Hypertool: A Resource Abstraction and Lifecycle Management Framework for Heterogeneous Cloud Environments

Michalis Loukeris[1], Nektarios Deligiannakis[1], Vassilis Papataxiarhis[1], Panagiotis Kechriniotis[1], Syed Mafooq Ul Hassan[2], Nicolas Louca[3], Herodotos Herodotou[2], Stathes Hadjiefthymiades[1]
[1] Department of Informatics and Telecommunications
National and Kapodistrian University of Athens, 15784 Athens, Greece
e-mail: {mloukeris, nekdel, vpap, pkehri, shadj}@di.uoa.gr
[2] Department of Electrical Engineering, Computer Science and Engineering
Cyprus University of Technology, 3036 Limassol, Cyprus
e-mail: {syedmafooqul.shah, herodotos.herodotou}@cut.ac.cy
[3] eBOS Technologies Ltd
2043 Nicosia, Cyprus
e-mail: nicolasl@ebos.com.cy

***Abstract*—Orchestrating modern applications across distributed cloud environments requires a unified, platform-agnostic representation of heterogeneous computational resources and a tool to automate their management. This paper presents an open-source framework, Hypertool, designed to automate node registration, discovery, and lifecycle management. Additionally, it provides an extensible model that enhances the default node representation with additional attributes, such as hardware acceleration, energy efficiency, and security posture, which are vital for next-generation workloads. By implementing a dual Command Line Interface (CLI) and DaemonSet architecture, the framework acts as an all-in-one utility for orchestrating complex functionalities, abstracting computing, networking, and economic metrics into declarative data models. The current implementation leverages core Kubernetes primitives, ensuring effortless integration into existing heterogeneous cloud environments. The framework is evaluated across local, on-premises, and public cloud environments, demonstrating high operational maturity, stability, and production-readiness.**

Keywords-Resource Abstraction; Distributed Cloud Environments; Kubernetes; Node Lifecycle Management; Heterogeneous Computing; Declarative Data Models; Orchestration.

I. INTRODUCTION

Orchestrating modern workloads across distributed cloud environments presents a twofold challenge. First, advanced cloud-edge applications, particularly those involving IoT, AI, and latency-sensitive tasks, require a more sophisticated representation of computational resources than the defaults provided by standard orchestration platforms such as Kubernetes [1][2]. To facilitate intelligent workload placement, systems require an extended node schema that captures dynamic attributes, such as energy efficiency, hardware acceleration, and security posture, across the cloud-to-edge continuum [3][4]. Second, the lifecycle management of these distributed resources remains a significant hurdle. As heterogeneous nodes are added to or removed from a cluster, existing manual procedures are often complex, multi-step, and error-prone [5].

To address these limitations, this paper introduces **Hypertool**, an open-source framework that automates the registration, discovery, and lifecycle management of nodes in heterogeneous cloud architectures. Hypertool provides a unified, platform-agnostic resource abstraction layer that simplifies the onboarding of diverse assets while alleviating vendor lock-in. It is also engineered for seamless deployment on any Kubernetes-compatible infrastructure, ensuring interoperability across local, on-premises, and public cloud environments.

The framework employs a dual-component architecture consisting of a cross-platform **CLI** and a persistent **DaemonSet**[6]. Within the Kubernetes ecosystem, a DaemonSet is a controller that ensures a copy of a specified pod runs on every node in the cluster, providing a resilient mechanism for background services. The CLI offers an intuitive interface for administrators to handle node onboarding, including a "dry run" mode to simulate operations before execution. Simultaneously, the daemon service runs on every node to continuously monitor health, recalculate resource attributes, and update the Kubernetes control plane via declarative Custom Resource Definitions (CRDs) [7].

Hypertool is built with a secure-by-design philosophy and adheres to industry best practices for authentication and authorization. To ensure high operational maturity, the project maintains comprehensive Continuous Integration/Continuous Deployment (CI/CD) pipelines for continuous testing and delivery, supported by extensive documentation and deployment guides.

