Cloud-Native Slurm Management on HPE Cray EX

Felipe A. Cruz
Swiss National Supercomputing Centre
Lugano, Switzerland
felipe.cruz@cscs.ch

Manuel Sopena
Swiss National Supercomputing Centre
Lugano, Switzerland
manuel.sopena@cscs.ch

Guilherme Peretti-Pezzi
Swiss National Supercomputing Centre
Lugano, Switzerland
guilherme.peretti-pezzi@cscs.ch

***Abstract*—This work introduces a cloud-native deployment of the Slurm HPC Workload Manager, leveraging microservices, containerization, and on-premises cloud platforms to enhance efficiency and scalability. Utilizing Kubernetes and Nomad's APIs alongside DevOps tools, the system automates system operations, simplifies service configuration, and standardizes monitoring. However, implementing a cloud-native architecture poses challenges, including complex containerization and resource management issues that are intrinsic to HPC.**

Our solution implements an automated deployment and management process of Slurm. Where some components are containerized, others are deployed natively and are managed via Kubernetes and Nomad, respectively. This approach ensures consistent, automated, reproducible, and resilient Slurm management. The full paper will detail this implementation and its benefits, underscoring the potential of modern software practices to evolve HPC systems to meet the performance and flexibility expected by our user base.

***Index Terms*—Cloud-native architecture, Workload Manager, Microservices, On-Premises Cloud Platforms, Automated Operations**

I. Introduction

Under a cloud-native approach [1], a High-Performance Computing (HPC) system is architected as a collection of small, loosely coupled services that can be independently delivered. Moreover, this approach leverages on-premise cloud platform deployments that enable a self-service model for engineers to introduce controlled changes to the cluster while streamlining service and infrastructure automation.

In this work, we present the application of cloud-native management principles to Slurm [2], a well-known HPC Workload Manager (WLM). This implementation puts in practice a microservices approach, containerization of software components, software-defined configuration for operation of the services, continuous integration and delivery (CI/CD), and on-premises cloud platforms to improve the efficiency and scalability of managing this service, with the goal of fully realizing the flexible capabilities of an HPE Cray EX system.

The foundation pilar of this work is achieved by leveraging on-premises cloud platforms for the administration of Slurm, making full use of Kubernetes [3] and Nomad [4] Application Programming Interfaces (APIs) to orchestrate Slurm services through declarative configurations. By doing so, we obtain several advantages, including:

- Automation of operations such as service deployment, updates, and self-recovery.
- Simplification of dynamic node configuration for large clusters.
- Enhanced efficiency through automated resource allocation of system services across the infrastructure.
- Automated governance of system services on the infrastructure.
- Standardized interfaces that can be leveraged for automatic detection of issues with services.

Often, implementations of the HPC software stack are monolithic in nature, where all the software components necessary for operation —such as OS, utilities, libraries, tools, applications, and various core HPC services— are integrated into a single, tightly coupled image. This approach is characterized by:

- A single unit for building, configuring, and deploying, which can make the cluster updates cumbersome and slow, especially as the complexity of the stack increases.
- Management of applications is centralized, leading to integration bottlenecks, which often have to compromise flexibility among the many diverse application needs.
- Upgrades are complex processes, due to interdependencies of software components all across the stack, often requiring extensive testing to ensure full stack compatibility.
- Issue propagation, as a problem on a single software, often requires full stack rebuild and redeploy.

In contrast, our cloud-native Slurm implementation is built upon an automated, version-controlled deployment and management process, utilizing GitLab [5] for CI/CD, Terraform [6] for service provisioning, Kubernetes for orchestrating the Slurm controller and database daemons, and Nomad for native slurmd agent deployment on the HPE Cray EX compute nodes. This approach ensures consistent, repeatable, and version-controlled infrastructure provisioning, significantly reducing human errors, streamlining change management, and providing a tangible structure to cloud infrastructure.

This paper outlines an innovative approach to managing Slurm deployments on HPC systems by harnessing the principles of cloud-native architecture. Through the deployment and orchestration techniques that we will describe, we present a modular and flexible approach for building and managing HPC services. The following sections will provide a detailed view of a deployment architecture of Slurm, the dynamic management processes that we have implemented, and the

