H2020-INFRAEOSC-2018-3

NI4OS-Europe

National Initiatives for Open Science in Europe

Deliverable D3.3
Recommendations for HPC centres on-boarding

Lead beneficiary(s):	IICT (editor)
Author(s):	IICT, KIFU, IPB, GRNET partners
Status –Version:	Final – f
Date:	February 26, 2021
Dissemination Level:	Public

Abstract: Deliverable D3.3 deals with the analysis of requirements and with the initial design of potential on-boarding approaches for HPC centres to EOSC, taking into account developments within EDI (European Data Infrastructure) and avoiding any potential duplications or misalignments. Operational and technical requirements and recommendations for the HPC centres are suggested, enabling the HPC centres to measure their EOSC readiness. Suggestions regarding the required policies are provided.

The NI4OS-Europe project is funded by the European Commission under the Horizon 2020 European research infrastructures grant agreement no. 857645.

The NI4OS-Europe project is co-funded by the European Union Horizon 2020 programme under grant number 857645.

Document Revision History

Date	**Issue**	**Author/Editor/Contributor**	**Summary of main changes**
September 25, 2020	a	E. Atanassov and T. Gurov	Initial ToC
November 6, 2020	b	E.Atanassov, T. Gurov, D. Vudragovic, T. Maray	First draft
January 10, 2021	b	Partners	Remarks and partners' contributions
January 25, 2021	c	E. Atanassov et al.	Major revision
February 7, 2021	d	A. Mishev, T. Kazinczy, E. Imamagic, K. Koumantaros, D. Vudragovic, N. Liampotis	Major revision, moved content to appendixes
February 18, 2021	e	E. Atanassov et al.	Minor revision, resolution of comments, editing
February 26, 2021	f	E. Atanassov et al.	Minor revision, additional inputs, QC

Table of contents

References

[1] Project NI4OS-Europe-857645 - Annex I - Description of the Action

[2] Ansible, https://github.com/ansible/ansible/releases

[3] Azab, Enabling Docker Containers for High-Performance and Many-Task Computing, 2017 IEEE International Conference on Cloud Engineering (IC2E), Vancouver, BC, 2017, pp. 279-285, doi: 10.1109/IC2E.2017.52

[4] EOSC-hub, https://www.eosc-hub.eu/

[5] FAIR Principles, https://www.go-fair.org/fair-principles/

[6] Gerhardt, L., Bhimji, W., Canon, S., Fasel, M., Jacobsen, D., Mustafa, M., Porter, J., Tsulaia, V., Shifter: Containers for HPC, (2017) Journal of Physics: Conference Series, 898 (8), DOI: 10.1088/1742-6596/898/8/082021

[7] Kubernetes, https://kubernetes.io/

[8] Kubespray, https://kubernetes.io/docs/setup/production-environment/tools/kubespray/

[9] Kurtzer GM, Sochat V, Bauer MW (2017) Singularity: Scientific containers for mobility of compute. PLoS ONE 12(5): e0177459. https://doi.org/10.1371/journal.pone.0177459

[10] Lustre® file system, https://www.lustre.org/

[11] Merkel, D., 2014. Docker: lightweight linux containers for consistent development and deployment. Linux journal, 2014(239), p.2

[12] NVIDIA GRID, https://www.nvidia.com/fr-fr/design-visualization/technologies/grid-technology/

[13] NVIDIA, Vingelmann, P. & Fitzek, F.H.P., 2020. CUDA, release: 10.2.89, Available at: https://developer.nvidia.com/cuda-toolkit

[14] OpenStack: Open source software for building private and public clouds, http://www.openstack.org/ Open Nebula, https://opennebula.io/

[15] Shifter, https://github.com/NERSC/shifter

[16] Singularity, https://www.sylabs.io/singularity/

[17] TOP500 Supercomputer Sites, https://www.top500.org/

[18] EOSC Profiles, version 3.00, https://eosc-portal.eu/providers-documentation/eosc-provider-portal-resource-profile

[19] NI4OS-Europe, D5.2 First report on provider and repository integration, 2021

[20] EGI Federation EC3, https://servproject.i3m.upv.es/ec3/

[21] NI4OS-Europe, D5.1 Provider landscape analysis and provider categorization, 2020

List of Figures

List of Tables

List of Acronyms

AAI	Authentication and Authorization Infrastructure
API	Application Program Interface
CIDR	Classless Inter-Domain Routing
CPU	Central Processing Unit
EOSC	European Open Science Cloud
FAIR	Findability, Accessibility, Interoperability, Reusability
GPU	Graphics Processing Unit
HPC	High-Performance Computing
KVM	Kernel Virtual Machine
LVM	Logical Volume Manager
MPI	Message Passing Interface
RAM	Random-access memory
SEE	South-East Europe
SLA	Service Level Agreement
VLAN	Virtual Local Area Network
VM	Virtual Machine

Executive summary

What is the focus of this Deliverable?

The main focus of this deliverable is to analyze the requirements and to provide initial design for the potential on-boarding of HPC centres to EOSC, taking into account both the specifics of EOSC and the established best practices in running HPC. The approaches provided should serve to facilitate the practical on-boarding of the HPC centres in the region, while also summarizing expertise and ideas that can be usable to wider audience, as the typical European supercomputing centres are not consistently integrated in EOSC at the moment, and the entire topic will be analyzed in the next 3 years in the wider European landscape by different initiatives. In order to accommodate the differences in both HPC hardware and types of available services, the deliverable provides different possible routes for opening access to the HPC services through EOSC, so as to enable centres with different levels of support and deployed services to become open for EOSC access. As the activity itself is defined as having a certain research component, the results and conclusions from testing certain deployment scenarios are also presented and discussed.

What is next in the process to deliver the NI4OS-Europe results?

This deliverable summarized the output of the task. Based on the conclusions and recommendations from this deliverable and the experience acquired throughout its preparation, the HPC centers which are interested can proceed to registering their resources in NI4OS-Europe service catalogue, choosing one or more of the provided options. Thus the results of this activity will support the refinement of the on-boarding approach that will be further described in D3.4 – Best practices for on-boarding and related policies 2nd version.

What are the deliverable contents?

The deliverable starts with analysis of the specifics of the available hardware in order to establish requirements and limitations for EOSC integration. Starting from there, we systematize the operational and administrative requirements that are to be satisfied. Two different routes for satisfying these requirements are presented in the next two sections – provisioning direct user-level access as is typical in HPC/supercomputing and provisioning access through virtualization and containerization. In the next section we explain the specifics about how to integrate HPC services as a type of NI4OS-Europe generic service, which should be the main way to achieve EOSC integration, while some other possibilities are also discussed. Due to the importance of dealing with data in EOSC, we also discuss various approaches and recommendations to facilitate easier access to data for users of HPC services. Considerations with regards to the policies that accompany EOSC services are provided. The deliverable conclusions summarize the most important considerations and conclusions obtained during this initial design phase.

Conclusions and recommendations

We present a comprehensive set of options in order to allow HPC centers to select their approach based on their operational organization and the expected usage. As a straightforward and highest performance option we recommend to provide direct access to the services, although access through virtualization or containerization is also a good option for more mature centres. The deliverable summarizes operational requirements, policy considerations and gives ideas about improving access to related research data following FAIR principles.

1 Introduction

Although there has been significant focus on the ensuring of access to research data in Europe, the data is meaningless without the capability to analyze it and attain new knowledge, using powerful computational resources. That is why it was important to allow HPC centers to expose their resources to researchers through the EOSC. It has to be noted that there is substantial "cultural" gap between the HPC world, led by powerful supercomputers with hardware that is tightly coupled, highly optimised to solve the envisaged scientific problems, and the world of distributed computing, where the goal is to minimize communication between largely independent tasks (jobs), with far more open data access patterns and protocols. That is why opening up the relatively closed HPC centers, especially supercomputers, for access through EOSC, presents unique challenges in different areas, like operations, policy, software, support.

Nevertheless, there are many technologies that allow for resolving these difficulties and the particular approach chosen by the HPC center will be determined by their level of maturity and availability of technical expertise and support, as well as the type of users that they expect to attract via EOSC. Our proposed approaches are motivated by the analysis of the hardware and software setups of the HPC centres in the region, where we notice substantial diversity. Representative resources and analysis of their setup is provided.

As the main option for offering HPC resources through EOSC, we consider providing direct access, as it safely allows the maximum performance of the system to become available to the end users. The usage of virtualization or even conteinerization technologies is also described in the Apendix A, as a possibility for HPC centers that have appropriate levels of support and expertise, as well as the desire to provide more flexibility and to support certain workloads that map naturally to such kind of environments. Notably, many HPC applications consist of single or multiple executions that use single powerful server (e.g., one that has advanced GPU), that can compensate for the introduced overhead on the account of ease of access and user friendliness.

We also tackle the question of data provisioning as part of the HPC services, mainly from technical point of view, with the main goal to ensure smooth experience for the user. Various options are provided, from simpler to more advanced approaches.

The operational requirements that are specifically resulting from ensuring EOSC access are also considered and suggestions and recommendations are provided. We expect that the HPC centre would integrate support of EOSC users as part of their regular operations, while some integration with outside services, e.g., monitoring, accounting, helpdesk, authentication, as provided by the NI4OS-Europe project, will be beneficial to lower the burden on the centre.

By providing comprehensive list of options, requirements and recommendations to the centres we hope to help especially the smaller HPC clusters to open-up towards EOSC users and research groups, thus allowing larger groups of scientists to obtain access to state-of-the-art equipment backed up by mature software and operations environment and adequate support.

