Scalable and replicable Kubernetes Platform for ML_INFN

Introduction

A large amount of highly valuable digital data is produced on a daily basis in the context of the scientific activities of the National Institute for Nuclear Physics. ML_INFN is an initiative of *Commissione Scientifica Nazionale 5* aiming at fostering the application of modern data analysis techniques, including machine learning, to a large spectrum of scientific applications with actions organized in three *work packages*:

- WP1 - Infrastructure. Providing users and scientific communities with a test bed for evaluating the suitability of modern data analysis technologies and techniques to their use cases. Specialized hardware, including accelerators, and customizable configurations of the environment are made available by federating a set of dedicated servers in INFN Cloud.
- WP2 - Stewardship. Organizing training events (*hackathons*) where the applications of machine-learning models to INFN research activities is discussed with the time for discussing implementation details and experimenting with the code.
- WP3 - Use cases and knowledge base. Activating a network of INFN units to collect example applications in a national knowledge base, fostering a network of competences and a forum for discussion transversal to the experiments and to the INFN units involved in the developments.

During the last few months, a significant increase in the number of requests to access the ML_INFN resources for development purposes is setting competition on the access, in particular, to CPU and GPU resources, calling for an evolution of the computing model to enable a more efficient and effective share of the resources, a more reliable monitoring and accounting infrastructure, and a more flexible setup enabling offloading tasks to other resources in periods of higher request, as for example during the hackathons.
In this document we discuss a possible evolution of the compute infrastructure of ML_INFN providing details on the motivation first, and then describing the technological solutions we have identified for short- and long-term developments.

Motivation

The current infrastructure providing resources to the development activities in ML_INFN and, at least partially, to the ML_INFN hackathons, relies on two servers physically located in the Tier-1 hall, at CNAF with the following setup:

- Server 1
    - 64 cores; 750 GB of memory; 24 GB of NVMe storage

    - GPUs: 8 nVidia T4; 5 nVidia RTX5000
- Server 2
    - 128 cores; 1024 GB of memory; 12 GB of NVMe storage
    - GPUs: 1 nVidia A30; 2 nVidia A100
    - FPGAs: 2 Alveo U50; 1 Alveo U250

Most of the resources were funded by CSN5, while some of the GPU was contributed by projects or INFN units aiming at a privileged access to ML_INFN resources while accepting opportunistic usage of their resources by other ML_INFN users.

The two servers are federated to INFN Cloud and dedicated TOSCA templates were designed to enable instantiating Virtual Machines (VMs) with access to GPUs and FPGAs.
The number and the model of the GPUs, the number of virtual cores assigned to each VM, the memory and storage resources are defined by selecting an *OpenStack* flavor when instantiating the VM.
User authentication and authorization is managed by multiple INFN Cloud IAM services. In particular a IAM instance is used to manage the identities of ML_INFN users accessing the resources for research purposes and external users accessing the resources for a limited amount of time for live demos or to practice with accelerators during hackathons and training courses
A restricted number of ML_INFN participants, with an official nominee to administrate cloud resources, is allowed to instantiate VMs, while most of the users access the resources via JupyterHub services running on each VM as defined by the TOSCA template.
Administrators are supposed to update the software packages running on the machines to preserve the functionalities upon updates of the INFN Cloud infrastructures and to fix vulnerabilities upon their discovery. In practice, most ML_INFN users request to become administrators to gain some flexibility on the definition of the resources assigned to their projects, by selecting their preferred OpenStack flavors or allocating additional VMs for example to prepare their application to move to scalable computing environments involving more than one hardware accelerator.

The current setup is subject some important limitations:

- Since VMs are statically assigned to projects and GPUs are statically assigned to VMs, **the amount of resources reserved to each project is fixed**: GPUs cannot be reassigned to other projects without destroying the respective VM, destroying the filesystem with possible user data losses. On the other hand, it is not possible to extend the amount of resources assigned to a project without re-instantiating the VM, which limits collaborative programming sessions.
- The VMs cannot be deallocated without an explicit consensus of the users, whose data would be lost. In the late phases of the projects, when the publication of the results approaches, users access resources sporadically, but consider as crucial the preservation of the exact compute environment, including temporary files and intermediate results. As GPUs cannot be

reassigned without destroying the file system, **GPUs may remain unused for long periods**, independently of the demand from other projects.

