# Computational Genomics platform: a Cloud-enabled approach

**Jacopo Gasparetto,[a,*] Letizia Magenta,[a] Francesco Sinisi,[a] Stefano Zotti,[a] Alessandro Costantini,[a] Barbara Martelli,[a] Tania Giangregorio[b] and Tommaso Pippucci[b]**

[a]*INFN CNAF,*
*viale Carlo Berti Pichat 6/2, Bologna, Italy*
[b]*IRCCS Azienda Ospedaliero-Universitaria di Bologna,*
*Via Giuseppe Massarenti 9, Bologna, Italy*
*E-mail:* jacopo.gasparetto@cnaf.infn.it, letizia.magenta@cnaf.infn.it, francesco.sinisi@cnaf.infn.it, stefano.zotti@cnaf.infn.it, alessandro.costantini@cnaf.infn.it, barbara.martelli@cnaf.infn.it, tania.giangregorio@aosp.bo.it, tommaso.pippucci@aosp.bo.it

Modern technologies for DNA and RNA sequencing allow for fast, parallel reading of multiple DNA fragments. While sequencing the first genome took 32 years, today with Next Generation Sequencing technologies we are able to sequence 40 genomes in about 2 days, producing 4 TB of text data (a file of about 100 GB per genome). This ability poses a challenge to computing infrastructures, which need to be able to ingest this amount of data and to process it through efficient genomics pipelines, exploiting heterogeneous resources such CPUs, GPUs, HPC clusters and storage exposing different Quality of Service (QoS) to perform the analysis with the optimal cost-performance balance.

This is the case of the Computational Genomic platform under development in the context of the collaboration between INFN and IRCCS AOU Sant'Orsola (the main research hospital in Bologna, Italy). The platform has been deployed on EPIC (Enhanced PrIvacy and Compliance) Cloud, the high security partition of INFN Cloud certified ISO 27001 27017 27018, where some sample genomic pipelines have been implemented and the needed security measures have been adopted to guarantee GDPR compliance.

*International Symposium on Grids and Clouds (ISGC2024)*
*24 - 29 March, 2024*
*Academia Sinica Computing Centre (ASGC), Institute of Physics, Academia Sinica Taipei, Taiwan*

*Speaker

## 1. Introduction

Recent advancements in computational genomics have revolutionized our understanding of genetics and its applications. From the refinement of genome sequencing technologies to the development of sophisticated algorithms for analyzing vast amounts of genomic data, the field has witnessed remarkable progress. Moreover, the integration of computational methods with experimental techniques has facilitated the exploration of complex biological systems at an unprecedented scale and resolution. These advancements hold great promise for uncovering the intricacies of the genome and translating genomic knowledge into improved healthcare outcomes.

However, these breakthroughs are posing significant challenges for computing centers, as the size of genomic data continues to grow exponentially, leading to technical difficulties in storage, processing, and analysis. For instance, the sheer volume of sequencing data generated from large-scale projects like the Human Genome Project and the ENCODE project strains computational resources, requiring advanced infrastructure and efficient algorithms to handle the data effectively. Additionally, the integration of multi-omics data, such as genomics, transcriptomics, proteomics, and metabolomics, further exacerbates the computational demands, necessitating innovative approaches for data integration and analysis.

In this context INFN, the Italian Institute of Nuclear Physiscs and the IRCCS Azienda Ospedaliero-Universitaria di Bologna, Policlinico di Sant'Orsola joined their forces in a collaboration to design and implement a Computational Genomics platform able to host and operate genomics pipelines to provide a set of cloud-enabled services available to the research communities from one side, and make a smart and efficient use of the resources from the other.

The paper is organized as follow. In Section 2 the current computing architecture of IRCCS AOU for genomics application is described, together with the inner limitations. In Section 3 the new approach consisting in the design and implementation of the Computational Genomics Cloud Platform is presented and in Section 4 both performances and further steps are provided. In Section 5 the conclusions are drafted.

