tommaso.tedeschi@cern.ch

K8s autoscaling based on custom metrics. Two examples of application: CMSWEB and HTCondor in the CMS Analysis Facility@INFN

Tommaso Tedeschi[1,2], Daniele Spiga[2], Diego Ciangottini[2], Valentin Y Kuznetsov[3], Muhammad Imran[4,5]

[1] Università degli Studi di Perugia, Perugia, Italy
[2] Istituto Nazionale di Fisica Nucleare - Sezione di Perugia, Perugia, Italy
[3] Cornell University, New York, USA
[4] National Centre for Physics, Islamabad, Pakistan
[5] CERN, Geneva, Switzerland

Also thanks to **INFN-cloud** team

Outline

The aim of the present work is to implement and evaluate **automated resource scaling based on custom metrics**

- We will show how we use it in 2 different cases:
  - CMSWEB for **DBS service**
  - ANALYSIS FACILITY @INFN for **HTCondor**

K8s - native autoscaling

- The K8s **Horizontal Pod Autoscaler** (HPA) automatically scales the number of Pods in a replication controller, deployment, replica set or stateful set based on some observed metrics.
- It is implemented as a Kubernetes API resource and a controller. The resource determines the behavior of the controller which in turn **periodically adjusts** the number of replicas in order to make the metrics value stay below a threshold value set by user.
- The controller manager usually obtains the metrics from the **resource metrics API** for per-pod resource metrics (CPU and memory usage) and from the **custom metrics API** (or from the external metrics API) for all other metrics (included in HPA resource beta version `autoscaling/v2beta2`).
- The controller then takes the current metric value and produces a ratio used to scale the number of desired replicas:

```
desiredReplicas = ceil[currentReplicas * ( currentMetricValue / desiredMetricValue )]
```

Our problem

To scale up and down systems and services on the basis of metrics whose effect on CPU or memory usage is **not predictable**. Concretely we wanted to address:

CMS DBS (a Long-running service)
a key point to scale such a service efficiently is to do it on the basis of the number of accesses by users, not directly mapped to resource usage!

Batch systems (e.g. HTCondor for analysis workflow)
Here scaling has to be based on information about the overall status of the system!

Autoscaling based on custom metrics

Key elements for scaling pods on the basis of custom metrics:

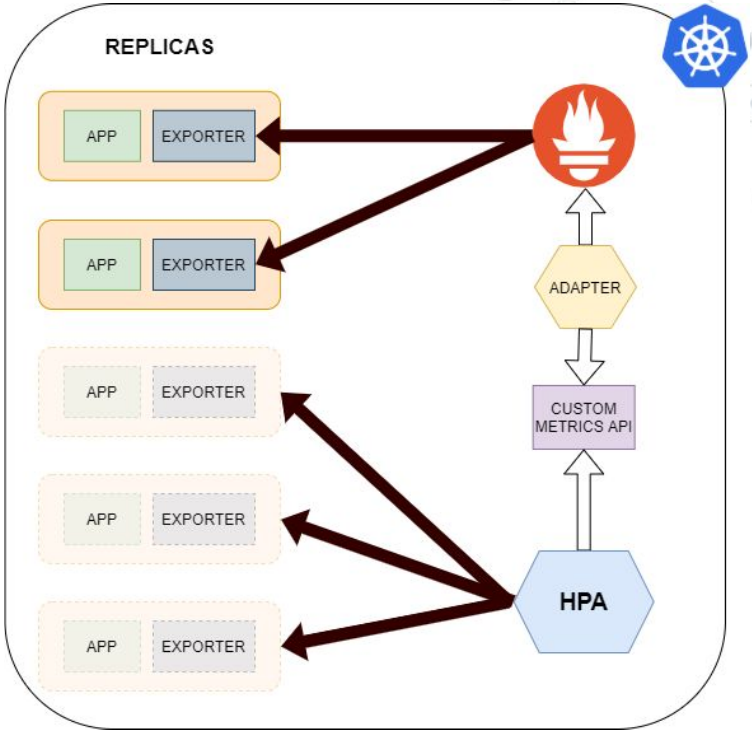
- a monitoring server to collect custom metrics
- an exporter, i.e. a web server that exports and makes metrics from the specific application available to the monitoring server
- an adapter that acts as a link between the monitoring server and Kubernetes exposing metrics through Custom Metrics API.