- The vCPUs assigned to a VM are strictly bounded to the GPUs assigned to the same VM. This implies a **difficult static tuning of the number of CPUs per GPU** which could be simplified with a setup with VMs equipped with multiple GPUs assigned to different projects by managing the allocation at container level. Note that the typical usage of CPU resources for development activities alternates bursts of activity, for example for preprocessing and loading a batch of data to the GPU, and significant periods of inactivity, for example for writing the code, reading the plots produced as output and studying the documentation. Sharing a batch of vCPUs among multiple users sounds therefore reasonable.
- Batch jobs exploiting GPUs would require dedicated VMs, becoming competitive with interactive access to the resources while, conceptually, they may exploit time slots unused for interactive development such as nights and weekends. In other words, **the current setup does not enable opportunistic, batch usage of the resources.**
- From an experiment perspective, **autoscaling policies are difficult to implement**, setting hard constraints on the amount of available resources, and representing a major hurdle for offloading tasks to external resources.
- Becoming administrators for the sole purpose of gaining flexibility on the resource provisioning results in a significant multiplication of critical administrative tasks on a large number of very similar deployments managed by different people. Supporting the administrators in their tasks is less efficient than managing a unique service platform and is more **prone to vulnerabilities due to outdated packages** at least in some of the deployments.

Most of these limitations can be overcome by enhancing the flexibility of a centrally-managed ML_INFN platform designed to decouple the access to compute resources from storage while preserving fast access to the local disk for caching purpose. While tailoring the service on the resources we manage at CNAF, we aim at a platform sufficiently general to be deployed, as separate services, on multiple sites. For example, we should be able to deploy a temporary version of the platform on *HPC bubbles* physically located outside CNAF, or at ReCaS, to respond to the high demand for interactive resources during training events.

The Kubernetes Overlay

The services composing the ML_INFN platform will be organized in a Kubernetes Overlay providing a dynamically extensible platform with a seed (i.e. a minimum amount of resources always available) including:

- A single and persistent JupyterHub service configured to spawn JupyterLab via Kubernetes on nodes that can be elastically assigned to the Kubernetes cluster. K8s nodes are dynamically instantiated depending on the needs. Initially we plan to start with manual approach (i.e. via a

new Tosca template) in order to achieve experience and validate the system, while the ultimate goal is to automate the whole process, for example relying on a k8s autoscaler triggering the k8s node instantiation.

- A NFS server that will serve the whole cluster, including the nodes eventually added dynamically. Kubernetes will mount the NFS volumes to the JupyterLab instances spawn on demand and local resources (on the node) will be used as cache for optimizing the I/O. The ultimate goal of this approach is to ensure that once a JupyterLab instance is deallocated (by explicit user action or by cull-off) the exact content of the file system is stored at service-level and can be automatically restored upon respawning of the JupyterLab by JupyterHub. In other words the user file system will be persistent in the context of the platform. Once the platform is destroyed the data will be deleted.
- A HTCondor local batch system, functional to exploit the opportunistic usage of the compute resources. Resources that are not used for interactive development can be exploited opportunistically to process jobs organized in queues defined by an instance of HTCondor part of the same Kubernetes cluster. HTCondor may be used to spawn Dask worker nodes to opportunistically exploit unused resources also in interactive mode.
- A dedicated PostgreSQL database enabling:
    - advanced monitoring and accounting metrics (before they are ported in INFN Cloud infrastructure)