## 2. State of the Art

The current architecture of IRCCS AOU consists of a modular hyperconvergent solution that exploits compute and storage resources into a unified system. It relies on an external storage area network (SAN) system for long-term storage and integrates infrastructural and backup services, compute node for genomics analysis, solutions for access and resource management, bioinformatics middleware and automation and versioning systems for bioinformatics software. For package and environment management IRCCS AOU adopted Anaconda/Conda [1], a software designed to manage the configuration of isolated execution environments and the quick installation of open source software. Multi-step reproducible analyses are executed using Snakemake [2] a workflow manager designed specifically for biomedical data. Snakemake optimizes resource usage and step parallelization, manages software installation and supports incremental build and re-entrancy. Resources within the computing infrastructure are allocated using Slurm [3], a job scheduling manager that launches jobs or queues job execution.

Despite properly serving the duty that is asked for, the current architecture suffers of some limitations, that are especially highlighted when the size of the data increases, as well as the number of scientists that will work on it. The usage of traditional virtual machines requires a meticulous configuration process to scale the computing cluster and a massive effort to manage all the different running services, starting from the scientific computations to some user-oriented service such as web dashboards and applications for data visualization and users management. From a system administrator perspective, it became difficult to manage the entire environment. Resources and services, in fact, are not self-constrained nor isolated with the result that different project leaders have access to all the resources. This approach results in important security issues that have to be addressed in case of data breach or other kind of security related violations.

Another limitation consists on the user management: carried out by plain Linux, the users home directories are shared over the network. Each user directory is created and configured manually to properly work with the infrastructure, including the permissions. Every time a user is created or removed from the system, a series manual actions have to be undertaken, together with the related inner errors.

In respect to software layer, the different packages and services are often installed and configured manually with the result that many different versions of similar conda environments are available in the cluster at the same time and shared among the different nodes. This approach makes extremely difficult to catalog the versions of the software currently available in the system. This aspect is also critical in terms of security vulnerabilities present in the software running in the pipelines.

For the above mentioned reasons, work have been carried out to provide a central, private, software registry to store, manage and share common and reliable software to be used by the community. Moreover, vulnerabilities can be detected and fixed easier, working on the piece of problematic software that, once patched, can be made available to the community.

## 3. Computational Genomics Cloud Platform

To address the limitations described in Section 2, a cloud-native approach has been defined and implemented providing greater flexibility in deploying workflows, enhancing resilience to computing cluster failures, and significantly boosting scalability across the architecture.

This section describes the solutions adopted to design and implement a Computational Genomics platform able to host and operate genomics pipelines via a set of cloud-enabled services made available to the research communities.

### 3.1 Cloud enabled approach

Cloud computing has become a cornerstone of modern IT infrastructure. In particular, Cloud infrastructure comes with a set of features that can harmonize and optimize the use of virtualized resources, providing the needed services to be adapted to different kind of computation. Some of this feature are listed hereafter:

- Scalability: organizations can scale computing resources up or down on demand.

- Flexibility and Agility: users can access resources from anywhere with an internet connection. This flexibility enables remote work, collaboration, and access to applications and data on multiple devices.
- Resilience and Reliability: a robust infrastructure with built-in redundancy and fail-over capabilities ensures high availability and reliability of services. This resilience minimizes downtime and data loss, enhancing business continuity and disaster recovery.
- Elasticity[1]: organizations can dynamically adjust resources to meet changing demands.

For such reasons, the infrastructure used to host the Computational Genomics Platform is based on OpenStack [4], the IaaS Cloud solution adopted within INFN and made available to support the development of project activities as well as production environments.

The need to adopt a standardized and widely accepted method for packaging the software to be distributed across the cloud became mandatory and the possibility to have a better control of the distributed software employed in genomics workflows, thereby enhancing scrutiny of security issues, is given by the use and adoption of container solutions.