2 Analysis of HPC resources in the region

We start by presenting a brief analysis of the available HPC systems in the region in order to establish the motivation and basis for the recommendations of the deliverable.

It is important to note the contradiction between the desire to open the systems for wide access by diverse user communities and to ensure security and preservation of not only data but also privacy in general, especially since some of these systems are unique at their national level and thus may perform research tasks that are sensitive in various aspects. Another area of contingency that is also discussed in detail is the importance of achieving the highest available performance, which typically is obtained by having direct access to the hardware, while ease of use is usually attained by providing some sort of standardized access, using technologies like virtualization or containerization.

These complex interactions are present not only in the region, but also in Europe in general and have been a significant hindrance to adoption of HPC technologies by researchers and in general.

2.1 Available HPC systems in the NI4OS-Europe partnership

There is substantial heterogeneity in the types of systems available in the region, their sizes and setups. As the aim of this deliverable is to deliver approaches and recommendations, we only present several systems that are representative of the type of resources and the challenges present when trying to expose them through EOSC.

First of all, we have supercomputers, defined as systems that have been present in the Top500 list of supercomputers [17]. Then we mention specifically one large system with shared memory. The rest of the systems can be considered as HPC clusters, sometimes equipped with InfiniBand and/or accelerators, for which systems we present some examples instead of exhaustive list.

Provider	System	Architecture	Cores	Interconnection	Memory	Storage	Accelerators
IICT-BAS	AVITOHOL	Intel	20700 (total)	FDR InfiniBand	9600 GB	96TB	Intel Xeon Phi 7120P
GRNET	ARIS thin	Intel	8520	FDR InfiniBand	-	1200TB	
	ARIS fat	Intel	1760		22TB		
	ARIS GPU	Intel	880 (CPU)		2816GB		NVidia K40 GPU
	ARIS Phi	Intel	360 (CPU)		1152GB		Xeon Phi 7120P
	ARIS ML	Intel	40 (CPU)		512GB		NVIDIA V100
KIFU	Debrecen 2 (Leo)	Intel	1344 (CPU)	FDR InfiniBand	10TB		Nvidia K20x and K40x
	Debrecen 3 (Apollo)	Intel	1056 (CPU)	FDR InfiniBand			Xeon Phi 7120P
	Budapest	Intel	768 (CPU)	FDR InfiniBand	2TB		
	Miskolc UV	Intel, ccNUMA	352 (CPU)	NUMALink	1.4TB		
CyI	CYCLONE	Intel	680 (CPU) + 640 GPU	HDR-100	6.5 TB	5 PB disk + 150TTB nvme	64 NVIDIA V100 GPU
	EPYC	AMD EPYC (Rome)	1024	HDR-100	2 TB	As above	
IPB	PARADOX-IV	Intel	1696 (CPU)	QDR InfiniBand			NVIDIA Tesla M2090
SRCE	ISABELLA		3100 (CPU)	FDR InfiniBand	16 TB	756TB	NVIDA Tesla V100

AVITOHOL (Bulgaria)

AVITOHOL consists of 150 HP Cluster Platform SL250S GEN8 servers with 2 Intel Xeon E 2650 v2 CPUs and 2 Intel Xeon Phi 7120P coprocessors.

Site: IICT-BAS/Avitohol

Manufacturer: Hewlett-Packard

Cores: 20700

Interconnection: FDR InfiniBand

Theoretical Peak Performance: 412.3 TFlop/s

RMAX Performance: 264.2 TFlop/s

Memory: 9600 GB

Operating System: Red Hat Enterprise Linux for HPC

Compiler: Intel Composer XE 2015

Storage systems: 96 TB storage (currently)

Top500 entry: https://www.top500.org/system/178609/

Additional resources: 8 thick nodes with 3TB RAM, 4 CPUs Intel Intel Xeon Gold 6238L 2,1GHz, 22 cores each, 12 GPU-based servers Fujitsu Primergy RX 2540 M4, NVIDIA Tesla V100 32GB 128 GB RAM, CPU 2x Intel Xeon Gold 5118 2.30GHz 24 cores, 2x800GB SSD, 3*12TB HDD
Interconnected with non-blocking FDR and now HDR InfiniBand (200Gbps linespeed).

Figure 1: AVITOHOL

ARIS (Greece)

The ARIS infrastructure consists of a total of five computing islands based on Intel x86 architecture interconnected into a single non-blocking InfiniBand FDR14 network that offers multiple processing capabilities and architectures.

- A thin node is based on the IBM NeXtScale platform and Intel Xeon E5-2680v2 Ivy Bridge processors. It has 426 computing nodes and offers a total of 8,520 cores (CPU cores).
- An island of large memory nodes (fat nodes) consisting of 44 Dell PowerEdge R820 servers. Each server offers 4 Intel Xeon E5-4650v2 Ivy Bridge processors and 512 GB of central memory
- An island of GPU accelerator nodes consisting of 44 Dell PowerEdge R730 servers. Each server contains 2 Intel Xeon E5-2660v3 Haswell processors, 64 GB of memory and 2 NVidia K40 GPU cards, and

- An island of Xeon Phi accelerator nodes consisting of 18 Dell PowerEdge R730 servers, each containing 2 Intel Xeon E5-2660v3 Haswell processors, 64 GB of memory and 2 Intel Xeon Phi 7120P co-processors
- A machine learning node island consisting of 1 server, which contains 2 Intel E5-2698v4 Broadwell processors, 512 GB of central memory and 8 NVIDIA V100 GPU cards
- Top500 entry: https://www.top500.org/system/178545/

Figure 2: ARIS

NIFF(KIFU) (Hungary)

The priority for NIIF Institute is to keep their systems using the most advanced technologies available to serve the widest range of the scientific community. This can only be achieved by joining projects aiming to integrate or develop our systems. Some of the NIFF clusters are:

- **Debrecen 2 (Leo) -** GPU cluster - The cluster have 1344 Sandy Bridge CPU cores, accelerated with 252 Nvidia K20x and K40x GPGPUs adding 3576 more (real) cores. The machine has more than 10Tbytes of RAM.
- **Debrecen 3 (Apollo) -** Phi cluster - The cluster has 1056 Sandy Bridge CPU core, accelerated by 90 Xeon Phi coprocessor, adding 5490 more cores to be available for computations. The machine has nearly 6Tbytes of RAM.
- **Budapest cluster -** The machine has 768 db Opteron CPU cores, and more than 2Tbytes of RAM.
- **Miskolc - UV machine -** This is a UV 2000 machine from SGI, which provides similar technology to regular shared memory architectures, called ccNUMA. The difference between UVs and clusters is that jobs can use all CPUs and full memory capacity for the same computation on the same OS on UVs.
- The machine has 352 Sandy Bridge CPU and 1.4Tbytes of RAM (which can be fully utilized by a single job, if needed). Despite the machine being visible as a single server, there is a batch system that manages the execution.

Figure 3: NIFF UV cluster

CYCLONE (Cyprus)

CYCLONE – 17 forty-core compute nodes, 16 forty-core compute nodes, each with 4 Nvidia V100 GPUs, 2 twenty-core sockets per node, each is Intel Xeon Gold 6248, 96 GB memory per CPU node, 192 GB memory per compute node, Aproximately 5 TB, 135 TB NVMe Storage, 3.2 PB Storage, HDR 100 interconnect, GPUs accelerator, CentOS, 600 TFlop/s

CYICLOUD – Based on OpenStack (Stein), 20 physical servers (24 cores, 128 GB RAM), CEPH storage backend with 200 TB.

EPYC Server – Based on AMD EPYC (Rome). 8 Servers x 128 cores x 2 threads.

Figure 4: CYCLONE

PARADOX (Serbia)

PARADOX Cluster at the Scientific Computing Laboratory of Institute of Physics Belgrade consists of 106 compute nodes (2 x 8 core Sandy Bridge Xeon 2.6GHz processors with 32GB of RAM + NVIDIA® Tesla™ M2090) interconnected by the QDR InfiniBand network.

PARADOX is an HP Proliant SL250s based cluster with the following components:

- Compute nodes: HP Proliant SL250s
- Processors Type: Intel® Xeon® Processor E5-2670 (Sandy Bridge, 8 Core, 20M Cache, 2.60 GHz)
- Number of nodes: 106
- Number of CPU cores: 1696

- Number of GPUs: 106 NVIDIA® Tesla™ M2090 (5375MB of RAM, 512 CUDA cores at 1.3GHz, Compute capability 2.0)
- RAM: 32 GB/node (4x8GB) DRR3 1600MHz
- Network infrastructure: InfiniBand QDR

Operating system:

The operating system on PARADOX cluster is Scientific Linux 6.4

Figure 5: PARADOX

<u>ISABELLA (Croatia)</u>

Isabella consists of 135 worker nodes with 3100 processor cores, 12 GPUs and 756 TiB data space. As a shared resource of all scientists in Croatia, it allows using significant computational resources in demanding data processing of scientific and research projects.

A total of 135 computer nodes provide users with:

- 270 Intel CPUs
- 3,100 CPU processor cores
- 12 NVIDA Tesla V100 with 16 GB
- 16 TB of RAM
- 756 TB of shared data space
- InfiniBand FDR

Cluster middleware used:

- Son of Grid Engine batching system
- BeeGFS parallel file system
- ScaleMP vSMP fat node with 160 cores and 2 TB RAM
- Singularity
- Ganglia monitoring system
- CentOS 7.

Figure 6: ISABELLA

2.2 Overview of the main hardware and software features of the HPC systems

On the basis of the systems that are present in the partnership, as outlined in the previous section, we can summarize their hardware and software features as follows:

Processors: Mostly Intel or compatible AMD. Options like ARM or IBM Power are not widespread in the region.

