PROCEEDINGS OF SCIENCE

Automating Distributed Cloud Infrastructures for AI-Driven Earth Observation Services

Marica Antonacci,[a,b,*] **Mohanad Albughdadi,**[a] **Vasileios Baousis,**[a] **Federico Fornari,**[a] **Tolga Kaprol**[a] **and Claudio Pisa**[a]

[a] *ECMWF,*
Shinfield Park, Reading RG2 9AX, United Kingdom
[b] *INFN Sezione di Bari,*
Via Orabona 4, 70125 Bari, Italy
E-mail: marica.antonacci@ecmwf.int, mohanad.albughdadi@ecmwf.int, vasileios.baousis@ecmwf.int, federico.fornari@ecmwf.int, tolga.kaprol@ecmwf.int, claudio.pisa@ecmwf.int

European projects such as EO4EU [1], MEDITWIN [2], CLIMRES [3] and BUILDSPACE [4], in which the European Centre for Medium-Range Weather Forecasts (ECMWF) is actively involved, develop AI-driven Earth Observation (EO) services that require multiple Kubernetes clusters to be provisioned and operated consistently across geographically distributed, heterogeneous cloud environments, under strict security and reproducibility constraints, while efficiently serving GPU-accelerated AI/ML workloads.

This paper presents the automation strategy adopted to meet these requirements. In production, cluster provisioning and lifecycle management rely on a Git-based workflow using Terraform [8], Ansible [9], and GitLab CI/CD. Operational experience with this model has revealed two structural limitations: duplication of infrastructure code across clusters and the lack of continuous reconciliation between desired and actual state. To address them, we are validating a Kubernetes-native architecture in proof-of-concept (PoC) environments. The architecture combines Cluster API [12], Crossplane [7] and Sveltos [10] to represent infrastructure, cluster lifecycle, and platform add-ons are expressed as declarative Kubernetes resources and continuously reconciled by controllers.

The PoC is designed to derive a complete cluster deployment, including GPU-related capabilities, from a high-level specification. This approach is intended to reduce duplication of code and configuration drift while preserving portability across cloud providers. We discuss the design, preliminary deployment experience and implications of a transition from pipeline-driven to controller-driven operations for the production platform.

International Symposium on Grids and Clouds (ISGC2026)
15-20, March, 2026
Academia Sinica, Taipei, Taiwan

*Speaker

© Copyright owned by the author(s) under the terms of the Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License (CC BY-NC-ND 4.0) All rights for text and data mining, AI training, and similar technologies for commercial purposes, are reserved. ISSN 1824-8039 . Published by SISSA Medialab.

PoS(ISGC2026)010

1. Introduction

The European Centre for Medium-Range Weather Forecasts (ECMWF) is an intergovernmental organisation supported by 35 European states, with operational sites in the United Kingdom, Italy, and Germany. Its mission is to develop and operate global numerical weather prediction systems, advance Earth system science, and maintain one of the world's largest meteorological archives, currently exceeding one exabyte of observational, model, and reanalysis data. Alongside its four-times-daily global forecasts, ECMWF implements the Copernicus Atmosphere Monitoring Service (CAMS) and the Copernicus Climate Change Service (C3S).

These activities rely on state-of-the-art high-performance computing (HPC) facilities, complemented by cloud-based infrastructures such as the European Weather Cloud [6] and the Copernicus Data Stores, which provide researchers, public institutions, and industry with scalable access to data, services, and computing resources.

Externally funded research and innovation projects extend these capabilities and act as technological accelerators, bridging scientific development and operational deployment. Together with ECMWF's involvement in Destination Earth (DestinE) [5], they have driven concrete advances in multi-cloud orchestration, AI/ML service deployment, and interoperable data workflows. Representative examples include EO4EU [1], an AI-enhanced platform for discovering and processing Earth Observation data from a wide range of heterogeneous sources; BUILDSPACE [4] and CLIMRES [3], which combine satellite observations, climate datasets, and AI-based analytics to support resilience and energy-efficiency assessment of the built environment; and MEDITWIN [2], which develops an AI-driven digital twin of the Mediterranean for the prediction and impact assessment of climate extremes.

The common goal of these initiatives is to deliver user-oriented, cross-domain services that democratise access to Earth Observation data and enable advanced analytics and decision support for expert and non-expert users alike. This places demanding requirements on the underlying infrastructure: it must handle very large data volumes, support heterogeneous and data-intensive workflows, orchestrate services across geographically distributed multi-cloud sites, interoperate with heterogeneous OpenStack-based deployments, satisfy strict security, reproducibility, and governance constraints, and efficiently execute GPU-intensive workloads for machine learning applications.

To address these challenges, our infrastructure strategy is centred on automation, reproducibility, scalability, and portability, enabling consistent deployment, operation, and evolution of services across diverse and distributed computing environments.

