PROCEEDINGS OF SCIENCE

HA Kubernetes cluster using Octavia Ingress Controller

Francesco Sinisi,[a,*] **Ahmad Alkhansa,**[a] **Diego Michelotto**[a] **and Alessandro Costantini**[a]

[a]*INFN CNAF,*
viale Carlo Berti Pichat 6/2, Bologna, Italy
E-mail: francesco.sinisi@cnaf.infn.it, ahmad.alkhansa@cnaf.infn.it, diego.michelotto@cnaf.infn.it, alessandro.costantini@cnaf.infn.it

With the widespread adoption of containers by various organizations and companies, Kubernetes (K8s), an open-source software dedicated to container management, has become the *de facto* standard in recent years for the deployment and operation of applications focused on this technological solution. K8s offers several advantages: workload balancing, dynamic resource allocation, automated rollout and rollback, storage orchestration, management of sensitive information, self-healing, etc. Of course K8s has some limitations, but they can be overcome thanks to the easy integration with third-party software. Thanks to its fame, in fact, there are many developers who create software that can be integrated with K8s.

Thanks to its flexibility and scalability features, K8s can be integrated with cloud solutions such as OpenStack, a modular cloud operating system capable of offering process and storage management services according to the Infrastructure as a Service (IaaS) model, deployed at INFN CNAF. The inner complementary relationship between K8s and OpenStack has pushed us to widely use this solution in our Cloud infrastructure. One aspect that made us lean towards using the two frameworks mentioned above is the possibility of exposing K8s services externally via a Load Balancer (LB) using Octavia, one of the many open-source modules that integrate into the OpenStack ecosystem. In addition to this, other measures have been implemented, integrating the cluster with external software that should, at the same time, simplify the system administrator's work and enhance security.

If the high reliability of a system is a requirement that is generally very welcome by any user of a service, security is of particular importance in our infrastructure. The architecture presented, in fact, has the purpose of hosting personal data, which requires a high degree of protection against external attacks and isolation between the various users of the infrastructure.

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

*Speaker

PoS(ISGC2024)028

1. Introduction

The National Institute for Nuclear Physics (INFN [1]) is a public research agency founded in the 1950s to further the nuclear physics research tradition initiated by Enrico Fermi. It deals with different topics, ranging from the physics of atomic nuclei to astrophysics. Within INFN, the Center for Research in ICT (CNAF) is committed to providing the needed computational power and expertise to deal with the new technological challenges arising in modern science.

Due to the recent pandemic, the European Union negotiated with Italy the National Recovery and Resilience Plan (Piano Nazionale di Ripresa e Resilienza, NRRP) as part of the Next Generation EU (NGEU) programme. The Plan is developed around three strategic areas shared at the European level: digitization and innovation, ecological transition, and social inclusion. It is an intervention that aims at repairing the economic and social damage caused by the pandemic crisis, contributing to addressing the structural weaknesses of the Italian economy, and leading the country along a path of ecological and environmental transition.

In this regard, INFN is actively participating in different projects concerning digitalization, innovation, digital and technical-scientific skills, research and technology transfer. Guided by these principles, INFN contributes to the DARE [2] project, which seeks to develop solutions for population surveillance, prevention, health promotion, and health security. The nature of the project, the research field in which it operates, involves the manipulation and management of medical data, which must be treated with care. For this reason, we designed and implemented a Cloud-oriented platform capable of hosting services aimed at operating and storing this particular type of data. During the design and implementation of the platform (and related services), different aspects have been taken into account: data resiliency, service availability, resource elasticity, and orchestration. The intent of this manuscript is to present a recipe for quickly and securely building a Kubernetes [3] (K8s) cluster, on top of an IaaS layer provided by OpenStack, to build an elastic and general purpose platform suitable for hosting different use cases and services, and fully integrate with OpenStack inner services.

The document is structured as follows. In Section 2 the new proposed platform is described together with the components adopted and in Section 3 the software services supporting the platform are presented. In Section 4 the basic operations and monitoring features are analysed. Finally, Section 5 draws conclusions.

