EPJ Web of Conferences **251**, 02045 (2021) https://doi.org/10.1051/epjconf/202125102045
CHEP 2021

First experiences with a portable analysis infrastructure for LHC at INFN

Diego Ciangottini[1], *Tommaso* Boccali[2], *Andrea* Ceccanti[3], *Daniele* Spiga[1], *Davide* Salomoni[3], *Tommaso* Tedeschi[1], and *Mirco* Tracolli[1]

[1] INFN Sezione di Perugia, Via Alessandro Pascoli 23c, 06123 Perugia (ITALY)
[2] INFN Sezione di Pisa, L.go B. Pontecorvo 3, 56127 Pisa (ITALY)
[3] INFN-CNAF, Viale Carlo Berti Pichat, 6/2, 40127 Bologna (ITALY)

Abstract. The challenges proposed by the HL-LHC era are not limited to the sheer amount of data to be processed: the capability of optimizing the analyser's experience will also bring important benefits for the LHC communities, in terms of total resource needs, user satisfaction and in the reduction of end time to publication. At the Italian National Institute for Nuclear Physics (INFN) a portable software stack for analysis has been proposed, based on cloud-native tools and capable of providing users with a fully integrated analysis environment for the CMS experiment. The main characterizing traits of the solution consist in the user-driven design and the portability to any cloud resource provider. All this is made possible via an evolution towards a "python-based" framework, that enables the usage of a set of open-source technologies largely adopted in both cloud-native and data-science environments. In addition, a "single sign on"-like experience is available thanks to the standards-based integration of INDIGO-IAM with all the tools. The integration of compute resources is done through the customization of a JupyterHUB solution, able to spawn identity-aware user instances ready to access data with no further setup actions. The integration with GPU resources is also available, designed to sustain more and more widespread ML based workflow. Seamless connections between the user UI and batch/big data processing framework (Spark, HTCondor) are possible. Eventually, the experiment data access latency is reduced thanks to the integrated deployment of a scalable set of caches, as developed in the context of ESCAPE project, and as such compatible with the future scenarios where a data-lake will be available for the research community.
The outcome of the evaluation of such a solution in action is presented, showing how a real CMS analysis workflow can make use of the infrastructure to achieve its results.

1 Introduction

As the technologies and the challenges evolve, a new approach is needed when providing HL-LHC communities with all the tools needed to get their analysis work done. In fact, not only the hardware requirement for both CPUs and storage will increase significantly, but also the capability of seamlessly porting the computational environment over different architecture and/or ephemeral infrastructure will become a key factor. The end-user data analysis workflow is evolving under many aspects, with a new event data format called NanoAOD [1] designed by the CMS Collaboration [2] in order to satisfy the needs of a large fraction of physics analyses, with a per-event size of order of 1 kB (more than a factor of 40 smaller than the MINIAOD format [3]); still, it contains all the top-level information typically used in the last steps of the analysis. NanoAODs also enable the use of a data-science-like approach, with frameworks that have almost no need of any experiment specific software, except for some specific configuration as, for instance, the calibration parameters. All this led to the possibility to bring important changes in the analysis paradigm which, in short, means that one can start shifting from a Grid based model where the barriers to entry are higher and the latency has a significant impact, toward an interactive or quasi-interactive way of data exploration that can provide a shorter time-to-insight, an important feature for an analysis workflow. In terms of computing infrastructure this evolution brings the opportunity for R&D around new solutions that, on the one hand, offer the possibility to exploit models based e.g., python based WebUIs and, on the other hand, allow to optimize the throughput, a key optimization aspect for the analysis at CMS. This translates into a model which foresees the usage of a well-equipped node with specialized hardware such as NVMe, many CPU and GPU cores enabling analysis activities at the MHz level and beyond. Several initiatives are arising in this context such as those at CERN [4] and in US [5], and an effort is being made at the Italian National Institute for Nuclear Physics (INFN) in leveraging modern cloud-native paradigms to serve as building blocks for the analysis infrastructure, with the main objective of deploying a platform to be challenged and optimized in preparation of the HL-LHC era, a solution fully compatible with resources provisioning model and service portfolio composition strategy of the INFN-Cloud. A first end-to-end prototype has been fully integrated with the analysis environment of the CMS experiment. A real-life workflow has been used to prove all the functionalities, specifically it is the ongoing studies on "*ssWW VBS with hadronic tau, mu/electron and two jets in final state"* which has been selected as it uses pre-filtered NanoAOD datasets where a cut-based analysis is applied, followed by a Machine Learning study that is performed on signal and the main sources of background in order to produce a more efficient signal vs background discriminator.

