HPE Slingshot in the Kubernetes Ecosystem

Caio Davi
caio.davi@hpe.com
Hewlett Packard Enterprise
Houston, Texas, USA

Jesse L Treger
jesse.treger@hpe.com
Hewlett Packard Enterprise
Houston, Texas, USA

Abstract

The convergence of traditional HPC systems with AI increases expectations for supercomputing sites to deliver new capabilities (beyond traditional batch scheduling, single tenancy, and bare-metal application deployment methodologies) for more dynamic provisioning. Convergence with enterprise cloud computing techniques such as containerized applications and Kubernetes have become a priority. But transitioning high-performance computing (HPC) environments and applications to Kubernetes is complex because of the critical requirement to maintain low-latency networking for high-performance. In this context, we have HPE Slingshot, a modern high-performance interconnect for HPC and AI clusters that delivers industry-leading performance, bandwidth, and low-latency for HPC, AI/ML, and data analytics applications through its innovations in the fabric that overcome congestion and its innovations in the NIC to significantly offload communications and message processing from the hosts. Because HPE Slingshot NICs run native Ethernet alongside an optimized RDMA transport using a connectionless protocol, ensuring that the RDMA transport is operating as intended is critical to delivering the high performance expected in HPC and AI. This requires careful configuration of Kubernetes because if not configured, the system can fall back to standard TCP/IP over Ethernet instead of achieving the expected HPC and AI RDMA performance. Our proposed solution for proper Kubernetes configuration is composed of a number of Kubernetes components such as device plugins, CNIs, Operator, and Admission Policies. These contributions represent a significant advancement in deploying and operating HPC applications within containerized environments and offering a robust framework for future developments in distributed computing, ensuring both high performance and ease of management for the continuing convergence of HPC/AI and cloud computing and the coming transition from siloed HPC interconnects to interoperable Ultra-Ethernet transport.

CCS Concepts

• **Computer systems organization** → **Cloud computing**; • **General and reference** → Design; • **Networks** → *Cloud computing*; • **Software and its engineering** → Software libraries and repositories; • **Applied computing** → **Enterprise computing infrastructures**.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
CUG'25, May 04–08, 2025, Jersey City, NJ
© 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-XXXX-X/18/06
https://doi.org/XXXXXXX.XXXXXXX

ACM Reference Format:
Caio Davi and Jesse L Treger. 2025. HPE Slingshot in the Kubernetes Ecosystem. In *Proceedings of Cray User Group Conference (CUG'25).* ACM, New York, NY, USA, 6 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction and Background

The increasing demand for complex Artificial Intelligence (AI) and Machine Learning (ML) workflows in HPC is driving a shift toward containerized applications and Kubernetes orchestration. Traditionally, HPC systems have focused on maximizing performance for numerical simulations on bare metal hardware using specialized tools like MPI [8]. However, with the rising demand for diverse scientific workflows incorporating AI/ML, data analytics, and in-situ visualization, managing complex software dependencies and configurations on bare-metal systems has become increasingly insufficient. Containers, operating at the OS level via namespaces, provide safe and lightweight virtualization and isolation, enabling researchers to package applications and dependencies into portable and reproducible units [7]. These units, usually called images, are widely available through private or public registries such as Dockerhub, NVIDIA NGC, Hugging face, etc. [11].

Managing the container lifecycle can be a challenging task. The many steps and components of this process make it error-prone. Users must manually execute complex command line operations, configure networking plumbing, manage storage bindings, and handle version updates, potentially leading to inconsistencies and deployment issues, especially in large-scale HPC environments [3].

Kubernetes enhances container management by orchestrating deployment, scaling, and networking and offering a robust platform for deploying and managing complex scientific applications on HPC systems. In this paper, we propose new Kubernetes components for deploying pods in an HPE Slingshot environment. These components can be roughly summarized as a device plugin for the device allocation, a set of CNIs (Container Network Interface) for the network interface management, an Operator for managing the NIC (Network Interface Card) resources, and a Helm Chart for installations and versioning management. The device plugin and CNIs are currently available in an alpha version, and are able to fulfill the requirements listed in Section 2. Their implementation, along with the Slingshot Kubernetes Operator, is described in Section 3. The deployment of the current released Slingshot Kubernetes solution is detailed in Section 4, and the limitations of the current solution (partially related to the partial release - lacking the Operator) are discussed in Section 5. Finally, the reader can find the conclusion and acknowledgments in Sections 6 and 7, respectively.

