of k8s

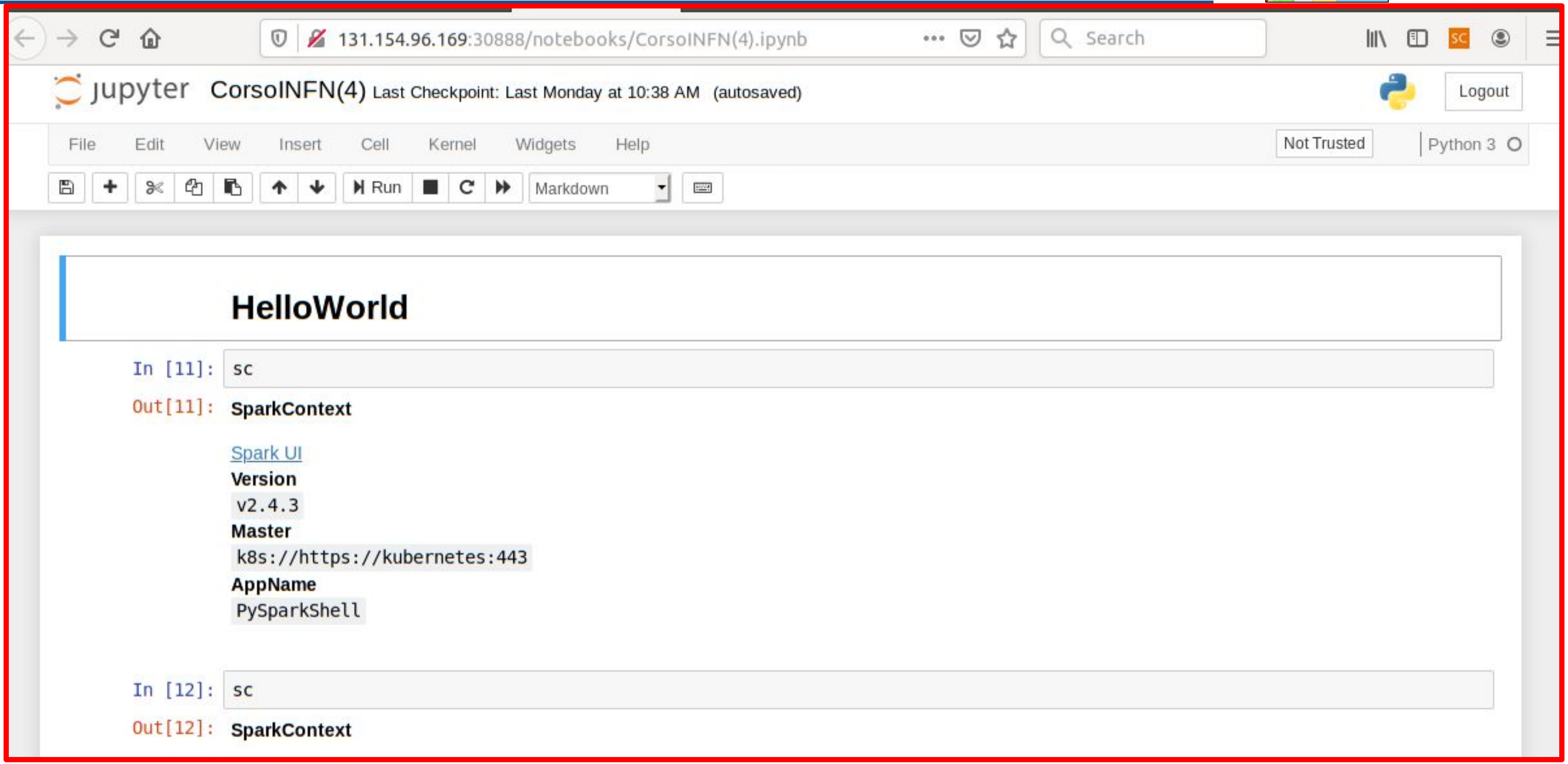
GitHub, Inc. [US] | https://github.com/Cloud-PG/ansible-role-helm/blob/master/tasks/kube.yml

23 lines (18 sloc) | 697 Bytes

Raw | Blame | History

```yaml
---
- name: Helm install cloudpg repo
  command: helm repo add {{ item.name }} {{ item.url }}
  with_items: "{{ repos }}"

# - name: Helm install cloudpg repo
#   command: helm repo add cloudpg https://cloud-pg.github.io/charts/

# - name: Helm install cache repos
#   command: helm repo add cache https://cloud-pg.github.io/CachingOnDemand/

- name: write values
  get_url:
    url: "{{ values_file }}"
    dest:  /tmp/values_{{ name }}-template.yml

- name: compile values
  template:
    src:  /tmp/values_{{ name }}-template.yml
    dest: /tmp/values_{{ name }}.yml

- name: Helm install chart {{ chart }}
  command: "helm install --name {{ name }} -f /tmp/values_{{ name }}.yml {{ chart }}"
```

And the result



CheckPoint #3

How DODAS install and deploy Spark?

It uses Helm chart based installation with values compiled at runtime, everything (obviously) on top of Kubernetes.

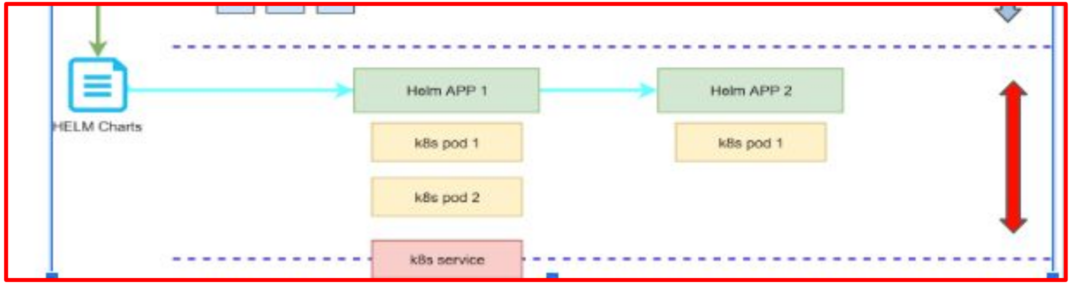
- DODAS install also Kubernetes
- And provision the Hardware (virtual HW)

TOSCA is where you defined all of that and where you pass the inputs you want goes down to the “fabric layer”