Our strategy: a Prometheus server (https://prometheus.io/), a Prometheus Adapter (https://github.com/DirectXMan12/k8s-prometheus-adapter) and application-specific exporters.

Implemented workflow

https://github.com/Cloud-PG/prometheus-hpa

1. **Exporter**: *exports internal metrics from the application of interest, converting them to predefined format*
2. **Prometheus Server**: *collects all the metrics from various exporters in the form of time series*
3. **Adapter:** *exposes manipulated Prometheus time series through* ***Custom Metrics API***
4. **Horizontal Pod Autoscaler**



CMSWEB: scaling on process open fds

CMSWEB is a cluster that hosts essential CMS experiment central services which are responsible for the CMS data management, data discovery, data bookkeeping tasks (20+ services maintained by CMS operators).

- CMSWEB cluster was recently migrated to K8s (*see Muhammad Imran talk tomorrow*)

⇩

Data Bookkeeping Service (DBS)*, a CMSWEB service, provides the necessary information used for tracking datasets (e.g. the data processing history, files and runs associated with a given dataset)*

Autoscaling has been applied to scale DBS service at CMSWEB using **process open file descriptors (fds)** metric, which is directly related to the number of accesses to service: when too many requests, a new pod has to be deployed

The flow

1) A **DBS-specific exporter** retrieves a custom metric `dbs_global_exporter_process_open_fds`

2) **Prometheus** scrapes that exporter and saves that metric as a time series.

```
rules:
  - seriesQuery: 'dbs_global_exporter_process_open_fds'
    resources:
      template: "<<.Resource>>"
    name:
      matches: "^(.*)"
      as: "${1}"
    metricsQuery: 'max(<<.Series>>) by (job)
```

3) The **Adapter** exposes `max(dbs_global_exporter_process_open_fds)` through Custom Metrics API

4) **HPA** resource scales `dbs-global-r` deployment

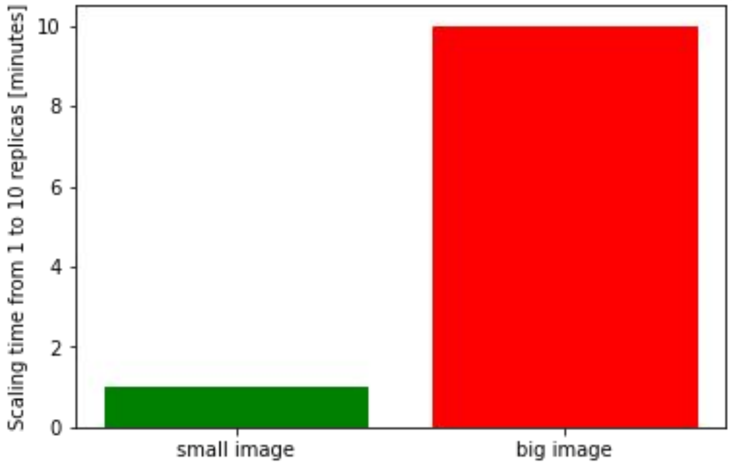
```
scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: dbs-global-r
  minReplicas: 8
  maxReplicas: 10
- type: Object
    object:
      metric:
        name: dbs_global_exporter_process_open_fds
      describedObject:
        apiVersion: batch/v1
        kind: Job
        name: dbs-global-r
      target:
        type: Value
        value: 800
```

Consideration: Quick scaling

Scaling must be quick..