2 Design Requirements

Here, we outline the design requirements for a robust and performant Kubernetes experience on top of the HPE Slingshot NIC architecture.

- **Device Discovery and Management**. The solution requires a robust mechanism to automatically discover and manage new devices and nodes attached to the Kubernetes cluster. The system should be able to identify the device type, available resources, and any specific configuration requirements. This is crucial for enabling efficient scheduling and allocation of device resources to appropriate workloads.
- **Resource Allocation**. The solution must also be able to effectively allocate device resources to containers, ensuring that workloads requesting specific devices are scheduled on nodes equipped with those devices [1]. This involves extending Kubernetes resource management capabilities to include device-specific resources and constraints in scheduling decisions.
- **Performance and Isolation**. For high-performance computing workloads, minimizing overhead and ensuring performance isolation are critical [14]. The device integration mechanisms should be designed to minimize latency and ensure that device access does not negatively impact the performance of other workloads running on the same node.
- **Multiple Network Interfaces**. Dense HPC clusters usually have multiple NICs per node, but Kubernetes Networking defaults to one interface per container [9]. Multiple network interfaces are required to exploit all available resources in the cluster.
- **Support RDMA**. A dedicated interface for inter-node communication using RDMA, bypassing the overlay network for low latency and high bandwidth.
- **Security**. Secure access to devices is of uppermost importance, especially in multi-tenant environments where different users or workloads may share the same device. Both Kubernetes [12] and HPE Slingshot [5] enforce multiple levels of security policies, and these policies must be complied with.
- **Monitoring and Troubleshooting**. Effective monitoring and troubleshooting tools are essential for diagnosing and resolving issues related to device usage and performance [13]. Kubernetes must periodically obtain insights into the device utilization, health status, and any potential errors or conflicts to aid in debugging and optimization.

3 Implementation

To fulfill the requirements listed in the previous section, we propose an architecture composed of several elements: (i) a device plugin for device allocation and management, (ii) a specific set of Container Network Plugins (CNIs) to handle the pods' networking plumbing, (iii) a centralized controller for resources management, and (iv) base images to support both admin and user applications.

The main task of the device plug-in for the proposed design is to provide the pod with all required CXI drivers, libraries, and environment variables. Meanwhile, the CNIs enable the Linux network interfaces in the pod, as well as the network management and the connections between pods, containers, and the host machine. There is no explicit communication between the device plugin and the CNIs. Thus, admission policies are in place to trigger the CNIs during pod creation. Lastly, the role of the centralized controller (referred to here as the Slingshot Operator) is to configure the right NIC level resource pools, such as TX/RX profiles and VNI tags. Table 1 lists all the main Kubernetes components described in this section.

Table 1: List of main Kubernetes components in the HPE Slingshot K8s solution.

Component	Role
Slingshot Operator[a]	Manage CXI resources.
CXI CRD	Describe CXI resources.
CXI device-plugin	Manage CXI device allocation.
IPVLAN CNI	Virtual Network in the host.
Multus CNI	Multiple Linux Network interfaces.
Whereabouts CNI	Cluster-aware IP management.
Admission Policies	Network automatic annotations.

[a]Slingshot Operator is currently under development and expected to be released in the first semester of 2025.