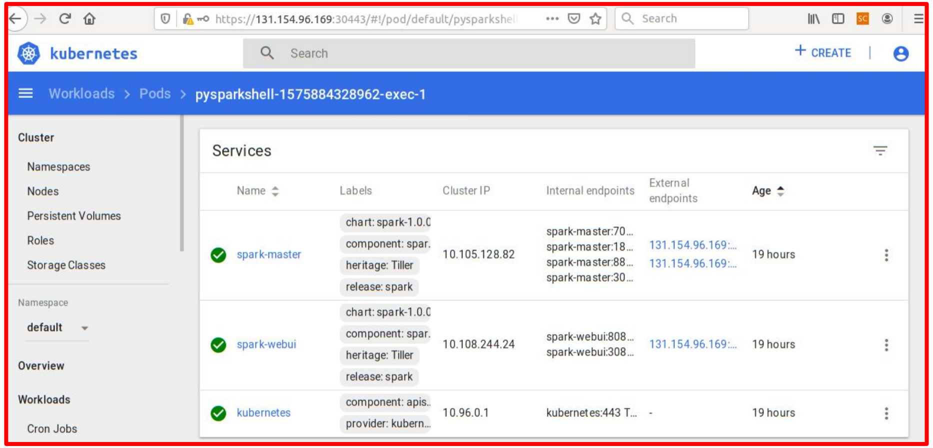
Spark and DODAS

Allow to remotely deployed clusters, possibly customized, managed through K8s

Applications Layer



- **No Dedicated** experiment **support**
- **No Dedicated site leve setup** required

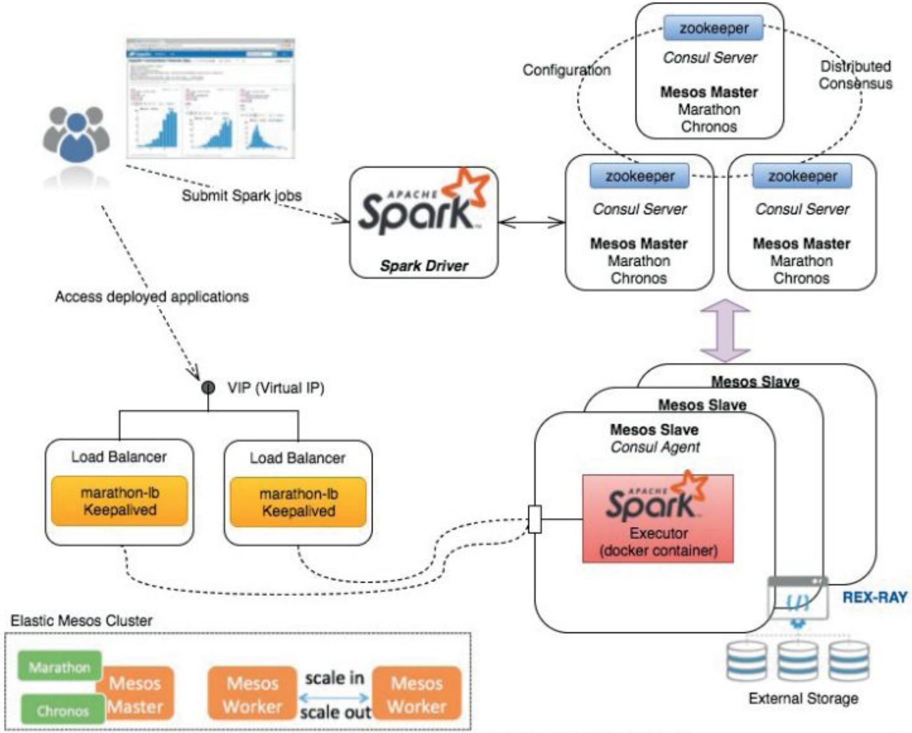


Also Mesos based Spark …

Courtesy of Marica Antonacci

EOSC-hub Use case: On-demand Big Data Analysis Platform

```
elastic_cluster_front_end:
  type: tosca.nodes.indigo.ElasticCluster
  properties:
    deployment_id: orchestrator_deployment_id
    iam_access_token: iam_access_token
    iam_clues_client_id: iam_clues_client_id
    iam_clues_client_secret: iam_clues_client_secret
    marathon_credentials:
      protocol: https
      token:  { get_input: marathon_password }
      user: admin
    chronos_credentials:
      protocol: https
      token:  { get_input: chronos_password }
      user: admin
    mesos_credentials:
      protocol: http
      token:  { get_input: mesos_password }
      user: admin
  requirements:
    - lrms: mesos_master
    - wn: mesos_slave
mesos_master:
  type: tosca.nodes.indigo.LRMS.FrontEnd.Mesos
  properties:
    mesos_masters_list: { get_attribute: [ HOST, private_address ] }
    mesos_password: { get_input: mesos_password }
    marathon_password: { get_input: marathon_password }
    chronos_password: { get_input: chronos_password }
  requirements:
    - host: mesos_master_server
```



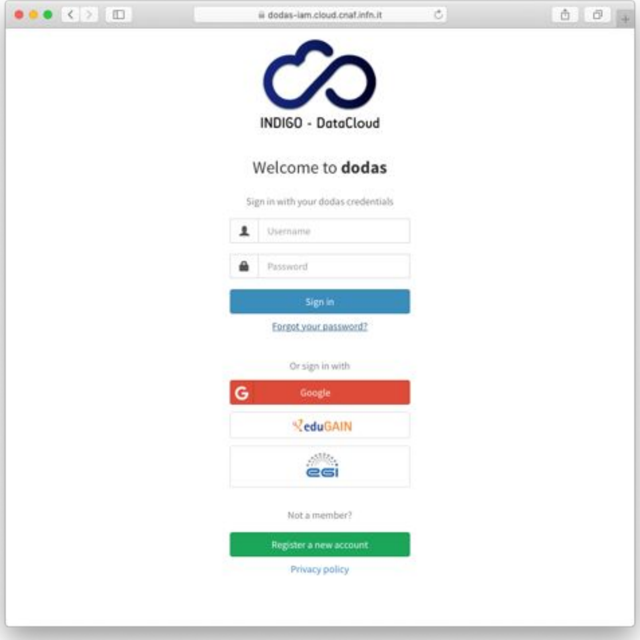
Infrastrutture Cloud. L'esempio pilota dell esperimento AMS. Perugia, 25-28 November 2019

Aside note : AuthN/Z

DODAS adopts INDIGO-Identity Access Management to manage Authentication and Authorization

A VO-scoped authentication and authorization service that

- supports multiple authentication mechanisms
- provides users with a persistent, VO-scoped identifier
- exposes identity information, attributes and capabilities to services via JWT tokens and standard OAuth & OpenID Connect protocols
- can integrate existing VOMS-aware services
- supports Web and non-Web access, delegation and token renewal



The final rush...