2. Related Work

Kubernetes has become a common building block of research computing infrastructures, while cluster and infrastructure lifecycle management have evolved along different paths. A first family of approaches provides Kubernetes as a managed service on OpenStack. CERN has operated several hundred user-facing clusters through OpenStack Magnum [20], whose Kubernetes support is transitioning from the original Heat-based implementation towards Cluster API (CAPI) drivers [21]. Similarly, the Azimuth portal [22], initially developed for the JASMIN Cloud and subsequently

PoS(ISGC2026)010

adopted by the IRIS research cloud in the United Kingdom, provides a self-service interface for provisioning and managing Kubernetes clusters, with Cluster API used for cluster lifecycle management. These approaches demonstrate the maturity of CAPI for declarative Kubernetes cluster lifecycle management, but primarily address the cluster layer rather than providing a unified abstraction across cloud infrastructure, cluster lifecycle, and platform services.

A second family relies on orchestration frameworks external to Kubernetes. INFN Cloud [23], for example, uses the TOSCA-based INDIGO PaaS Orchestrator together with Ansible and Helm to provision Kubernetes clusters across federated sites. Similar to the Terraform-based workflow described in Section 3, these approaches treat infrastructure changes primarily as provisioning operations: the orchestrator executes the requested deployment, but the resulting infrastructure is not continuously reconciled through a Kubernetes-native control loop. This distinction is particularly relevant for distributed infrastructures, where drift detection, recovery, and dependency management become important operational requirements.

A complementary line of work explores Crossplane as a Kubernetes-native control plane for infrastructure management. D'Agati et al. [25] use Crossplane with OpenStack to manage compute, network, and IoT resources in the SLICES cloud-continuum infrastructure, demonstrating its applicability to distributed research-oriented environments. The SIMPL initiative [24] adopts Crossplane for infrastructure provisioning across multiple cloud providers as part of an open-source middleware platform for cloud-to-edge federations and European data spaces. Beyond these infrastructure-specific applications, Kumar and Thanga Selvi [26] investigate Crossplane for declarative multi-cloud infrastructure management, highlighting continuous reconciliation, drift correction, automated recovery, and dependency management. These works establish Crossplane as a suitable foundation for Kubernetes-native infrastructure management, but do not address the coordinated lifecycle of the underlying infrastructure, Kubernetes clusters, and platform services as a single higher-level resource.

Our work combines these complementary capabilities into a unified declarative lifecycle model for distributed research infrastructures. Crossplane manages the underlying cloud infrastructure, Cluster API manages Kubernetes cluster lifecycle on OpenStack, and Sveltos manages platform add-ons. In the architecture described in Section 4.1, these components are exposed through a high-level *XCluster* abstraction whose specification drives the provisioning and reconciliation of infrastructure, cluster topology, and platform services, including GPU enablement. Rather than treating these layers as independent provisioning steps, the proposed approach composes their controllers into a single Kubernetes-native control plane, extending declarative reconciliation from individual infrastructure resources and cluster objects to the lifecycle of a complete distributed Kubernetes environment. To the best of our knowledge, this combination has not previously been documented for research infrastructures supporting Earth Observation and AI workloads.

3. Git-Based Infrastructure Automation and Operational Model

Given these infrastructure challenges, establishing a robust and reproducible operating model became essential to ensure consistency, scalability, and traceability across distributed environments.

To address this need, we adopted a Git-based workflow (see Figure 1) as the foundational operational paradigm. In this model, Git acts as the single source of truth for all infrastructure and

PoS(ISGC2026)010

configuration definitions. Every change — whether related to infrastructure provisioning, platform configuration, or application deployment — is introduced through a Git commit, ensuring full traceability, auditability, and reproducibility of system states over time.

In current cloud-native practice, GitOps has emerged as a reference model for operating platforms because it naturally combines version-controlled declarative configuration, automated and auditable delivery pipelines, and policy-driven enforcement mechanisms. In parallel, established Infrastructure as Code (IaC) tools such as Terraform, together with configuration management frameworks such as Ansible, continue to play a central role in production environments, particularly in multi-cloud scenarios that require fine-grained provisioning and post-deployment configuration management.

Within this operational model, CI/CD pipelines provide the execution backbone. We rely on GitLab CI/CD to trigger deployment workflows automatically upon changes to Git repositories. This automation reduces manual intervention, minimises configuration drift, and ensures consistency across development, staging, and production environments. From an implementation perspective, Infrastructure as Code is realised through a combination of Terraform and Ansible. Terraform is responsible for provisioning the underlying infrastructure resources, including OpenStack virtual machines, networking components, floating IPs, security groups, and the initial setup of RKE2 Kubernetes clusters. Ansible complements this layer by performing post-provisioning configuration, including the deployment and configuration of Rancher [13], which serves as a central management plane for orchestrating multiple Kubernetes clusters and higher-level platform services. GitLab is additionally used as the remote backend for Terraform state management, ensuring centralized state storage, locking, and encryption.