For this purpose, we've opted to containerize the software using open container formats [5], utilizing standard Docker [6] manifest files. This approach provides finer control over software compilation and its dependencies compared to Conda Environments. Packages from Anaconda/Bioconda are less transparent in their construction and are more challenging to analyze with security scan tools. Additionally, container usage enables the creation of lightweight, high-performance, tailor-made images containing only essential software and dependencies.

The definitions (i.e. Dockerfiles) for the container images used in genomics workflows are stored in a self-hosted instance of GitLab [7] on EPIC Cloud. Continuous Integration/Continuous Delivery (CI/CD) pipelines automatically build the images upon any changes and then push them to the private container registry provided by GitLab itself. The images are thus accessible by any standard container runtime, such as Docker or Containerd [8].

### 3.2 Container orchestration engine

Running on top of Openstack IaaS, Kubernetes [9] has been chosen as the service orchestrator software due to its predominant role as the leading cloud solution for container-based micro-services provisioning. Among its key features, it offers high scalability, enabling effortless horizontal expansion of cluster size in terms of worker nodes. The combination of OpenStack [4] and Kubernetes allows for rapid implementation of vertical scaling; a resizing of a VM on the infrastructure side is sufficient to allow Kubernetes to have a greater number of computational resources available. Kubernetes also provides high resilience by automatically relocating services from degraded nodes to healthy ones. Moreover, it facilitates complex and flexible storage management through the usage of Persistent Volume Claims, enabling data sharing between each workflow step and ensuring long-term storage retention.

[1]Scalability and elasticity are both important concepts in cloud computing, but they address slightly different aspects of resource management: the first is about the ability to handle growth by adding resources, while the second specifically focuses on the automatic provisioning and de-provisioning of resources to match fluctuating demand, ensuring optimal performance and cost efficiency in cloud environments.

The Kubernetes cluster has been built using the Rancher Kubernetes Engine 2 (RKE2) [10] distribution, chosen specifically for its emphasis on security features. RKE2 is a lightweight Kubernetes distribution designed to streamline the deployment and management of Kubernetes clusters, particularly in production environments. Among its strengths we find:

- Simplicity and Automated Operations: RKE2 simplifies the installation and operation of Kubernetes clusters. It offers a straightforward setup process, making it accessible to users with varying levels of Kubernetes expertise, and automates various aspects of cluster operations, including node deployment, scaling, and upgrades.
- High Availability: RKE2 supports high availability configurations out of the box, allowing clusters to withstand node failures and maintain service availability. It employs features like ETCD clustering and distributed control planes to achieve resilience against failures.
- Scalability: RKE2 scales seamlessly to accommodate growing workloads and user demands. It supports horizontal scaling by adding or removing worker nodes dynamically, enabling clusters to adapt to fluctuating resource requirements.
- Extensibility: RKE2 is highly extensible and integrates seamlessly with other Rancher products and third-party tools. This enables users to extend the functionality of their Kubernetes clusters and integrate them with existing workflows and systems.
- Security: Security holds utmost importance in contemporary IT landscapes, particularly concerning containerized applications. RKE2 benefits from a vibrant community dedicated to promptly addressing vulnerabilities. They employ tools like Trivy [11] for regular scanning of cluster images to detect CVEs. One notable advantage of RKE2 is its seamless facilitation of cluster creation aligned with Center for Internet Security (CIS) benchmarks. By default, the cluster incorporates several benchmark requirements, while others are automatically activated via RKE2 configuration settings. However, some benchmarks necessitate manual activation by the administrator.