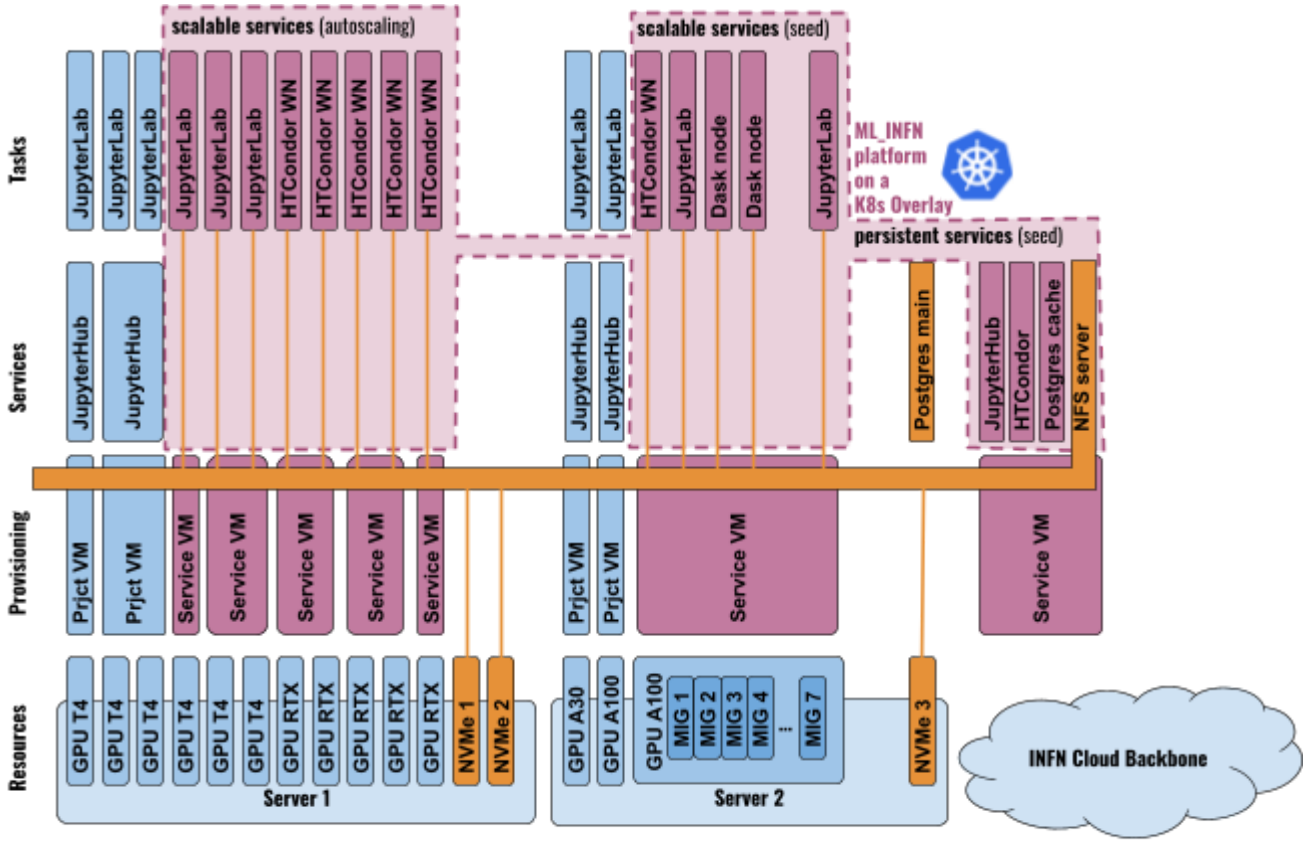

Figure 1. Schematic overview of the ML_INFN platform

- continuous data ingestion from remote sources to be processed with batch jobs for example to keep models updated (online learning)
- a persistent backend for kubernetes instead of `etcd` to reduce the points of failure.

At least in a first transition phase, this new platform will live side-to-side with the current resource provisioning model. It will be the responsibility of the administrators to arbitrate the resources between use cases requiring access to the VMs (as of today) and assigning resources to either a project or to the main platform by instantiating new VMs with the appropriate Tosca template. Hence, starting with a manual approach will be perfectly suitable to commission and evaluate the platform. In the future we aim at automating the process of allocating and destroying resources to be assigned to the service cluster based on the demand according to the principle of autoscaling.