The duration of scaling procedure depends on image size

- We measured autoscaling time for two different deployments based on two httpgo server images with same functionalities but different sizes
  - **(19.17 MB vs 325.96 MB) effect of ~10X**



Analysis Facility at INFN

The DODAS project is working on building CMS **analysis facility** (**AF**):

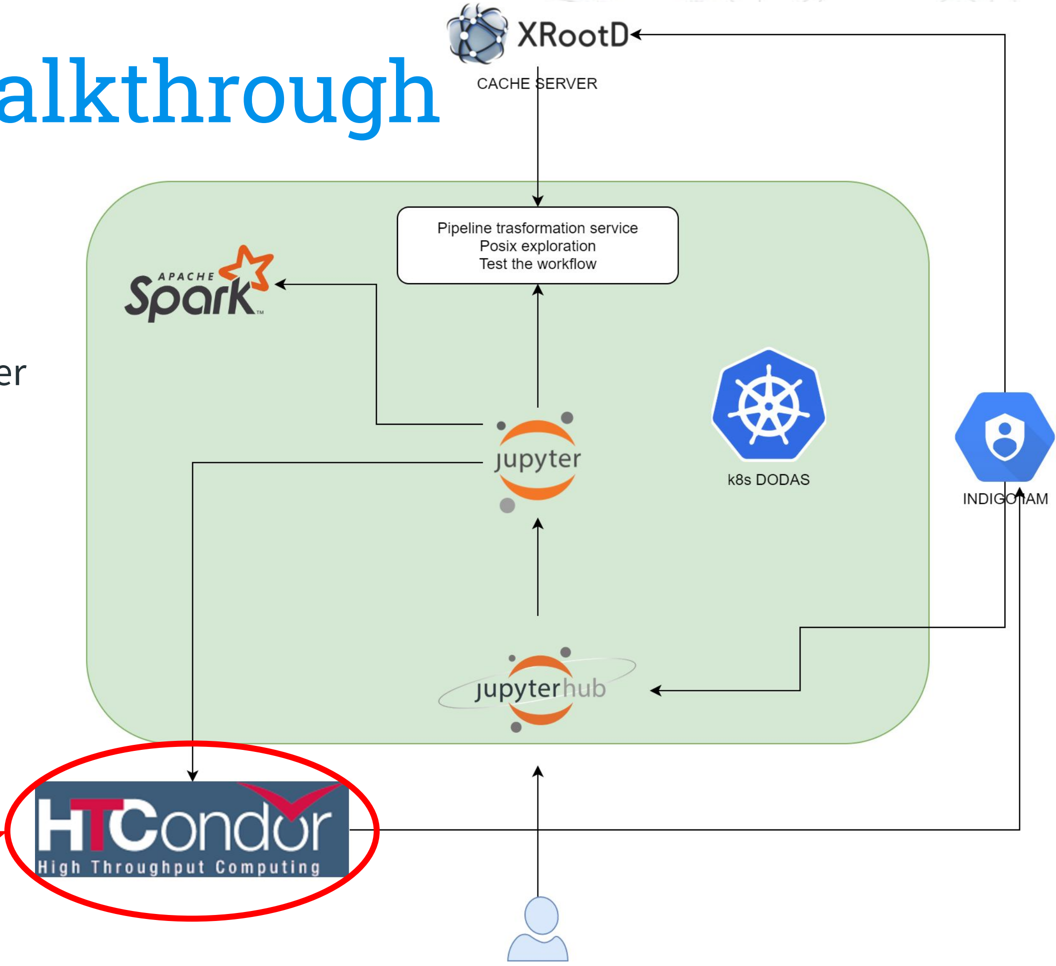
- Highly based on services composition model (see later)
- Focus on nanoAOD based workflows
- Facilitate Python ecosystem exploitation
- Support exploitation of Machine Learning pipelines
- Targeting CMS but not CMS specific

Fully integrated into the INFN-Cloud (the national federated Cloud infrastructure) Portfolio