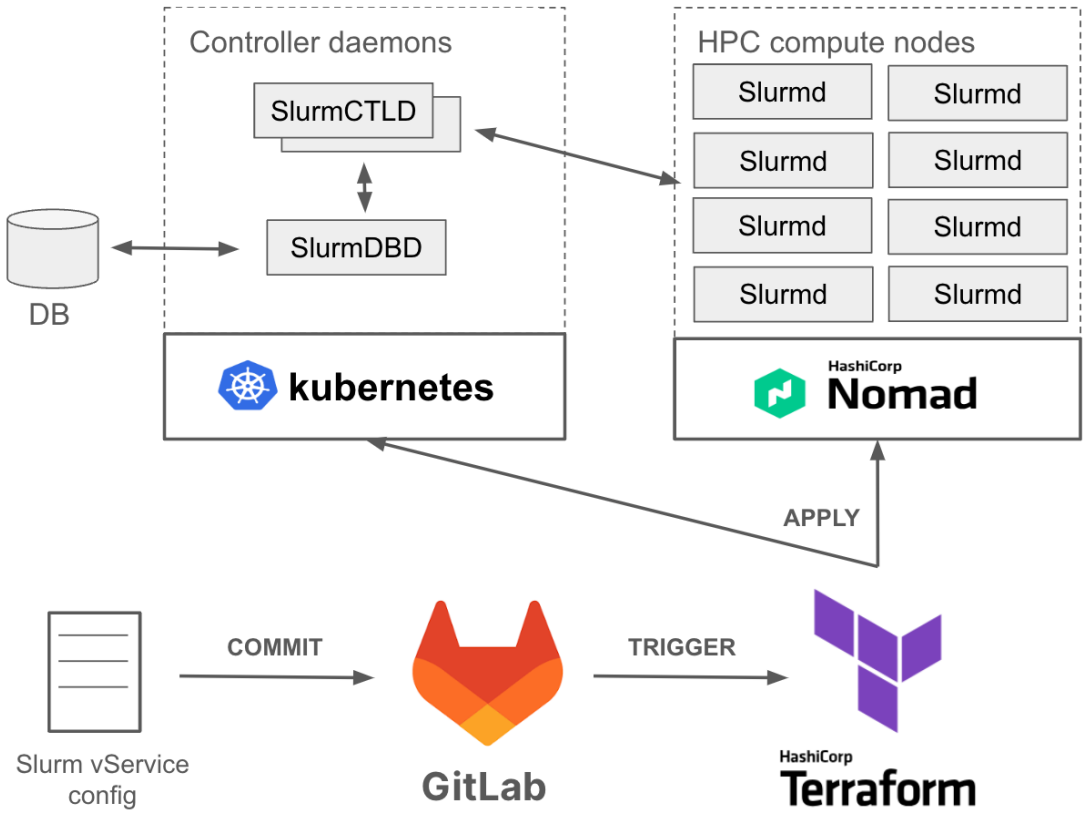

Fig. 1. Diagram of the automated, version-controlled deployment and management process for a Slurm WLM environment using GitLab for version control and CI/CD, Terraform for infrastructure provisioning, Kubernetes for container orchestration of slurm control daemons, and Nomad for deploying native slurmd agent on the HPC cluster, with Slurm itself managing HPC job scheduling on HPC compute nodes..

operational results of integrating cloud-native principles to HPC management. The overarching goal is to present a template for HPC service delivery and management that can make optimal use of HPC resources but, more importantly, allow us agility to respond to the ever-growing computational needs of performance and flexibility expected by our user base.

II. Cloud-Native Principles and HPC

In [7] paper, we introduced a novel approach by setting up Cloud-Native High-Performance Computing (HPC) clusters on HPE Cray EX systems. This approach applies cloud-native principles to the deployment and management of HPC clusters by leveraging the following key principles in a cloud-native architecture: microservices, containerization, continuous integration and continuous delivery (CI/CD), and DevOps practices. By following such an approach, our objective is to realize the same advantages that cloud computing has gained from the application of these principles and apply them to the management of the complex and dynamic demands of HPC systems.

In a nutshell, the cloud-native HPC architecture is characterized by several core principles:

- Microservices Architecture: In a cloud-native approach, we break down the Cluster monolith into a collection of microservices. The goal of the decomposition is to obtain smaller, loosely coupled services that can be independently developed and deployed for greater flexibility and responsiveness to changes. Ideally, each service manages its own dependencies and interacts with other services through well-defined interfaces.
- Containerization: When possible, microservices are packaged into containers, which include all necessary executables, binary code, libraries, and configuration files. Containers provide an isolated environment for each service, ensuring consistency across different development and production environments.
- Continuous Integration and Delivery (CI/CD): Leveraging CI/CD pipelines to support deployment and testing processes. This automation supports team collaboration and transparency of deployments, while allowing for more frequent updates and consistent deployment practices, reducing the risk of errors and, when possible, deployment of new features or updates without significant downtime.
- DevOps Practices: Integrating development and operations teams through DevOps practices enhances collaboration and streamlines workflows. This integration facilitates a culture of continuous improvement, enhances productivity, and reduces the time to deploy new features

or resolve issues.