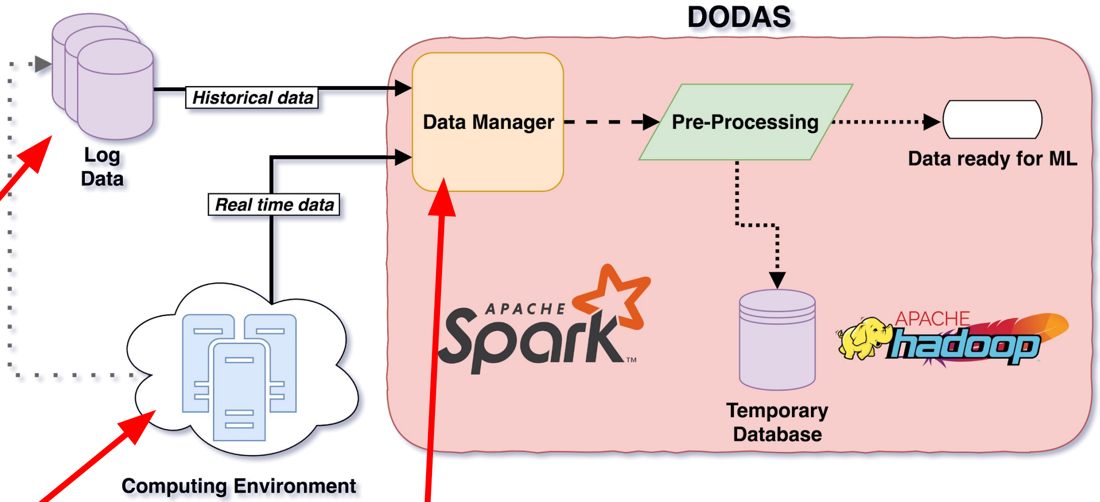
- A real usage of such system (with combination of services)

Data flow and Pre-processing

The **CMS available logs are the key** to the success of the model development

A **Primary data** source is historical data of infrastructure utilization:

- **Data logs** are in JSON format, **stored** in a **Hadoop** file system and **serialized using Avro**.

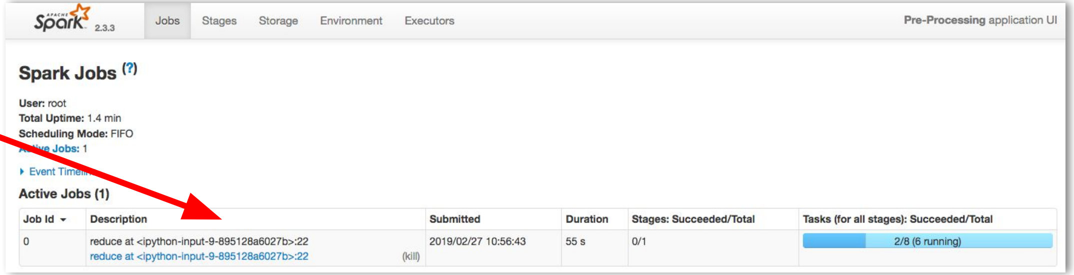


- The **Secondary data** source are **real-time information**
  - Info of hardware, clusters, network and the cache system (content and status)
  - Streaming information feed
- The **Data Manager** can be used by end-users to **pre-fetch data** into DODAS environment **or to get a stream** of data in real-time.

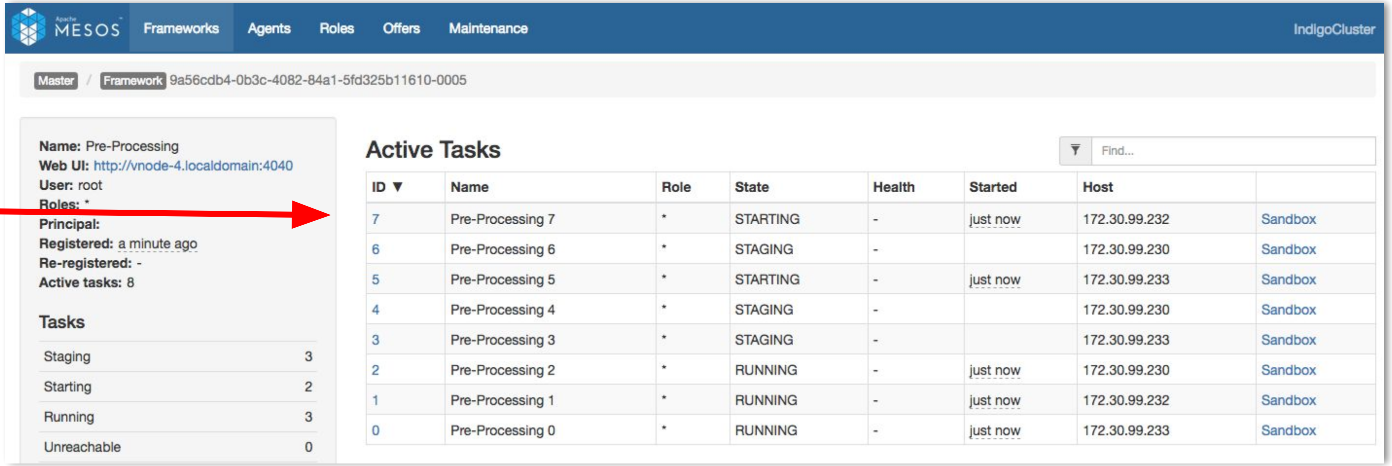
Courtesy of Mirco Tracolli

Pre-processing step

Spark is deployed by of DODAS

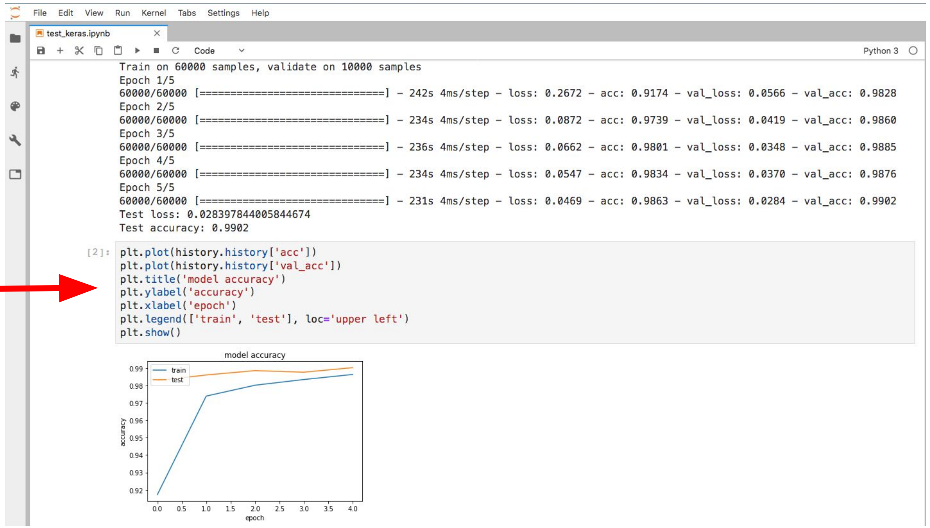


The service is **completely transparent** to the user, **Mesos** will manage the Spark's job.