In Sec 2.1 the infrastructure is described, with focus on the main characterizing traits as a "single sign on"-like experience regarding the AAI (Sec 2.2), a compute cluster capable of scaling up and down depending on the work queue pressure (Sec 3.) and fully integrated with a caching system to drastically reduce the impact of remote read latency (Sec.2.3), evolution of the DODAS [6] model. The model of deployment is also introduced in Sec. 4 followed by the results of the first use case exploration (Sec. 5) and the final consideration on the improvements brought by the infrastructure.

2 The infrastructure

The primary motivations of the proposed architecture are to satisfy the shift toward interactivity as opposed to the GRID batch approach, as well as to maximize the throughput while analysing the experiment data. Although this can be done by using a single node with specialized hardware, it is also important to exploit scale out capability as well and, possibly, embedding everything in the very same deployment. In this paper, we present the current solution which foresees the support to a scale out over local resources. Moreover since the early design phase, one of the objectives of the proposed solution has been the portability, aiming at abstracting the infrastructure away from underlying heterogeneous resources, being them on multiple architectures, different providers, or different orchestration technologies. A third aspect taken into account has been the capability to reproduce, on demand, a setup with a minimal effort, something that becomes particularly important in order to reduce the costs for the user with no system administration skills to provision resources (e.g., on cloud resources, HPC centres etc). To cope with all these aspects we adopted a full containerization model, where all the components of the architecture are deployed as Docker containers. This approach does not preclude the use of Singularity at the application level.

2.1 Architecture overview

The core of the proposed solution is a JupyterHUB [7] instance capable of providing the user with all the knobs to set up its preferred workspace, including the usage of its own Docker image to work with. This is particularly important in order to allow users to customize their Jupyter notebook software with their preferred Python tools. Each user will then be redirected to its own running instance of Jupyter Notebook where its credentials have been previously forwarded to be used for further authentication with downstream services. From there onward is it possible to play with interactive approach and whenever ready to scale up leveraging batch system framework like HTCondor [8] or Spark [9]. Also in case of the distributed software the users get a certain freedom in terms of customization, meaning that starting from a set of provided base images they can add any piece of software they need and then distribute it through the cluster. The scaling out capability has been naturally delegated to Kubernetes [10] which has been selected as the orchestrator, providing a battle-tested environment that we plan to use to provide interactive interfaces and to scale out over a local cluster. The software stack is shipped in the form of a set of HELM [11] charts and, in addition, all the charts involved are collected using the Helmfile tool [12] that describes the interdependencies between different blocks and allows to disable some components when not needed. The main traits of the proposed solution are summarized in Fig.1, where it is visible that the only source of AAI is concentrated on an external instance of INDIGO IAM [13] leveraging the OpenID Connect standards and compliant with the WLCG plan of adoption. All authentication and authorization processes have been configured to follow this standard. The cluster is also provided with an ingress controller to redirect the traffic from the public endpoint to the correct service running inside the cluster, the present work adopts the NGINX [14] solution. In order to expose services through HTTPs, the Cert-Manager [15] technology is used to retrieve and manage the certificate automatically. A shared file system is provided by Longhorn [16], a simple and effective Kubernetes-native software to share and store files in a persistent manner across the cluster,

while an integration with external S3 storage based on Min.io [17] to guarantee a long term storage area can also be provided, if required.