2. Platform design and deployment

CNAF has extensive experience in distributed computing and cloud-native technologies and solutions. In this regard, we decided to investigate Kubernetes, a *de-facto* standard for container orchestration, and to explore the integration features with OpenStack and its related services, to design and implement a cloud-enabled platform flexible enough to host different use cases.

2.1 IaaS layer provisioning

The Infrastructure as a Service (IaaS) layer offers significant advantages: it is very easy and fast to create virtual machines (VMs), modify existing VMs, and connect all relevant devices to the

network. By leveraging the characteristics of this infrastructure, virtual machines can be optimally sized to meet specific requirements and efficiently distributed across the servers in the data center.

We have adopted some measures for the cluster nodes, especially for the masters. We decided to install fast disks for the latter (SSD), to support the high I/O coming from the ETCD [5] service, present only on this type of node. The worker nodes, on the other hand, have been equipped with slower but less valuable disks (HDD). As for the computational resources used, the approach is reversed: the workers, the nodes that will perform the most demanding tasks, are equipped with more CPUs and more RAM.

What happens if we realize that we have undersized our VMs or that their number is not sufficient to meet the needs of the cluster? We can leverage the high elasticity of this type of architecture. The computational resources of a single VM can be increased in a few steps (**vertical scaling**) as well as their number (**horizontal scaling**). This allows for resource regulation, avoiding unnecessary waste.

It is not only the computing power or the number of VMs that is important, but also their installation *location*. When creating the VMs, it is advisable to distribute the cluster nodes across different hypervisors to minimize disruption to VMs hosted on the same physical hardware. This is especially true for the Kubernetes master nodes, which are critical to preserving the health of the cluster.

For our purposes, we used the OpenStack [4] framework to manage with our IaaS, and Kubernetes was deployed on various VMs provided by Cloud@CNAF. OpenStack is a modular cloud operating system capable of offering process and storage management services according to the IaaS model, and it is widely used at INFN CNAF, particularly in the Cloud@CNAF private cloud.

2.2 Kubernetes deployment solution

Due to its simplified installation procedure and to the virtualization enhancements, the Rancher Kubernetes Engine (RKE) has been chosen for deployment. In particular, the RKE2 [6], which differs from the previous version with regard to the container runtime and the control plane pods management [7]. RKE2 automates and simplifies the creation of a Kubernetes cluster. Using a script, it creates all the necessary components of the cluster based on the role that the node will assume within the cluster (server or agent)[1]. The most important ones are:

- **kubelet**, an agent that runs on each node in the cluster and makes sure that containers are running in pods;
- **Container Runtime** (CR), such as containerd or CRI-O, is the software responsible for running containers, and **CR Interface** (CRI) is the protocol for the communication between kubelet and CR;
- **Container Network Interface** (CNI), such as Calico [8] or Flannel [9], is a plugin for cluster networking;
- some useful services, such as HELM [10] (the package manager for Kubernetes) and Nginx [11], are installed by default.

[1]The terms "master-worker" and "server-agent" are interchangeable. The latter pair is preferable due to ethical language considerations.

To manage the cluster, the tool starts a service on systemd-based systems on both the server(s) and agent(s) nodes. This service supervises the core components of the cluster (kubelet, etcd, controller, scheduler, Helm), which are created as **static Pods**. Static Pods are managed directly by the kubelet daemon on a specific node. The Figure 1 summarizes the main components of an RKE2 cluster and the services deployed on both the server(s) and agent(s) nodes.