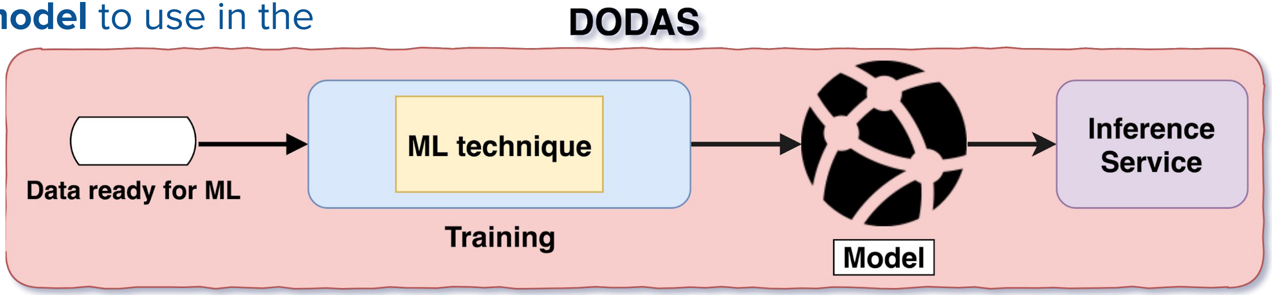
Training models over reduced data

- Reduced data are automatically available for training ML models
- The developed environment is ready with the most used **ML frameworks:**
  - **Jupyter, Keras** and **TensorFlow**
  - Highly **customizable**: e.g. **Intel BigDL framework** has been added to use **alongside Spark** for the **training** phase.



The **output** of this phase is a **model** to use in the inference step.

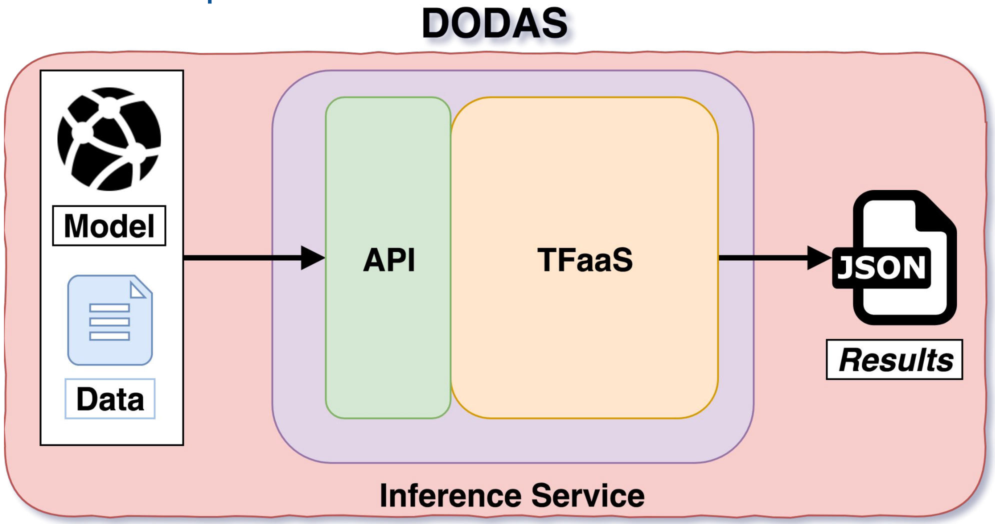
Trained model is **automatically loaded** into the **inference service.**



Performing Inference

The inference service is implemented using the **CMS TFaaS**, embedded in DODAS.
It is a **Software as a Service** based on **TensorFlow framework** for Machine Learning and exposes an **API** through the **HTTP** protocol:

- ***/models***: to view existing models on TFaaS server
- ***/json***: to serve TF model predictions in JSON data-format
- ***/upload***: to push a model to TFaaS server
- ***/delete***: to delete your model



DOI: Valentin Kuznetsov. (2018, July 9). vkuznet/TFaaS: First public version (Version v01.00.06). Zenodo. http://doi.org/10.5281/zenodo.1308049

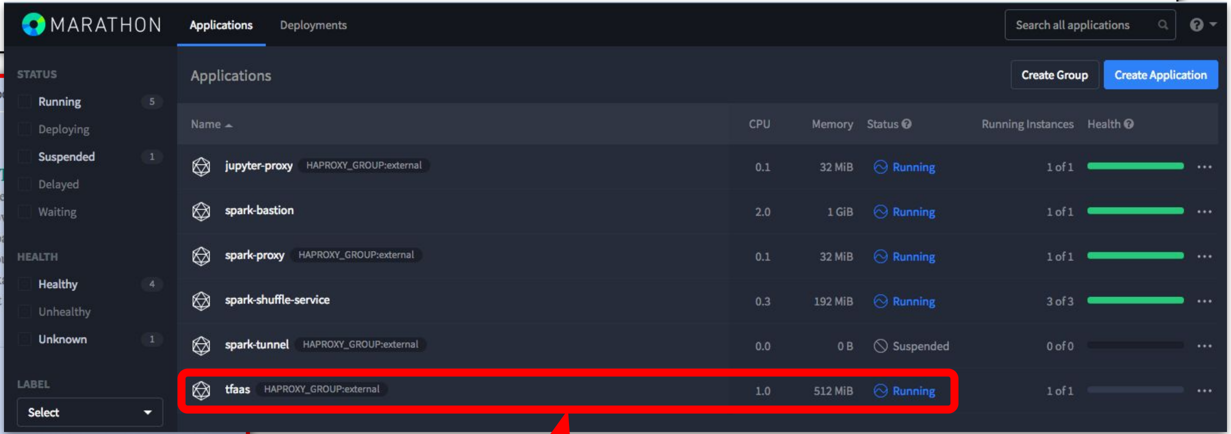
Inferencing with TFaaS

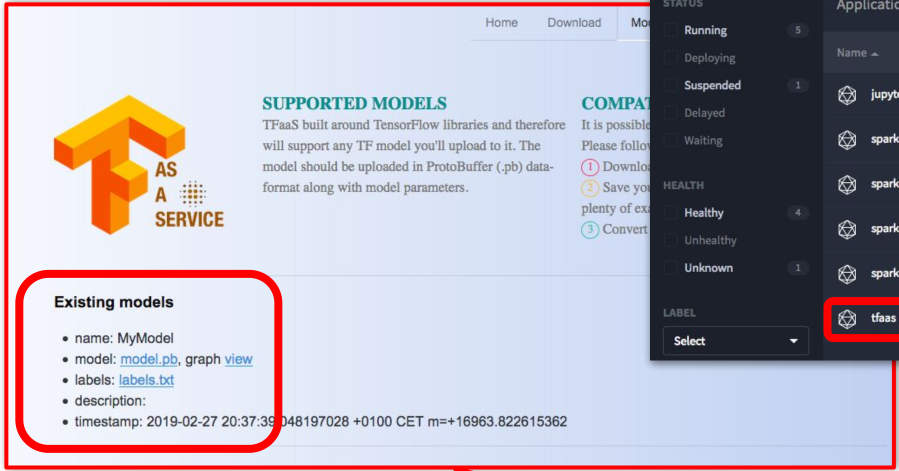
Call the model:

```
curl -X POST http://tfaas/json -d @data.json -H "Accept: application/json" -H "Content-Type: application/json"
```