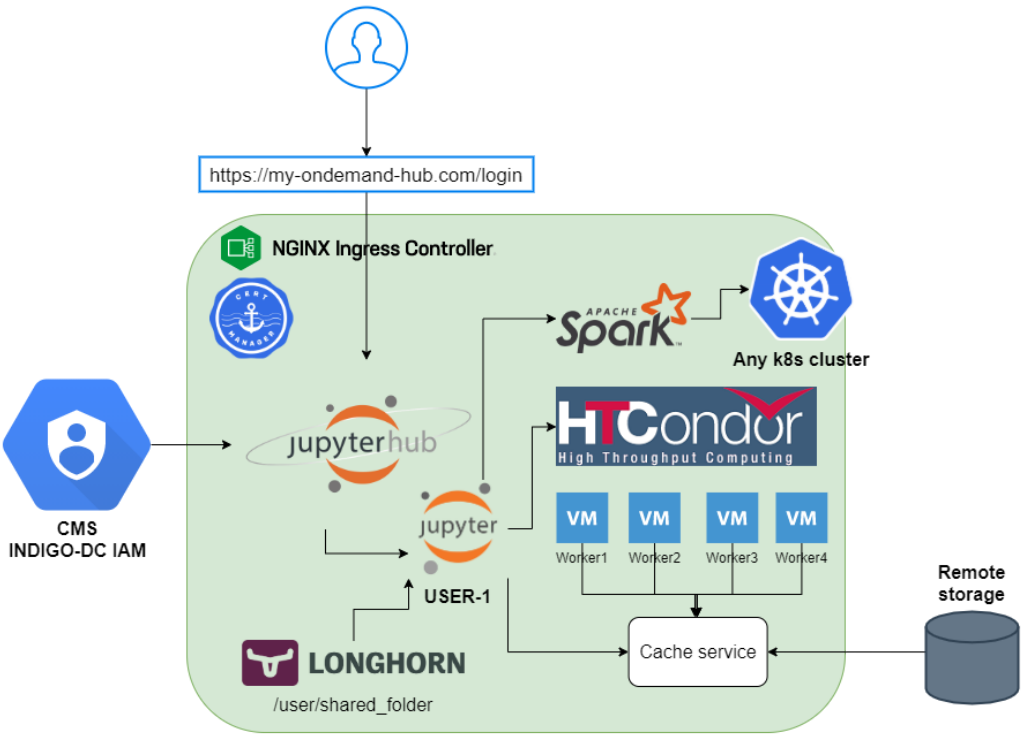

Fig. 1. Schema of component interactions in a generic analysis infrastructure at INFN

For the CMS use case, the deployment configuration has been customized as shown in Fig.2. The experiment software is shared through a repository hosted on CVMFS [18], which needs to be accessible from Jupyter notebooks and HTCondor worker nodes. A cache server configured to interact with the CMS remote storage federation (AAA [19]) is also present.