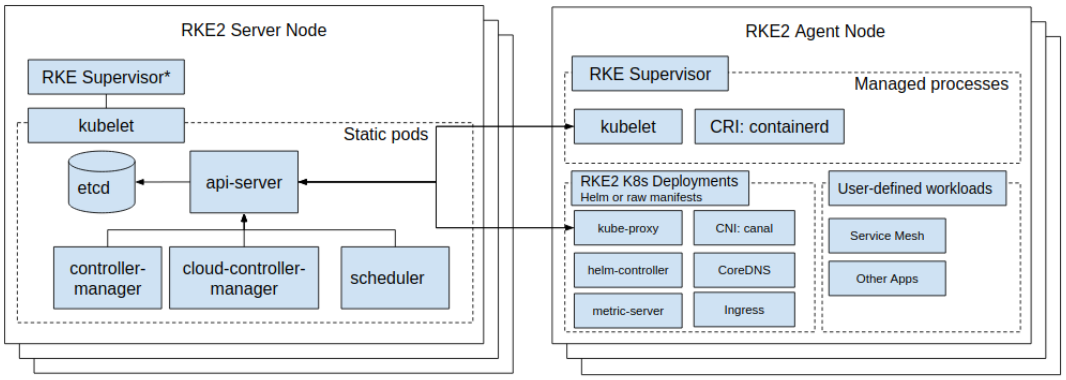

Figure 1: The diagram shows the basic components of a server node (on the left-hand side) and an agent node (on the right-hand side). RKE2 automatically installs these components when the service starts. From RKE2 [6].

The advantages of using RKE2 can be summarized as follows:

- it is supported by an active community, which continuously updates the software;
- vulnerabilities are periodically fixed, using trivy to scan regularly for CVEs on the images used in the cluster;
- offers a good user guide, which significantly reduces the learning curve regarding its use;
- RKE is also a stable CNCF-certified (Cloud Native Computing Foundation) Kubernetes distribution that can be used in any kind of production environment.

RKE2 is a fully compliant Kubernetes distribution designed with a focus on security and regulatory requirements for the U.S. Federal Government sector. As a consequence, one of the features of adopting RKE2 is the possibility to create clusters that meet the **Center for Internet Security** (**CIS**) benchmarks [12]. At the time of creation, the cluster already has some of the requirements listed in the benchmark by default. Some are activated automatically through a simple configuration of RKE2. Still others must be activated manually by the administrator. Below are some of the most important changes required to adhere to the benchmark:

- **Protection of kernel parameters**: this is a kubelet flag that will cause the kubelet to exit if the required kernel parameters are unset or are set to values that are different from the kubelet defaults.
- **Ownership of ETCD folders**: the ETCD data directory should be owned by the ETCD user and group.

- **Limitation of communication between namespaces**: the NetworkPolicy (§ 3.1) used will only allow pods within the same namespace to communicate with each other.
- **Default service accounts**: the default service account should be configured so that it does not provide a service account token and does not have any explicit rights assignments.

Creating the first Kubernetes cluster node and adding subsequent nodes is very fast thanks to the RKE2 functionalities. It provides a bash script that installs all the necessary software, based on the node role. We decided to take an additional step, integrating RKE2 into Puppet [13]. This combination speeds up the creation of a cluster or its re-creation in the event of a downtime. It also constantly checks the state of the system and maintains the desired configurations. For example, changing an RKE2 configuration file, subject to Puppet control, is automatically applied to all nodes of the cluster. Conversely, any inadvertent changes made by the administrator on one of the nodes are rolled back.

2.3 Networking and Load Balancing

OpenStack provides advanced networking solutions able to improve and harmonize the different services needed by Kubernetes to be connected. In this regard, Octavia [14] is an OpenStack module dedicated to the creation and management of Load Balancers (LB). Figure 2 presents a schematic illustrating the components required for the operation of an LB.