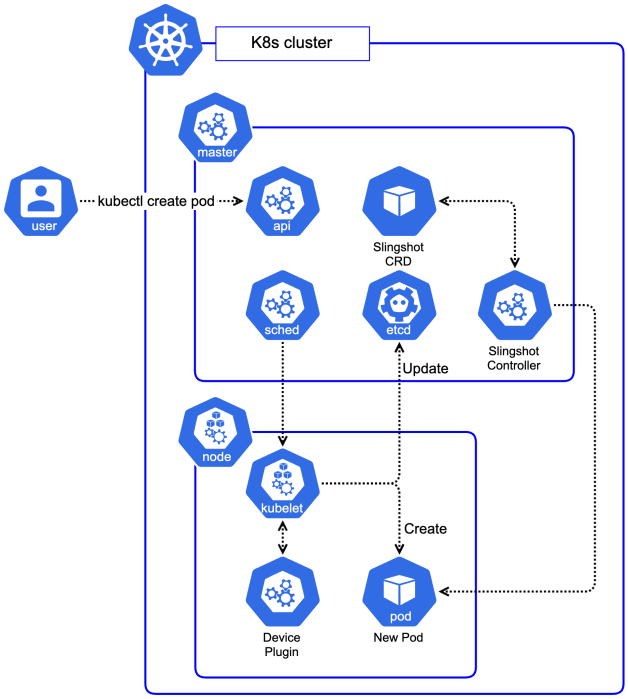

Figure 1: Slingshot Kubernetes components during pod creation process.

Figure 1 illustrates the Slingshot Kubernetes components in operation during the pod creation process. Upon receiving the pod creation request, the KubeAPI will initiate the creation workflow (not fully depicted in the figure for the sake of simplicity), which will

ultimately reach the node's Kubelet. The Kubelet will then interact with the cxi-k8s-device-plugin to allocate the necessary resources. Once the pod is registered in the Kubernetes etcd database, the Slingshot Operator is notified and will configure the HPE Slingshot fabric resources within the pod/host, including Virtual Network Interfaces (VNIs) and resource pools.

3.1 The CXI device plugin

A Kubernetes device plugin is a specialized component that allows Kubernetes to manage and utilize hardware resources that require vendor-specific setups, such as GPUs, FPGAs, and high-performance NICs. These plugins enable Kubernetes to extend its capabilities beyond the default supported hardware (vCPU/vMEM), making it more adaptable to the underlay infrastructure. Device plugins run on each node with the corresponding hardware; thus, they are often deployed as K8s daemonSets. They usually communicate with the Kubelet via a gRPC service, registering the hardware resources and making them available for scheduling and allocation to pods. Kubernetes device plugins are flexible enough to support a variety of vendor-specific requirements, but they should at least perform the following roles:

- Register hardware resources and make them available for scheduling.
- Provide the device requirements to Kubelet during pod creation/resource allocation.
- Continuously run health checks on the node and report the device(s) status back to Kubernetes.

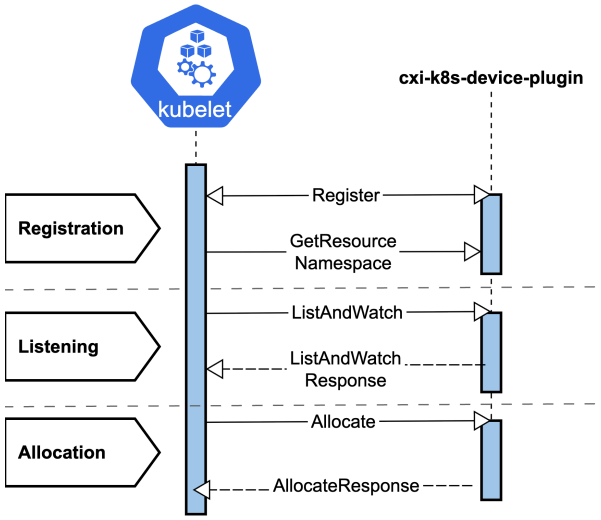

Figure 2: cxi-k8s-device-plugin lifecycle.

The lifecycle of the K8s device plugin is constituted of three major events/phases: registration, listening, and allocation. Upon installation of the cxi-k8s-device-plugin, the plugin will ascertain the number of available CXI devices on the host machine and promptly communicate this information to the Kubelet (Registration). Once the Kubelet is informed of the new resources, it will invoke the KubeApi to register these resources. Subsequently, the device plugin will continuously monitor Kubelet's health check requests and update the device status within the Kubernetes (K8s) environment (Listening). When the KubeApi receives a new pod creation request, and following the completion of all K8s security checks, the KubeScheduler will identify nodes with sufficient available resources and allocate the new pod to an appropriate node. The Kubelet will then be activated and will request the device list and resources from the device plugin (Allocation). Figure 2 illustrates the sequence diagram for the three phases of the cxi-k8s-device-plugin lifecycle.