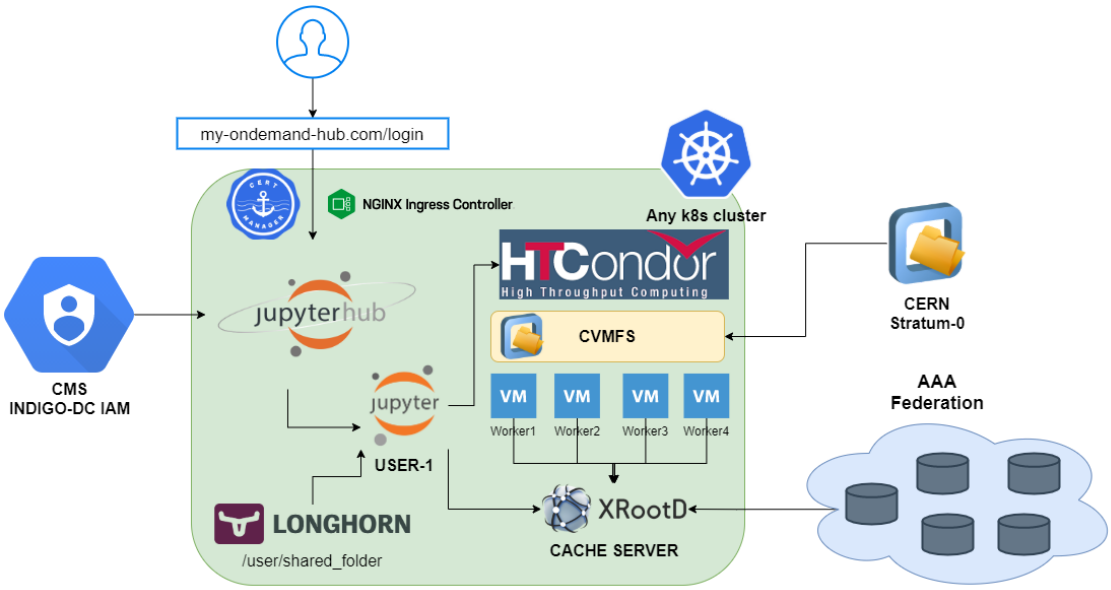

Fig. 2. Schema of component interactions with CMS customization if the analysis infrastructure at INFN

2.2 Identity and access management

In order to provide analysts with a "single sign on"-like experience, the JupyterHUB instance is configured to authenticate the users through an instance of the OpenID Connect provider, as implemented by INDIGO IAM; in the current testbed we rely on the dedicated service deployed for the CMS experiment at CERN. Upon successful authentication, hub checks the ID token and performs the authorization step based on the group claims in order to spawn a Jupyter notebook server - via a standard Kubespawner implementation [20]. JupyterHUB is also configured to pass, transparently to the users, all the IAM-related information to the spawning process via environment variables, for all further interactions with additional services, such as the cache and batch systems. The token management tasks are performed using an oidc-agent [21] daemon configured at the start-up.

2.3 Caching

A key objective of the system is the capability to grant access to the experiment's data, with an as high as possible throughput. The access to data can be performed from outer storages via remote read; still, in order to reduce the latency impact on the analysis performance, we used a local cache system. Starting from the experience achieved on data-lake prototyping at INFN, within the ESCAPE project [22], and extending the work previously done in XDC project [23], a cache solution has been integrated. The caching system must be seen as an intermediate layer between the client requesting data and the remote storage federation of the CMS experiment. The INFN strategy is to adopt and deploy a multi-level caching system with a central national cache e.g., deployed at Tier-1, and another level meant to be closer to the analysis facilities; the

cache should be implemented using fast disks (SSD, NVMe). From the authentication perspective, the cache can be configured as a translation layer between the legacy X509-based authentication and authorization model, and the CMS IAM framework. The cache server hosts a service X509 certificate authorized to read experiment data from AAA using XRootD [24] protocol, while the clients contacting the cache are authenticated providing a bearer json web token to an HTTPs endpoint, thus leveraging the CMS IAM identity. The described system is aiming at the realization of a X509-free setup.
In Fig. 3 the detailed implementation of the cache server is shown with particular focus on the authorization pattern; in this implementation the translation of the CMS IAM based identity into a legacy X509 flow is completely hidden to the user by the cache server, providing an environment where he will no longer need to manage his own certificates.

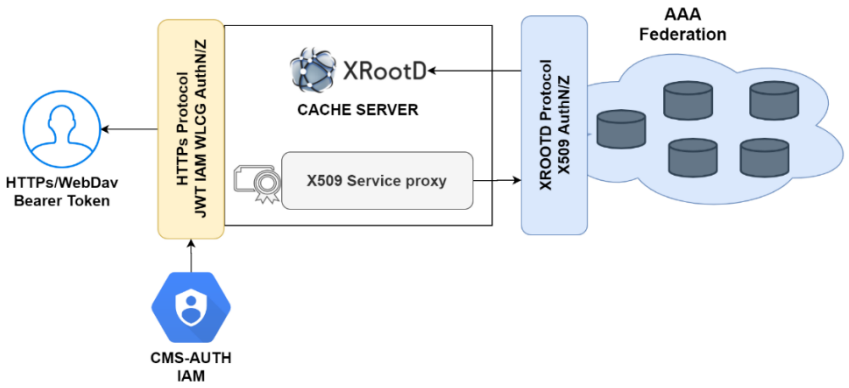

Fig. 3. Schema of a typical data access from the analysis infrastructure cluster point of view. The protocols used and the authentication and authorization flow are shown.