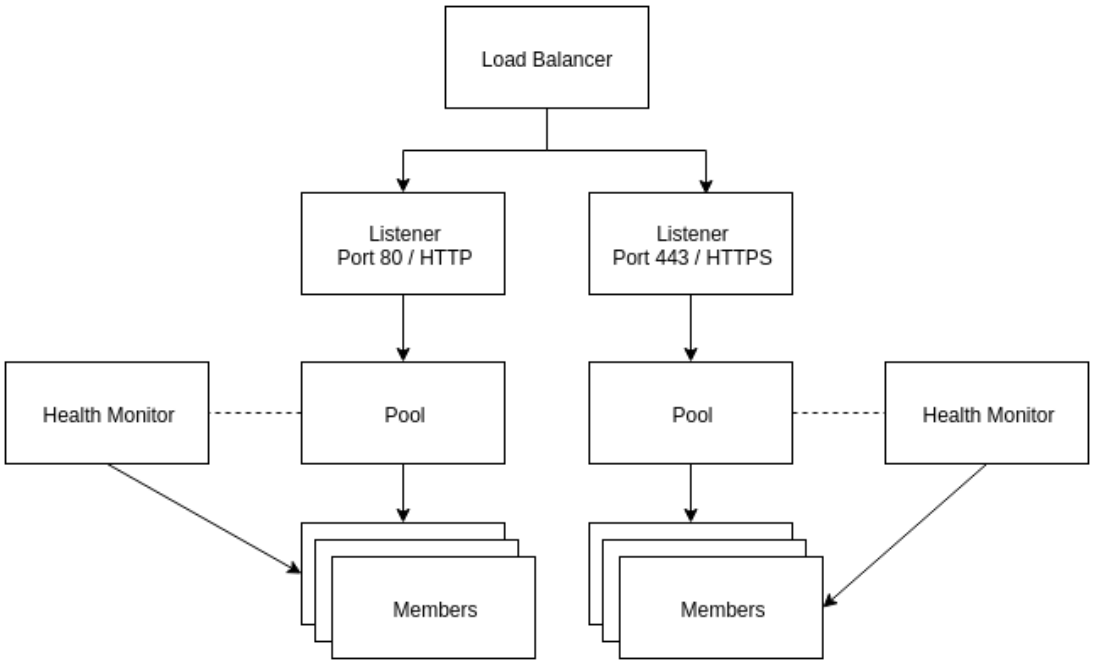

Figure 2: Diagram showing the components of a LB: each port of the LB needs a listener; each listener, in turn, is associated with a pool that represents the set of nodes intended for managing incoming traffic. The Health Monitor, on the other hand, monitors the status of the pool. From OpenStack [4].

When an LB is generated via this module, two identical VMs are automatically created: the first with the role of **master**, the second with the role of **backup**. The master is responsible for

receiving and sorting all incoming requests. In case the master goes down, the backup takes over the role of master, thus keeping the LB service active. The master VM is associated with a Public IP which acts as the external communication channel.

As can be seen from the Figure 2, it is necessary to create a **Listener** for each port. For each Listener, in turn, it is needed to associate a **Pool** and a **Health Monitor**. The Pool is the set of nodes to which traffic coming from outside will be directed and it is advisable, for obvious redundancy reasons, that it is made up of several nodes. The Health Monitor ensures that the Pool is healthy, meaning that there is at least one node belonging to the pool capable of accepting the request. Furthermore, it is possible to choose the traffic routing algorithm between the listener and the pool nodes (e.g., Round Robin, Least Connection, Source IP).

The Octavia service allows for fine-tuning some parameters, such as the configuration of Listener timeouts in case of non-response from the Pool members, and limiting incoming traffic for a subset of IP addresses (CIDR). Finally, one of the advantages of using an LB is to reduce the use of public IPs, which are a rare and therefore precious resource. Without an LB, several public IPs would be required to maintain the same level of service reliability.

3. Software services

Different software services and solutions have been deployed to support our new Kubernetes-based platform. An overview of the software solutions adopted, both those running on the cluster as well as those deployed externally, is provided and explained in the following sections.

3.1 Kyverno

Kyverno is a policy engine capable of automatically **imposing limits**, chosen by the administrator, on storage, CPU and RAM, both at namespace and container levels. The tool allows customization of the namespaces and, more generally, it can also be applied to other kinds of Kubernetes resources.

When creating a namespace, in fact, the administrator can set a maximum limit for each resource (CPUs, RAM, storage) for the entire newly created namespace. For correct deployment of resources on the cluster, it is good practice to set at least CPU and RAM values. Setting the computational requests needed for the correct running of an application allows the scheduler to intelligently create Pods, choosing the freest nodes as destination. Furthermore, not setting a maximum limit for the Pods could lead to excessive consumption of the underlying resources, affecting the functioning of the other Pods and the host node. The administrator can set default values for the CPU and RAM of the containers, as well as a maximum limit per single container.