### 3.3 Workflow engine solutions

A workflow consists in a set of computing tasks represented as nodes of a complex direct acyclic graph (DAG), which describes how the data flows between the graph nodes. Independent tasks will run in parallel, while dependent tasks will wait until the previous tasks complete, forwarding their output data as input. Figure 1 shows an example of DAG produced by Nextflow for a typical genomic pipeline designed to detect germline variants from whole genome sequencing data. Although genomics workflows are each tailored to a particular application, overall they follow similar patterns, typically comprising two analysis phases: alignment and calling. The first phase involves pre-processing the raw sequence data provided in FASTQ format to produce analysis-ready BAM files. This involves alignment to a reference genome as well as some data cleanup operations to correct for technical biases and make the data suitable for analysis. The second phase proceeds from analysis-ready BAM files and produces variant calls. This involves identifying genomic variation in one or more individuals and applying filtering methods appropriate to the experimental design. The output is typically in VCF format. Each node of the graph in Figure 1 corresponds to

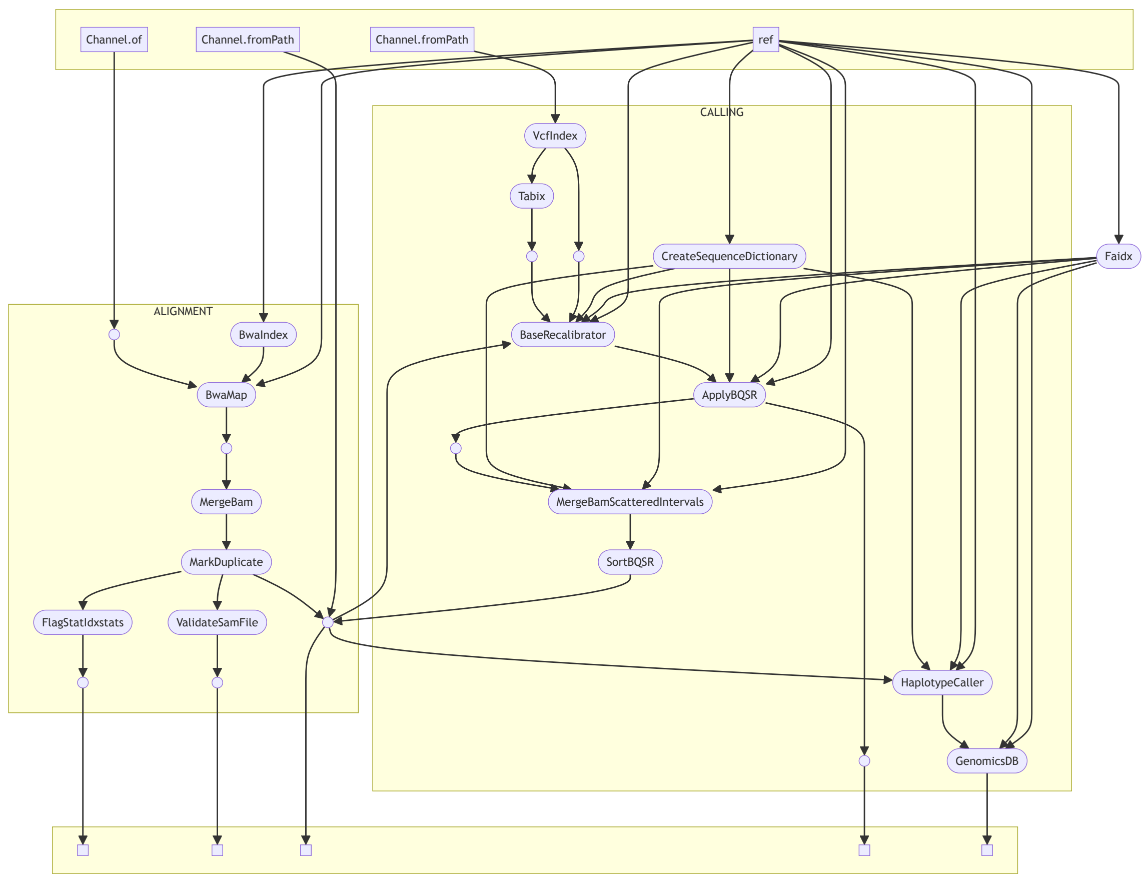


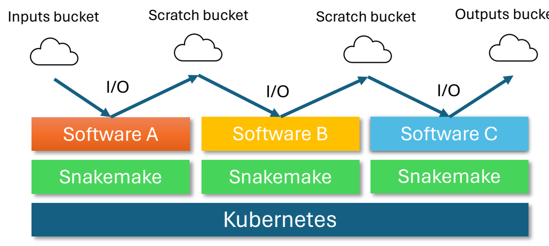
**Figure 1:** Example of Direct Acyclic Graph (DAG) produced by Nextflow for the Whole Genome Computational pipeline.

a computing task and therefore a Kubernetes pod. Each pod is associated with a single container based on the docker images containing the necessary software to perform that precise task. It should be noted that each computing task can be simultaneously invoked in multiple instances or "clones", depending on how many samples are processed.

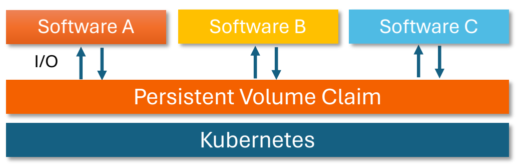
Since the nature of pods/containers is ephemeral and so is their storage, a data transfer mechanism to correctly share data between consecutive tasks is needed, as well as to collect the final results. Figure 2 shows how Snakemake and Nextflow work differently on Kubernetes, as described in the following sections.

### 3.3.1 Snakemake

Snakemake has been initially adopted as a workflow engine for genomics workflow. Despite its support to Kubernetes, was clear from the beginning that it was not the proper solution for the platform needs, especially in terms of data transfer and how it supports containers. At the time of writing, Snakemake does not provide support for Kubernetes Persistent Volume Claims (PVC). Consequently, data generated by a single unit task (i.e., pod) must be transferred over the network using one of the supported protocols. For our tests, a self-hosted instance of Ceph Object Storage