The primary contributions of this work are as follows:

- **Resource Abstraction Layer**: A declarative data model that abstracts resources across the continuum, continuously calculating dynamic attributes to provide an enhanced representation of heterogeneous computational power.
- **Automated Lifecycle Management**: A robust mechanism to manage the complete lifecycle of distributed resources, significantly reducing the complexity of node registration and cluster integration.
- **Production-Ready Framework**: An open-source, secure-by-design utility featuring a dual CLI/DaemonSet architecture, validated through automated CI/CD

pipelines and evaluated across diverse cloud environments for stability and performance.

The remainder of this paper is organized as follows: Section II reviews related work. Section III details the Hypertool framework architecture. Section IV presents the system's validation across diverse environments, and Section V concludes the paper.

II. Related Work

Existing operational frameworks and modern orchestration platforms, such as Kubernetes [1], OpenStack [8], and HashiCorp Nomad [9], serve as the industry standard for deploying distributed applications. However, while these systems provide a generic model for representing compute resources, their default models are fundamentally basic, typically limited to static allocations of CPU cores, memory, and storage capacity. Although extensibility mechanisms exist to enhance these representations (such as Custom Resource Definitions in Kubernetes or custom metadata flavors in OpenStack), the baseline schema remains too simplistic [7][10]. It cannot meet the multidimensional needs of modern applications, which increasingly require real-time awareness of complex attributes such as energy efficiency, specialized hardware acceleration, and network performance metrics to make optimal scheduling decisions.

Furthermore, current procedures for managing node lifecycles within these platforms inherently assume that the underlying infrastructure is highly stable and static. When heterogeneous resources need to be dynamically added to or removed from a cluster, the default lifecycle management processes prove inadequate [2][5]. The built-in mechanisms for registering, updating, or deleting nodes typically involve multiple disjoint, intricate steps. Consequently, these operations are highly complex, error-prone, and demand significant domain expertise, often requiring a certified cluster administrator to manually execute and verify the changes safely. This rigidity prevents automated scaling in modern distributed cloud architectures, underscoring the critical need for a more dynamic, automated approach to resource abstraction and lifecycle governance.

III. Hypertool Framework Architecture

A. Hypertool as a CLI Tool

To effectively resolve the twofold challenge of providing a sophisticated resource representation and automating dynamic node lifecycle management, the framework introduces a dual-component architecture comprising a Command Line Interface (CLI) utility and a management DaemonSet, as depicted in Figure 1. This structural separation of concerns ensures that heterogeneous computing resources are first thoroughly profiled and abstracted into declarative data models prior to cluster inclusion, and subsequently continuously monitored, updated, and managed during their operational lifecycle.

The CLI utility serves as the primary interface for cluster administrators to govern the end-to-end lifecycle of computational nodes, as illustrated in Figure 2. It is specifically designed to manage the critical phases of resource onboarding and offboarding, including initial discovery, self-advertisement, node registration, and unregistration.

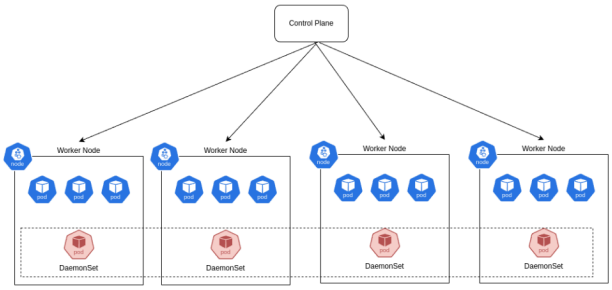

Figure 1. The dual-component architecture of the Hypertool framework, showing the interaction between the CLI profiler and the management DaemonSet.

```
> hypertool run --dry-run
IP address of the master node: 8.8.8.8

[DRY-RUN] The following attributes would be applied:

[+] GEOLOCATION
    City:    Athens
    Region:  Attica
    Country: GR

[+] PERFORMANCE & COST
    Cost Category:     Very Low
    Energy Efficiency: Very Low
    FLOPS/sec:         11808288288.29
    Latency Class:     A

[+] NETWORK
    Network Type: Ethernet
    Interface:    enp6s0
```

Figure 2. Overview of the CLI utility and node lifecycle management.