Figure 1 reports a schematic overview of this platform. We have represented in blue the elements of the current setup, including resources, resource provisioning through static VMs, services instantiated on the static VMs and tasks that today are limited to Single-Node-JupyterLab containers spawned by JupyterHub instances on the same VM.
Grape-colored blocks represent compute elements of the proposed service, providing resources to multiple projects. Storage elements are represented in orange.

Project-level resource sharing and experiment-level autoscaling

In the discussion above there are two different concepts of resource scaling that may lead to confusion so that it is worth making the due distinctions.
From the user perspective, automatically scaling the resources made available to a project means enabling use cases in which multiple users aim at collaborating on the same set of notebooks, on the same shared filesystem with a GPU assigned to each user. As of today, either the whole amount of GPUs is statically assigned to the VM at the time of the deployment, or it is impossible to add and remove resources to the file-system of the user's project.
This approach enhances resource sharing between members of the ML_INFN initiative and has the potential of increasing the effectiveness of the GPU resources drastically reducing the time needed to reallocate the resource from a project to another.
However, this project-level resource sharing has no impact on the amount of resources provisioned to the ML_INFN initiative: as long as we hold the GPU for usage within the ML_INFN platform we pay for it independently of whether it is assigned to a project or not.
An experiment-level auto scaling implies the ability of releasing unused resources to the benefit of other initiatives relying on INFN Cloud but independent of ML_INFN and its platform. Clearly, the complexity of a system able to acquire and release the resource from a pool of resources accessed by multiple activities increases both in terms of technologies and policies.

From a technological perspective, handling project-level resource sharing implies assigning compute resources to different users with very similar use cases, accessing the resources with JupyterHub, and sharing a unique distributed file system. On the other hand, experiment-level autoscaling implies automatically instantiating VMs via INFN Cloud.
Concerning the policies, to the purpose of ML_INFN it is reasonable to assume that interactive development has much higher priority than batch processing, which is intended for opportunistic usage of resources in time slots unsuitable for interactive development. Other experiments and activities on the Cloud may have a priority higher than interactive development, so that the policy should rely on concepts of pledged and opportunistic resources as developed in the context of the WLCG.

File-system hosted on ML_INFN resources and distributed via NFS

A crucial requirement to decouple the compute and storage resources is to deploy a file system that can be transparently accessed by the user independently on the machine where the compute task is spawned.
As mentioned in the introduction, we would like to rely on the fast storage resources installed in the ML_INFN cluster. An NFS server will be part of the Kubernetes Overlay as a persistent service part of the cluster seed. Kubernetes will then mount the NFS volumes to the JupyterLab sessions and to the HTCondor worker nodes spawn on demand. Local storage resources will be used as cache for optimizing the I/O.
This approach should be sufficient to ensure that once a JupyterLab instance is deallocated (by explicit user action or by cull off) the exact content of the file system is stored at service-level and can be automatically restored upon respawning of the JupyterLab by JupyterHub.
In other words the user file system will be persistent in the context of the platform. Once the platform is destroyed the data will be deleted.
A frequent backup policy might be adopted, to persist on INFN Cloud resources (for example in MinIO) a copy of the entire NFS-served file system to make it possible to recover the whole setup in case of scheduled or accidental interruption of the service.

Database

A PostgreSQL database server will be part of the Kubernetes Overlay as a persistent service in the cluster seed. While it is important to have such a service included in the platform to make it easy to replicate the whole setup in other sites, we should aim at a configurable level of data persistency.
For example, we may configure the PostgreSQL database in the overlay to mirror a main PostgreSQL instance running in a dedicated VM in INFN Cloud. The main advantages of this setup are:

- the services on the overlay interacting with the database can address the server independently of the site where the overlay is deployed relying on cluster network

- deploying the cluster on a new site does not require a separate effort to configure a compliant database
- data committed to the database are immediately stored in a persistent way, independently of the overlay itself. This is of particular importance especially during the commissioning phase, when frequent restart of the Kubernetes overlay should not cancel the history of the events logged in the database.