**(a)** Snakemake on K8s



**(b)** Nextflow on K8s

**Figure 2:** Different behaviours between Snakemake and Nextflow. (a) Snakemake asks K8s to spawn pods starting from the same Snakemake docker image. Specific images are downloaded and started at runtime. In addition, data must be downloaded/uploaded at each step of the computation. (b). Nextflow directly spawn the desired image making it *cache friendly* to k8s. In addition, contrary to Snakemake, data is not required to be uploaded/downloaded to an external storage provider. The *Snakemake* and *Software X* rectangles represent different containers.

[12] with the RADOS Gateway [13] S3 compatible REST API extension has been selected. Figure 2a shows the data transfer model of Snakemake: 1) a pod is invoked using the Snakemake Docker image, 2) a Snakemake instance runs inside the just born pod and it downloads the requested Docker image, which is run as a Singularity [14] container, 3) data is downloaded from the remote S3 storage, 4) the actual computation runs and, 5) the computation's outputs are uploaded to the S3 storage, becoming available to next computing tasks. The first test immediately reveled the non-feasibility of this approach. Since each unit task can deal with up to hundreds of gigabytes of text data, both as input and output, this results in extensive data transfers across a traditional LAN network. Consequently, a pod may spend 30 to 60 minutes solely downloading the required data before initiating the real computation. Subsequently, it may similarly wait to upload the generated data to the object storage, thereby making it accessible to the subsequent unit of work.

Moreover, Snakemake heavily orbits around Conda and has a limited support of containers and how they are managed, increasing the total overhead of the system. All pods spawned by Snakemake are clones of the exact same container image. This image must provides Snakemake runtime and by default by the official Snakemake Docker image is used. Subsequently, after the pod has fully spawned, the Snakemake instance running within the pod downloads the actual image containing the software necessary for the computational task. As consequence, Kubernetes is completely blind in respect of all the images (more than a few) needed by the workflow. Because Snakemake asks Kubernetes to span replicas of itself, Kubernetes will cache on nodes only the Snakemake image. Each spawned pod must then, at runtime, download the correct image, convert it to Singularity and spawn a container in a container which will run the computation. Considered that a single image can weight between hundreds of megabytes to gigabytes and that tens of pods are spawned per each workflow, it easy to image how heavy is the impact of this issue in terms of additional overhead. These limitations leaded to the decision of exploring alternatives that have a better integration with the cloud paradigm.