By encapsulating the underlying complexity of the native orchestration platform, the application translates intricate, multi-step Kubernetes API [1] interactions into a centralized suite of intuitive commands (e.g., `register`, `unregister`, `self-advertisement`). This design significantly reduces the operational overhead and domain expertise traditionally required to manage heterogeneous clusters. Furthermore, while the default execution abstracts these complex mechanisms to provide a streamlined user experience, the tool includes a configurable verbose mode. When enabled, this mode surfaces granular, low-level diagnostic logs and API responses, equipping administrators with deep visibility for advanced troubleshooting.

Crucially, the CLI component is engineered with sophisticated error management and fault tolerance to ensure operational continuity. Recognizing the inherent unpredictability of distributed infrastructures, the tool employs robust exception handling and automatic rollback mechanisms to prevent catastrophic failures. Rather than experiencing a total system crash when an error occurs during command execution, the application logs an informational warning and automatically reverts the system to its previous stable state.

B. Hypertool as a DaemonSet

While the CLI utility handles the initial registration, the nodes' continuous lifecycle management is maintained by deploying Hypertool as a Kubernetes DaemonSet [6]. By utilizing this native orchestration resource, the framework ensures that a dedicated monitoring pod is automatically scheduled and deployed to every new worker node that joins the cluster. This approach leverages Kubernetes's inherent self-healing and reconciliation loops, ensuring that the Hypertool remains operational and resilient against localized failures.

To ensure that resource information remains up to date and accurately reflects a node's current status, Hypertool runs a continuous polling routine. By default, this execution loop runs every five minutes, strictly aligning with the native `-node-status-update-frequency` of the Kubelet mechanism [11]. During each interval, the pod dynamically recalculates an extensive suite of system attributes and patches the orchestration API, transitioning the node representation from a static asset to a highly dynamic, robustly profiled entity. The primary metrics dynamically calculated during this phase include:

- **Computational Performance**: The attribute is computed by first executing a quick subsecond artificial workload to estimate floating point operations per second, and then computing GFLOPs (Giga Floating-Point Operations per Second). It serves as a hardware performance indicator derived from synthetic workloads and is crucial for benchmarking, load balancing, and compute-aware scheduling decisions.
- **Energy Efficiency**: This attribute quantifies performance-per-watt (GFLOPs/Joule). It is calculated by correlating the dynamically estimated FLOPs with the node's Thermal Design Power (TDP). The TDP is either retrieved via hardware introspection databases or predicted using a trained Random Forest regressor [12] based on core count, CPU clock speed (GHz), and L3 cache size from Intel- and AMD-provided datasets.
- **Monetary Cost**: To enable cost-aware scheduling, a pre-trained K-Means clustering [13] model evaluates the instance's CPU and memory specifications. It maps the node's Euclidean distance against curated cloud pricing centroids to assign a qualitative monetary tier (ranging from *very low* to *very high*) without requiring real-time pricing APIs.
- **Trust Score**: This metric provides a quantified assessment of a node's operational stability. It computes a weighted score by aggregating internal stability signals, such as container restart frequencies, CPU throttling periods, pod evictions, and Out-Of-Memory (OOM) kills, classifying the node's reliability as high, medium, or low.
- **Security Posture**: To safeguard sensitive workloads, the framework calculates a composite security level. This weighted metric evaluates system patch compliance, Center for Internet Security (CIS) benchmark passing rates [14], open CVEs [15] detected via rootfs scanning, and overly permissive RBAC rules.
- **Network Performance & Geolocation**: The framework periodically executes active networking tests to quantify download/upload bandwidth, Round-Trip Time (RTT), and packet loss percentage. Additionally, it approximates the node's geographic location (city, region, country) utilizing IP geolocation mappings to facilitate edge-aware workload placement.
- **Hardware Accelerators**: It dynamically scans and exposes the exact capacity of specialized hardware natively available on the host, such as Graphics Processing Units (GPUs), Tensor Processing Units (TPUs), and Field-Programmable Gate Arrays (FPGAs), enabling schedulers to target nodes specifically suited for accelerated machine learning tasks.
- **Node Category**: This attribute provides a concatenated architectural summary of the node's hardware profile. Following a lightweight system scan, the framework generates an acronym representing the node's primary capabilities, specifically evaluating CPU core count, RAM capacity, disk size, and the presence of hardware accelerators.
- **Node Pool**: Inspired by major cloud provider architectures, this attribute enables the logical grouping of nodes based on their hardware capabilities, functional purpose, or both. It serves as an administrative abstraction layer for organized cluster management.
- **Node Uptime**: This attribute captures the operational lifespan of the node, serving as a temporal fingerprint within large-scale, dynamic clusters where nodes are frequently decommissioned and replaced. Given its uniqueness in volatile environments, the metric can serve as a supplementary identifier for the node.