Co-processors and accelerators: Most of the time these are NVIDIA CUDA GPUs [13], while substantial amount of compute power is present in Avitohol via Intel Xeon PHI.

Memory: mostly distributed memory systems (clusters), at least one shared memory machine (single system image).

Interconnect: InfiniBand is the most popular interconnect for the larger installations. Pure Ethernet-based smaller clusters are also present.

Operating system: Different versions of Redhat Enterprise Linux, including RHEL for HPC or compatible (like CentOS), rarely SUSE or others.

Shared Storage: Lustre, NFS, others

Virtualization solutions: OpenStack, OpenNebula, NVIDIA Grid (commercial software specific for virtualizing access to NVIDIA GPUs [12]).

Batch systems: PBS/Torque, SLURM, others.

Although there is substantial diversity in hardware, it is encouraging that the operating systems are mostly compatible. The batch systems are also very similar in their functionality. Most popular middleware for managing virtualization seems to be OpenStack. We do not deal here with application software and libraries, but usually they are deployed under a shared filesystem and their compatibility depends on the usage of the same operating system and type of library for support of accelerators (e.g. CUDA version) rather than anything else.

2.3 Main operational and administrative requirements

When provisioning access to certain HPC resources to outside users, the corresponding HPC centre has to take into account its own governance rules and technical limitations. Here we list the main points that have to be taken into consideration.

Security

The HPC resources are expensive facilities, usually unique at the national level for the respective country and as such should be properly protected from unauthorized access. The constant ongoing attacks that are easily observed for any system that is even partially visible on the Internet dictate that the HPC systems should be closely guarded and monitored. In most cases firewalls limit access to the system and certain services may be available only locally.

Traceability

Stemming from the above points, for each system it is important to be able to trace each action to the specific user that initiated it. Sometimes the reason this is desirable is that the execution of certain codes leads to various kinds of technical problems. The increased complexity of the HPC systems increases the difficulty of understanding and solving such problems and also creates situations where undesirable interactions between workloads from different users happen. Apart from traceability towards different users, it is also important to be able to tell which applications are active and to which projects their use has to be attributed.

Availability

Due to the complexity of HPC systems and their high resource use (e.g., electricity) and dependence on the proper functioning of several systems like cooling, fire detection/prevention, etc. their constant availability cannot be guaranteed. Complex maintenance works may require sometimes extended downtime periods. While large-scale commercial providers may be able to design their centres with high availability in mind, typical HPC and especially supercomputer systems for research are designed mostly for performance and cost-effectiveness. That is why applications that absolutely require 24x7 availability with very low downtimes cannot rely on a single centre 100%. Another consideration in this area is that some high-profile applications require the use of the whole system or a substantial part of it. Thus even if the whole system is fully operational, it can be tied-up in certain workload for substantial periods of time. This is an issue for applications that work in online or quasi-online regime.

2.4 Analysis of the requirements, stemming from EOSC

In order to expose HPC services through EOSC, one has to ensure the requirements that are specific to EOSC are satisfied. Although to some extent they overlap with these previously mentioned, we try to focus on those that are specific to EOSC and the possible gaps.

Policies

A comprehensive set of policies should be clearly outlined. Although most centres already have these, they have to be updated and in some cases translated. The templates provided from NI4OS-Europe can be useful in this respect. Some issues that are specific to HPC and have both technical and administrative aspect, are considered in Chapter 5.2.

Availability

The services should be made available to an audience that is wider than the current user-base. Geographically this means to extend beyond national borders. There should be clear understanding of the limitations of the offering, for example whether and how business/commercial use is allowed.

Accounting

It is important to be able to distinguish EOSC users from other users. It is also necessary to ensure correct and comprehensive measuring of their use of resources. Usually certain limits to the resource usage are established, like 5% or 10% from the available CPU core hours.

Pricing

A comprehensive pricing scheme should be established. It is customary to measure the resource utilization in terms of CPU core hours and the data usage in TB/month. As different types of storage may be available, it is acceptable to have different prices for the different types. Additional services that are offered will also necessitate their own pricing model. For some more advanced types of services the measurement can be in number of queries per month. It is advisable to avoid complex and difficult to understand measures.

Monitoring

The availability of the service and its operational state should be constantly monitored. It is advisable to make the results of the monitoring visible to users.

3 Technical recommendations for provision of direct HPC access

3.1 Technical approaches for provisioning of direct access to the underlying HPC infrastructure

The EOSC environment is more conducive to provision of access to the HPC infrastructure using one of the virtualization or containerisation options or packaging the desired workloads as a thematic service or function-as-a-service. However, the highest performance and the biggest flexibility when using HPC is obtained via unencumbered direct access. That is why the direct option should be the first to consider.

In this subsection we discuss some high-level considerations that determine the technical solution, while more concrete steps are described later.

Note that once the issue of how exactly to provide the user account to the system is solved (see next subsection), there are very few technical changes that are to be implemented.

The usage accounting should be ensured to provide enough information in order to properly measure and attribute the use of resources (usually CPU-core-hours) to EOSC.

The level of availability of resources should be decided. It is advisable to create a different queue or queues for EOSC users. Some batch systems have the option to manage accounts separately, which gives good high-level information about the distribution of usage among projects, communities or applications. If such an option is enabled, it should be clearly described in the usage manual so that its use is consistent.

Since HPC resources are usually closely guarded, it is expected that users are to obtain access for login through ssh only. Some batch systems allow web interface for job submission, in which case users may have access to that too. It is desirable to discourage and even disable the use of passwords for access with preference to public/private keys.

The management of access to data is discussed much more in detail in another section.

3.2 Simplified access to the HPC infrastructure

The NI4OS-Europe AAI allows users to associate their public SSH key(s) to their profile using the User & VO/Group Membership Registry (COmanage Registry) at https://aai.ni4os.eu/registry. The keys can then be made available to connected services. Specifically, SSH keys are released through the sshPublicKey (urn:oid:1.3.6.1.4.1.24552.500.1.1.1.13) in the case of SAML services and the ssh_public_key claim in the case of OpenID Connect clients.

It should also be possible for the NI4OS-Europe AAI to provision the user's SSH keys to Unix based systems. There is no requirement for custom SSH clients or servers, though tight integration with the VO/group which can access these systems is implied. Specifically, the user identifier, VO/group information and SSH keys maintained in the NI4OS-Europe Membership Registry can be provisioned to LDAP (or another suitable location), and the Unix servers would need to be configured to read their account and authentication data from LDAP using standard PAM, NSS, SSSD, and/or SSHD configuration. In this manner, accounts may be provisioned and configured on the fly.

An alternative approach would require the installation of a non-standard PAM module implementation (e.g. https://github.com/stfc/pam_oauth2_device) that would allow users to log in via SSH using OpenID Connect/OAuth2, instead of SSH Keys. Such PAM implementations use the OAuth2 Device Flow (https://oauth.net/2/device-flow/), which is already supported by the NI4OS-Europe AAI OpenID Connect Provider. Specifically, upon SSH login, users will be presented with a URL which they can open in a web browser and authenticate using their preferred academic/social credentials registered with RCIAM. Upon successful authentication, the PAM module can check if the user is in the right VO/group(s) or have a specified username and allow or deny access.

3.3 Operational recommendations for advanced provisioning of direct access to the underlying HPC infrastructure

There are at least two general approaches that HPC systems take to the management of user accounts. One is to manage accounts using standard system tools, probably with some degree of automation. The other is that they implement a form of Single-Sign-On (SSO), usually using an LDAP directory as the backend for authentication.

The procedures around the procurement of user accounts can vary from site to site depending on local policies and configuration of storage systems, batch systems, etc.

Certain degree of automation is beneficial to increase the take-up of HPC services via EOSC. However, a heavyweight implementation may impose undesirable unification requirements on the individual centres and clash with local policies and setups. Thus the next procedure is an option that some centres could implement, depending on how many users they expect to handle. In the Figure 7 we see a schematic description of the process, presuming that the user management system is essentially a web application/front end, which interacts with the User management system/back end. It is acceptable and probably advisable in such a setting to have also a manual step where the requests are accepted by the support staff individually. The users can submit their signed forms to the HPC Account procurement service and, once their request is accepted, they can also receive their SSH keys. Alternatively, a manual procedure (using email) can be utilized by some centres. We consider that it is more important to establish contact with the user and enable secure and easy submission of the required forms.