Due to its advanced hardware and robust system management capabilities, the HPE Cray EX is well-suited for implementing a cloud-native HPC approach. A key feature is its support for Infrastructure as Code (IaC), enabling us to manage the infrastructure via automation and programmable interfaces. This functionality allows for the deployment of independent configurations that can be individually updated without affecting the entire system. Moreover, the HPE Cray EX simplifies operations and enhances flexibility by enabling the management of multiple clusters from a single, highly configurable system. Deploying HPC clusters using a cloud-native approach on HPE Cray EX brings multiple operational and strategic benefits:

- Enhanced Flexibility: The microservice architecture enables us to build a single system stack that can easily deploy and configure multiple independent clusters, where each can respond to specific needs. This flexibility is crucial for serving a variety of needs where computational, storage, and system service configuration and composition can vary significantly between clusters.
- Increased Resilience: The use of microservices enhances system resilience. In case of a service failure, it can be updated and restarted independently with limited impact on other services of the system, thereby improving overall system response and reliability.
- Faster Innovation and Deployment: CI/CD and DevOps practices reduce the cycle time for the development and deployment of new hpc services, user applications, and updates. This capability enables engineers to work independently and more freely, accelerating iteration cycles in service development.

Improved System Management and Maintenance: Cloud-native principles facilitate better system management through automation, monitoring, and maintenance capabilities. Tools like Hashicorp's Nomad and Kubernetes simplify service deployment and management.

III. Slurm Deployment Architecture

The architecture of the cloud-native Slurm implementation makes use of multiple cloud technologies to enhance the management of the workload manager. Kubernetes, Nomad, Terraform, and GitLab are the core components of this approach towards dynamic and robust system management. The system is architected using a microservices approach where internal components of the Slurm are modularized and managed using different cloud platforms and tools:

A. Kubernetes

Kubernetes orchestrates containerized instances of Slurm components such as the Slurm controller (slurmctld), Slurm database daemon (slurmdbd), and Slurm rest daemon (slurmrestd), each running within dedicated pods in a Kubernetes tenant namespace. This setup enhances management by:

- Automating the deployment of Slurm components, ensuring high availability and fault tolerance.
- Utilizing Kubernetes' secrets management to handle sensitive configurations securely, ensuring that components such as Munge [8] (for authentication) are correctly configured across deployments.

B. Nomad

While Kubernetes enters around managing containerized applications, Nomad is employed to manage the deployment of native services on Compute Nodes, a crucial aspect of HPC operations. Nomad orchestrates the deployment of the Slurm daemon (slurmd) on compute nodes, extensively using its capability to manage non-containerized tasks. Key benefits include:

- Nomad can manage native tasks using the "rawexec" task driver. This allows for direct interaction with compute node hardware, ensuring performance and no overheads. This is crucial for components that are not suitable for containerization due to their specific requirements or complexities.
- Streamlined management of compute resources by Nomad agents on cluster's compute nodes, allowing us a flexible mechanism to manage resources via software without minimal latency on reconfiguration.

C. Terraform

Terraform plays a pivotal role in provisioning and managing the underlying infrastructure for both Kubernetes and Nomad environments. By using IaC, Terraform ensures that all service components are deployed consistently and are reproducible. This process is critical for:

- Quick and consistent setup of the required services, reducing manual configuration errors and increasing deployment speed.
- Central management of both containerized and native service components, simplifying the complexity of operating diverse technological stacks while ensuring that the component deployment order is respected.

D. GitLab

GitLab integrates the entire deployment process through GitLab runner implementation of CI/CD pipelines, automating the deployment and version control of the Slurm implementation. This ensures that all deployments are consistent, reproducible, and automated, streamlining the management of service changes:

- Automated pipelines deploy updates and new configurations of services without significant downtime, improving the responsiveness to changes.
- Version control of configurations provides us with improved tracking and management for all service changes.

The Slurm service deployment workflows begin with the cluster owner pushing updates or configuration changes to the GitLab repository that holds the Cluster configuration. These changes trigger GitLab's CI/CD pipelines, which manage the deployment across the Kubernetes and Nomad platforms. Artifacts used by Slurm are maintained in a JFrog container