As mentioned in the introduction, the main use cases for a platform-dedicated database are:

- advanced metrics for monitoring and accounting, completing and extending the information made available by the INFN Cloud infrastructure, as for example:
  - allocation of VMs to to either the main platform or single projects
  - GPU-specific metrics, such as power consumption as a function of time
  - platform-specific metrics, such as number of concurrent users and overall load on the allocated resources
- data ingestion from remote services for example submitting experimental data to be processed in batch or interactive jobs in the cloud, a properly configured data ingestion layer close to compute resources may represent a no-effort option for small experiments geographically distributed producing data that need to be monitored on a regular basis (*e.g. radon monitoring networks, muography, …)*. TensorFlow provides the tools to stream data from a PostgreSQL database directly into the GPU memory.
- a persistent backend for kubernetes instead of `etcd` to reduce the points of failure
- a database backend for HTCondor

As of today, it seems that PostgreSQL does not support user authentication based on OIDC tokens. For most applications this is not needed as users are not supposed to access the database directly. For the task of data ingestion, however, letting users to authenticate with AIM-provided OIDC tokens may be a better solution than managing PostgreSQL users separately. We have started investigating postgrest, a REST-mediated access to the PostgreSQL database that may provide a solution for token-authenticated access to the database.

Opportunistic usage of the resources with HTCondor and Dask

As mentioned above, providing evaluation and interactive development to the INFN communities implies the availability of a sufficient amount of resources to satisfy the demand of multiple users trying to simultaneously access the resources, enforcing a queuing time which cannot be much longer than a handful of seconds. Such a computing model implies that a large fraction of the compute time that a resource (CPU, GPU or FPGA) could provide cannot be exploited in interactive mode.
However, longer training procedures, or periodic CI/CD workloads validating updates to machine learning pipelines, or updates of statistical models based on new data, could easily run in batch

mode, during nights and weekends when the demand for interactive access to the resources is negligible. At the same time, users accessing the resources interactively may profit from unused resources to speedup their applications by offloading part of the processing to other resources available in the cluster with libraries such as Dask Distributed.
To increase the effectiveness of the resources granted to ML_INFN, we aim at providing a solution to enqueue tasks in local batch system configured to run with a much lower priority than interactive usage. We identified HTCondor as batch system.
The HTCondor service should be able to discover idling GPU resources on the cloud and to run jobs from a queue of opportunistic payloads on those resources.
We may use HTCondor, or Kubernetes directly, to spawn Dask worker nodes on the cluster to speed up interactive analysis.
From a technological point of view, we rely on Kubernetes to assign compute resources to either interactive JupyterLab instances, worker nodes processing tasks from the HTCondor queues, and worker nodes connected to Dask to speedup interactive data processing and analysis.
A possible alternative that has been discussed consist of defining three different HTCondor queues for these three classes of payloads and rely on HTCondor priority system also to spawn JupyterLab instances and Dask worker nodes. This second alternative would not change the overall picture of the platform which would still rely on Kubernetes to manage the resource provisioning, while relying on HTCondor to define and manage a unique and consistent priority system, relying on widely adopted metrics for accounting.
While in the initial phase of the project we plan at including the HTCondor setup in the Kubernetes Overlay, the maintenance costs of the setup could be significantly reduced by relying on the Tier-1 HTCondor service, properly instructed to use ML_INFN resources to process ML_INFN workloads.

Monitoring and accounting

In a mixed setup with project-dedicated VMs sharing the available resources with the ML_INFN platform described here, we aim at three levels of monitoring & accounting:

- **Resource provisioning accounting**. The amount of resources granted to ML_INFN allocated by OpenStack to the benefit of any activity related or unrelated to ML_INFN, must be made available to management of the initiative and, upon request, to the referees of CSN5. In other words, we should be ready to answer the question: *“how were the resources granted to ML_INFN allocated during the last year?”*.
- **Resource provisioning monitoring**. The effective usage of the compute resources granted to ML_INFN and used by the ML_INFN community, including the energy consumption of the GPUs per time slice, must be made available to the management of the initiative and, upon request, to the referees of CSN5. In other words, we should be able to answer the question: “*are the allocated resources employed or idling?*”. This second question is particularly relevant

to prioritize the eviction of project-dedicated VMs in case of pressure on the resources. Such monitoring metrics should enable measuring the level of success of the effort migrating to the platform proposed in this document for the purpose of enhancing the efficiency of GPU resources.