3 On-demand computation and scale out

One of the design pillars of the system is a scaling capability in order to satisfy the needs of the analyses, possibly exploiting hybrid providers (HPC, clouds, ...). At the time of writing both HTCondor and Spark have been integrated as K8s applications; ongoing work is foreseen towards further native Kubernetes integration for batch payload distribution (e.g., exploring kube-batch [25] functionalities) as well as the exploitation of software alternatives as the distributed ROOT framework RDataFrame [26] developed at CERN and columnar analysis tools as coffea-casa [5]. The setup includes two pods for the cluster management: the first runs the Collector and Negotiator, while the second executes a Schedd instance. Daemons authenticate through a shared secret stored as a Kubernetes secret resource. The Schedd relies on Longhorn volume to store the spool directory content persistently around the cluster: in case of pod restart, the spool content will be recovered without any loss. Only users that show a valid CMS IAM bearer token are authorized to submit a job to the queue, a feature available in the version 8.9.9 that is currently installed. In addition, at submission time, the users can also ask HTCondor to manage their IAM identity on their behalf, in order to make WNs able to get an access token to retrieve data through the cache system. This makes the whole delegation and

refresh of the credentials completely transparent for the end user upon successful authentication with JupyterHUB as explained in Sec 2.2.

3.1 Autoscaling on custom metrics

With the aim to optimize the scale out based on workflow needs, we decided to work on implementing the support for autoscaling on custom metrics. The cluster configuration comes with an integrated HTCondor Worker Node Pod autoscaling that is based on custom metrics collected via Prometheus [27]. The solution leverages Kubernetes native Horizontal Pod autoscaling (HPA) [28] which automatically scales the number of Pods in a replication controller, deployment, replica set or stateful set based on some observed metrics. It consists essentially in setting up a Kubernetes API resource and a controller: the latter is determined by the former, periodically adjusts the number of replicas in order to decrease the metrics value, which is required to stay below a user-defined threshold value. Currently, just resource usage metrics are automatically exported by Kubernetes, thus making it necessary to use third-party apps to export and expose useful HTCondor custom metrics. Key elements for scaling pods based on custom metrics are a Prometheus exporter (i.e., a web server that exports and makes metrics from the specific application available to the Prometheus server), and an adapter that acts as a link between the monitoring server and Kubernetes exposing metrics through a so-called Custom Metrics API. In Fig. 4 you can see in a schematic form how the Prometheus server, the adapter, and the HTCondor-specific exporter are implemented.

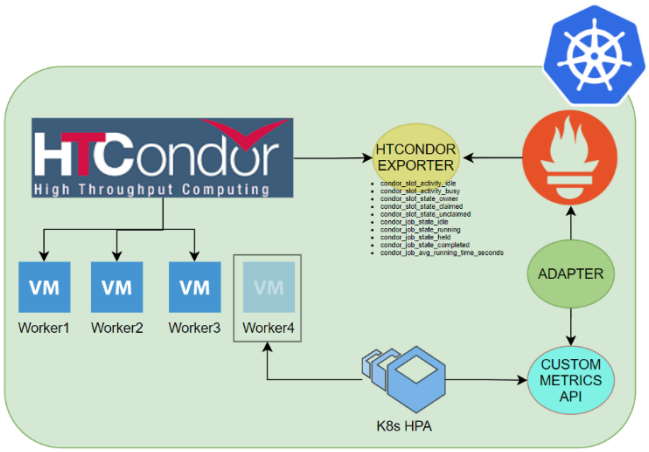

Fig. 4. A schema with the detailed implementation of the autoscaling workflow in the presented infrastructure