Result:

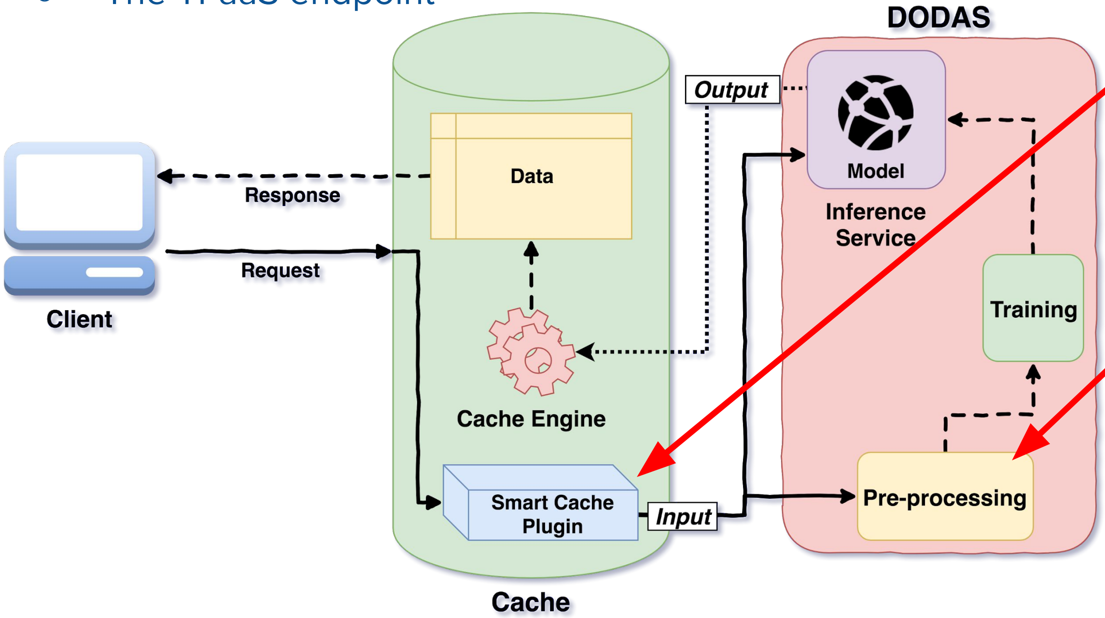

```
{"labels":[{"label":"a","probability":1},{"label":"b","probability":2.815438e-8},{"label":"c","probability":4.65911e-18}]}
```



Integration with Data Cache

- The plan is to **extend the XRootD cache** (XCache) with a specific **plugin** which queries against the developed **AI Service**
  - The TFaaS endpoint

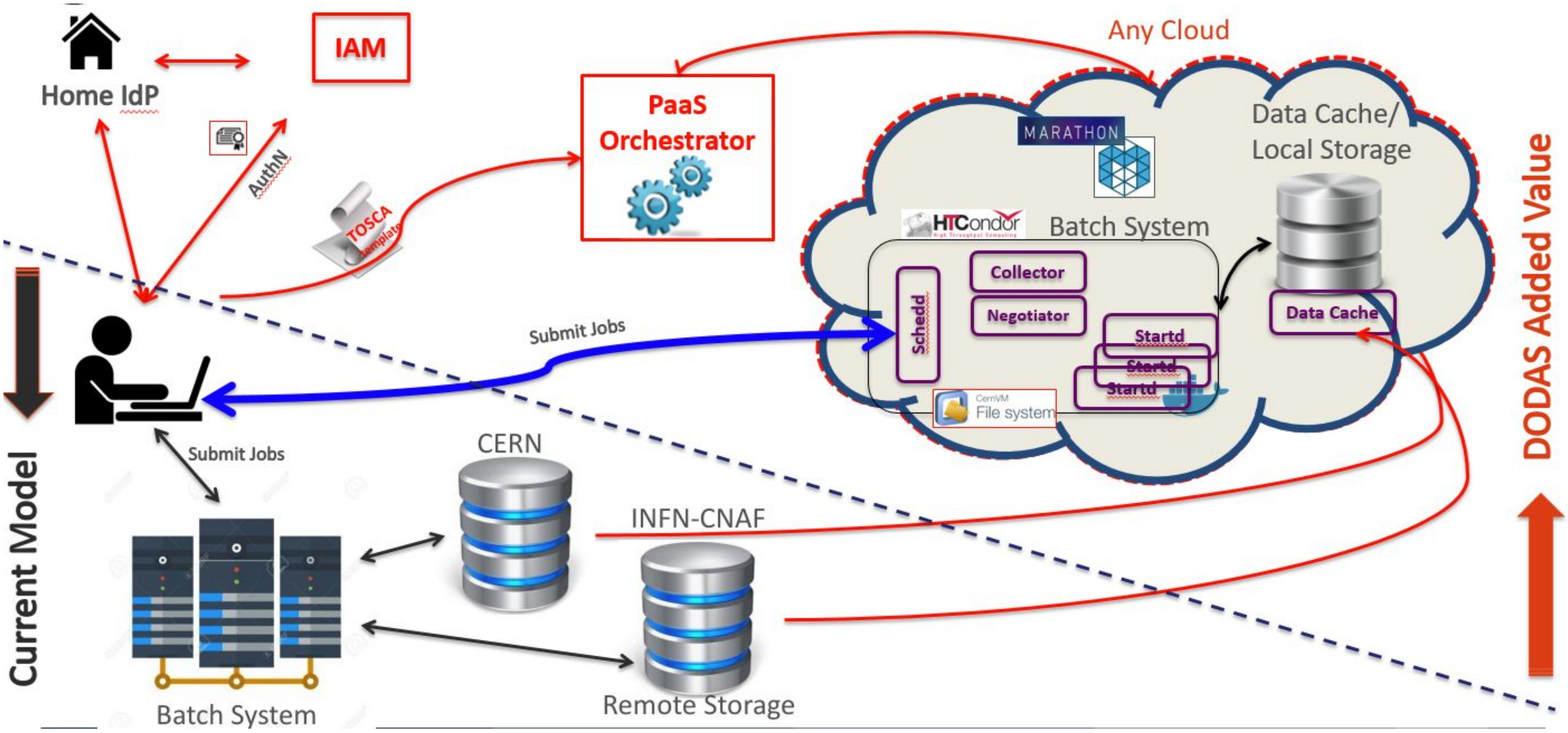


Runtime information are used to **continue** the **training** of the **model**

However DODAS is not only BigData...