### 3.3.2 Nextflow

While Snakemake offers a straightforward and intuitive syntax based on Python, Nextflow relies on a less common Groovy-based syntax. However, Nextflow provides significantly more advanced features, particularly for cloud computing. Despite being a general purpose workflow manager, as Snakemake is, both are very popular among the bio-informatics community and almost equivalent.

Nextflow boasts robust support for native containers, eliminating the need for containers within containers. Each computational unit task can be executed as a native container directly on the host machine by injecting a simple shell script into the spawned container. When deployed on Kubernetes, Nextflow will instantiate pods with the required images, enabling Kubernetes to cache these images on the worker nodes, as shown in Figure 2b. Consequently, all pods will spawn almost instantly once the images are downloaded for the first time. This characteristic yields an additional benefit: the ability to seamlessly execute the same workflow on both a large distributed system for production purposes and on a small laptop for development and testing.

With its native support for Kubernetes Persistent Volume Claims, Nextflow eliminates the need to transfer large amounts of storage from and to S3 (or similar) storage providers before and after executing computing tasks. The Kubernetes cluster has been configured to include at least three Network File System (NFS) volumes: one for read-only input data, one as a scratch volume, and one for collecting output results. These volumes are mounted to each worker node. Once the NFS volumes are accessible to all worker nodes, Nextflow can be instructed to spawn pods that mount these volumes as PVCs. Consequently, input data is readily available to the pod upon startup, eliminating the need to download previous data or upload current computation results, as illustrated in Figure 2b.

Another important Nextflow's feature is the possibility to tag as single unit of work with custom node selector labels. This allows the run of certain units of work on some defined worker node, for example with GPUs or different types disks (HDD or SSD).

The first tests of this approach resulted to be very promising, making Nextflow a good candidate to manage workflows as containerized micro-services run on a Kubernetes cluster.

## 3.4 Monitoring

Monitoring is provided by the well-known Prometheus [15] - Grafana [16] software stack, deployed on Kubernetes as Helm [17] chart. These tools not only facilitate resource monitoring, but also provides profound insights into workflow performance and bottlenecks.

In Figure 3 is highlighted that a particular heavy job uses up to 40 CPU cores, while other processes that run in parallel use only few cores. Such information enables the fine tuning of the node selectors for different steps of the same pipeline. By adopting this approach it is possible to recognize with a glance which job is performing badly and optimize the entire workflow resources allocation. Thanks to the combination of Kubernetes' node selectors and the possibility to tag individual unit of work provided by Nextflow, it has been possible to properly design the cluster with one heavy duty worker node, and several smaller worker nodes for light jobs.