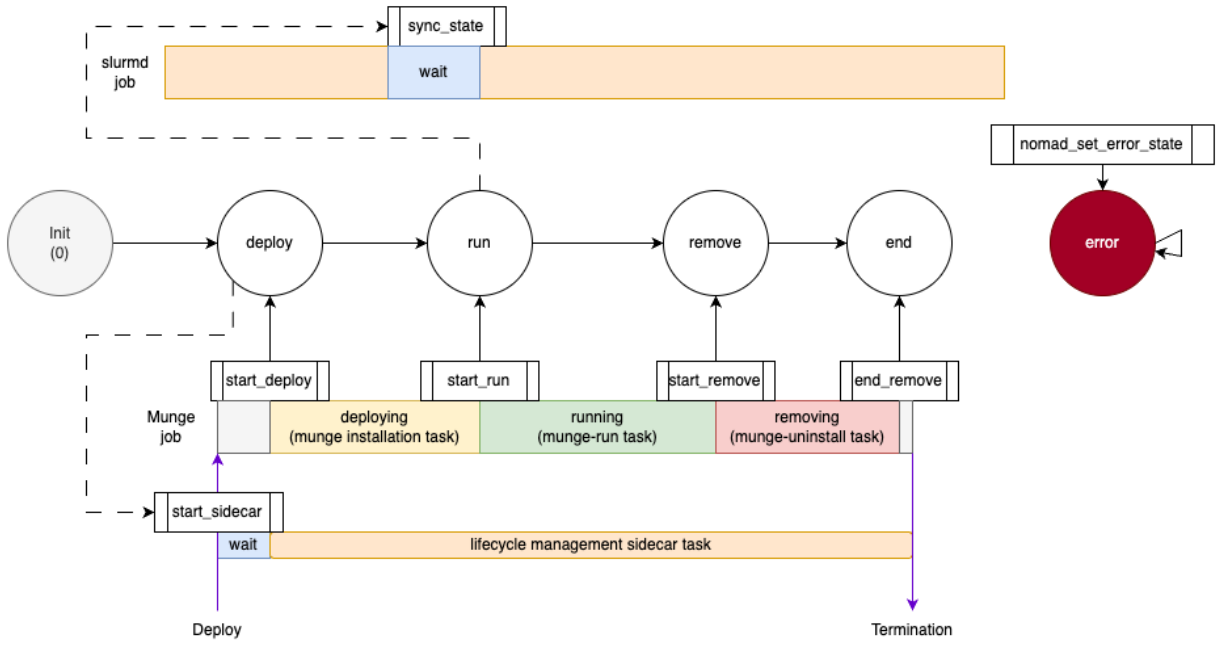

Fig. 2. This diagram depicts the state transitions and synchronization of Nomad jobs, providin an example on the interaction between a slurmd job and a munge job within a Nomad deployment. The slurmd job's synchronization depends on the state transitions of the munge job, highlighting the dependency between jobs being successfully deployed and operational before the next job can proceed. This interaction ensures that jobs dependent on each other respect the necessary sequence for a stable and efficient deployment process. Note the sidecar lifecycle management task that starts concurrently with the munge job's deploy phase and it waits until end for termination. Consider now that the same illustration applies for any service that need to implement lifecycle and interdependency management.

registry [9], with Kubernetes orchestrating the deployment of the containerized application. Simultaneously, Nomad schedules and manages native Slurm daemon tasks directly on the compute nodes.

IV. Deployment and Management Process

The architecture of the Slurm service is designed to efficiently deploy and manage Slurm on vCluster environments, it uses a hybrid model where services are deployed both natively on compute nodes and containerized on a Kubernetes cluster. Table I provides an overview of each component and its deployment mechanism.

A. Containerization and Orchestration of Daemons on Kubernetes

The Slurm components deployed via Kubernetes have been containerized to isolate dependencies and simplify deployment procedures. The containerized components include slurmctld, slurmdbd, mariadb [10], slurmrestd, and necessary authentication services like munged and ssd.

1) General Details for Kubernetes Deployment Configuration:

- Containers are executed within Kubernetes Pods, encapsulating them in a controlled environment.
- Volumes:
  - ConfigMaps: Used to store shared configuration data, allowing uniformity across deployments.
  - Secrets: Employed to securely manage sensitive data, such as authentication keys.
  - PersistentVolumeClaims: Utilized for storing persistent data that needs to survive pod restarts or failures.
- Environment Variables passed on to Pods are used for providing configurations specific to each deployment, such as `SLURM_CLUSTERNAME`, `SLURM_SPOOLDIR`, and timezone settings.
- Component interoperation is achieved via Networking and shared storage:
  - Ports are used for communication between components, with `slurmctld` listening on port `TCP:6817` and `slurmdbd` on `TCP:6819`.
  - Shared Mounts: Utilized across containers to facilitate inter-container communication and data sharing. Common mounts include read-write access on directories like `/etc/munge` from munge-confdir, `/munge` from munge-secret, `/var/run/munge` from munge-rundir, and `/var/spool/slurm` from slurm-data.
- Mounts: Key for operational consistency, these Kubernetes volume mounts manage configuration files (`/etc/munge`, `/etc/slurm`), secrets, and data directories across components, ensuring they are secure and consistent across the deployment.
- File Permission Settings: Handled by an initialization container which configures file ownership and permissions to suit operational requirements, such as setting