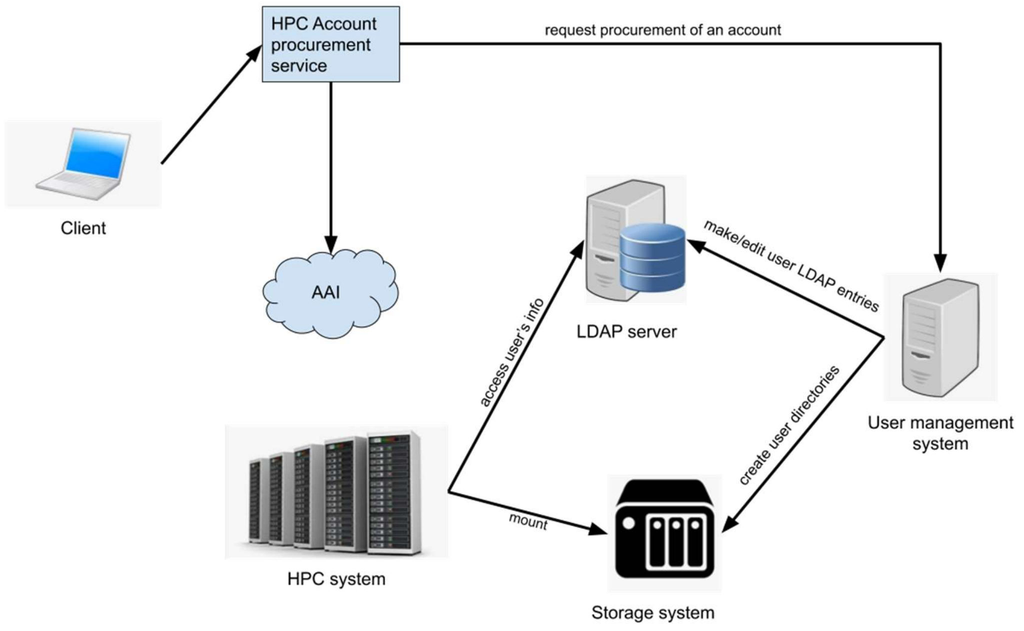

Figure 7: Overview of the HPC user account procurement scheme

3.4 Approaches related to the data provisioning

This section describes the various issues related to the provision of local data access, interplay with other data services, backup, repositories, etc. and provides approaches to deal with these.

The provisioning of data is strongly intertwined with the provisioning of access to HPC services. In view of the usual tight security profiles of HPC systems, this presents certain challenge for smooth integration of HPC to EOSC. In this chapter we consider the relevant issues and approaches for dealing with them.

First of all, HPC users that obtain access through EOSC should obtain adequate access to the fast local storage. Since this storage is usually a limited and highly contested resource, it is advisable to establish and enforce quotas, so that the EOSC users have automatically limited access to this resource. Certain amount of space can be provided automatically and higher allowances should be subject to additional negotiations/SLAs, etc. Group/project quotas are recommended, whichever is applicable.
In general, standard quota management tools can be used to setup/enforce quotas. Note that for Lustre file systems quota enforcement can be enabled on a per-filesystem basis. Sites should take into consideration also some privacy issues. They are advised to use numeric user names and to make sure that access for different projects/users should be clearly separated by ownership and ACLs, using the available technical means. Additionally, for sensitive data caching should generally be disabled.

Usually this kind of storage was provided via NFS and this is still the case for smaller clusters without specialized low-latency interconnection. However, larger production installations use high-performance parallel filesystems.

One popular solution for this question, also in the research community, is the Lustre filesystem [10]. The Lustre filesystem serves local storage through several dedicated nodes, divided into two types. The management type nodes that deal mostly with metadata are MDS/MDT, while the OST nodes serve the actual data. It is advisable to enable the high-availability features of Lustre, if there is enough hardware for this. Using active-active HA server pairs and multi-path storage connections certainly improve expected availability and minimize possible downtimes. Large centers like Avitohol already have this types of installation. However, enabling quotas is another optional step during installation that is well-described in the Lustre manual and should be performed. It is also advisable to enable the management of additional user groups as this is helpful in managing access to software with restricted rights (e.g., if certain commercial software should be accessible only by certain users).

Due to the vastly higher performance of flash storage vs HDD and the decreasing cost, having a flash storage layer becomes preferred and essential option for provision of local storage. There are other filesystems that have varying performance features and give additional flexibility.

In the context of EOSC the important questions are related to other access protocols that can be available, long-term storage (essentially object storage as opposed to the usual block storage provided by Lustre/NFS, etc.), automatic backups and so on.

For most applications having POSIX-compliant filesystem is a necessity, so we definitely discourage consideration of options that are not POSIX-compliant.

Fortunately, NFS for small clusters and Lustre for large installations are POSIX compliant. Other possible option that deserves consideration is the WekaIO filesystem, which has a POSIX interface to MatrixFS with optimal performance, intended as the primary ingest and operational interface for clients. The recommended cluster design also incorporates a sample on- or off-premises S3-compatible object storage interface, which enables to potentially provide snapshot to cloud. As most of the research HPC centers will have only on-premises object storage, they could use dense storage servers, coupled with on-premises object storage software like Scality RING or SUSE Enterprise Storage. Although WEKA Matrix is not considered as a replacement for traditional NFS or SMB solutions, it does to support these protocols. Non-POSIX clients share a common namespace with POSIX clients, and provide best possible performance within the limitations of SMB and/or NFS protocols.

4 Initial design for technical and operational integration in EOSC

4.1 Initial design for technical integration of HPC systems with the NI4OS-Europe core services as a generic service

In order to integrate an HPC system as NI4OS-Europe generic service, there are two technical integration steps to be made. Some centers may provide an endpoint for a partially or fully automated HPC account provisioning service, as described before. Other centers will provide a web page, which allows for requesting the service and establishing contact between the user and support staff in order to complete the whole account creation process.

The second technical step is to install the accounting data export scripts that will extract usage data relevant for the NI4OS-Europe projects. This is a python script that will have a minimal number of dependencies (possibly none), which will upload the data to a NI4OS-Europe service endpoint. Its invocation should be handled by the *cron* service. The already provided accounting publisher code can be used as a template for this. Sites are advised to do also storage space accounting. The frequency of data collection is up to each site but publishing shall be done periodically. The method used by the site must be consistent, e.g. daily average or maximum. Usage data may be aggregated for months. Again, the NI4OS-Europe accounting service has templates for publishing such data.

The process for provisioning of user accounts is well researched and effectively solved at each of the high-performance resource centers. For the purpose of streamlining the process in the case when the user requests and obtains access through EOSC, we consider several useful options. First of all, it is advisable to maintain pre-configured user accounts, dedicated for EOSC users, and hand them out as the users requests are accepted. Since due to GDPR requirements these accounts should not contain user's names or other specific information, this will be easy to follow.

Until the process is very well tested in production, we assume that manual review of the requests will be performed and that is why we do not propose full automation.

What could be done, however, is for the resource center to maintain a web page where the users can log-in using Single Sign-On (SSO) credentials, possibly upload signed forms or input other technical or administrative information that is needed, and download access credentials (i.e., ssh public/private key pairs) when their request is granted. This process could work also with the NI4OS-Europe authentication system, since the issue how to enable this authentication to work for a web-based application is already used in production.

4.2 Policies for access to HPC

When preparing the policies for the services that expose HPC resources, the resource owners may use the generic policy templates that have been provided by the NI4OS-

Europe project. Here we discuss several issues that are specific for the HPC resources and combine technical and administrative considerations. Our recommendations are not mandatory, but we try to summarize best practices and to give an idea about the minimum that has to be done in order to ensure proper use of the otherwise highly valuable HPC resources. Most of these issues are the same for both local and EOSC users.

1. Forbidding multiple accounts and also account sharing. For cases when a group needs to share data between themselves, appropriate shared directory should be provided by the center instead.
2. Ensuring possibility to contact the user in case of urgency, e.g., technical or security incident. Such data should be kept strictly confidential, but usually is necessary in case when user workload causes disturbance in the operations of the center.
3. Multi-factor authentication if there is a case of highly sensitive data or usage. Nevertheless, such situations should be resolved on case by case basis, since the usual technical means of ensuring security in the HPC center may not be adequate for higher security requirements.
4. Consider forbidding unencrypted data exchange in/out of the centre. Especially forbid as a minimum the unencrypted data exchange of sensitive data.
5. Forbid explicitly storing of any dubious kind of data. As an example, credit card information, as wells as medical data and similar should not be allowed except in very limited circumstances. Offensive content should also be clearly banned.
6. Forbid explicitly certain kinds of mostly illegal activity or activity that can lead to reputational damage to the data center. As an example, usage of HPC facilities for breaking encryption should be forbidden in most cases.
7. Commercial usage of the HPC facilities in most cases is not accepted.
8. In most cases users cannot run services from inside the data centre, because of firewall configuration. However, there may be situations when such use is acceptable and required. For such kind of use there should be separate rules, explaining the responsibilities of the users, e.g., with regards to security of the underlying system providing the service.
9. User's responsibilities in case of security incident should be clearly stated.
10. Forbid security testing. There have been cases of users running software that tests system security automatically. Without explicit written permission running such kinds of tests is undesirable and should be prohibited.
11. Forbid evasion of resource utilization controls. Although technical measures are usually in place to ensure that users do not login to random nodes that are not part of their jobs, it is useful to explicitly forbid such attempts. Users shall not purposely engage in activities to circumvent computer security or system administrative measures (for example batch queue control settings).
12. Certain nodes are usually available for development/testing purposes. These are usually the login nodes. It should be clearly stated what kind of usage is acceptable on these nodes and what is not.
13. Users should not expose information that they obtained accidentally through the use of the system. For example, if they came across data from other users' files, they are not allowed to share it.

14. Usage of the system for political purposes is highly undesirable and should be forbidden.
15. Proper acknowledgement of the use of the centre's resources should be ensured.
16. In some cases a technical report of the results of the use should also be requested.
17. Use of unlicensed software should be prohibited.
18. Users should understand that the use of the system is a privilege and not a right. As such, it can be revoked at any time. Access to the system is not guaranteed and should not be assumed.
19. The HPC resources should not be used for activities with high operational requirements that cannot be guaranteed. For example, usage of the system for management of nuclear plants, air traffic controls, medical procedures.
20. Collection of accounting and monitoring data – users should be aware that such collection is being made. It should also be stated that the HPC center adheres to the GDPR and the relevant legislation and does not sell personal data to 3rd parties.
21. FAIR principles [5] – reflect the adherence to the FAIR principles for the research data (this issue is not discussed here because it is more relevant for data-oriented services and is tackled in different work packages of the project).