AF@INFN: a quick walkthrough

Automated (on-demand) deployment via DODAS

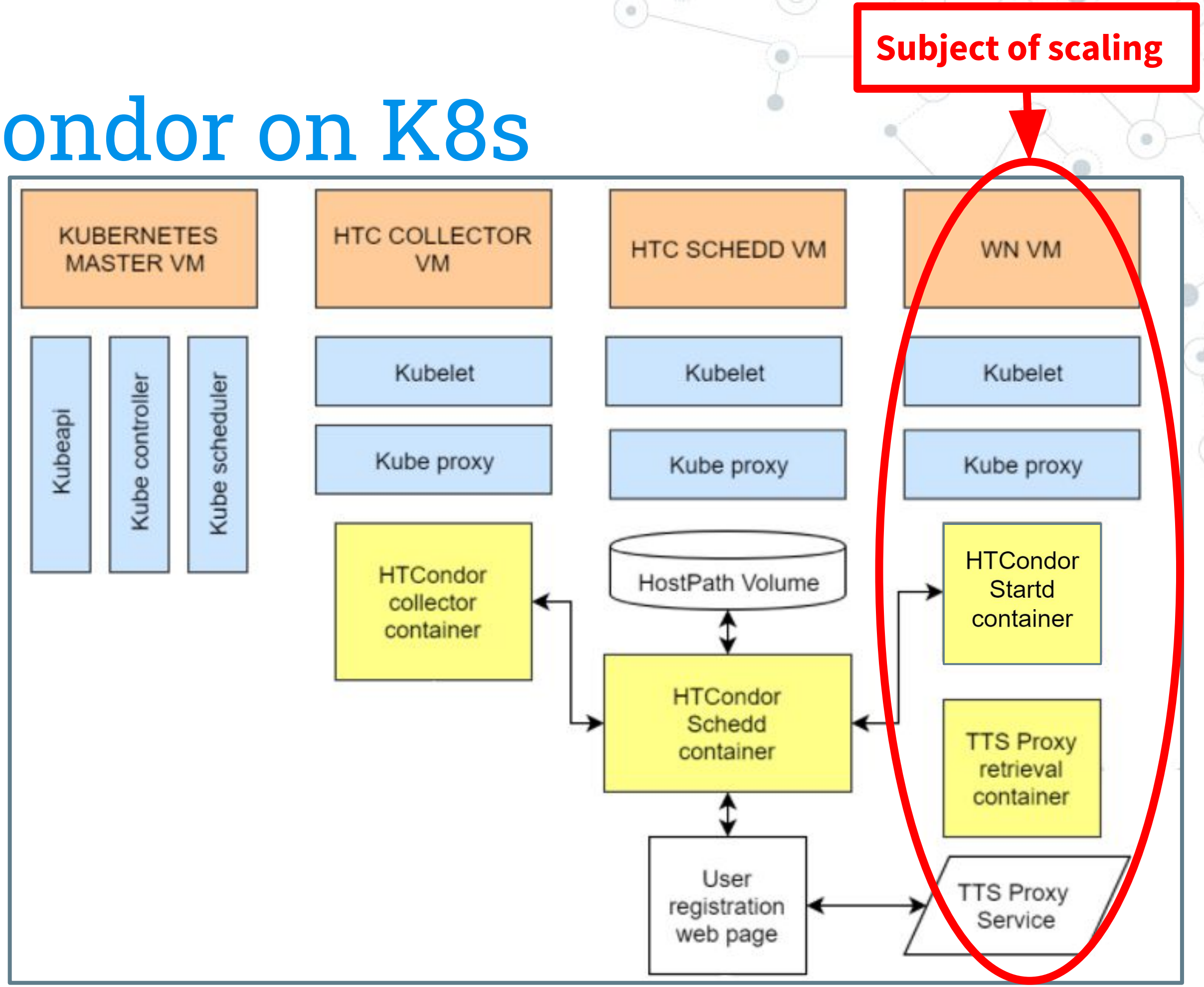
- JupyterHub/Jupyter (+ Spark) on K8s cluster
- HTCondor on K8s (containerized experiment software deployed on worker nodes)
- Submission to **HTCondor** via Jupyter
- XRootD cache server at CNAF
- Token-based authentication via Indigo-IAM
- Pipeline transformation service + Posix exploration + Test the workflow



Subject of scaling

Scaling HTCondor on K8s

- HPA targets **WN deployment** on the basis of metrics summarizing **jobs or machines status**



Helm chart:
https://github.com/DODAS-TS/helm_charts/tree/master/stable/htcondor
Values:
https://gist.github.com/ttedeschi/e1dc94561e0de9ce34ccfdaa41bbd53e