TABLE I
OVERVIEW OF KEY SLURM COMPONENTS: DEPLOYMENT METHODS AND FUNCTIONAL DESCRIPTIONS

Component	Deployment	Functionality
Slurm Control Daemon (slurmctld)	*Containerized and managed as Pod by Kubernetes*	Acts as Slurm's central management daemon, controlling all other service daemons and allocating resources in the HPC cluster. Key service for orchestrating workload management across the compute infrastructure.
Slurm Database Daemon (slurmdbd)	*Containerized and managed as Pod by Kubernetes*	Provides an interface to the database that records Slurm job data.
Database for Slurm Data (mariadb)	*Containerized and managed as Pod by Kubernetes*	Serves as the backend database for Slurm, storing all related data.
Slurm REST API (slurmrestd)	*Containerized and managed as Pod by Kubernetes*	Provides a REST API for Slurm, offering a programmatic interface through a web-based API.
Slurm Compute Node Agent (slurmd)	**Native on Compute Nodes, managed by Nomad**	Manages the execution and monitoring of tasks on each compute node, communicating with slurmctld to manage workloads.
Munge Authentication (munged)	Support component deployed natively on Compute Nodes and containerized on Kubernetes pods	Provides authentication services by creating and validating credentials, ensuring secure cluster operations.
Slurm Client Commands	Support component deployed natively on Compute Nodes and containerized on Kubernetes pods	Includes commands like squeue, srun, sbatch, scontrol, sacct, sinfo, etc., enabling users to submit and manage jobs on the cluster.

chmod 0755 on necessary directories, compensating for Kubernetes' default volume permission settings.

2) Specific Kubernetes Pod Configurations: Each component is configured within its pod to optimize performance and security.

3) Slurm Control Daemon:

- **Init Container:** Prepares shared mounts.
- **Containers:**
  - **slurmctld:** Utilizes `configmap` for `slurm.conf`, `sssd.conf`; `secret` for `munge.key`, `mail.rc`; environment variables like `SLURM_CLUSTERNAME`, `SLURM_SPOOLDIR`, `TZ`; and multiple mounts for configuration and runtime data.
  - **munged:** Handles authentication, mounting `/etc/munge` and `/var/run/munge`.
  - **sssd:** Manages system security services directory, mounting `/etc/sssd` and `/var/lib/sss`.

4) Slurm Database Daemon:

- **Init Container:** Sets up shared mounts.
- **Containers:**
  - **slurmdbd:** Similar to `slurmctld`, with specific mounts for its configuration.
  - **munged** and **sssd:** As above, supporting authentication and security services.

5) Slurm REST API on Kubernetes Pod:

- **Init Container:** For mount preparation.
- **Containers:**
  - **slurmrestd:** Uses secrets and mounts similar to `slurmctld` but tailored for REST API needs.
  - **munged** and **sssd:** Provide necessary backend support.

6) Backend Database on Kubernetes Pod:

- **MariaDB:** Employs `PersistentVolumeClaims` to ensure data persistence across sessions and deployments, crucial for maintaining stateful application data.

B. HPC Service Deployment Using Nomad

HashiCorp Nomad is a multi-purpose workload orchestrator that facilitates the deployment and management of both non-containerized and containerized applications in on-premises and cloud environments. It supports our cloud-native strategy by providing flexible and efficient tools for deploying and managing traditionally complex services that are difficult to containerize. Key aspects of using Nomad, which bring significant benefits, include:

- Nomad Job Specifications. HPC services deployed through Nomad follow specific job specifications and are designed to follow a microservice architecture. The goal is to have smaller, independently manageable components, to simplify integration, updates, and maintenance.
- Task Drivers. Nomad offers various task drivers for executing jobs. However, on compute nodes we mostly rely on the Raw Exec Driver to execute commands without isolation. This allows tasks to interact directly and tightly with the host HPC system, which is central to many HPC services.
- HCL for Job Definitions. Services configurations in Nomad are written using the HashiCorp Configuration Language (HCL). This declarative language streamlines the specification of job settings and parameters, enhancing clarity and maintainability.