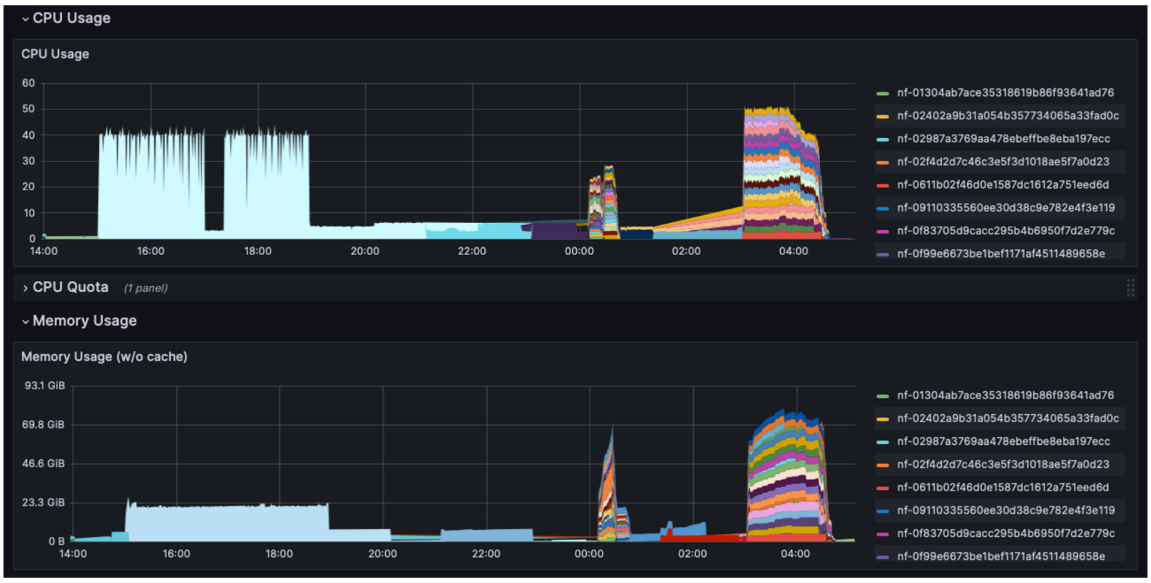


**Figure 3:** Resources used by the Whole Genomics Pipeline workflow, captured by Grafana. Each color/identifier represents a different pod, and thus, a different unit of work of the pipeline.

## 4. Performances and further steps

A single pipeline takes around 13 hours to run on a relatively small cluster composed by 72 vCPUs, 144 GB of RAM and 3 TB of storage. The performances are comparable with those measured on the local batch cluster.

Thanks to the combination of Kubernetes node selectors and the possibility to tag individual unit of work provided by Nextflow, we have been able to use the platform in an efficient way and properly tune and redirect the computation for the most suitable resource (see Figure 3 for more details).

Once the all the details of the future cloud-based architecture presented in this work are complete, it will be straightforward to enlarge the cluster size, firstly allocating more resources to the OpenStack tenant, and secondly exposing more nodes to the Kubernetes cluster.

Due to the nature of the computational genomics science, processing data coming form real clinical patients requires a strong and secure mechanism to authenticate and authorize users to handle such data. For this purpose, it is possible to configure Nextflow such that it instructs Kubernetes to run a job as a particular user id, with its associated group id. In this way, it is possible to protect files using regular Unix permissions. To centralize and simplify the authentication and authorization process, a single sign-on solution based on JSON Web Tokens (JWT) [18] and the OAuth protocol [19], using Keycloak [20] as Identity Provider (IdP), is currently under study. The goal is to have a single and simple mechanism for administrators to manage their users, allowing specific groups to work on the data associated with projects.

At the time of writing, users can interact with the proposed infrastructure only via Command Line Interface (CLI) tools, such as SSH and kubectl. Even tough launching the pipelines is already quite convenient (i.e, using kubectl paired with a bunch of pre-configured yaml files defining the pipelines to deploy), the final goal is to provide graphical user interfaces to improve the user experience in submitting even complex computational workflows.

For this purpose we are exploring Galaxy [21], one of the most adopted software in the bio-informatics community, to manage life science data and pipelines. The aim, in fact, is to enable the user to interact with a pipeline view a web browser.

As an added value, the output results can be made available on the most common S3-like object storage and analyzed via web-based tools like, in the present case, the Integrative Genomics Viewer (IGV) Web Application [22], an open-source web service for genomics visualization deployed in the same cloud oriented infrastructure as a Kubernetes micro-service.