During pod creation, if the referred pod manifest has a resource allocation matching the namespace advertised by the device plugin and the node has enough available resources, Kubelet will trigger an allocate call to obtain the required information for pod deployment. Then, cxi-k8s-device-plugin will reply to this call with an object of type `ContainerAllocateReponse` [10], which contains all the device-related information, as described in the following Go code snippet.

```
type ContainerAllocateResponse struct {
  // List of environment variable
  // to be set in the container to access
  // one of more devices.
  Envs map[string]string ``

  // Mounts for the container.
  Mounts []*Mount

  // Devices for the container.
  Devices []*DeviceSpec

  // Container annotations
  // to pass to the container runtime
  Annotations map[string]string ``
}
```

3.2 Network Plugin

The cxi-k8s-device-plugin, similar to other device plugins, is responsible for providing the functional requirements necessary for the pod/container to access and utilize the host device. This encompasses drivers, libraries, and any other essential components for the device. In the specific case of a Network Interface Card (NIC) device, containerized applications may require more than just the device being available within the pod/container. At a minimum, they will require a network interface to connect to other subnets, which could be either an internal Kubernetes network or an external subnet.

In order to provide this infrastructure to Kubernetes, we take advantage of a series of CNI plugins, admission policies, and network attachment definitions, as described below.

3.2.1 IPVLAN CNI. IPVLAN CNI is the CNI implementation of a Linux IPVLAN network and it is utilized to virtualize the host interface, thereby enabling the creation of new virtual interfaces within the container. Unlike MACVLAN, IPVLAN networks share

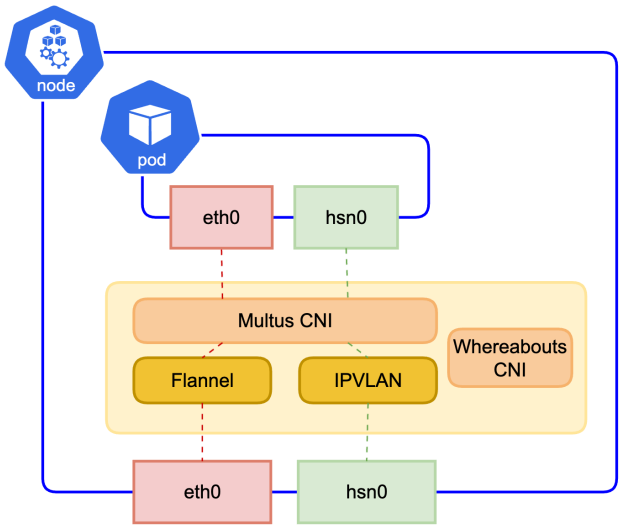

Figure 3: Slingshot Kubernetes network components.

the same Media Access Control (MAC) address, utilizing Layer 3 (L3) for multiplexing and demultiplexing among the virtual network interfaces. The kernel driver examines the IP address of each packet to determine the appropriate virtual interface for packet processing [2].

3.2.2 Multus CNI. Multus CNI is a CNI plugin for Kubernetes that facilitates the attachment of multiple network interfaces to pods. Typically, in Kubernetes, each pod is provisioned with a single network interface (excluding the loopback interface). However, with Multus, it is possible to create a pod with multiple interfaces. This functionality is achieved by Multus operating as a "meta-plugin," a CNI plugin capable of invoking multiple other CNI plugins. Here, Multus CNI is used to create the virtual hsn network interfaces inside the pod/container.

3.2.3 Whereabouts CNI. The Whereabouts CNI is responsible for managing dynamic IP address assignments for the pods within a cluster by using the Whereabouts IP Address Management (IPAM) solution. It ensures that each pod's network interface gets a unique IP address from the specified IP address range. It also handles IP address releases when pods are deleted or scaled down.

3.2.4 Network Attachment Definitions. It is important to note that while Multus will create new network interfaces, these interfaces will not necessarily be part of the default Kubernetes network. Network Attachment Definitions (NADs) can be employed to declare new subnets to which these interfaces will belong. Below is an example of how to describe a Slingshot Kubernetes NAD.