More in detail an exporter that interacts with the cluster via HTCondor Python bindings has been used. In the adopted implementation, condor_slot_activity_busy (indicating whether a slot is busy or not) metric values are averaged across all the WNs available in HTCondor pool, and in turn this value, hereafter referred as the *avg(condor_slot_activity_busy)*, is exposed by the adapter and made available to Horizontal Pod Autoscaler. It identifies the ratio between the

number of busy machines and the total number of machines, meaning that in first approximation it can be used to evaluate the status of pressure of the job queue. The Horizontal Pod Autoscaler resource is then created targeting WNs deployment, meaning that when the metrics value goes above a certain threshold value, the deployment is scaled up until the metrics value is taken back to below-threshold values. Then, after a cool-down period, HPA starts to gradually downscale the deployment.

4 Portability of the system and deployment strategy

From the deployment perspective, the main objective is to provide a highly portable system. Our vision in this respect is to support three alternative ways: it is possible to follow recipes for a docker-compose instantiation of the software stack for a single machine, while in the Kubernetes setup all the application layer configurations and their dependencies are described using HELM templates. For the sake of the genericity the deployment must be compatible both with a scenario with a pre-existent cluster or, in alternative, with an end-to-end automation that includes the cluster provisioning. To cope with the latter the plan is to provide a full integration of Helm with TOSCA [29]. The applications are structured in such a way that, through the very same base template structure, different flavors of the same cluster can be deployed. For instance one can activate a certain type of shared filesystem to be used by putting a flag at Helm configuration level (so called "Helm values"). In addition, multiple applications can be combined as needed with the Helmfile tool, where the child application will wait for the parent to be completely deployed before starting its own installation.
The Helm charts integration in the TOSCA template has been possible thanks to the usage of Ansible [30] roles which take care of compiling Helm values only when the cluster has been automatically created and thus all the parameterized information is known. This is how we plan to integrate the system in the services portfolio of the INFN-Cloud.

5 First user experiences

The development model follows a user feedback driven approach and thus at the time of writing we are testing the early prototype within a real analysis at CMS: the ongoing "*ssWW VBS with hadronic tau, mu/electron and two jets in final state*" analysis working group that run the ML steps of the analysis. The goal of the analysis is to measure the cross section of the considered signal and to evaluate possible BSM effects in an EFT framework [31], and it has a two-stage workflow: a first step is represented by the pre-processing of original Monte Carlo (MC) samples (stored in the NanoAOD format) that is done via CRAB [32]. The resulting samples are then processed via HTCondor jobs and eventually used to produce control and result plots and to develop a ML-based signal-vs-background discriminator. Therefore, the analysis infrastructure prototype easily meets the computing requirements of the second part of the analysis. The setup includes PyROOT [33] and NanoAOD-tools [34] libraries for cut-based analysis workflow on HTCondor Worker Nodes; Uproot [35], scikit-learn [36], Tensorflow [37] libraries for Machine Learning (ML) studies via Jupyter notebook and available cached data for the analysis.

As a first use case exploration, the signal MC sample (200GB) and a W + Jets background MC sample (1.2TB) were processed via HTCondor jobs, producing a set of plain ROOT ntuples. Multiple job submissions triggered the auto scaling process, starting from a single worker node, several additional wn-pods were deployed in order to keep the *avg(condor_slot_activity_busy)* metric below the 0.75 threshold that has been set. Also the data access was transparent for the user in fact no further authentication was needed thanks to the IAM token integration, and the cache system was able to serve all the clients with no errors, and, at the same time, was caching on disk the required data. On Fig. 5 it is also clear how, thanks to the cache capabilities, repeated requests of a single file block were served directly from the cache without the need for a new remote call. In fact a peak above 1 Gbps has been reached for the transmitted data while only around half of that amount for the incoming traffic.