HTCondor exporter

- Exporter based on https://github.com/niclabs/htcondor-monitor: interaction with HTCondor cluster via HTCondor Python bindings which expose a Pythonic interface to the HTCondor client libraries (https://htcondor.readthedocs.io/en/latest/apis/python-bindings/index.html).
- Exported metrics:

condor_slot_activity_idle	Is this slot idle
condor_slot_activity_busy	Is this slot busy
condor_slot_state_owner	Is this slot in the owner state
condor_slot_state_claimed	Is this slot in the claimed state
condor_slot_state_unclaimed	Is this slot in the unclaimed state
condor_job_state_idle	Number of jobs on the idle state for a given cluster and submitter
condor_job_state_running	Number of jobs on the running state for a given cluster and submitter
condor_job_state_held	Number of jobs on the held state for a given cluster and submitter
condor_job_state_completed	Number of jobs on the completed state for a given cluster and submitter
condor_job_avg_running_time_seconds	Average running time for completed jobs for the specific cluster and submitter

Much information about HTCondor ClassAds can be retrieved and exposed: **any exposed parameter could be a metric for scaling, great flexibility**

Our first approach: condor_slot_activity_busy

```
rules:
  - seriesQuery: 'condor_slot_activity_busy'
    resources:
      template: "<<.Resource>>"
    name:
      matches: "^(.*)"
      as: "${1}"
    metricsQuery: 'avg(<<.Series>>) by (job)'
```

- `avg(condor_slot_activity_busy)` is then exposed by **Adapter** and made available to Horizontal Pod Autoscaler. This value identifies the ratio between the number of busy machines and the total number of machines.
- The **Horizontal Pod Autoscaler** resource is then created targeting wn-pod deployment: when the metrics value goes above the threshold value, wn-pod deployment is scaled up.

```
scaleTargetRef:
  apiVersion: apps/v1
  kind: Deployment
  name: wn-pod
minReplicas: 1
maxReplicas: 10
metrics:
- type: Object
  object:
    metric:
      name: condor_slot_activity_busy
    describedObject:
      apiVersion: batch/v1
      kind: Job
      name: htcondor-pod
    target:
      type: Value
      value: 0.75
```

This is only a first approach: ongoing optimization studies allowed by aforementioned flexibility

Testing environment

*nanoAOD file:
https://cmsweb.cern.ch/das/request?input=file%3D%2Fstore%2Fmc%2FRunIISummer16NanoAOD%2FTTJets_TuneCUETP8M1_13TeV-madgraphMLM-pythia8%2FNANOAODSIM%2FPUMoriond17_05Feb2018_94X_mcRun2_asymptotic_v2-v1%2F40000%2F2CE738F9-C212-E811-BD0E-EC0D9A8222CE.root&instance=prod/global