- Managing stateless sites to execute experiment workflows for scientific experiments
  - Batch system as a Service , and their federations
    - See Corso formazione here: https://agenda.infn.it/event/20268/

Batch system on demand

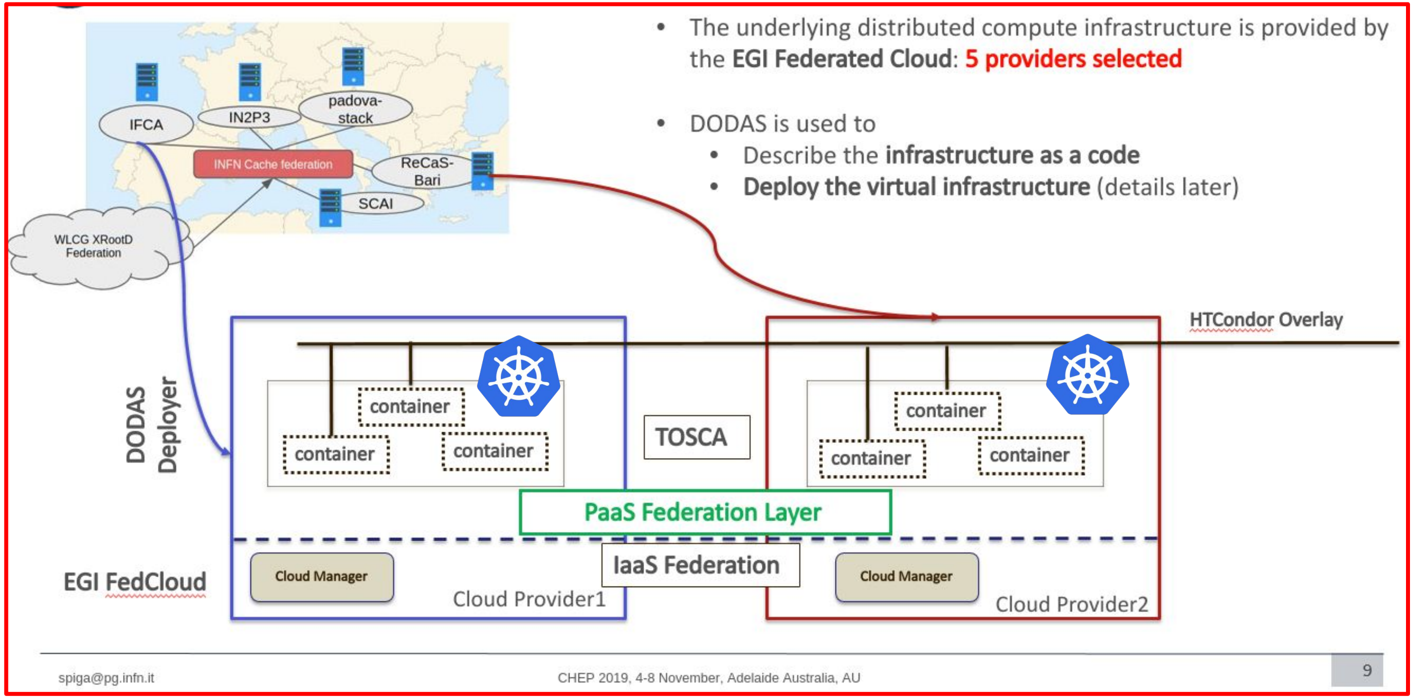


A recent setup

Many tests and deployments in the past two years (see results @ CHEP2018).

Recently we Used DODAS to manage **5 stateless sites**… a virtual batch via **HTCondor overlay**

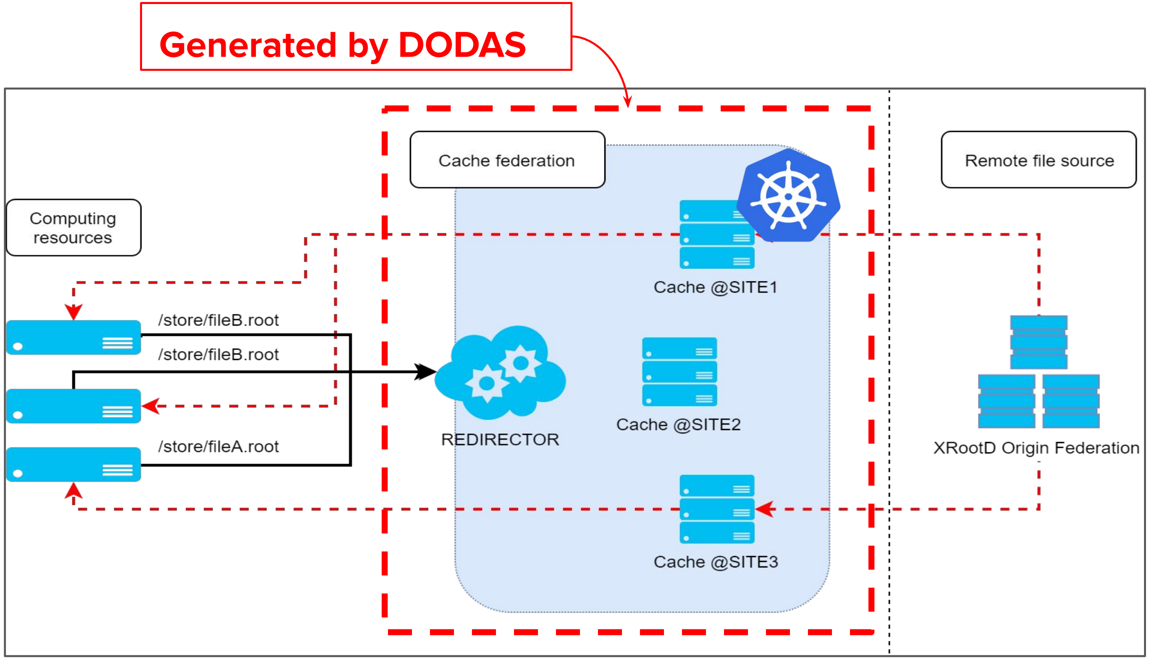
From CHEP 2019



DODAS and Data Caches

We rely on **XRootD** technology and we support configuration of a variety of services.