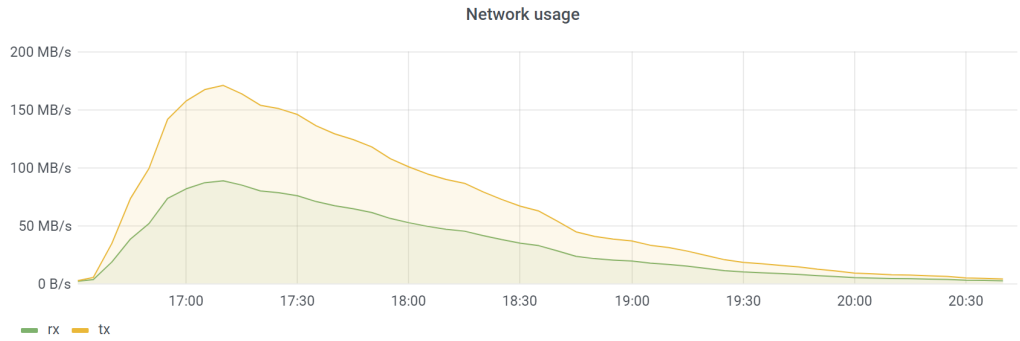

Fig. 5. Network activity as seen by the cache server, in green the data coming in and in yellow the outbound traffic. Please notice that the two distributions are shown overlapped and not stacked.

A preliminary ML study was then performed via Jupyter notebook: original ROOT files (200GB after the previous step) were translated to Pandas dataframes using Uproot library and then saved as compressed CSV files. Subsequently, a 900-estimator AdaBoost Classifier [38] was trained on a 8312 -event balanced dataset obtained from those CSV files and the training interactive notebook requested to run on a VM on the cluster labeled as having a GPU and having at least 64GB of RAM. More importantly, in order to achieve this, the end user had only to select the proper GPU and RAM flag from the JupyterHUB login page and to prepare its own notebook.

5.1 Current experiences and lessons learnt

These first tests shed light on the key elements that make the usage of such Analysis Facility preferable to grid-based computation. In fact, not only I/O time is eventually reduced by the usage of cached data, which could be critical when input data sizes become bigger, but also the user, when using his own Jupyter instance, has access to an all-in-one solution: submission to HTCondor and interactive python-based programming. The adoption of a X509-free model, also significantly reduces the configuration effort for the end user in order to access the experiment data. Besides, thanks to its template and containerized nature, it is very easy to adapt Analysis Facility features to user needs, which make this prototype a real general-purpose solution, which could expand well beyond the CMS use case. Finally, its elastic and auto scaling

features, although in their preliminary basic form, ensure an efficient resource usage, enhancing cost reduction.

6 Conclusion and plans

The GRID based approach to the distributed analysis at CMS has been extremely successful and allowed us to cope with the scientific programs of the collaboration during Run 1 and Run 2. It will keep playing a key role even during the upcoming Run 3, although several R&D initiatives to exploit new models are arising within CMS. We have presented our vision toward an evolution of the analysis infrastructure in view of HL-LHC discussing the current prototype at INFN, under test with a real analysis. A key to the success is now to evolve the system in order to establish feasibility for HL-LHC workflows at scale and to this end the plan is to provide an instance of the facility for the physicists at INFN. From the implementation point of view, future investment will be done toward the transparent exploitation of heterogeneous hardware and hybrid providers (e.g., Clouds and HPC) becoming important considering the expected increase in ML load in the future analysis workflows. In the short term we plan to push further the integration with INFN-Cloud resources starting a comprehensive test campaign at national level.

This work has been partially supported both by the EOSC-hub EU project G.A 777536, and by ESCAPE EU project, G.A. 824064.