Infrastructure

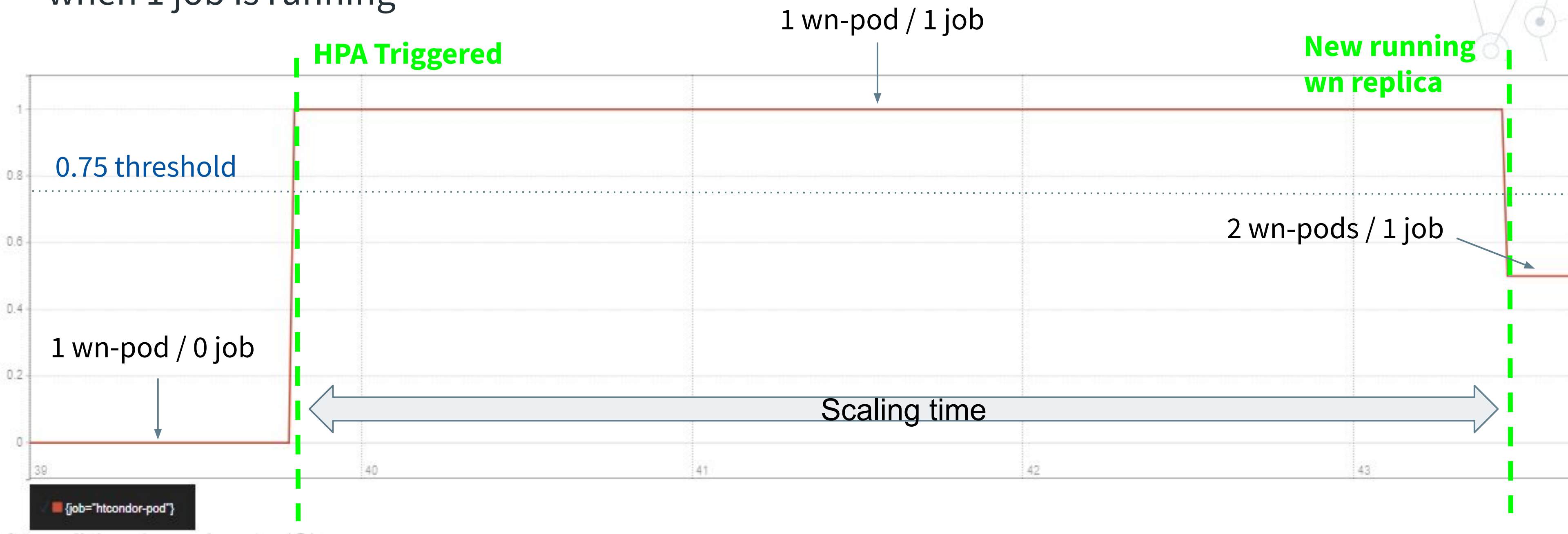
- HTCondor cluster @ReCas Bari (Italy):
  - 1 K8s master: 2 cores, 4 GB
  - 6 K8s slaves: 4 cores, 8GB each
- Deployment via DODAS (using Tosca template and helm charts)

Job

- 1.897GB nanoAOD mc file*
- Example nanoAODtools analysis code from https://github.com/cms-nanoAOD/nanoAOD-tools

Initial tests

`avg(condor_slot_activity_busy)` as a function of time, scaling from 1 to 2 pods if when 1 job is running



Open point: under investigation

- Explore solutions to get a **“targeted” down-scaling**, deleting only wn-pods that are not busy.
  - Right now K8s does not follow any rule to choose pod to be deleted when metrics are below threshold value: *so running WNs may be killed*.
  - The problem of killing running WN is now mitigated by the usage of *long-time cooldown*, but there’s room for improvement

Summary and next steps

- The implemented scaling approach is fully **generic and reusable**: seems all ok from all our early testing.
- Keep consolidating the described **Analysis Facility** and fine tuning scaling strategy evaluating various parameters
  - Including automation procedure
  - The goal is to support first analysis (in production): Vector Boson Scattering SSWW with hadronic tau in final state (Jan 2021)
- Investigate **scaling over distributed cluster,** fitting the INFN-Cloud topology
- Keep integrating with **DataLake** testbeds
  - Synergies with ESCAPE and DOMA Access and IDDLS (INFN national project)

BACK UP

Requirements and further details

- Prometheus exporter should be written following some general rules (https://prometheus.io/docs/instrumenting/writing_exporters/) and expose metrics in a predefined text-based format (https://prometheus.io/docs/instrumenting/exposition_formats/)
- Many ready-to-use exporters (https://prometheus.io/docs/instrumenting/exporters/, https://github.com/prometheus/prometheus/wiki/Default-port-allocations)
- Prometheus targets are defined via scrape_configs (https://prometheus.io/docs/prometheus/latest/configuration/configuration/)
- Prometheus Adapter is configured via a set of rules (Discovery, Association, Naming and Querying https://github.com/DirectXMan12/k8s-prometheus-adapter/blob/master/docs/config.md)

INFN-Cloud in a nutshell

INFN-Cloud: is a national distributed infrastructure

Architecture:

- **An INFN Cloud backbone** spanning the two main INFN computing sites (CNAF and Bari).
- **A set of distributed, federated cloud infrastructures** connecting to the backbone.
  - Several INFN sites are already connected, with a few others in the pipeline.