1) On slurmd Service:

```
job "slurm-cn" {
  priority = 95
  datacenters = ["${var.datacenter}"]
  type = "system"
  group "slurmd-cn" {
    # Each task is scheduled on a
        ↪ different node
    constraint {
      operator = "distinct_hosts"
      value = "true"
    }
    task "slurmd" {
      driver = "raw_exec"
      user = "root"
      config {
        command = "/usr/sbin/slurmd"
        args = ["-D", "-Z", "--conf-
            ↪ server", "${var.slurm-ctld
            ↪ -host}", "--conf", "
            ↪ Feature=compute"]
      }
    }
    network {
      port "slurmd" {
        static = 6818 # host linked port
            ↪  to TCP 6818
}}}}
```

Listing 1. With filename 'slurmd.hcl'. Contains a Nomad job description of Slurm daemon on compute nodes for Slurm

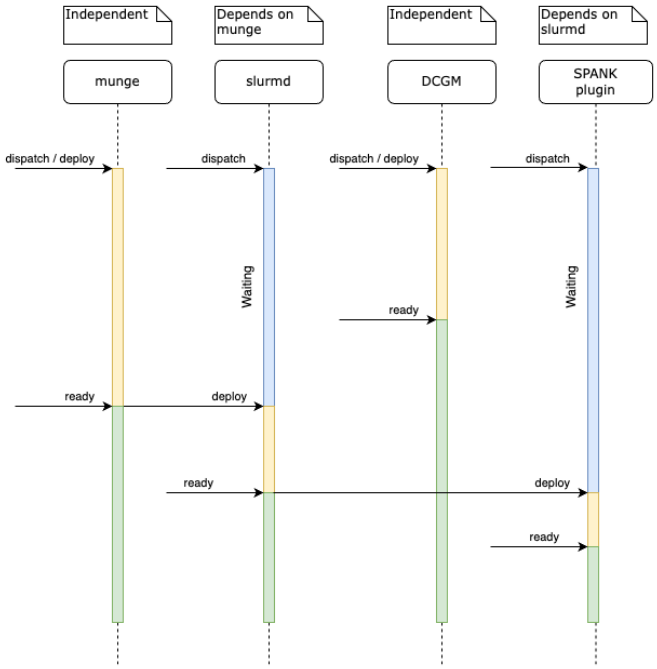

Fig. 3. This diagram illustrates the dependency and deployment sequence of Nomad jobs for various services. Each vertical line represents a different service, with the sequence of actions shown from top to bottom. This deployment sequence ensures that services with dependencies are correctly synchronized based on their respective readiness states, maintaining operational stability and coherence. In this example, the deployment of SPANK plugins depends on Slurm, which in turn depends on Munge while other services like DCGM start independent from the rest.

The Slurm daemon (slurmd), a key component of the Slurm cluster, is responsible for managing job execution and resource allocation on the compute nodes of an HPC system. When deploying slurmd with Nomad, the deployment adapts to accommodate its specific requirements, convoluted to containerize due to its complex and central role in many other HPC operations. As such, we opted to deploy this component natively and manage it via Nomad in the following way:

- Job Specification. The deployment of the slurmd via Nomad follows a Nomad job specification. This specification details how the Slurm daemon should be deployed, configured, and managed. The job uses Nomad's HashiCorp Configuration Language (HCL) to define the deployment parameters, see Listing IV-B1.
- Raw Exec Driver. Due to the slurmd need for close control of host system resources, it is deployed using Nomad's raw exec task driver. This driver executes the slurmd directly on the host without any form of virtualization or containerization, providing the daemon complete access to the host's resources. This is required by the current daemon implementation in order to manage system resources and schedule jobs on the compute nodes.
- Deployment Strategy:
  - Job Deployment Type System. The Slurm daemon (slurmd) runs as a Nomad Job type System. This setup ensures that the slurmd is run on every eligible compute node within the cluster.
  - Configuration and Command Parameters. The Nomad job specification for slurmd includes any configuration parameters and command-line arguments necessary for the daemon's operation.
  - Service Dependency. Before the Slurm daemon starts, other essential services, such as Munge (for authentication), must already be running. Service job dependency is discussed further in the following section.

2) On Services Nomad Jobs Synchronization: Nomad job synchronization plays a crucial role in managing the deployment sequence and interactions of services orchestrated by Nomad, especially during scenarios like service restarts or compute node reboots. In case of these events, the orchestrator automatically reschedules services, making it essential to ensure the correct start order of services is maintained.

Our approach to synchronization is at the level of individual Nomad Jobs, not across the entire microservices deployment. Each Nomad Job follows a sequence of states from the deployment to the termination shown in Figure 2. This design choice preserves the decoupled nature of deployments. To facilitate this, we've developed a synchronization library that provides standardized functions for managing job states throughout their lifecycle, including deployment, running/ready, and removal phases. This library also includes mechanisms for state

locking and unlocking, supporting sequential service startups based on interdependencies. For example, the deployment of SPANK plugins must precede that of Slurm, which may depend on Munge, see Figure 3. By means of this synchronization library, Nomad can effectively enforce the deployment sequence of services on compute nodes, ensuring operational coherence and reliability.