4.3 Approaches for provisioning access to HPC services packaged as a thematic service

The main focus of the deliverable is the provision of access to "raw" HPC services, i.e. users to be able to launch their workloads directly on the underlying hardware infrastructure. However, some kinds of application workloads are sufficiently popular so as to make it worthy to expose them as thematic services that hide the complexity of the execution and improve the user experience. On the other hand, in order for such services to be considered as proper HPC services, they have to be resource intensive enough. The partnership that is currently in NI4OS-Europe has vast experience in using certain generic portals in the Grid environment, like P-GRADE and its descendant technologies. Other job execution schemes that are community-agnostic, achieved certain popularity. However, with the maturing of the user communities, at certain point they achieve the capability to use HPC services that are available through ssh-based user access only, by developing the corresponding codes/scripts and hooking them up with their own community-specific portals. In such case it is not necessary for the HPC provider to develop a thematic service. The most high-probability situation where such a thematic service would be usable is when there is a sizable cluster of local users that utilize an internationally recognized open-source application. In such case the HPC service could be developed mainly for them and then exposed through EOSC to gain additional international users. An example of such workloads can be the calculation of meteorological prognosis or longer term climate modelling. Since such computations also generate significant amount of output data, it is important to ensure good accessibility of these data. From point of view of integration to EOSC such thematic services follow the usual route for thematic services, with added consideration for proper accounting of the usage, since it may happen that different queries use different amounts of computing resources and thus the measurement will not be based on number of queries, but most probably on number of CPU hours and TB/month of data. It is also important to evaluate the expected network utilization in order to be sure that network usage does not lead to network overload, for the countries that have more limited network connectivity.

4.4 On-boarding the HPC resources through the AGORA service catalogue

4.4.1 Requirements for service providers

The provider is defined [18] as an EOSC system user responsible for the provisioning of one or more resources to the EOSC. Since some partners within the NI4OS-Europe consortium are resource providers, including the providers of HPC resource, they are registered within the Agora. In this case, the on-boarding team aims to fully describe, in terms of EOSC profiles mandatory information, providers which resources are also on-boarded. The details for on-boarding of the providers are described in the deliverable D5.2 First report on provider and repository integration [19].

4.4.2 Requirements for HPC as generic services

As described in the NI4OS-Europe deliverable D5.1 Provider landscape analysis and provider categorization [21], the following details are collected in the EOSC profiles regarding the HPC resources:

- Basic information
  - Peak performance [TFlops] - Theoretical peak performance of the service in TFlops, including CPUs and Accelerators
  - Server specification – Vendor specific information about servers
  - Number of servers
- CPU details
  - CPU Specification – Vendor and model of CPU
  - CPUs per server
  - Cores per CPU
  - RAM per server [GB]
  - RAM per core [GB]
  - Total number of CPU-cores – Total number of CPU cores for the entire system
  - Max number of parallel processes – maximum number of parallel processes allowed for end users
  - CPU peak performance [Tflops] - CPU theoretical peak performance of the system
- Accelerator details
  - Accelerator specification – Vendor and model of the accelerator
  - Total number of accelerators
  - Accelerators per server

  - Maximal number of accelerators per server
  - Accelerators peak performance [Tflops] - Accelerators theoretical peak perfomance of the system
- Interconnection
  - Interconnect type – Interconnection technology between servers
  - Interconnect latency [μs]
  - Interconnect bandwidth [Gbps] - Interconnection bandwidth between nodes
- Filesystem details
  - Local filesystem type – Shared Filesystem used for interconnecting nodes
  - Total storage [TB]
- Software details
  - Operating system
  - Batch system/scheduler
  - Development tools
  - Libraries
  - Applications

The metadata is organized in six groups for more clarity as well as to provide users with more structured information and help them select the most suitable HPC resource to their processing needs.

4.5 Operational requirements to support the integration of an HPC system

The provisioning of HPC access to EOSC users imposes additional requirements to the operations of the HPC data centre. Once the question of user account provisioning is solved as outlined in the previous section, we have several other points of purely operational nature that need to be solved.

A support ticket queue should be opened for the system on the HelpDesk service and support accounts should be opened for the service provider's support staff. By leveraging NI4OS-Europe helpdesk we can save ourselves the trouble of maintaining a dedicated helpdesk instance when the HPC center does not already run one.

The HPC system should have a short user guide available and a link to it should be provided in the EOSC page.

The system availability in general is already tackled by the staff, but with relation to EOSC the staff needs to ensure that adequate resources are available for actual execution of workloads. In the various batch systems there are well established ways to solve this problem, e.g., by ensuring dedicated resources available or quotas. The most desirable solution is to define certain percentage of the resources as available for EOSC and then ensuring that this percentage is available *on the average*, along certain period of time. Such a solution is better than full dedication of resources, because HPC resources are typically oversubscribed and should not be left in idle state without good reason.

The one exception to this approach is if an SLA is negotiated with a user/user community where the dedication of resources is guaranteed.

In most cases of HPC systems some queue with shorter wait times and limited execution time is available (i.e., for short jobs). It is advisable to make this queue available also to EOSC users.

5 Conclusions

We described several viable options for how to expose HPC resources in EOSC. The NI4OS-Europe core services are useful to facilitate the process. Some of the options like virtualization, containerization, and packaging as a thematic services, are more complex and require more effort, while actually by definition they cannot offer the same performance attainable through direct access. On the other hand, provisioning of direct access to the HPC facilities is relatively straightforward to implement and well supported operationally, via the services like helpdesk or accounting. Thus it is to be the first option under consideration for production implementation, while the other possibilities are dependent on the maturity of the data center operations and the needs of potential user communities that can come to use the resources from EOSC. Some datacenters that already have implemented e.g., OpenStack, can proceed to offer HPC services also from there.

In all cases HPC services are coupled with appropriate data access. Some data centers that are in the process of expansion can consider upgrading their technology for offering shared storage with the aim of achieving better performance and more flexibility in view of the FAIR principles.

Overall there are sufficient applicable options for offering HPC services and the data centers should start doing this, leveraging the capabilities of NI4OS-Europe core services on one hand and operational teams on the other.

However the actual availability of HPC capacity through EOSC will have to be resolved at the later stage when long-term cross-border reimbursement schemes are established at the pan-European level, in collaboration with EuroHPC, PRACE and similar large HPC initiatives.

6 Appendix A: Options for provisioning of HPC access through virtualization and containerization

6.1 Recommendations for provisioning of HPC access in EOSC through virtualization

For many years now virtualization is integrated in the data centres, including in the HPC facilities for research. It has many advantages, especially in the context of EOSC. Some of the disadvantages are related to possible drop in performance or lack of access to advanced HPC features that are available natively. The most important examples for the latter are the access to low-latency networks like InfiniBand, which have been notoriously difficult to virtualize, or the access to all the performance of the available accelerators like GPUs or Xeon Phi. Especially the Xeon Phi have been a hindrance to adoption of virtualization in our experience running the Avitohol supercomputer.

Traditionally in SEE region and also in Europe in general, Openstack has become the dominant way of provisioning virtualized resources for researchers. When the servers at certain datacenter are equipped with powerful accelerators or are otherwise well endowed for computations (thick nodes), they can be provisioned as single virtual machines with some appropriate designation. Provisioning this type of resources is well researched and practically in use, for example, in EGI Federation, and thus does not need special investigation.

The other important option that we need to consider for the purpose of this deliverable is the provisioning of interconnected machines (virtual cluster). In HPC such kinds of resources are usually utilized for execution of MPI applications.

One example of such architecture is EGI Federation EC3 [20], which enables simple deployment of HPC clusters on OpenStack.

The MPI standard remains the most used framework in the world of the High-Performance Computer Systems but enabling a virtualization platform to run such workloads is not a trivial task especially when it comes to “As a Service” offerings.

Having a user catalog to provision an on-demand MPI Cluster with specific configuration for the exact workload is something that falls under the domain of Infrastructure Management Solutions. OpenStack, while being among the most adopted platforms providing higher-level resources abstraction, also provides enough granularity to be tuned for specific use cases. As is the case with such middleware types, it reduces user complexity in favor of operational complexity. Fortunately, the deployment work got significantly reduced by projects like RDO Project, providing a packaged and tested OpenStack platform for CentOS. Reducing the setup time and effort combined with the support of diverse hardware positions such a platform as the natural choice to building an MPI Cluster offering.

The proposed OpenStack deployment architecture follows the best practices defined by the *RDO Project* for medium size infrastructure assets. The suggested environment consists of two management nodes, at least one storage node and 8 or more worker nodes.