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: hsn0-ipvlan
spec:
  config: '{
      "cniVersion": "0.3.1",
      "type": "ipvlan",
      "master": "hsn0",
      "mode": "l2",
      "ipam": {
          "type": "whereabouts",
          "range": "10.10.10.1/28"
        }
  }'
```

The metadata name is the major reference to the NAD. It will be used by Multus to set up the additional interfaces in the pod. The `spec.config` field holds all major networking configurations, including the CNI to be used (`type`) and information about which interface in the host should be used (`master`). In `spec.config.ipam`, we specify the Whereabouts IP Address Management (IPAM).

3.2.5 Mutating Admission Policies. Mutating Admission Policies (MAPs) offer a declarative method to modify the pod definition right before the pod creation. Highly configurable, these policies enable authors to create parameterized and resource-scoped policies tailored to the requirements of cluster administrators. In the context of Kubernetes support for HPE Slingshot, Mutating Admission Policies are employed to include metadata annotations into the pod/container manifest. These annotations will trigger Multus CNI to create extra network interfaces in the pod/container.

3.3 Slingshot Operator

Kubernetes operators are software extensions that utilize custom resources to manage applications and their components. They leverage the Kubernetes API to monitor the state of the application and make real-time adjustments to ensure optimal performance. By defining custom resources and controllers, operators facilitate the automation of operational tasks that would otherwise require manual intervention, thereby enhancing the reliability and efficiency of managing Kubernetes-native applications. Further information on operators can be found in the Kubernetes Operator documentation.

The primary purpose of the Slingshot Operator is to manage the Remote Direct Memory Access (RDMA) security features of HPE Slingshot, similar to those found in the SLURM Slingshot "Plug-in" (that acts as the privileged entity to configure the NIC resources on compute nodes in a SLURM managed cluster). For example, using the HPE Slingshot NIC RDMA packet label security feature necessitates configuration of the Cassini Network Interface Card (NIC) for each pod/container on every compute node where the distributed application will execute. This NIC configuration must be performed by a privileged entity on the compute node, as it cannot be incorporated into the pod launch process, which operates under Kubernetes user Role-Based Access Control (RBAC) privileges. Additionally, due to the flexibility of the operator pattern, the role of the Slingshot Operator may be expanded as new features are developed.

The Slingshot Operator consists of two primary components:

(1) The controller, which is responsible for managing the available Virtual Network IDs and monitoring the creation, deletion, and update of Kubernetes (K8s) pods.

(2) The configuration repository, represented by a Custom Resource Definition (CRD).

During the installation of the Kubernetes Slingshot support components, the Kubernetes cluster administrator will instantiate a new object to hold the default configuration values (an instance of the CRD). The administrator will also instantiate the Slingshot Controller, which adheres to the operator design pattern. This Controller will monitor the state of the K8s cluster and, upon the creation, deletion, or update of a pod, will determine if the pod requires CXI fabric resources (see Figure 1). If so, the Controller will request the necessary configurations from the configuration repository and apply them to the new pod (post-creation) or release the VNI ID (post-deletion).

Although Slingshot Operator is currently in development, users can use HPE Slingshot on Kubernetes now. As mentioned before, the major role of the Operator is to manage the security policies related to RDMA traffic. Enabling the default `cxi service` in the host node would let pods/containers share the same VNI ID which allows the traffic through Slingshot fabric.

4 Deployment

This section address the installation and deployment of the Slingshot solution for Kubernetes. Since this deployment effectively happens in two moments, the infrastructure building and pods deployment, this section was conveniently divided into two subsections: deployment for system administrators and the deployment for final users.

4.1 For System Administrators

The device plugin is publicly available in [4], and the HPE Slingshot team released an application note for guidance on the cxi-k8s-device-plugin installation [6]. Both the repository and the application note have detailed steps for installation, and we strongly suggest the read of these documents before installation. They are constantly reviewed and up to date with the most recent releases.

Here, we briefly list the steps, for more detailed instructions :

(1) *[optional]* If you are not running Slingshot Controller, disable CXI Services for each CXI NIC in Kubernetes nodes.
(2) Enable Mutating Admission Policy in the Kube API Server.
(3) Install Multus CNI.
(4) Install Whereabouts CNI.
(5) Install Network Attachment Definition.
(6) Install Mutating Admission Policies.
(7) Install cxi-k8s-device-plugin (usually by deploying hpecxi-device-plugin-ds Kubernetes daemonsets).
(8) Deploy Slingshot Controller.

In the future, we intend to create a Helm Chart to simplify the deployment and management of the Slingshot K8s components. Beyond the installation process, it would facilitate updates by managing releases and rollbacks, handling dependencies between applications, and enabling version control for your Kubernetes resources.

4.2 For Users

The Slingshot Kubernetes Solution was designed to enable a straightforward deployment of CXI-enabled pods by the final user. Provided that all Kubernetes components are running and in a healthy state and the cluster has nodes with enough available CXI resources, the user must only state on the YAML manifest the requests (and preferably the limits) for CXI devices in the new component. Below, we show a pod manifest file as an example:

```
apiVersion: v1
kind: Pod
metadata:
  name: cxi-example
  labels:
    app: example
spec:
  containers:
    - name: cxi-example-container
      image: busybox
      command:
        - sleep
        - "3600"
      resources:
        requests:
          beta.hpe.com/cxi: 4     #Four NICs
        limits:
          beta.hpe.com/cxi: 4     #Four NICs
```

Notice the only fields added to this manifest were the `hsn` requests in the `spec.containers.resources`.

5 Limitations

The current release is composed of the device plugin and a set of CNIs. It is in alpha version, and specific design decisions implemented for this initial release have resulted in restricted features and limitations in the usage of HPE Slingshot fabric in Kubernetes.

5.1 No Shared CXI Devices across pods

Although the HPE Slingshot Network Interface Card (NIC) is capable of sharing its resources across multiple jobs/processes, this behavior is not expected for the first release of the Slingshot Kubernetes Operator. In its first release, the Slingshot Operator is expected to deal with pod/container and a NIC as a $1 : n$ (where n is an integer, $-1 < n < 5$) relationship. This means that each pod has the CXI device fully allocated to its workload. The processes running inside the pod may share CXI resources, though, similar to what is commonly seen in HPC workloads. This limitation will be addressed in future versions of the Slingshot Operator. If the user requires shared resources between pods in the meantime, they may enable the default `cxi service`, as mentioned in Section 3.3.

5.2 Full Load Always

At present, we support only the full allocation of all CXI devices in the node for a single pod. This restriction is related to the Kubernetes pod creation workflow and the role of Mutating Admission Policies (MAPs) within it. MAPs are verified and applied to the new component manifest immediately after the authentication process and before it reaches the Kubernetes `etcd` database. This means that the CXI-related information created by the device plugin is

not present in the request at this stage. At this point, Kubernetes is only aware of the user's intention to have CXI devices available in the pod and the quantity being requested. Consequently, neither MAP nor etcd has any means to identify which CXI devices are available and provided by the device plugin. For example, if a user requests two CXI NICs, the MAP policy cannot specify which device will be allocated to the pod, making it impossible to inform Multus CNI which network interfaces should be created. This is an undesirable behavior, and we intend to address this requirement in future versions of the cxi-k8s-device-plugin.

5.3 Software Versioning

As detailed in Section 3.1, the `cxi-k8s-device-plugin` mounts all necessary HPE Slingshot components from the host into the pod. This includes drivers, libraries, environment variables, and other elements required for accessing HPE Slingshot's high-speed networking. The Slingshot Kubernetes solution depends on these host components and assumes they are correctly installed and up to date. While this approach ensures an operational software stack within the pod by replicating the node, it introduces a potential challenge: certain upper-layer components may require specific underlay software versions. This issue is particularly relevant for Libfabric-dependent components, such as MPI, NCCL, and RCCL. If these components were compiled against a newer version of Libfabric than what is available on the host node, the pod's OFI may lack the necessary API endpoints. The HPE team recognizes this as a liability of the Slingshot Kubernetes solution and is actively working to address it. Meanwhile, several workarounds can be implemented to fix the versioning between the image and the pod.

6 Conclusion