Kyverno does not simply set a maximum limit on existing resources, but also allows the **creation of resources from scratch**. As an example, to comply with the rules present in the CIS (cf. § 2.2) benchmark (i.e., isolate cluster components belonging to different namespaces) some network policies have been adopted. To achieve this configuration, the `globalNetworkPolicy` has been used to block communication between all namespaces. To allow communication towards system namespaces (e.g. `kube-system`), the `networkPolicy` has been adopted. Its higher priority, compared with the `globalNetworkPolicy`, and the fact that it can be considered as exception to

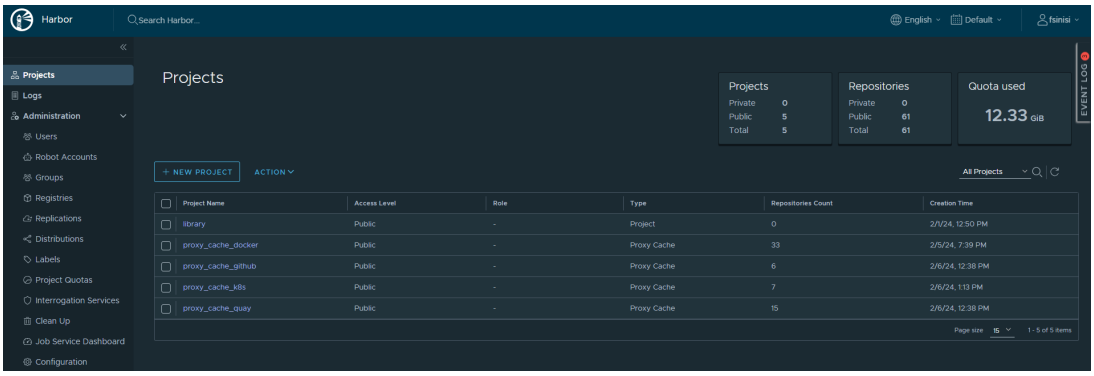

Figure 3: The Harbor dashboard shows different projects hosting images coming from other registries.

the global policy, enable such functionality. On the other hand, this component needs to be created for each namespace, but this is handled by Kyverno, which automates the process.

3.2 Harbor

Harbor [15] is used by Kubernetes for image management. It is an image registry capable of storing container images. We created an instance of Harbor and configured the cluster to request images from it. Harbor, acting as a **proxy**, requests images from the official registry, if they are not present in the application. By using Harbor as a **cache**, images are downloaded only once from the official registries. This is important because official image registries usually impose a limit on download requests. Harbor can speed up the deployment of the cluster by caching the images and reducing downloads. Figure 3 shows a screenshot of the Harbor dashboard where projects hosting the images coming from other registries (DockerHub, Quay.io, GitHub Container Registry and K8s Registry) are made available.

3.3 Ceph-FS as a Container Storage Infrastructure (CSI)

Among the various recommendations in the CIS guide (§ 2.2), the one suggesting avoiding the use of `HostPath` is the most relevant: Pods should not mount data present on the cluster nodes. For this reason, we decided to adopt Ceph-FS [16] as the CSI for data storage. It is a plugin, installed on Kubernetes, that allows communication between Kubernetes and a Ceph cluster.

To allow communication between the two clusters and to fulfill its purposes, the plugin must have in its configuration the ID and key pair, generated in Ceph, in such a way as to contact the portion of storage reserved for Kubernetes, and a Storage Class, who should be contacted whenever a volume is needed (Figure 4). With this configuration it is possible to dynamically generate PVCs (Persistent Volume Claims), which will then be associated with Pods, respecting the storage limits imposed by Kyverno (§ 3.1) for each single namespace. Similarly, when a PVC is destroyed in Kubernetes, the corresponding part of storage present in Ceph will also be deleted.