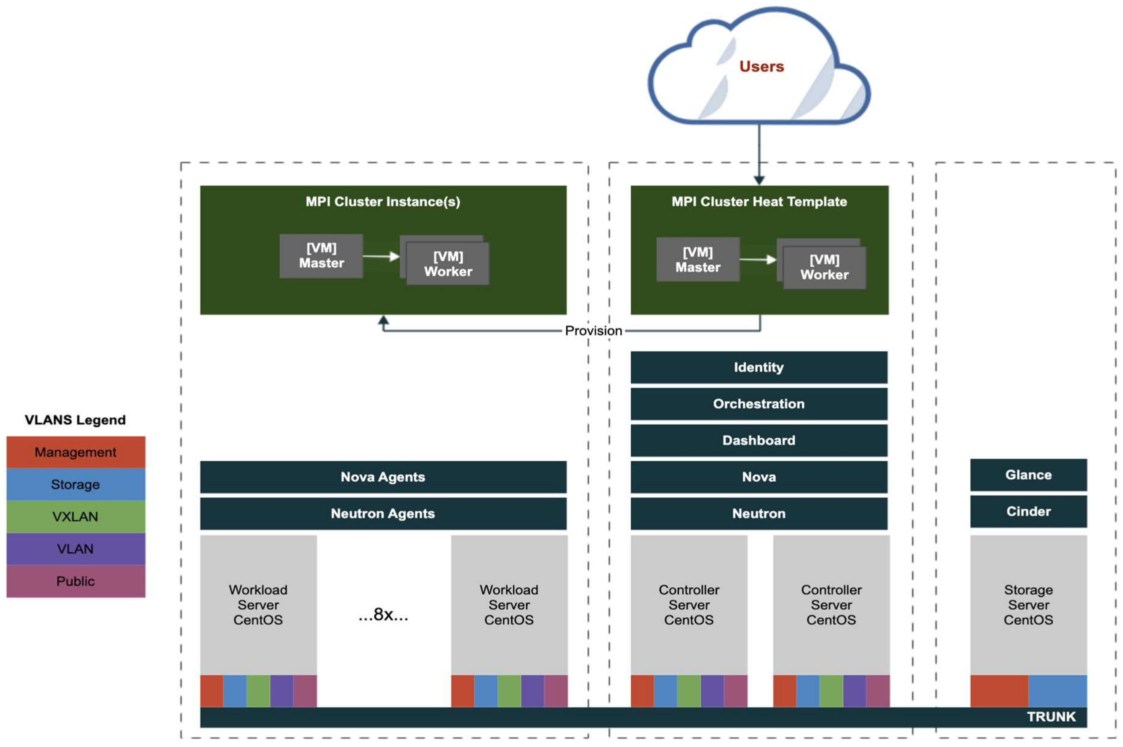

Figure 8: System architecture diagram

All nodes are connected to a standard L2 segment enabled for port trunking with five VLANs segmenting the traffic by purpose usage.

VLAN Name	VLAN Description	CIDR
Management	Carrier for management traffic, the link between the management nodes and the worker and storage nodes.	172.26.236.0/22
Storage	Carrier for storage traffic, the link between worker nodes consuming a block storage and images from the storage node.	172.26.244.0/22
VXLAN	Carrier for on-demand user defined networks, stretch between the worker nodes.	172.26.240.0/22
VLAN	Carrier for physical and external networks, stretch between the worker nodes.	N/A
Public	Carrier for public traffic to and from the internet, stretch between the worker nodes.	194.***** (example, in practice depending on the available external IPs)

Table 1: Networking setup

The storage system for this template installation consists of a single storage node connected with a fiber-channel to a storage array system. Both Cinder (block storage) configured with LVM driver and Glance (imaging) services are running on this host. It is advisable to expand into using more storage nodes for a larger installation. In such case dedicated nodes will be necessary for Cinder and Glance.

The compute system is the heart of the installation. In the template proposed there are eight worker nodes hosting KVM and QEMU, forming the compute virtual infrastructure required to run the OpenStack workloads. A production installation should have at least 20 worker nodes.

All regular HPC users are accessing the platform from the internet via HA Reverse proxy configured with a publicly trusted certificate.

The north-south firewall is configured to expose only Identity, Dashboard and Orchestrator services, blocking the internet interaction with the rest of the OpenStack Service's ecosystem.

The east-west management traffic is secured by purpose with services port communication limits and VLAN segmentation.

The east-west workload traffic is isolated by the boundaries of the OpenStack environment, not enforcing any default firewall profiles. The security of the individual workloads needs to be secured by the users.

For the purpose of this investigation, the OpenStack infrastructure management platform is used mainly for High-Performance Computing Workloads, exclusively for MPI workloads or combined MPI/OpenMP.

In order to address the demands of these workloads and the individual requirements we exposed the MPI Cluster as a Service via Heat Template. The template engine allows to combine the description of the MPI master and worker machines, their network communication, and security in a single human-readable textual form. It also makes it possible to dynamically select the size of the MPI Cluster depending on the workload requirements.

Example MPI Cluster Heat Template:

BAS::IICT::MPIClusterService

```
heat_template_version: 2016-10-14


description: MPI & OpenMP Cluster as a Service


parameters:

 name_prefix:

   type: string

   label: The name prefix for MPI machines

 image_id:

   type: string

   label: The ID of the MPI Node Image
```

```
  flavor:
    type: string
    default: m1.small
    constraints:
    - allowed_values:
      - m1.small
      - m1.large
      description: The MPI Node Flavor
  workers_count:
    description: The number of MPI Worker Nodes
    type: number
    default: 9
  internal_network_id:
    type: string
    label: Internal Network ID
    description: The ID of the internal network
  external_network_id:
    type: string
    label: External Network ID
    description: The ID of the external network
  known_keys:
    type: string
    description: The list of user/known keys for SSH
    default: []
    hidden: true

resources:


  workers:
    type: OS::Heat::ResourceGroup
    properties:
      count: { get_param: workers_count }
      resource_def:
        type: BAS::IICT::MPIWorker
        properties:
          name:
            str_replace:
              template:
                $name-$count
              params:
                $name: { get_param: name_prefix }
                $count: "%count%"
          image_id: { get_param: image_id }
          flavor: { get_param: flavor }
          internal_network_id: { get_param: internal_network_id }
          known_keys: { get_attr: [ known_keys ]  }

  master:
    type: BAS::IICT::MPIMaster
    depends_on: workers
    properties:
      name: { get_param: name }
      image_id: { get_param: image_id }
      flavor: { get_param: flavor }
      internal_network_id: { get_param: internal_network_id }
      external_
 network_id: { get_param: external_network_id }
      known_keys: { get_attr: [ known_keys ]  }
      workers: { get_attr: [workers, ip] }

 outputs:
    external_ip:
      description: The external IP address for MPI Master
      value: { get_attr: [master, ip] }
```

Example of definition of MPI Cluster Resources in a heat template:

BAS::IICT::MPIWorker

```
heat_template_version: 2016-10-14


description: MPI & OpenMP Worker Node


parameters:
 name:
   type: string
   label: The name of MPI machine
 image_id:
   type: string
   label: The ID of the MPI Node Image
 flavor:
   type: string
   default: m1.small
   constraints:
   - allowed_values:
     - m1.small
     - m1.large
     description: The MPI Node Flavor
 internal_network_id:
   type: string
   label: Internal Network ID
   description: The ID of the internal network
 known_keys:
   type: string
   description: The list of user/known keys for SSH
   default: []
   hidden: true

resources:
 wait_condition:
   type: OS::Heat::WaitCondition
   properties:
     handle: { get_resource: wait_handle }
     count: 1
     timeout: 360
 wait_handle:
   type: OS::Heat::WaitConditionHandle
  security_group:
     type: OS::Neutron::SecurityGroup
     properties:
       name: security_group
       rules:
         - protocol: tcp
           port_range_min: 22
           port_range_max: 22
```

```
          - protocol: tcp
            port_range_min: 1024
            port_range_max: 65535
  port:
   type: OS::Neutron::Port
   properties:
     network: { get_param: internal_network_id }
     security_groups:
       - { get_resource: security_group }

 worker:
   type: OS::Nova::Server
   properties:
     name: { get_param: name }
     image: { get_param: image_id }
     flavor: { get_param: flavor }
     networks:
       - port: { get_resource: port }
     user_data_format: RAW
     user_data:
       str_replace:
         params:
         template: |
           #!/bin/bash
           # MPI Worker installation script here ...

outputs:
 ip:
   description: The internal IP address for MPI Worker
   value: { get_attr: [worker, first_address] }
```

BAS::IICT::MPIMaster

```
---
heat_template_version: 2016-10-14


description: MPI & OpenMP Master Node


parameters:
 name:
   type: string
   label: The name of MPI machine
 image_id:
   type: string
   label: The ID of the MPI Node Image
 flavor:
   type: string
   default: m1.small
   constraints:
   - allowed_values:
     - m1.small
     - m1.large
     description: The MPI Node Flavor
 internal_network_id:
   type: string
   label: Internal Network ID
   description: The ID of the internal network
 external_network_id:
   type: string
   label: External Network ID
   description: The ID of the external network
 known_keys:
   type: string
   description: The list of user/known keys for SSH
   default: []
   hidden: true

resources:
 wait_condition:
   type: OS::Heat::WaitCondition
   properties:
     handle: { get_resource: wait_handle }
     count: 1
     timeout: 360
 wait_handle:
   type: OS::Heat::WaitConditionHandle
  security_group:
     type: OS::Neutron::SecurityGroup
     properties:
       name: security_group
       rules:
         - protocol: tcp
           port_range_min: 22
           port_range_max: 22
         - protocol: tcp
           port_range_min: 1024
           port_range_max: 65535
  internal_port:
   type: OS::Neutron::Port
   properties:
     network: { get_param: internal_network_id }
     security_groups:
       - { get_resource: security_group }

 external_ip:
   type: OS::Neutron::FloatingIP
   properties:
     floating_network: { get_param: external_network_id }

 external_ip_assoc:
   type: OS::Neutron::FloatingIPAssociation
   properties:
     floatingip_id: { get_resource: external_ip }
     port_id: { get_resource: internal_port }

 master:
   type: OS::Nova::Server
   properties:
     name: { get_param: name }
     image: { get_param: image_id }
     flavor: { get_param: flavor }
     networks:
       - port: { get_resource: port }
     metadata:
       workers: { get_param: workers }
     user_data_format: RAW
     user_data:
       str_replace:
         params:
         template: |
           #!/bin/bash
           # MPI Master installation script here ...

outputs:
 ip:
   description: The external IP address for MPI Worker
   value: { get_attr: [external_ip, floating_ip_address] }
```

6.2 Options for use of containers and systems for automated management of containerized applications.

Containerization of applications on Linux has made tremendous breakthroughs in enabling portability of programs by packaging them and their network of dependencies in an environment similar to a virtual machine. However, unlike a virtual machine, it does not impose the performance penalty of emulating hardware or running another operating system on top of the host system. Containers use features of Linux (LXC and cgroups) to create an isolated root file system for an image that can hold all the necessary parts of a Linux distribution, above the kernel level, that are required by the target application. This image together with the application runs on top of the existing system kernel.