## 5. Conclusions

In the present work the design and implementation of the proof of concept related to a cloud-based computational genomics platform has been presented. Based on Cloud-oriented solutions deployed over the INFN EPIC Cloud IaaS, the Computational Genomics platform is aimed to provide a set of cloud-enabled services available to the research communities and make a smart and efficient use of the provided resources.

The intent of the presented computational model is to improve the control over the resource allocation, providing the optimal amount of resources to each computational task and subtask composing the genomics pipelines embracing the micro-services approach.

The initial results of this work show that the proposed modern, scalable and agile computing model based on the cloud paradigm, leads to a much more resilient and dynamic processing architecture from which the omics, and more in general the life science communities, may benefit. In fact, the performances achieved on the cloud platform are almost comparable with those obtained in a local batch cluster having similar hardware resources. As an added value, our approach provides the needed flexibility and resilience typical of a modern cloud computing environment properly tuned to manage and process medical genomics data.

Activities are still in progress to achieve single sign-on for authentication and authorization. In such respect, a solution based on JWT, OAuth and Keycloak integrated with Kubernetes is currently under investigation. Once the feasibility of this approach will be proved, the platform will be finally ready to scale up to full size on a production environment.

## 6. Acknowledgements

The work presented in this paper has been supported and co-funded by the following projects:

- NextGenerationEU European initiative through the Italian Ministry of University and Research, PNRR Mission 4, Component 2 ICSC [23]: Investment 1.4, Project code CN00000013 - CUP I53C21000340006;

- The Italian Complementary National Plan PNC-I.1 "Research initiatives for innovative technologies and pathways in the health and welfare sector” D.D. 931 of 06/06/2022, "DARE - DigitAl lifelong pRevEntion" initiative [24], code PNC0000002, CUP: B53C22006480001”.

## References

[1] *Anaconda*, https://www.anaconda.com/open-source. Last seen, April 2024.

[2] *Snakemake*, https://snakemake.github.io. Last seen, April 2024.

[3] *Slurm*, https://slurm.schedmd.com/overview.html. Last seen, April 2024.

[4] *OpenStack*, https://www.openstack.org. Last seen, April 2024.

[5] *Open Container Initiative*, https://opencontainers.org. Last seen, April 2024.

[6] *Docker*, https://www.docker.com. Last seen, April 2024.

[7] *GitLab*, https://www.gitlab.com. Last seen, April 2024.

[8] *Containerd*, https://containerd.io. Last seen, April 2024.

[9] *Kubernetes*, https://kubernetes.io. Last seen, April 2024.

[10] *Rancher Kubernetes Engine 2*, https://ranchergovernment.com/products/rke2. Last seen, April 2024.

[11] *Trivy*, https://trivy.dev. Last seen, April 2024.

[12] *Ceph*, https://ceph.io/en/. Last seen, April 2024.

[13] *RADOS Gateway*, https://docs.ceph.com/en/latest/rados/2. Last seen, April 2024.

[14] *Singularity*, https://sylabs.io/singularity/. Last seen, April 2024.

[15] *Prometheus*, https://prometheus.io. Last seen, April 2024.

[16] *Grafana*, https://grafana.com. Last seen, April 2024.

[17] *Helm*, https://helm.sh. Last seen, April 2024.

[18] *JSON Web Token*, https://datatracker.ietf.org/doc/html/rfc7519. Last seen, April 2024.

[19] *OAuth 2.0 Authorization Framework*, https://www.rfc-editor.org/rfc/rfc6749.html. Last seen, April 2024.

[20] *Keycloak*, https://www.keycloak.org. Last seen, April 2024.

[21] *Galaxy Project*, https://galaxyproject.org. Last seen, April 2024.

[22] *IGV-Web App*, https://igv.org/doc/webapp/. Last seen, April 2024.

[23] *ICSC*, https://www.supercomputing-icsc.it/en/icsc-home/. Last seen, April 2024.

[24] *DARE*, https://www.fondazionedare.it/en/dare-digital-lifelong-prevention/. Last seen, April 2024.