3.4 ArgoCD

ArgoCD [17] is a continuous-delivery framework specifically designed to be integrated into Kubernetes. It is capable of synchronizing the code present in a repository and the related application. It allows setting black/white-lists at the cluster or namespace level and user-level policies. It is

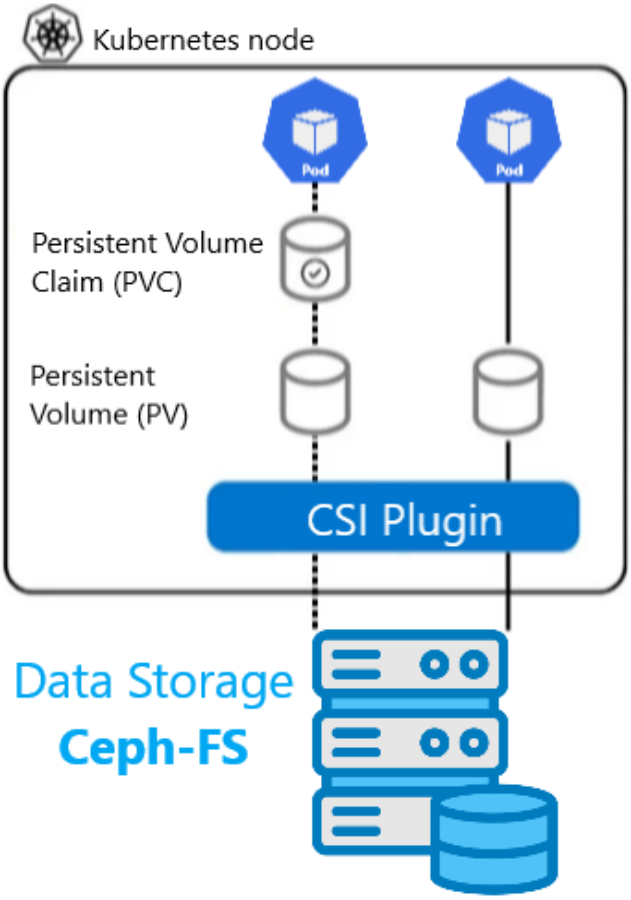

Figure 4: Thanks to the Ceph plugin, it is possible to connect the PVCs created dynamically in Kubernetes with the portion of Ceph storage intended for the Kubernetes cluster.

possible to integrate it with an OpenID Connect (in our case, INDIGO-IAM [18]) and Prometheus, so that users can conveniently monitor the metrics of their containers in the ArgoCD dashboard (Figure 5). ArgoCD can interact with the container terminal and enable log visualization of the pods directly from the dashboard.

Another interesting functionality of ArgoCD is that it allows users to use the cluster **without exposing the Kubernetes APIs**. The user simply makes changes and improvements to the code via classic commits to their repository. Whenever the code changes, ArgoCD will implement the changes on the cluster. Due to the functionalities mentioned above, the user has at their disposal tools (metrics, terminal, log) to analyze their applications.

4. Operations and Monitoring

As already mentioned, public IPs are usually few and precious. For our cluster, we only used two: one for the LB and the other for a VM, not belonging to the cluster, which acts as a bastion, entry point or jump host. Access to this VM is allowed via SSH only to administrators, and the software necessary for management is installed on it: `kubectl` to contact the API, `HELM`, used