IV. VALIDATION

To ensure operational maturity, robustness, and ease of adoption, the proposed framework was validated across multiple heterogeneous environments. Hypertool is officially packaged and published as a Helm chart [16]. This standardized packaging guarantees that the framework can be effortlessly integrated, updated, and maintained within any compliant Kubernetes cluster with minimal administrative overhead. By following this cloud-native deployment strategy, the framework ensures a validated and repeatable installation process regardless of the underlying infrastructure.

The testing methodology was divided into three distinct operational phases to evaluate different architectural requirements:

- **Local Validation**: This phase utilized Minikube [17] to validate the primary features of Hypertool in a lightweight, single-node Kubernetes setting. The machine used for these tests was a 12-core system with 16 GB of RAM and a 250 GB SSD. Hypertool successfully registered the single node and correctly computed all resource metrics listed in Section III.B.

- **On-Premises Validation**: To evaluate the framework under standard multi-node conditions, an on-premises virtualized cluster was provisioned. This cluster consisted of a dedicated control plane (4 vCPUs, 6 GiB RAM, 64 GiB of HDD storage) and multiple worker nodes (3 vCPUs, 4 GiB RAM, 32 GiB HDD storage), enabling rigorous testing of resource isolation, inter-node coordination, and the DaemonSet's ability to seamlessly scale and attach to newly joined nodes.
- **Public Cloud Validation**: The final validation phase was executed on public cloud infrastructure to simulate a highly heterogeneous, production-grade continuum. The cluster incorporated a diverse mix of compute resources, including general-purpose instances (4–16 vCPUs, 8–64 GiB RAM), GPU-optimized hardware, and more. This phase successfully demonstrated the framework's capability to accurately profile varying raw compute capacities, detect specialized hardware accelerators, and dynamically estimate cost tiers in a highly varied topological setup.

V. Conclusions and Future Work

This paper presented Hypertool, an open-source, platform-agnostic framework designed to address the critical challenges of resource abstraction and dynamic node lifecycle management in heterogeneous distributed cloud environments. By implementing a dual-component architecture comprising a CLI utility and a Kubernetes-native DaemonSet, the framework successfully bridges the gap between static default orchestrator schemas and the complex, multidimensional requirements of modern workloads. The proposed declarative data model extends native node representations with rich, continuously updated attributes spanning computational capacity, energy efficiency, economic cost, operational trust, and security posture, without introducing heavy custom controllers that burden the control plane. Extensive validation across local, on-premises, and public cloud infrastructure demonstrates the framework's operational maturity, stability, and ease of integration.

Future work will focus on expanding the abstraction schema to capture deeper network topology metrics and more granular hardware accelerator telemetry (e.g., multi-GPU interconnect bandwidth). Additionally, we plan to integrate lightweight, predictive machine learning models into Hypertool's core logic to enable proactive anomaly detection and predictive node self-healing.