References

[1] A. Rizzi, G. Petrucciani, M. Peruzzi, *A further reduction in CMS event data for analysis: the NANOAOD format*, EPJ Web Conf., 214 (2019), doi: 10.1051/epjconf/201921406021

[2] S. Chatrchyan et al. "*The CMS Experiment at the CERN LHC*". In: JINST 3 (2008), S08004. DOI: 10.1088/1748-0221/3/08/S08004

[3] CMS Offline Software and Computing, *Evolution of the CMS Computing Model towards Phase-2,* CMS-NOTE-2021-001, https://cds.cern.ch/record/2751565

[4] E. Bocchi, L. Canali, D. Castro, P. Kothuri, H. G. Labrador, ScienceBox Converging to Kubernetes containers in production for on-premise and hybrid clouds for CERNBox, SWAN, and EOS, EPJ Web Conf. 245 (2020) 07047, doi: 10.1051/epjconf/202024507047

[5] Shadura, Oksana. (2020, July). *A prototype U.S. CMS analysis facility.* Presented at the PyHEP 2020 Workshop, Zenodo. http://doi.org/10.5281/zenodo.4136273

[6] D. Spiga, and others, *Exploiting private and commercial clouds to generate on-demand CMS computing facilities with DODAS,* EPJ Web Conf. 214 (2019), doi: 10.1051/epjconf/201921407027

[7] https://jupyter.org/hub

[8] https://research.cs.wisc.edu/htcondor/

[9] https://spark.apache.org/

[10] https://kubernetes.io/

[11] https://helm.sh/

[12] https://github.com/roboll/helmfile

[13] https://indigo-iam.github.io/docs/v/current/about.html

[14] https://kubernetes.github.io/ingress-nginx/
[15] https://cert-manager.io/
[16] https://longhorn.io/
[17] https://min.io
[18] Buncic, P. and Aguado Sanchez, C. and Blomer, J. and Franco, L. and Harutyunian, A. and Mato, P. and Yao, Y., *CernVM: A virtual software appliance for LHC applications,* J. Phys. Conf. Ser., 219 (2010), doi: 10.1088/1742-6596/219/4/042003
[19] K. Bloom and others, *Any Data, Any Time, Anywhere: Global Data Access for Science,* arXiv physics.comp-ph , 1508.01443,8 (2015)
[20] https://jupyterhub-kubespawner.readthedocs.io/en/latest/index.html
[21] https://indigo-dc.gitbook.io/oidc-agent/

[22] https://projectescape.eu/
[23] D. Ciangottini, G. Bagliesi, M. Biasotto, T. Boccali, D. Cesini, G. Donvito, A. Falabella, E. Mazzoni, D. Spiga, M. Tracolli, *Integration of the Italian cache federation within the CMS computing model,* PoS ISGC2019, p. 014 (2019), doi: 10.22323/1.351.0014
[24] https://xrootd.slac.stanford.edu
[25] https://github.com/kubernetes-sigs/kube-batch
[26] V. E. Padulano, J. C. Villanueva, E. Guiraud, and E. T. Saavedra, *Distributed data analysis with ROOT RDataFrame,* EPJ Web Conf. Volume 245, 2020, doi: https://doi.org/10.1051/epjconf/202024503009
[27] https://prometheus.io/
[28] https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/
[29] https://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.0/csprd01/TOSCA-Simple-Profile-YAML-v1.0-csprd01.html#_Toc430015628
[30] https://www.ansible.com/
[31] Green, D. R., Meade, P., & Pleier, M. A. (2017). *Multiboson interactions at the LHC*. Reviews of Modern Physics, 89(3), 035008
[32] Spiga, D., Lacaprara, S., Bacchi, W., Cinquilli, M., Codispoti, G., Corvo, M., ... & Kavka, C. (2007, December). *The CMS remote analysis builder (CRAB)*. In International Conference on High-Performance Computing (pp. 580-586). Springer, Berlin, Heidelberg
[33] https://root.cern/manual/python/
[34] https://github.com/cms-nanoAOD/nanoAOD-tools
[35] https://uproot.readthedocs.io/
[36] https://scikit-learn.org/stable/
[37] https://www.tensorflow.org/
[38] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html