A key aspect of the architecture is the integration of secrets management into the provisioning workflow. Terraform interacts securely with HashiCorp Vault [15] to retrieve and manage sensitive credentials, including OpenStack application credentials, AWS Route53 DNS credentials, SSH keys, and other environment-specific secrets. This integration ensures that sensitive information is never hardcoded in repositories and is accessed in a controlled and auditable manner during runtime.

We organize the code structure into multiple repositories following a clear separation of concerns. A dedicated repository is used for the Terraform code responsible for deploying the Rancher management cluster, which acts as the central control plane. In addition, each downstream RKE2 cluster is managed through its own dedicated repository, following the same Git-based workflow. For downstream clusters, CI/CD pipelines are triggered through structured commit messages, which encode the intended operation (e.g., create, update, or destroy). Based on these triggers, the pipeline automatically executes the appropriate Terraform actions (apply or destroy), enabling fully automated cluster lifecycle management. This mechanism supports iterative operations; for instance, an initial commit may provision a new cluster, while subsequent commits can modify configuration layers such as networking policies, add-ons, or platform extensions.

3.1 Motivation for a Kubernetes-Native Transition

This Terraform-centric, Git-based workflow has proven effective in supporting multi-cluster infrastructure provisioning and management at scale. However, several limitations have emerged as the system has evolved.

PoS(ISGC2026)010

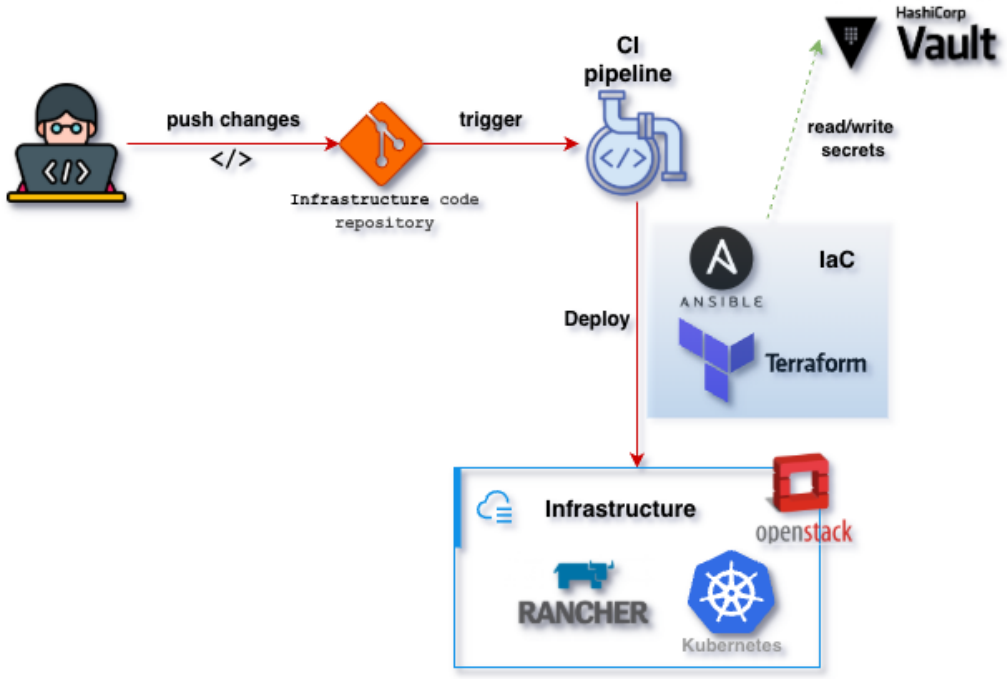

Figure 1: Git-based infrastructure automation workflow.

A first limitation is code duplication across repositories. Since each cluster is managed in a separate repository, common infrastructure patterns and modules are frequently replicated. This increases maintenance overhead and introduces the risk of configuration divergence over time. For example, applying a fix or enhancement—such as a correction in ingress configuration or a networking policy update—often requires manual propagation across multiple repositories, reducing operational efficiency and increasing the likelihood of inconsistencies.

A second limitation concerns the lack of a native reconciliation model in Terraform. Unlike Kubernetes-native controllers, Terraform operates primarily as a declarative provisioning tool without a continuous reconciliation loop. As a result, it does not continuously enforce convergence between desired and actual state once a deployment has been completed, limiting its ability to self-heal or dynamically adapt to external changes.

These limitations motivated the exploration of a more scalable, maintainable, and Kubernetes-native operational model, capable of reducing duplication, improving lifecycle consistency, and aligning more closely with continuous reconciliation principles inherent to modern cloud-native architectures.

These observations are also reflected in the scale of the current production environment. The team currently operates approximately a dozen of RKE2 clusters across different OpenStack-based cloud environments: ECMWF European Weather Cloud installations, WEkEO/CloudFerro, and the CINECA AdaCloud infrastructure. The production footprint also includes five NVIDIA A100 GPU nodes at ECMWF, supporting both full-GPU and MIG configurations. The number of managed clusters is expected to grow further as infrastructure requirements evolve across ongoing and new projects.