C. On Terraform, Nomad, and Kubernetes

We use Terraform as the front-end interface where all cluster and service definitions are expressed. This allows our engineering teams to manage the configuration and control of cluster services as code. Nomad and Kubernetes, on the other hand, provide the back-end execution platforms to physically deploy and manage the slurm components. They ensure that the declared state of the services deployed matches the desired state declared in Terraform. This allows us to streamline infrastructure management while also enhancing the flexibility for managing cluster services.

1) Terraform as IaC interface: Terraform serves as the Infrastructure as Code (IaC) interface for the HPC cluster, enabling the definition and configuration of HPC infrastructures through code within a Terraform description file (main.tf), which outlines the resources of the HPC cluster and its services as modular Terraform components. Each Terraform module represents a deployable service for the HPC cluster. The terraform configuration files are maintained in a version-controlled GitLab repository, using tags to manage versions of services, enhancing reusability, updates, and rollbacks. Moreover, Terraform actively manages the state of the infrastructure services, ensuring that it always aligns with the configurations declared in the main.tf file. When changes are detected, Terraform coordinates with providers like Nomad or Kubernetes to apply necessary updates, maintaining the infrastructure in its desired declared state.

2) Service Providers: Nomad and Kubernetes act as service providers to Terraform, taking on the role of executing deployments as specified by Terraform. Both providers receive job specifications from Terraform, to then schedule and run these jobs on suitable infrastructure. Moreover, the providers are responsible for dynamically managing resources and scheduling tasks across the distributed infrastructure, ensuring that services operate reliably and according to the configurations defined in the infrastructure code.

3) Synchronization Management: The Jobsync tool, discussed in the previous Nomad section, manages deployment synchronization and is crucial for maintaining inter-service dependencies. This tool ensures that service deployments are executed in the correct order and state, as defined by the Terraform configuration.

V. Challenges and Solutions

Transitioning to a cloud-native architecture is not devoid of challenges. Integrating Slurm, traditionally an HPC service, into a microservices architecture presents several complexities. Challenges identified include:

- Components can be hard to containerize due to complex inter-process communications.
- Conflicts between cgroups of microservices and container runtime.
- Conflicts involving Linux namespaces usage by microservices with container runtime.
- Complex management of containers' underlying infrastructure resource access (crucial when accessing specialized hardware).
- Difficulties in executing privileged applications within containers.
- Challenges in maintaining long-lived, stateful applications not originally designed for dynamic redeployment.
- Subtleties in managing Linux namespaces for services like the slurmd agent on compute nodes, which spawns child processes.
- Complex services interdependencies, non trivial management of service lifecycle for automated scripted deployment.

As we have shown, our implementation addresses these challenges by architecting Slurm microservices deployment to leverage the strengths of different on-premise cloud platforms and tools, ensuring that key slurm components are automatically deployed and managed:

- Slurm Control Daemon: This central management entity, responsible for resource allocation, job monitoring, and overall cluster management, is containerized and managed through Kubernetes.
- Slurm Database Daemon: This component interfaces with databases for job, node, reservation data storage, etc., and is also containerized and Kubernetes-managed.
- Slurmd daemon on compute nodes: Functioning natively on compute nodes to monitor and manage tasks, this daemon, due to its non-container-friendly nature and associated challenges, is deployed and managed through Nomad's "rawexec" task driver, complementing Kubernetes' orchestration capabilities.

VI. Cloud-Native Operation

We will now illustrate some of the key operational scenarios for cloud-native Slurm deployment.

A. Cluster Bootstrapping

Terraform is used to define and configure the services for the cluster through a version-controlled specification. Ensuring that every component of the slurm cluster is described in a consistent, repeatable manner. Such descriptions, configurations, and associated artifacts are stored and versioned, enabling not only reuse but also sharing across different cluster deployments, or even different organizations.

When a cluster administrator decides to deploy a new HPC cluster, they leverage Terraform that automates the bootstrapping process. Terraform acts by requesting service providers like Nomad and Kubernetes to orchestrate the instantiation of all services described in the Terraform configuration. Because the infrastructure resources are already in an alive state,

ready and waiting for deployment commands, the process of spinning up a new cluster becomes significantly faster and more streamlined. This orchestration includes setting up the entire infrastructure in a manner that adheres to predefined configurations, ensuring that the deployed service is promptly operational and configured for HPC. This infrastructure as code approach accelerates the deployment process while also enhancing the reliability and consistency of the HPC services deployed.

B. Service Reconfiguration On A Live Cluster

A well-designed Slurm microservice can be effectively managed using a cloud-native approach to minimize impact on end-users while enhancing operational flexibility and efficiency. Consider that the Slurm workload manager utilizes a slurm.conf configuration file, which is a central component for defining the behavior of the Slurm cluster, including scheduling parameters and job management. This file is a versioned artifact, stored in a repository to ensure that any changes made are traceable, reversible, and consistent.