Acknowledgments

This work has been partially supported by the HYPER-AI project, funded by the European Commission under Grant Agreement 101135982 through the Horizon Europe research and innovation program (https://hyper-ai-project.eu/).

References

[1] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes, "Borg, omega, and kubernetes," *ACM Queue*, vol. 14, no. 1, pp. 70–93, 2016. DOI: 10.1145/2898442.2898444. [Online]. Available: https://queue.acm.org/detail.cfm?id=2898444.

[2] S. Yang, Y. Ren, J. Zhang, J. Guan, and B. Li, "KubeHICE: Performance-aware container orchestration on heterogeneous-ISA architectures in cloud-edge platforms," in *2021 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)*, IEEE, 2021, pp. 81–91.

[3] J. Rey-Jouanchicot, J. Á. L. Del Castillo, S. Zuckerman, and E. V. Belmega, "Energy-efficient online resource provisioning for cloud-edge platforms via multi-armed bandits," in *2022 International Symposium on Computer Architecture and High Performance Computing Workshops (SBAC-PADW)*, IEEE, 2022, pp. 45–50.

[4] L. Xiao, H. Shan, J. Zhu, R. Mao, and S. Pan, "Lightweight cloud service composition and orchestration for edge intelligence," *Intelligent Decision Technologies*, vol. 19, no. 2, pp. 844–861, 2025.

[5] N. Deligiannakis et al., "DACCA: Distributed Adaptive Cloud Continuum Architecture," *Future Internet*, vol. 18, no. 74, 2 2026, ISSN: 1999-5903. DOI: 10.3390/fi18020074. [Online]. Available: https://www.mdpi.com/1999-5903/18/2/74.

[6] The Kubernetes Authors, "Daemonset | kubernetes," 2026, Accessed: Mar. 16, 2026. [Online]. Available: https://kubernetes.io/docs/concepts/workloads/controllers/daemonset/.

[7] The Kubernetes Authors, "Custom resources | kubernetes," 2026, Accessed: Mar. 16, 2026. [Online]. Available: https://kubernetes.io/docs/concepts/extend-kubernetes/api-extension/custom-resources/.

[8] OpenStack Foundation, "Openstack open source cloud computing software," 2026, Accessed: Mar. 16, 2026. [Online]. Available: https://www.openstack.org/.

[9] HashiCorp, "Nomad: Flexible workload orchestration," 2026, Accessed: Mar. 16, 2026. [Online]. Available: https://www.hashicorp.com/products/nomad.

[10] The OpenMetal Authors, "Manage flavors in openstack," 2026, Accessed: Mar. 16, 2026. [Online]. Available: https://openmetal.io/docs/manuals/tutorials/manage-flavors#:~:text=How%20to%20Update%20Flavor%20Metadata,select%20the%20option%20Update%20Metadata..

[11] The Kubernetes Authors, "Kubelet | kubernetes," 2026, Accessed: Mar. 16, 2026. [Online]. Available: https://kubernetes.io/docs/reference/command-line-tools-reference/kubelet/.

[12] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001. DOI: 10.1023/A:1010933404324.

[13] S. Lloyd, "Least squares quantization in PCM," *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 129–137, 1982. DOI: 10.1109/TIT.1982.1056489.

[14] Center for Internet Security, "CIS benchmarks," 2026, Accessed: Mar. 16, 2026. [Online]. Available: https://www.cisecurity.org/cis-benchmarks/.

[15] The MITRE Corporation, "Common vulnerabilities and exposures (CVE)," 2026, Accessed: Mar. 16, 2026. [Online]. Available: https://cve.mitre.org/.

[16] The Helm Authors, "Helm | the package manager for kubernetes," 2026, Accessed: Mar. 16, 2026. [Online]. Available: https://helm.sh/.

[17] The Kubernetes Authors, "Minikube start | minikube," 2026, Accessed: Mar. 16, 2026. [Online]. Available: https://minikube.sigs.k8s.io/docs/start/.