PoS(ISGC2026)010

Preliminary deployment tests, not intended as a controlled performance benchmark, suggest that provisioning times are broadly comparable to those of the existing Terraform-based approach. At the same time, the Kubernetes-native model introduces reusable declarative abstractions, continuous reconciliation, and automatic recovery from transient failures, providing a more streamlined operational model as the number of managed clusters grows.

4. Kubernetes-Native GitOps Architecture for Declarative Multi-Cluster Management

To overcome the limitations of the previous Terraform- and repository-centric approach, we are designing and validating a Kubernetes-native architecture in experimental PoC environments. The architecture uses declarative control loops and shared unified abstractions to manage infrastructure and cluster lifecycle through continuous reconciliation.

At the core of this candidate architecture lies a Rancher-based management cluster, which acts as the central control plane for multi-cluster operations. To ensure consistency and continuous alignment with the desired state, we adopt Fleet, Rancher's native GitOps engine [14]. Fleet continuously monitors Git repositories and automatically reconciles configuration changes, ensuring that platform components, controllers, operators, and add-ons are consistently and reliably deployed on the management cluster.

From a state-of-the-art perspective, the cloud-native ecosystem is increasingly converging towards controller-based and reconciliation-driven control planes. In this context, Cluster API (CAPI) has become a de facto standard for declarative Kubernetes lifecycle management, enabling programmatic provisioning, configuration, and operation of Kubernetes clusters. In parallel, Crossplane extends this paradigm beyond cluster management, providing a universal control plane for infrastructure resources and enabling higher-level abstractions that unify infrastructure provisioning and platform composition. Complementing these, add-on orchestration frameworks such as Sveltos address the operational complexity of consistently deploying and managing applications, policies, and configurations across large and heterogeneous cluster fleets.

Within this ecosystem, three core components define the orchestration layer:

- **Crossplane**, used as a universal control plane to provide a unified abstraction layer where infrastructure resources and higher-level platform definitions are expressed as declarative Kubernetes custom resources;
- **Cluster API**, responsible for declarative Kubernetes cluster lifecycle management, including provisioning, configuration, upgrades, and deletion of clusters in a cloud-agnostic manner;
- **Sveltos**, a Kubernetes-native add-on controller that enables consistent deployment and lifecycle management of platform services, applications, and policies across multiple clusters.

Argo CD [11] was considered as an alternative for add-ons management. The choice of Sveltos was based on an evaluation of the features and configuration models of both solutions against the requirements of our architecture. Sveltos provides native integration with Cluster API, allowing newly created clusters to be automatically discovered and registered for add-on management. Its

PoS(ISGC2026)010

label-based cluster selection also provides a natural mechanism for targeting add-ons to clusters based on their characteristics. Since Fleet already provides Git synchronisation on the management cluster, using Sveltos also avoids introducing a second GitOps engine specifically for add-on delivery.

The resulting workflow is implemented today in experimental PoC deployments as an end-to-end GitOps-driven (see Figure 2), declarative, and continuously reconciled model. This means that, in the PoC environment, the system no longer depends on external imperative pipelines; instead, it continuously converges toward the desired state through Kubernetes-native control loops, improving resilience, consistency, and operational simplicity.

A key architectural shift in this model is the relocation of the control plane inside Kubernetes itself, enabled by Crossplane. Instead of relying on external provisioning pipelines, infrastructure and cluster definitions are expressed as Kubernetes-native resources.

4.1 The XCluster Abstraction

We introduce a high-level custom resource abstraction called *XCluster*, which represents the desired state of a complete cluster deployment. This abstraction encapsulates cluster topology, infrastructure parameters (including networking, load balancers, virtual IPs, routers, and other cloud-specific resources), as well as the set of platform add-ons to be deployed.

Crossplane, in combination with Cluster API, translates these high-level declarative specifications into concrete infrastructure and Kubernetes clusters. In parallel, Sveltos ensures that the required add-ons, policies, and platform services are consistently applied across the resulting cluster environments (Figure 3).

To illustrate the XCluster abstraction, Figure 4 shows a simplified example of a declarative cluster specification used in our platform.

This example highlights the declarative nature of the XCluster abstraction. The user specifies high-level parameters such as the number and type of nodes (including GPU-enabled workers), the Kubernetes distribution version, and the set of platform add-ons to be enabled.

From this compact specification, the control plane automatically derives the full infrastructure and platform configuration. Crossplane and Cluster API handle infrastructure provisioning and cluster lifecycle management, while Sveltos ensures that the selected add-ons are consistently deployed across the clusters.

4.2 Reusable Cluster Blueprints and Topology Variants

To further enhance modularity and reuse, we leverage Cluster API *ClusterClasses* to define reusable cluster blueprints (Figure 5). These templates encapsulate infrastructure and bootstrap logic, enabling a clear separation between cluster intent and implementation details. This approach allows clusters to be instantiated from standardised building blocks while still supporting customisation through declarative configuration.