As an acknowledgment of the recent increase in Kubernetes interest by the supercomputing community, we propose an HPE Slingshot solution for the Kubernetes Ecosystem in this paper. While yet in its alpha release, it has already been deployed and tested in different systems under diverse user cases with distinct requirements. The solution proposed here is composed of a device plugin for device discovery, allocation, and monitoring, and a number CNI plugins for network interface management. These components are available today to enable Kubernetes clusters on HPE Slingshot fabrics. Additional components are also part of the proposed solution and in development, including the Slingshot Operator and the Helm chart for automatic node labeling and cluster deployment. Together, these components intend to create a more robust, manageable, and safe environment for Kubernetes clusters running on top of HPE Slingshot high-speed fabrics.

7 Acknowledgments

The authors would like to acknowledge the use of AI tools to assist in reviewing the grammar, style, and overall coherence of this manuscript. Specifically, Microsoft Copilot 365 provided support in refining the language and ensuring clarity in the presentation, and Grammarly contributed with grammar checks. These tools' contributions have been instrumental in enhancing the quality of this paper.

References

[1] Angel M Beltre, Pankaj Saha, Madhusudhan Govindaraju, Andrew Younge, and Ryan E Grant. 2019. Enabling HPC workloads on cloud infrastructure using Kubernetes container orchestration mechanisms. In *2019 IEEE/ACM International Workshop on Containers and New Orchestration Paradigms for Isolated Environments in HPC (CANOPIE-HPC)*. IEEE, 11–20.

[2] CNI. 2024. IPvlan Plugin. https://www.cni.dev/plugins/current/main/ipvlan/.

[3] Felipe A. Cruz and Alberto Madonna. 2024. Containers-first user environments on HPE Cray EX. *Proc. Cray User Group (CUG)* (2024).

[4] Hewlett Packard Enterprise. 2017. CXI Kubernetes device plugin. https://github.com/HewlettPackard/cxi-k8s-device-plugin.

[5] Hewlett Packard Enterprise. 2024. *HPE Slingshot Administration Guide*.

[6] Hewlett Packard Enterprise. 2025. HPE Slingshot & Kubernetes. https://hpe.seismic.com/Link/Content/DCJcgP2Q66d4gG4DQ4pp36VQfRpV.

[7] Dan Fulton, Laurie Stephey, R Canon, Brandon Cook, and Adam Lavely. 2023. Containers everywhere: Towards a fully containerized HPC platform. *Proc. Cray User Group (CUG)* (2023).

[8] Lise Jolicoeur, François Diakhaté, and Raymond Namyst. 2025. Leveraging private container networks for increased user isolation and flexibility on HPC clusters. In *International Conference on High Performance Computing*. Springer, 415–426.

[9] Kubernetes. 2017. Kubernetes Network Design Proposal. https://github.com/kubernetes/design-proposals-archive/blob/main/network/networking.md.

[10] Kubernetes. 2025. Go k8s.io/kubernetes Device Plugin API. https://pkg.go.dev/k8s.io/kubernetes@v1.16.15/pkg/kubelet/apis/deviceplugin/v1beta1#pkg-index.

[11] Amit Ruhela, Matt Vaughn, Stephen Lien Harrell, Gregory J Zynda, John Fonner, Richard Todd Evans, and Tommy Minyard. 2020. Containerization on petascale HPC clusters. State of Practice talk in International Conference for High Performance ….

[12] John Smith. 2024. Exploring the Role of Kubernetes in MLOps and DevOps for Containerized Machine Learning Model Management. *Hong Kong Journal of AI and Medicine* 4, 2 (2024), 88–94.

[13] Nitin Sukhija and Elizabeth Bautista. 2019. Towards a framework for monitoring and analyzing high performance computing environments using kubernetes and prometheus. In *2019 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*. IEEE, 257–262.

[14] Madan Timalsina, Lisa Gerhardt, Nicholas Tyler, Johannes P Blaschke, and William Arndt. 2024. Optimizing Checkpoint-Restart Mechanisms for HPC with DMTCP in Containers at NERSC. *arXiv preprint arXiv:2407.19117* (2024).