The most popular software that brought containers into the spotlight is Docker [11]. It is the defacto standard for specifying containerized images of applications, and it is used in many production deployments. For more complex deployments, i.e. execution of multiple containerized applications in a coordinated way, Docker Swarm and Kubernetes [7] are available, but these systems are not very suitable for HPC environments. They integrate poorly with existing job scheduling platforms and their scheduling subsystem is not as flexible and could not coexist with jobs scheduled on these platforms. Regardless of that, the increasing popularity of Kubernetes in the IT industry motivates us to strongly consider Kubernetes-based provisioning of HPC services. Some approaches to that will be outlined later on.

Docker's model of deployment implies that the container image must be run and managed as a superuser, which is not a viable option for HPC use cases. The main focus of the community that has developed Docker was on advancing DevOps approaches to easier application deployment, and security management and portability across different systems were not primary goals.

Several solutions are more or less compatible with Docker and avoid the aforementioned pitfalls. In the rest of this section we will consider the following:

- **Socker** (Azab 2017) [3] is a lightweight wrapper around Docker which allows images to be run by unprivileged users. It also integrates the container control with Slurm. This is a small project from University of Oslo, and besides the github project page and the accompanying paper, the project does not seem to have wide deployment or community. This maturity level makes it less than ideal solution for NI4OS-Europe project.
- **Shifter** (Gerhardt et al. 2017) [6] is a system from the National Energy Research Scientific Computing Center (NERSC). It is a containerization system specifically developed for HPC use cases. It is not based on Docker but it supports import and conversion of Docker images to its custom format. Besides Docker, it can also import virtual machines and CHOS images. It has been used in production on HPC clusters at NERSC, but its user base is still smaller than the option we cover next.
- **Singularity** (Kurtzer, Sochat, and Bauer 2017) [9] is a container platform developed by Sylabs (https://sylabs.io). It has been designed from the beginning to address all the pain points that are preventing Docker from wider adoption on HPC systems. Among the options described in this section, this one is the most widely deployed in production HPC systems. It focuses on verifiable reproducibility of experiments, easier integration with the host system (i.e. use of GPUs or high speed networks), portability of compute with its single file images, and a simpler security model. Unlike Docker, the user that executes a container is the same user within the container so no additional privileges are exposed. Like Docker it has its SingularityHub which is a central repository for sharing container images. Also, Singularity can import, shell, run and exec standard Docker images.

When considering Kubernetes specifically, we note that one of its main advantages is that it accepts the possibility at any moment a component of the application to fail and thus the need to be automatically reconciled, abstracting the application management activities and letting the owners to concentrate on the business logic. When Kubernetes orchestration is merged into the world of HPC, it makes it possible to utilize a single platform for hybrid workloads with flexibility and ease of use.

The most interesting HPC workloads that can benefit from Kubernetes use applications like TensorFlow, Apache Spark, Hadoop as part of their operation. One of the biggest challenges they face is the upgrade and scalability of these components, as there was no commonly adopted standard for orchestration until the Kubernates got traction. Today most of the application software vendors are creating Helm Charts, a Kubernetes resource packages that are easy to maintain, secure, scale and upgrade across compatible platforms. Such flexibility motivates the idea to incorporate Kubernetes into HPC by forming the next generation of HPC Infrastructure, one has to consider the following requirements:

Application Friendly - ability to host a wide range of applications with minimal effort.

API First - provide programmatic access to the infrastructure configuration and runtime orchestration

Feedback Aware - actively monitor infrastructure and runtime components, provide immediate feedback and perform proactive reconciliation

Workload Reactive - monitor the use of platform components and reactively apply scale-in or scale-out operations

Cloud Agnostic - workload independence of cloud provider

Repeatable - easy workload recreation on platforms with similar interface

In order to fulfill these requirements we outline the following approaches:

Application Friendly - Linux Containers is the most widely adopted applications packaging and runtime solution, and Kubernetes is the framework providing the best orchestration for container workloads.

API First - every aspect from infrastructure configuration to application orchestration is handled by Kubernetes API constructs.

Feedback Aware - is in the nature of desired state configuration and the enforcement of this state at any time.

Workload Reactive - the auto scaling is as an extension of Feedback Awareness of Kubernetes resources that automatically scales-in or scales-out resource instances based on usage.

Cloud Agnostic - majority of public cloud providers and private cloud management products provide a Kubernetes Cluster support with its standard API.

Repeatable - as Kubernetes API enforces a desired state on managed resources and it is common across compatible platforms, the workload migration is seamless.

Applying these approaches, we obtain an architecture for Kubernetes-based HPC services as follows:

The traditional Kubernetes infrastructure consists of a cluster including a number of hosts each exposing certain hardware and software capabilities via labels, where a certain number of the hosts acts as management nodes and the rest as workload nodes.

The management set of the cluster runs the vital Kubernetes services such as Etcd, Kube API, Kube Scheduler and Controller, which are common across all implementations serving the operation of the built-in resources. In addition to the standard resource types Kubernetes offers powerful extensibility hooks in the face of the Custom Resources and the Operator Pattern.

A Custom Resource is an extension of the Kubernetes API representing a customization of a particular Kubernetes installation bringing capabilities like virtual infrastructure resources, HPC datasets or jobs under centralized management, where an operator is a software extension to Kubernetes that makes use of these Custom Resources following Kubernetes principles, notably the control loop.

Utilizing these two principles makes it possible to represent any Application or HPC resource as a Kubernetes resource, leveraging a standard API for its state transition, and implementing a control loop (identify, apply, feedback) for the enforcement of the state.

This is represented in the following scheme:

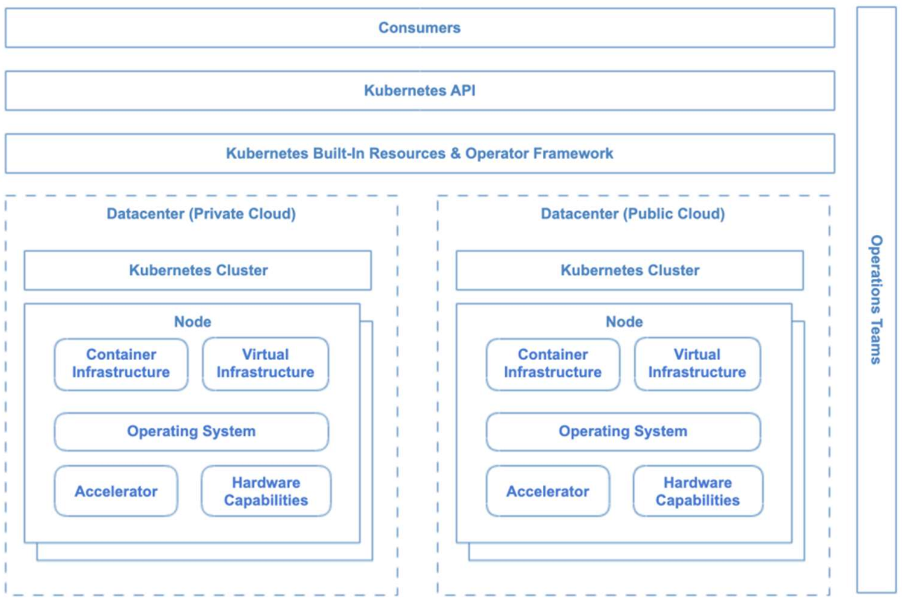

Figure 9: Deployment scheme using Kubernetes

The adoption of Kubernetes brought a dozen solutions for provisioning cluster lifecycle automation, but Kubespray [8] surfaced above the others for this wide range of supported infrastructure providers in face of GCE, Azure, OpenStack, AWS, vSphere, Packet (bare metal), Oracle Cloud Infrastructure (Experimental) or Bare Metal (HPC). That is why it forms the basis of our proposed setup strategy.

The underlying basic infrastructure for the installation consists of a number of workload nodes and at least three management nodes all pre-installed with CentOS 7.x (or equivalent like RHEL 7).

Depending on the available infrastructure the initial provisioning can be achieved via PXE following *Install CentOS via PXE* guide.

Kubespray is utilizing Ansible [2] as configuration management technology to assign the appropriate Kubernetes role to each node part of the infrastructure inventory.

The target servers must have access to the Internet in order to pull docker images.