This modular design enables flexible adaptation of cluster topologies without modifying the underlying core definitions. For instance, the same base cluster definition can support different ingress strategies simply by selecting an alternative ClusterClass configuration.

We currently support two ingress topologies: a standard architecture based on OpenStack Octavia [16] load balancing, and an alternative topology designed for environments where Octavia

PoS(ISGC2026)010

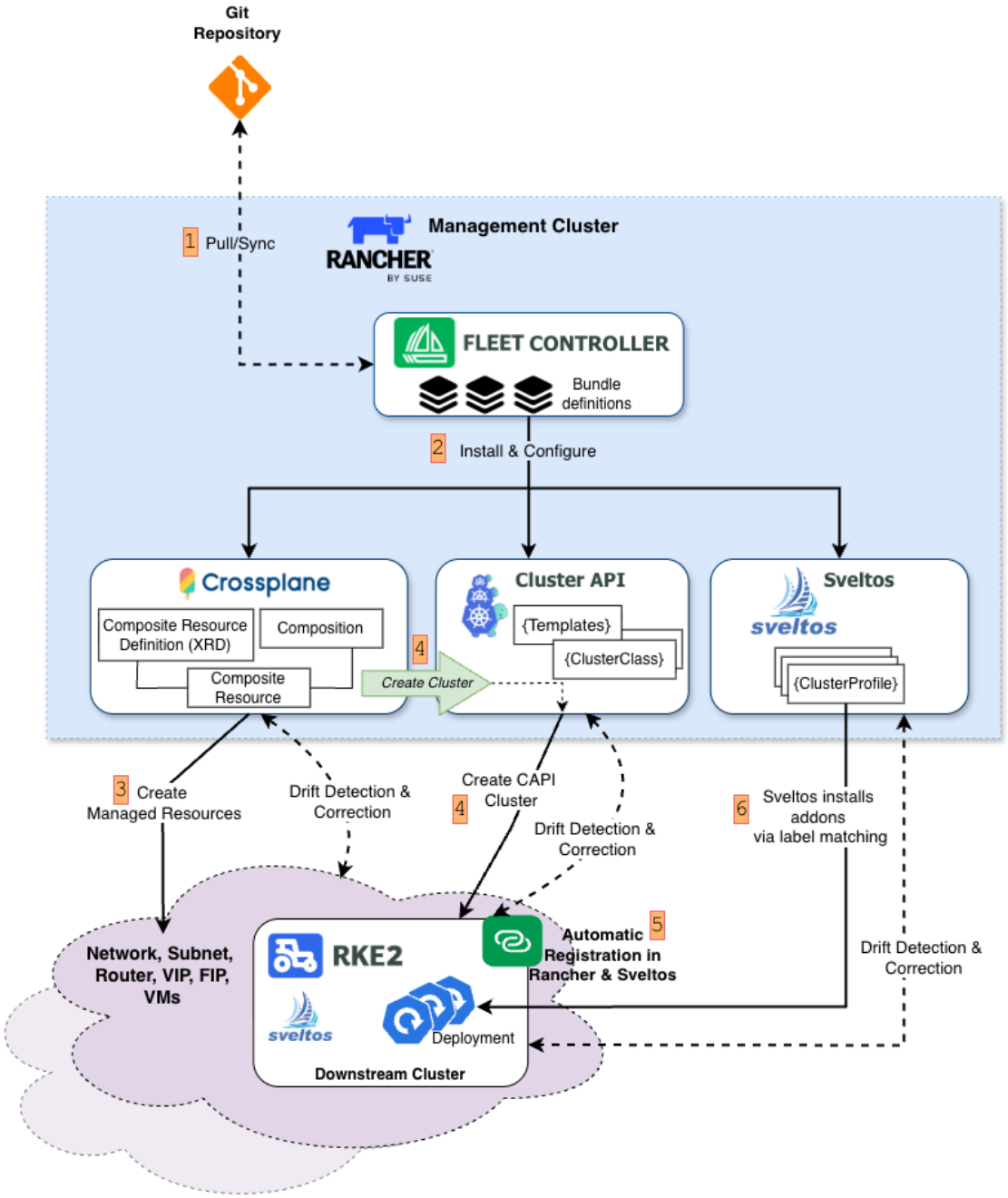

Figure 2: Kubernetes-native declarative multi-cluster workflow.

is not available or has shown instability in practice. In the latter case, dedicated ingress nodes are provisioned and automatically configured for high availability using Keepalived [17].