- **Service accounting**. Enabling batch access to the compute resources requires accurate monitoring and accounting to have clear the share of the resources among projects and, in case of pressure, to define rules that increase the priority of users with a history of lesser impact on the resources. Monitoring should include metrics for interactive usage of the resources designed to assess the peak demand for CPU and GPU resources which are relevant to the tuning of the resources allocated for interactive sessions. However, we do not plan to apply any penalty to interactive access to compensate for high consumption of resources in batch mode, so the two monitoring systems for batch and interactive usage might be completely independent. In other words, we should be able to answer the question: "*who is using the ML_INFN platform, to do what and how much we could shrink the CPU and RAM resources allocated to a single-accelerator task without an evident penalty in performance?"*.

Main drawbacks, limitations and possible mitigations

The proposed platform is intended for running on a single resource provider. The proposed overlay is not designed to be deployed on multiple interacting instances distributed across multiple sites (*e.g.* different INFN units). The whole platform will be designed to be easily replicated on other instances and resource providers, but the replicas will be completely independent of each other for all aspects concerning the access to the compute and storage resources. Identity management might be shared or not depending on the external IAM service selected for each replica.
The proposed solution for data persistency implies that the file system is shared among different services and tasks within the same replica of the platform. We might consider and extension to provide platform-managed persistency to project-dedicated VMs otherwise independent of the platform, as long as these VMs are deployed on resources from the same provider.

Also, dedicated services for managing the NFS master nodes, PostgreSQL and HTCondor permanent services would run continuously with a non-negligible impact on the accounting metrics of the ML_INFN project.
While this would still represent a crucial improvement over the current setup where several VMs are allocated for months without a single user access, this is still suboptimal and may be improved by enlarging the user base for these services, for example relying on similar services hosted and managed by INFN Cloud. This might be achieved in the foreseeable future for the Postgres deployment, for the storage in case of support from INFN Cloud for a national distributed file system, while the other services should be replicated together with the platform.

Another limitation worth mentioning is the small redundancy on the persistence of multiple users on a single storage resource. A very similar problem has been mitigated in the setup of the Sync&Share as a Service solution within INFN Cloud, by setting up frequent backups to MinIO. We may adopt similar strategies to increase the reliability of the platform.

Finally, we notice that moving from the current setup to the platform discussed here results into a significant modification of the “shared” data. Currently, users can move or copy their data to a shared folder in the jupyter filesystem to make them accessible to the other users accessing the same machine, or collaborating on the same project. With the proposed platform, data in the shared folder is shared among all the ML_INFN users.

Ongoing and future developments in INFN Cloud

Some R&D activities in INFN and INFN Cloud are designing a set of tools enabling the offloading of tasks, such as running JupyterLab notebooks, across multiple sites and possibly relying opportunistically on constrained submission systems. To make those developments reliable for interactive development of machine learning or data analysis tasks it is critical to distribute across the sites the users home.
In such a scenario, the platform described above could profit of cross-site resources replacing NFS with the same mechanism used to implement a national, cross-site home service.
In a preliminary investigation, we identified a limitation in the rclone package as a show stopper for using it to distribute computing environments.

Conclusion

We propose an ML_INFN platform, tailored to the needs of interactive development of machine learning models on the resources granted to the experiment by INFN, enabling an opportunistic usage of the resources temporarily not involved in interactive user access.
We expect the migration to this platform to significantly increase the effectiveness of the hardware accelerator resources granted to the initiative without any significant penalty in terms of user experience and we proposed a set of monitoring metrics and an accounting infrastructure to measure the impact of the proposed platform.
Future developments in INFN and INFN Cloud may profit of the experience obtained by building this infrastructure and also on the platform itself which might be replicated, and possibly tuned, to respond to similar needs for interactive access to hardware accelerated compute resources.