The Ansible public SSH key must be trusted by all of the servers part of the Ansible inventory.

```
# Generate SSH key
ssh-keygen -t rsa -b 4096

# Copy Public Key to all Nodes, where X represents the node index
ssh-copy-id root@nodeX
```

The following script will fulfill the bare minimum of requirements for creating the proper Ansible tasks.

```
# Enable the Ansible Engine repository for RHEL 7
sudo subscription-manager repos --enable rhel-7-server-ansible-2.9-rpms

# Install Required Packages
sudo yum install ansible
sudo yum install python-netaddr
Sudo yum install python-jinja2

# Allow IPv4 forwarding
sysctl -w net.ipv4.ip_forward=1

# Disable Firewall during installation
# Note, firewall can be enabled and configured baed on the provider needs

# Stop the FirewallD service
sudo systemctl stop firewalld

# Disable the FirewallD service to start automatically on system boot
sudo systemctl disable firewalld

# Mask the FirewallD service which will prevent the firewall from being started by other
services:
sudo systemctl mask --now firewalld
```

After that one can proceed with the actual Kubernetes on-boarding. First all of, some prerequisites are to be installed

```
# Setup Python virtual environment
python3 -m venv venv
source venv/bin/activate

# Clone Kubespray Git repository
git clone https://github.com/kubernetes-sigs/kubespray.git
cd kubespray

# Set active branch
git checkout release-2.14

# Install Playbook requirements
pip install -r requirements.txt
```

Kubespray Ansible playbook requires an infrastructure inventory file as an input to initialize the Kubernetes cluster.

```
# File path ${kubespray}/inventory/hpc/hosts.yaml
# The node that is not a etcd member do not need to set the value, or can set the empty string value.
[all]
node1 ansible_host=EX.Y.Z.1 ip=IN.Y.Z.1 etcd_member_name=etcd1
node2 ansible_host=EX.Y.Z.2 ip=IN.Y.Z.2 etcd_member_name=etcd2
node3 ansible_host=EX.Y.Z.3 ip=IN.Y.Z.3 etcd_member_name=etcd3
node4 ansible_host=EX.Y.Z.4 ip=IN.Y.Z.4 etcd_member_name=etcd4
node5 ansible_host=EX.Y.Z.5 ip=IN.Y.Z.5 etcd_member_name=etcd5
node6 ansible_host=EX.Y.Z.6 ip=IN.Y.Z.6 etcd_member_name=etcd6
node7 ansible_host=EX.Y.Z.7 ip=IN.Y.Z.7 etcd_member_name=etcd7
node8 ansible_host=EX.Y.Z.8 ip=IN.Y.Z.8 etcd_member_name=etcd8
node9 ansible_host=EX.Y.Z.9 ip=IN.Y.Z.9 etcd_member_name=etcd9
node11 ansible_host=EX.Y.Z.11 ip=IN.Y.Z.11 etcd_member_name=etcd11
node12 ansible_host=EX.Y.Z.12 ip=IN.Y.Z.12 etcd_member_name=etcd12

# Configure a bastion host if nodes are not directly reachable via Ansible Host
# bastion ansible_host=x.x.x.x ansible_user=some_user

[kube-master]
node1
node2
node3

[etcd]
node1
node2
node3

[kube-node]
node4
node5
node6
node7
node8
node9
```

```
[calico-rr]
node11
node12

[k8s-cluster:children]
kube-master
kube-node
calico-rr
```

After that the actual cluster installation can proceed

```
# Execute Kubernetes Cluster Installation
ansible-playbook -i inventory/hpc/hosts.yaml -u root -b -v --private-key=~/.ssh/id_rsa
cluster.yml
```

Once the cluster has been installed, one can proceed with a verification of the installation.

```
# Copy permission file from any of the management nodes
scp root@node{1,3}:/etc/kubernetes/admin.conf access.conf
# Change the service access IP from the internal to the external in
vi access.conf

# Setup Client
export KUBECONFIG=$PWD/access.conf

# List cluster nodes
kubectl get nodes
```

The next step is to initialize a Certificate Manager. An example here of how to deploy a CA and Issuer resources in order to provide certificate as a service to the Kubernetes users is provided:

```
# Create CA Spec
cat > ca-csr.json << EOF
{
    "CN": "HPC CA",
    "key": {
        "algo": "rsa",
        "size": 4096
    },
    "names": [
        {
            "C": "BG",
            "L": "Sofia",
            "O": "BAS",
            "OU": "HPC",
            "ST": "Sofia"
        }
    ]
}
EOF

# Build Docker Image to generate CA
read -r -d '' DOCKERFILE << EOF
FROM photon:3.0
RUN tdnf -y install wget
RUN    wget    -O    /bin/cfssl    -q    --show-progress    --https-only    --timestamping
https://pkg.cfssl.org/R1.2/cfssl_linux-amd64 && chmod +x /bin/cfssl
RUN   wget   -O   /bin/cfssljson   -q   --show-progress   --https-only   --timestamping
https://pkg.cfssl.org/R1.2/cfssljson_linux-amd64 && chmod +x /bin/cfssljson
EOF


echo "${DOCKERFILE}" | docker build --tag cfssl:R1.2 -
docker run --rm -it -v $(pwd):/tmp cfssl:R1.2 sh -c "cd /tmp; cfssl gencert -initca /tmp/ca-
csr.json | cfssljson -bare ca"
kubectl create namespace cert-manager
helm repo add jetstack https://charts.jetstack.io
helm repo update
helm install \
    cert-manager jetstack/cert-manager \
    --version v0.15.0 \
    --set installCRDs=true
# Use default
#     --namespace cert-manager \

# Create CA Issuer spec
read -r -d '' CA_ISSUER_SECRET << EOF
apiVersion: v1
kind: Secret
metadata:
    name: hpc-bas-key-pair
data:
        tls.crt: $(cat ca.pem | base64 -w 0)
        tls.key: $(cat ca-key.pem | base64 -w 0)
EOF


echo "${CA_ISSUER_SECRET}" | kubectl apply -f -
read -r -d '' CA_ISSUER << EOF
apiVersion: cert-manager.io/v1alpha2
kind: Issuer
metadata:
    name: hpc-bas-ca-issuer
spec:
    ca:
        secretName: hpc-bas-key-pair
EOF
echo "${CA_ISSUER}" | kubectl apply -f -


# Validate

read -r -d '' TEST_CERTIFICATE << EOF
apiVersion: cert-manager.io/v1alpha2
kind: Certificate
metadata:
    name: example-hpc
spec:
    # Secret names are always required.
    secretName: example-hpc-tls
    duration: 2160h # 90d
    renewBefore: 360h # 15d
    organization:
    - bas
    commonName: example.hpc
    isCA: false
    keySize: 2048
    keyAlgorithm: rsa
    keyEncoding: pkcs1
    usages:
        - server auth
        - client auth
    # At least one of a DNS Name, URI, or IP address is required.
    dnsNames:
    - example.hpc
    - www.example.hpc
    uriSANs:
    - spiffe://cluster.local/ns/sandbox/sa/example
    ipAddresses:
    - 192.168.0.5
    # Issuer references are always required.
    issuerRef:
        name: hpc-bas-ca-issuer
EOF
echo "${TEST_CERTIFICATE}" | kubectl apply -f -
kubectl describe certificates
```

Create a Load Balancers based on Metal LB to provide Load Balancer as a service capability to the Kubernetes users.

```
# Create separate namespace for load balancers
kubectl create ns metallb-system

# Initialize Metal LB
kubectl apply -f https://raw.githubusercontent.com/google/metallb/v0.9.2/manifests/metallb.yaml -n metallb-system

# Generate Metal LB secret
kubectl create secret generic -n metallb-system memberlist --from-literal=secretkey="$(openssl rand -base64 128)"

# Provide Metal LB IP Range / K8S external IPs
read -r -d '' METALLB_CONFIG_MAP << EOF
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: metallb-system
  name: config
data:
  config: |
    address-pools:
    - name: default
      protocol: layer2
      addresses:
      - 192.168.252.1-192.168.252.252
EOF



# Validate
kubectl create deployment echo --image=inanimate/echo-server
kubectl scale deployment echo --replicas=3
kubectl expose deployment echo --port=8080 --type LoadBalancer
```

Once these steps are completed, one is able to provide services as Custom Resources. This is an example of how a custom resource definition of type HPC job can be created. This configuration will enable the users of the Kubernetes clusters to create HPC Job instances. But in order these instances to be realized an operator needs to be implemented and deployed to fulfill the control loop. Typically the operator runs as a Kubernetes pod under system namespace and monitors the resources it has understanding for.

```
cat > ${CUSTOM_RESOURCE_JOB} << EOF
apiVersion: apiextensions.k8s.io/v1
kind: HPCJob
metadata:
  # name must match the spec fields below, and be in the form: <plural>.<group>
  name: jobs.stable.example.hpc
spec:
  # group name to use for REST API: /apis/<group>/<version>
  group: stable.example.hpc
  # list of versions supported by this HPCJob
  versions:
    - name: v1
      # Each version can be enabled/disabled by Served flag.
      served: true
      # One and only one version must be marked as the storage version.
      storage: true
      schema:
        openAPIV3Schema:
          type: object
          properties:
            spec:
              type: object
              properties:
                template:
                  type: string
                priority:
                  type: number
                runtime:
                  type: string
  # either Namespaced or Cluster
  scope: Namespaced
  names:
    # plural name to be used in the URL: /apis/<group>/<version>/<plural>
    plural: jobs
    # singular name to be used as an alias on the CLI and for display
    singular: job
    # kind is normally the CamelCased singular type. Your resource manifests use this.
    kind: Job
    # shortNames allow shorter string to match your resource on the CLI
    shortNames:
    - job
EOF

# Create Resource Definition
kubectl apply -f ${CUSTOM_RESOURCE_JOB}


# Validate Object Creation
cat > ${MY_JOB} << EOF
apiVersion: "stable.example.hpc/v1"
kind: Job
metadata:
    name: my-job
spec:
    template: "Tenzor"
    priority: "High"
    runtime: "Bare Metal"
EOF

# Create Resource
```

```
kubectl apply -f ${MY_JOB}
```

We believe that although Kubernetes is not widely used currently in the HPC for research, due to its high industry support and wide user community it has great potential and can be used to expose HPC services through EOSC.