In practice, even at sites where Octavia is available, we have frequently used the ingress-node topology because Octavia load balancers have shown instability during infrastructure events and slow recovery in some cases. The ingress-node topology instead provides service exposure through dedicated nodes, with Keepalived managing the virtual IP and the ingress controller serving traffic

PoS(ISGC2026)010

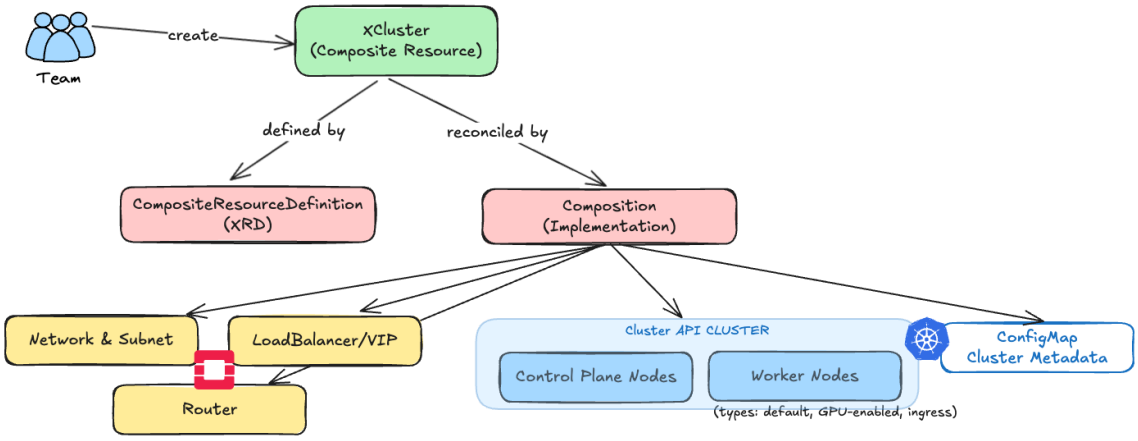

Figure 3: Crossplane resource model and XCluster abstraction.

```
apiVersion: crossplane.ecmwf.int/v1alpha1
kind: XCluster
metadata:
  name: cluster-a
spec:
  parameters:
    numberOfMasters: 3
    numberOfWorkers: 3
    numberOfGpuWorkers: 3
    rke2Version: v1.33.3+rke2r1
    subnetCidr: 10.0.50.0/24
    # other infrastructure parameters...
    addons:
      - nginx-ingress
      - external-dns
      - gpu-operator
      - grafana-operator
```

Figure 4: Example of XCluster declarative specification.

directly. Based on this experience, we currently prefer this option for production-facing services, while retaining Octavia as an alternative.

Crossplane is responsible for provisioning the underlying OpenStack networking resources, including virtual IPs, floating IPs, routers, and related networking components. When the Octavia-based topology is selected, services are exposed through managed load balancers; when it is not available, traffic is routed through the dedicated ingress node layer.

Sveltos dynamically adapts the deployment of ingress components depending on the selected topology, configuring Kubernetes services either in LoadBalancer mode or via host networking on dedicated ingress nodes.

PoS(ISGC2026)010

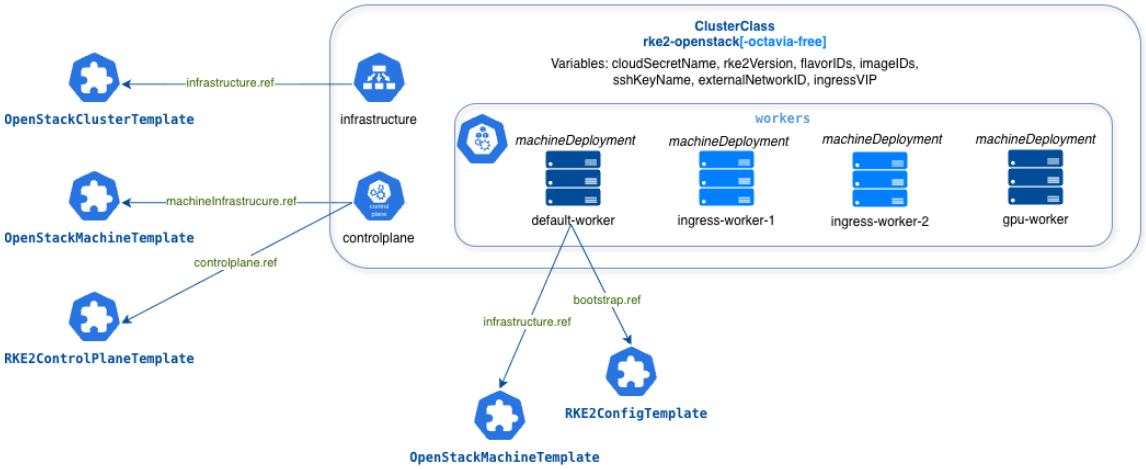

Figure 5: Cluster API *ClusterClass*-based reusable cluster blueprint.

This design enables consistent service exposure semantics across heterogeneous environments while preserving flexibility in underlying infrastructure implementations.

The same modular and declarative approach is extended to compute resources, enabling support for specialized node types such as GPU-enabled instances. These are managed through dedicated machine deployment classes, which define specific images, flavors, and scheduling constraints.