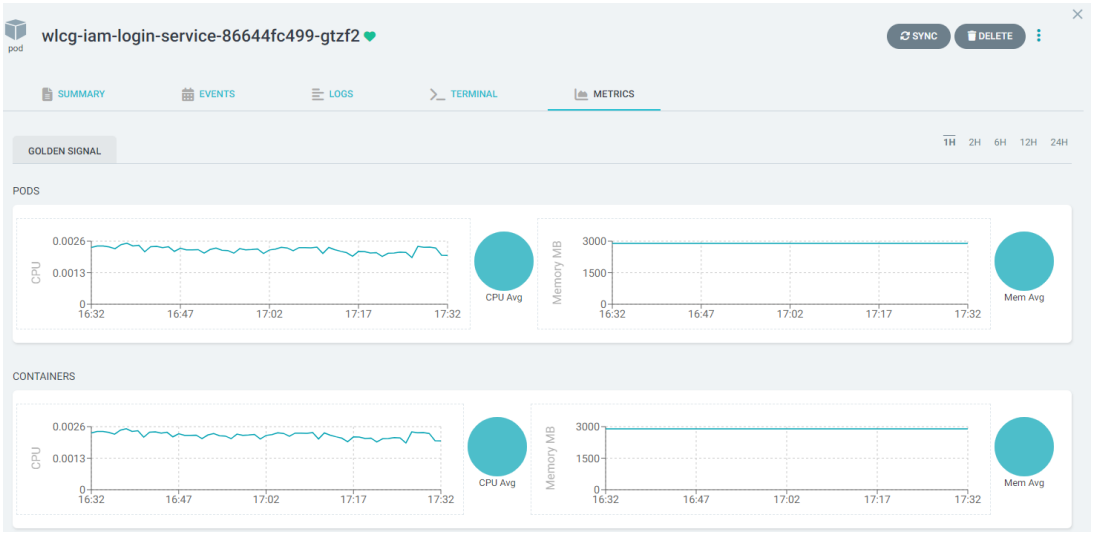

Figure 5: Screenshot of the ArgoCD dashboard. Thanks to the integration between ArgoCD and Prometheus, it is possible to view the metrics of Kubernetes resources.

to deploy most of the software seen so far (Kyverno, ArgoCD, Ceph-FS), and `K9s`, a convenient Kubernetes textual UI.

Furthermore, the ETCD backup is saved on the bastion node: it periodically collects the data from all 3 server nodes, keeping the last 5. For greater preservation over time, these backups are taken from the node, and, thanks to the BackupPC [19] software, they are saved externally on tape via TSM.

Everything is monitored both at the VM level and at the cluster/application level, thanks to software such as Sensu [20] and InfluxDB [21] as regards the infrastructure side, Prometheus [22] and Grafana [23] for software side. Prometheus takes care of scraping the cluster metrics and exporting them to Grafana, which instead takes care of displaying them in a pleasant graphical format. There is a wealth of information that we can obtain from Prometheus, such as the consumption of computational resources (CPU and RAM), of the network (bandwidth and rate of packets), of storage (IOPS and throughput). This data, in addition to keeping the status of the cluster under control, can also be useful for debugging or optimizing an application, identifying, for instance, bottlenecks. In the screenshot of Figure 6, as an example, you can see the graphs and a generic dashboard related to the CPU and RAM consumption of the entire cluster in the last hour. Finally, monitoring is connected to an alarm system (Slack, Teams and emails), to allow the administrator to intervene promptly on the infrastructure.

5. Conclusions

INFN is actively participating in different projects concerning digitalization, innovation, digital and technical-scientific skills, research and technology transfer, most of them started from the Italian National Recovery and Resilience Plan (Piano Nazionale di Ripresa e Resilienza, NRRP) as part of the Next Generation EU (NGEU) programme.

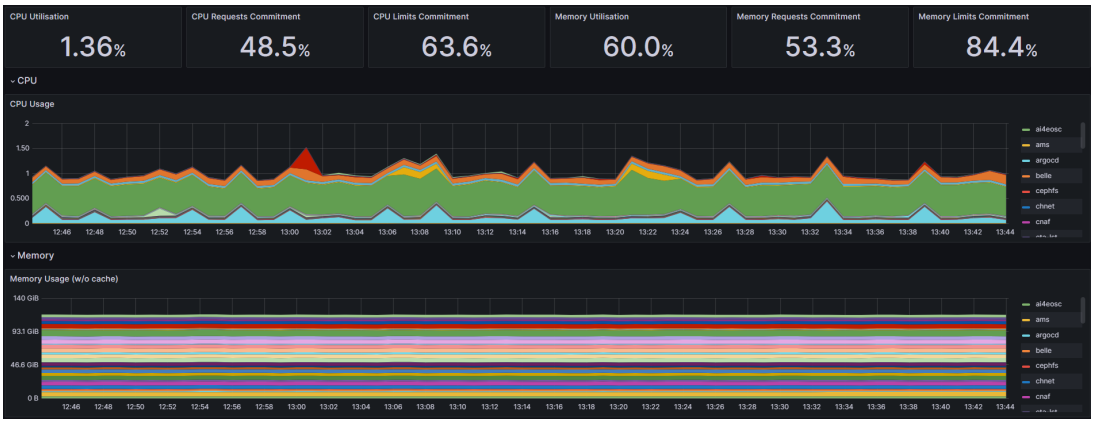

Figure 6: Screenshot of the Grafana dashboard. The image shows the CPU and RAM consumption of the entire cluster, grouped by namespace.