- **Data Server:**
- **Redirector:**
- **XCache:**




```
$> helm repo add cache https://cloud-pg.github.io/CachingOnDemand/
$> helm repo update
$> helm install cache/cachingondemand
```

Summary

DODAS is a high modular deployer manager build on the concept of Infrastructure as a code. Today we discussed:

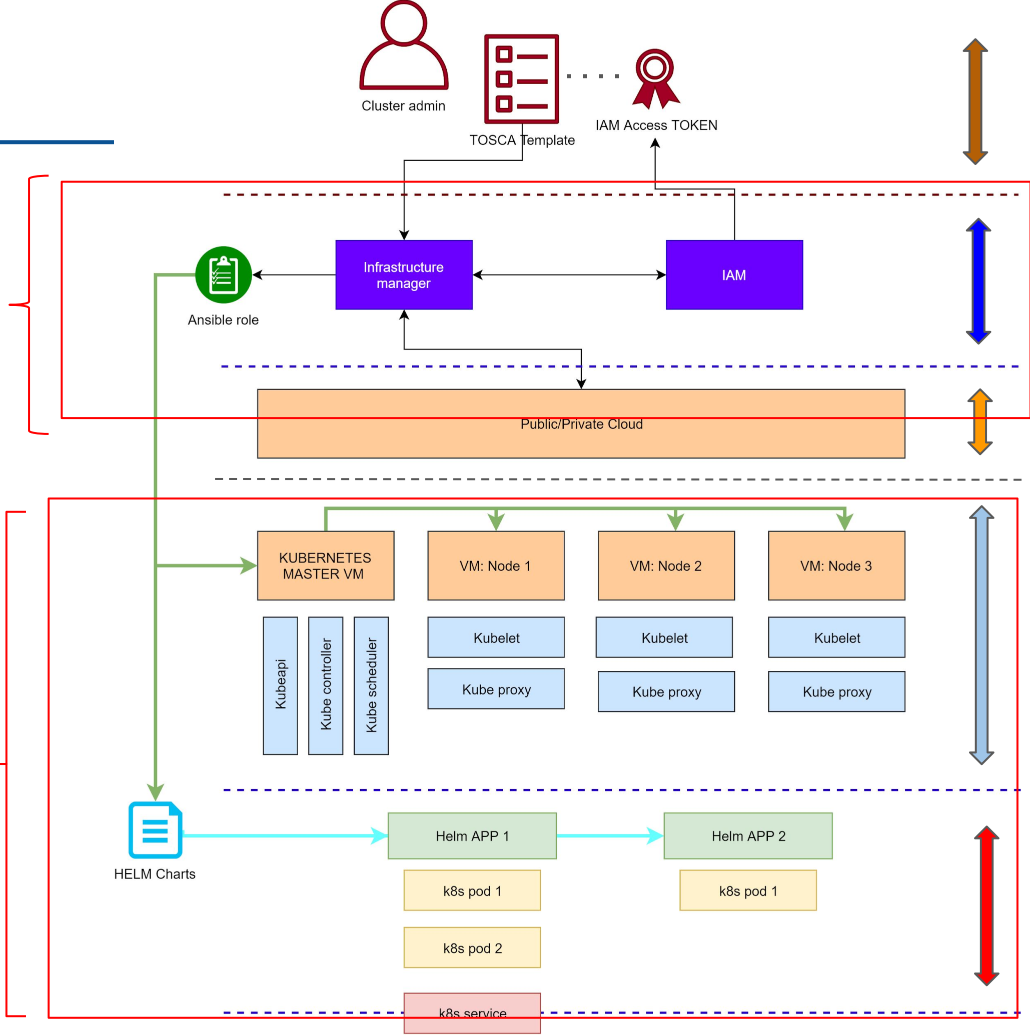
- How to **automatically deploy Spark on any cloud environment**
  - Allowing the customisation of the underlying environment (e.g. dependencies) and to compose the computational stack (DBs, FS, storages/caches)
- How we support more generic use case such as **K8s on demand**
  - includes compute and data creation and orchestration and federation

DODAS also support **on-demand analysis facility** (on top of K8s):

- HTCondor batch on demand, including HTCondor federations
  - Floking, routing and HTC/HPC mixing (CHEP2019)
- Spark cluster
- TFaaS (reference)