Overall, this architecture maximizes reuse and consistency across heterogeneous infrastructure topologies, while preserving a fully declarative, GitOps-driven operational model that unifies cluster lifecycle, infrastructure provisioning, and application delivery.

The transition to a controller-based model is also motivated by failure modes observed in the existing Terraform-based workflow. Interrupted or partially failed pipeline executions have occasionally resulted in inconsistencies between Terraform state and the actual infrastructure resources, requiring manual state recovery and re-execution of the affected pipelines. Similarly, transient failures in cloud services can leave a deployment in an intermediate state until an operator intervenes. The controller-based model handles many of these situations differently: reconciliation continuously compares the declared and observed state and retries failed operations, so transient failures can recover without re-running a pipeline. However, continuous reconciliation introduces failure modes of its own. For example, unavailable dependencies, exhausted quotas, or missing images may prevent an *XCluster* from reaching a ready state indefinitely, requiring explicit monitoring of controller conditions to distinguish temporary failures from persistent problems. Automated drift correction also means that manual changes to resources managed by Crossplane or Sveltos may be reverted to the declared state. We therefore follow a Git-first operational model, treating manual interventions as break-glass procedures and ensuring unambiguous ownership of managed resources to avoid conflicting reconciliation loops.

5. GPU-Enabled AI Workloads in a Kubernetes-Native Platform

Building on the Kubernetes-native and GitOps-driven architecture introduced in the previous section, we extend the platform to support AI and machine learning workloads at scale, including

PoS(ISGC2026)010

transparent access to GPU resources across heterogeneous clusters. Rather than being implemented as an external extension, this capability is treated as a first-class feature of the same declarative control plane, where infrastructure, cluster lifecycle, and workload enablement are expressed uniformly and continuously reconciled.

A key enabler of this end-to-end model is the XCluster abstraction introduced earlier, which defines the desired state of a complete platform deployment, including not only cluster topology and networking configuration, but also the set of enabled capabilities such as add-ons and hardware-specific features. Within this model, GPU support is expressed as a declarative requirement at the platform level, rather than as a post-provisioning configuration step.

In practice, GPU enablement is propagated through the same control chain: Crossplane and Cluster API materialise the infrastructure and cluster definitions described in the XCluster resource, while Sveltos continuously enforces the desired state of platform add-ons, including GPU-related components. This ensures that GPU readiness becomes an inherent property of a cluster instance, fully aligned with the GitOps-driven reconciliation loop that governs the entire system.

In current cloud-native practice, production AI platforms increasingly rely on Kubernetes-native GPU lifecycle management rather than ad hoc node configuration. The **NVIDIA GPU Operator**[18] has therefore become a de facto standard solution, as it encapsulates best practices for driver management, container runtime configuration, and device discovery while improving portability across heterogeneous infrastructures.

Within the experimental architecture, the NVIDIA GPU Operator is deployed automatically through Sveltos-based GitOps policies. Clusters or cluster profiles derived from an XCluster specification can be labelled (e.g., addons.gpu-operator: enabled), triggering the automatic installation and configuration of the operator. This mechanism ensures that GPU capabilities are consistently provisioned across all eligible clusters without manual intervention, fully aligning with the declarative lifecycle defined at the platform level.

Once deployed, the GPU Operator manages the complete GPU software stack. It performs node-level GPU discovery, identifies hardware characteristics (including model type, memory size, and compute capabilities), and configures the container runtime to use the NVIDIA runtime. Depending on the selected configuration, it can also automate driver installation, CUDA toolkit provisioning, and lifecycle upgrades, thereby abstracting one of the most complex operational domains in AI infrastructure management.

As a concrete example of this integrated workflow, we deploy JupyterHub [19] through a GitOps-managed Helm release. Users interact with a unified environment that exposes multiple server profiles, ranging from CPU-only notebooks to GPU-enabled environments preconfigured with frameworks such as PyTorch and CUDA. GPU allocation is handled natively through Kubernetes resource requests, enabling seamless scheduling and isolation across shared multi-tenant infrastructure.

Overall, GPU enablement is fully integrated into the same end-to-end declarative pipeline that governs cluster provisioning, infrastructure orchestration, and application deployment. Through the combination of XCluster abstractions, Crossplane-managed infrastructure, Cluster API lifecycle control, and Sveltos-based add-on enforcement, GPU computing becomes a native and composable capability of the platform, rather than an externally managed extension.

PoS(ISGC2026)010

6. Conclusions

This work presents a Kubernetes-native approach for progressively transitioning from pipeline-driven infrastructure automation to a controller-based operational model for distributed cloud environments supporting Earth Observation and AI workloads.

In traditional Terraform-centric workflows, infrastructure is provisioned through CI/CD pipelines that apply changes in discrete execution steps. While this approach provides explicit control and reproducibility, it lacks continuous reconciliation between the desired and actual system state once deployment has been completed.