In particular, within the DARE project, INFN has had the mission to design and implement a Cloud-oriented platform hosting services that process and store partially-sensitive data produced by DARE. In this process, different aspects have been taken into account: data resiliency, service availability, resource elasticity and orchestration.

In the present work, we have presented recipes and best practices aimed at deploying and making operational a highly available and resilient Kubernetes cluster (master redundancy, presence of the LB and decoupling storage from applications). Different aspects such as the design of the infrastructure, the definition of the platform and the related services have been addressed and discussed. Other aspects and functionalities have been analyzed, such as the automatisms implemented in RKE2 and those implemented with Puppet, the CIS benchmark and the related security settings, the services and tools deployed as a support to the platform.

We are now implementing new improvements and features: sending the logs produced by the cluster to a BDP (Big Data Platform), using OpenSearch [24] as data collector; integrating Kubernetes APIs with OpenID Connect (INDIGO-IAM); and considering adopting backup solutions in the Ceph cluster.

The platform has been designed to work with particular data types, such as those of the DARE project, but it is flexible enough to host services and data from other areas or collaborations.

6. Acknowledgement

This research was co-funded by 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, code PNC0000002, CUP: B53C22006480001.

References

[1] *INFN home page*
https://home.infn.it/en/ - last seen Sep 2024

[2] *DARE project*
https://www.fondazionedare.it/en/dare-digital-lifelong-prevention/ - last seen Sep 2024

[3] *Kubernetes guide*
https://kubernetes.io/docs/home/ - last seen Sep 2024

[4] *OpenStack*
https://www.openstack.org/ - last seen Sep 2024

[5] *ETCD*
https://etcd.io/ - last seen Sep 2024

[6] *RKE2*
https://docs.rke2.io/ - last seen Sep 2024

[7] *RKE1 vs RKE2*
https://ranchermanager.docs.rancher.com/how-to-guides/ - last seen Sep 2024

[8] *Calico*
https://www.tigera.io/project-calico/ - last seen Sep 2024

[9] *Flannel*
https://github.com/flannel-io/flannel - last seen Sep 2024

[10] *Helm*
https://helm.sh/ - last seen Sep 2024

[11] *Nginx*
https://nginx.org/ - last seen Sep 2024

[12] *CIS hardening guide*
https://docs.rke2.io/security/hardening_guide - last seen Sep 2024

[13] *Puppet*
https://www.puppet.com/ - last seen Sep 2024

[14] *Octavia module*
https://docs.openstack.org/octavia/2024.1/ - last seen Sep 2024

[15] *Harbor*
https://goharbor.io/ - last seen Sep 2024

[16] *Ceph-FS*
https://docs.ceph.com/en/latest/cephfs/ - last seen Sep 2024

[17] *ArgoCD*
https://argo-cd.readthedocs.io/en/stable/ - last seen Sep 2024

[18] *INDIGO-IAM*
https://indigo-iam.github.io/v/current/ - last seen Sep 2024

[19] *BackupPC*
https://backuppc.github.io/backuppc/ - last seen Sep 2024

[20] *Sensu*
https://docs.sensu.io/sensu-go/latest/ - last seen Sep 2024

[21] *InfluxDB*
https://www.influxdata.com/ - last seen Sep 2024

[22] *Prometheus*
https://prometheus.io/ - last seen Sep 2024

[23] *Grafana*
https://grafana.com/ - last seen Sep 2024

[24] *Opensearch*
https://opensearch.org/ - last seen Sep 2024