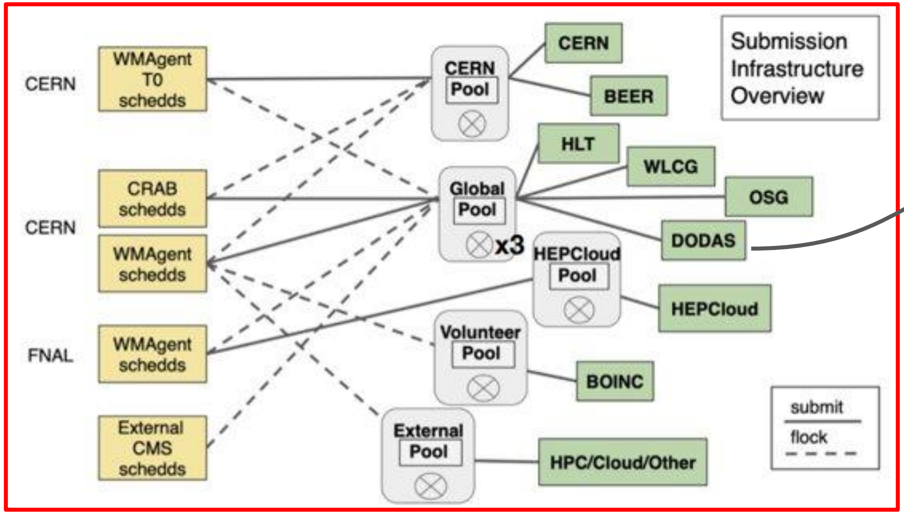
Future work and R&D

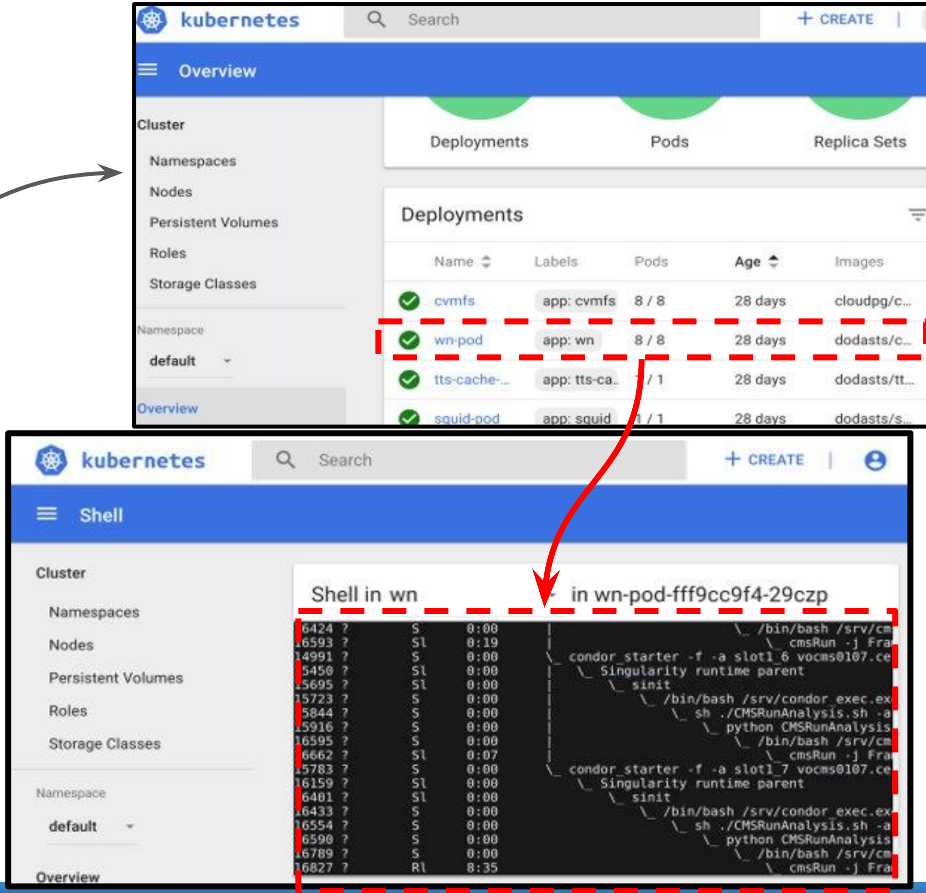
- DODAS specific
  - Improve and evolve the support for bare metal (instead of Cloud API)
  - Improve/evolve User interface (GUI/CLI)
- K8s oriented:
  - Autoscalers with custom metrics
  - Federated k8s
  - Integrated Authentication layer



Extra slides

Example: The CMS Integration





A Key component is the AuthN/Z:

DODAS is based on JWT (INDIGO-IAM). To integrate the CMS GlobalPool:

- start with JWT Token as incoming auth credential
- Implements security via IAM token exchange
- Cache and return X509 certificates to grant access to CMS

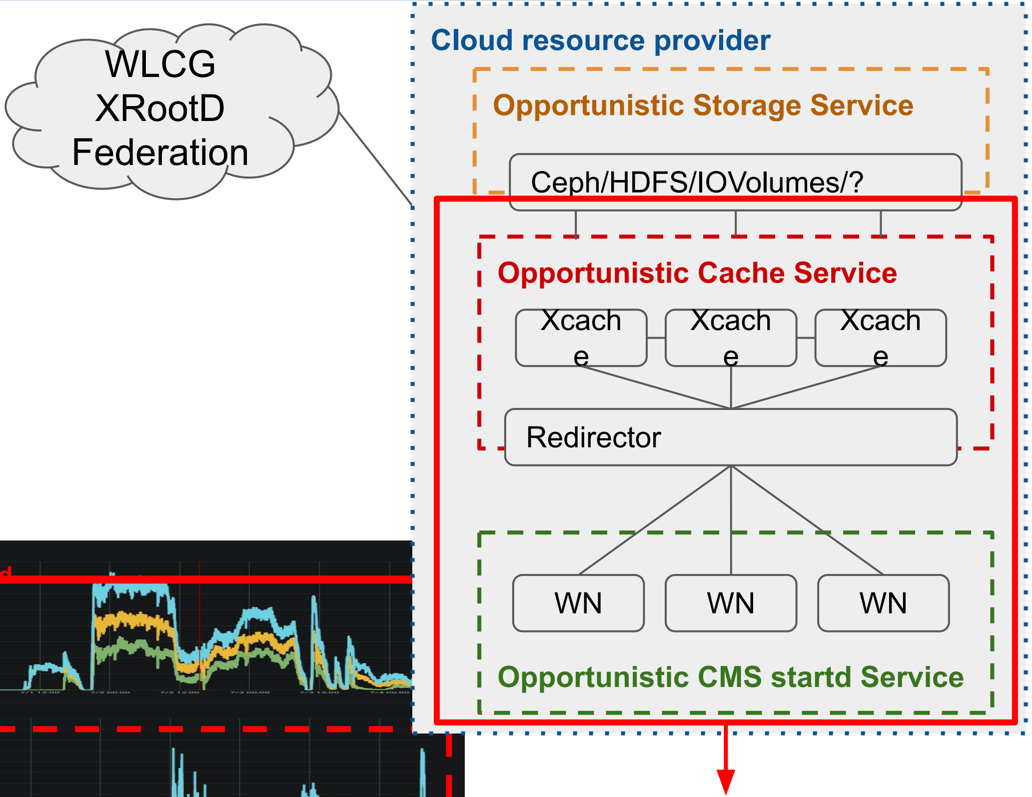
--> Global Pool authorization is based on DN mapping

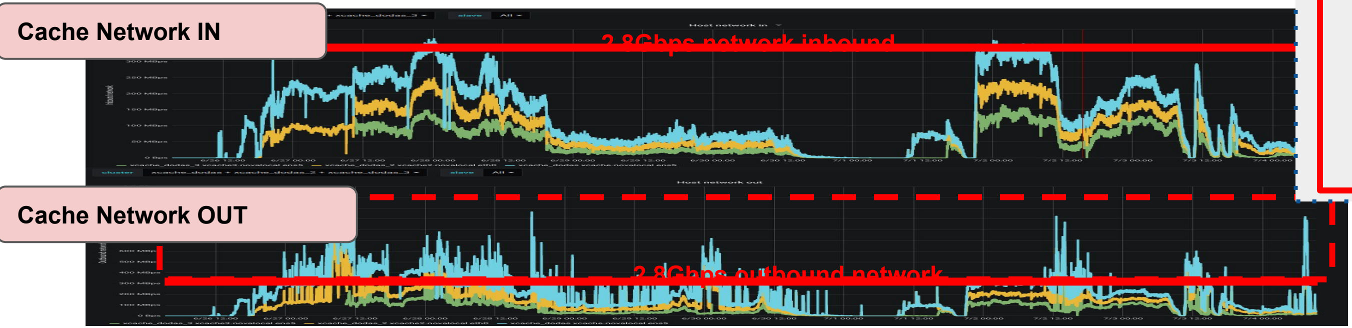
Finally: Not Only CMS

DODAS is also under evaluation (at different level of testing and integration) by **communities other than CMS**

- **AMS Experiment** is already testing/evaluating DODAS to run analysis over opportunistic resources
- **Fermi** analysts are already using (for daily activities) DODAS
- **Virgo** is integrating a pipeline for testing the whole flow

Putting everything together





Generated by DODAS