In contrast, the proposed architecture adopts Kubernetes as the central control plane, where infrastructure, cluster lifecycle, and application configuration are expressed declaratively and continuously reconciled by dedicated controllers. This shift enables a unified operational model in which system state is continuously aligned with its specification, reducing configuration drift and improving operational consistency across distributed environments.

The presented PoC provides a foundation for the progressive evolution of the production platform into a fully GitOps-driven ecosystem where infrastructure provisioning, cluster management, and workload deployment are governed through a single declarative paradigm. This model improves consistency and provides a basis for greater scalability, portability, and resilience in multi-cluster and multi-cloud scenarios.

Beyond the architectural transition itself, this work highlights the convergence of modern cloud-native control-plane technologies, including Cluster API, Crossplane, and Kubernetes-native GitOps engines, which together enable a composable and extensible infrastructure model suitable for heterogeneous environments.

Looking forward, this architecture provides a foundation for the progressive introduction of AI-assisted operations. However, fully autonomous or agentic infrastructure control introduces important risks related to predictability, safety, and unintended system behaviour.

To address this, we advocate a human-in-the-loop operational model, where large language models are integrated through constrained and auditable tool interfaces (e.g., MCP-based systems). In this setup, AI systems act as bounded assistants for tasks such as debugging, diagnosis, and operational analysis, while final control remains explicitly under human supervision.

In parallel, the enrichment of platform observability is a key enabler for these capabilities, providing the telemetry and context required for reliable AI-assisted reasoning in operational environments.

Overall, this stepwise approach enables a safe evolution from declarative automation toward AI-augmented infrastructure management, preserving strong guarantees in terms of governance, reproducibility, and operational control while progressively enhancing system intelligence and usability.

Acknowledgments

This work has been funded by the European Union's Horizon Europe programme through the MeDiTwin project (Grant Agreement No. 101159723) and the CLIMRES project (Grant Agreement No. 101147777).

PoS(ISGC2026)010

References

[1] EO4EU Project. https://www.eo4eu.eu/

[2] MEDITWIN Project. https://meditwin-project.eu/

[3] CLIMRES Project. https://climres.eu/

[4] BUILDSPACE Project. https://www.buildspaceproject.eu/

[5] Destination Earth (DestinE). https://destination-earth.eu/

[6] ECMWF and EUMETSAT, The European Weather Cloud. https://europeanweather.cloud/

[7] Crossplane. https://crossplane.io/

[8] Terraform. https://www.terraform.io/

[9] Ansible. https://www.ansible.com/

[10] Sveltos. https://projectsveltos.io/

[11] Argo Project, *Argo CD: Declarative Continuous Delivery for Kubernetes*. https://argo-cd.readthedocs.io/

[12] Kubernetes Cluster API. https://cluster-api.sigs.k8s.io/

[13] Rancher. https://rancher.com/

[14] Fleet. https://fleet.rancher.io/

[15] HashiCorp Vault. https://www.vaultproject.io/

[16] OpenStack Octavia. https://docs.openstack.org/octavia/latest/

[17] Keepalived. https://www.keepalived.org/

[18] NVIDIA GPU Operator. https://github.com/NVIDIA/gpu-operator

[19] JupyterHub. https://jupyter.org/hub

[20] OpenStack Magnum. https://docs.openstack.org/magnum/latest/

[21] M. Pryor, *Using Cluster API to provide managed Kubernetes on OpenStack*, OpenInfra User Group meeting, CERN, June 2024. https://indico.cern.ch/event/1376907/contributions/5831719/

[22] Azimuth: HPC and AI platforms on demand. https://github.com/azimuth-cloud/azimuth

PoS(ISGC2026)010

[23] M. Antonacci and D. Salomoni, *Leveraging TOSCA orchestration to enable fully automated cloud-based research environments on federated heterogeneous e-infrastructures*, PoS(ISGC&HEPiX2023)020 (2023). https://doi.org/10.22323/1.434.0020

[24] European Commission, *Simpl-Open: Mechanisms to configure and provision infrastructure resources*, Simpl Programme, 2025. https://simpl-programme.ec.europa.eu/book-page/83-mechanisms-configure-and-provision-infrastructure-resources

[25] L. D'Agati, G. Tricomi, F. Longo, G. Merlino, A. Bujari, P. Bellavista and A. Puliafito, *Building a Federated Cloud Continuum Infrastructure via Crossplane*, in Proc. iCities 2024, Messina, Italy (2024). https://icities24.unime.it/papers/52.pdf

[26] N. Kumar and R. Thanga Selvi, *Crossplane: A Unified Framework for Multi-Cloud Infrastructure Management*, in Proc. 6th Int. Conf. on Intelligent Communication Technologies and Virtual Mobile Networks (ICICV 2025), IEEE (2025), pp. 25–34. https://doi.org/10.1109/ICICV64824.2025.11085680

PoS(ISGC2026)010