When the administrator of the cloud-native cluster needs to modify parameters within the Slurm configuration, these changes are implemented by updating the versioned slurm.conf file and the corresponding service definition within the deployment architecture managed by Kubernetes. The updated configuration can then be redeployed to the Slurm controller via Kubernetes without shutting down or disrupting the active Slurm worker nodes. This means that while the controller is being updated, the compute nodes continue to operate undisturbed, processing jobs already in the queue.

This method of managing updates exemplifies a delicate aspect of HPC services and automation: the design of the system has to leverage platform functionality by means of version control, service decomposition, and dynamic resource management in order to ensure minimal disruption to scientific users. The ability to update a central component like the Slurm controller without impacting the broader HPC system enables seamless, continuous service to users, even during configuration transitions. This approach can enhance the user experience by maintaining service availability while allowing for more agile and responsive management of the HPC environment, accommodating evolving user community needs.

C. Node Management

One important operational capability is to dynamically reconfigure HPC cluster resources to meet varying needs at runtime. This is facilitated by Nomad's scheduling and resource management features. In this case, we can use the code description of Nomad jobs to arrange or assigning compute nodes across different target usage pools, such as batch computing, high-throughput computing, interactive computing, and cluster services.

Moreover, Nomad allows service developers to specify constraints within the job definition, to define infrastructure conditions under which jobs should run. For instance, these constraints can be used to ensure that compute nodes are assigned to Slurm queues that match their capabilities. For example, nodes equipped with high-memory can be programmatically constrained to join memory-intensive queues.

Additionally, the datacenter attribute of a Nomad job configuration can be dynamically used for targeted deployment of compute nodes across different stages for CICD pipelines, such as testing, staging, or production environments. This enables a flexible and efficient use of resources, where nodes can be dynamically reassigned between different software-based redeployments based on current operational needs, without requiring physical reconfiguration or significant downtime.

The described flexibility in resource allocation and job scheduling makes possible HPC clusters that adapt to changing requirements and priorities. Nomad features can be leveraged to manage resource management smoothly via software configuration, as such, it is possible to use computational resources efficiently, reduce operational costs, and maintain high levels of service availability and performance across various clusters.

VII. Conclusion

The implementation of a cloud-native architecture for Slurm on the HPE Cray EX supercomputer presents a novel approach to the deployment and management of HPC systems' services. This innovative approach to HPC decomposes the often-used monolithic management of an HPC system into a more dynamic and modular framework, enabling agile system deployments to respond to the evolving demands of modern HPC.

The transitioning to a microservices-based architecture for Slurm deployment offers granular control over each of its components, allowing for independent development and maintenance by different engineering teams. This separation of concerns reduces the complexity associated with managing large-scale systems while opening the door toward the ability to introduce updates and improvements with minimal disruption to the overall system. By following the proposed approach for managing Slurm to other HPC services, engineers can be given the autonomy to innovate with limited risk of conflicts across services.

The integration of Kubernetes, Nomad, and the CSM within this architecture can support a range of cloud-native features such as automated deployments, self-healing processes, live service reconfigurations, and compute node management. These capabilities facilitate a robust operational approach to adapting via code to varied dynamic needs. Moreover, the use of infrastructure as code through Terraform further streamlines the deployment process, ensuring consistency, reproducibility, and efficiency in the use of deployed HPC resources.

Moreover, the adoption of a cloud-native strategy promotes a collaborative environment across teams by leveraging continuous integration and delivery pipelines. This setup helps us accelerate the deployment iteration time while ensuring that changes are approved and tracked, helping operate reliable and robust HPC services. Additionally, the flexibility of this

architecture supports rapid response and resolution of problems given by changing requirements.

In conclusion, the presented cloud-native Slurm deployment on HPE Cray EX is a forward-looking approach that harnesses the benefits of modern software practices and tooling. It reshapes HPC cluster management by offering a scalable, resilient, and efficient platform to meet the needs of contemporary HPC, optimizing HPC cluster management while also aligning with CSCS's strategic goals of enhancing flexibility and improving operations to enable world-class scientific innovation.

REFERENCES

[1] Cloud Native Computing Foundation, "Who we are. Cloud Native Definition." [Online]. Available: https://www.cncf.io/about/who-we-are/. [Accessed: Jan. 20, 2024].

[2] A.B. Yoo, M.A. Jette, and M. Grondona, "SLURM: Simple Linux Utility for Resource Management."

[3] Kubernetes, "Production-Grade Container Orchestration." [Online]. Available: https://kubernetes.io/. [Accessed: Jan. 20, 2024].

[4] Hashicorp, "Nomad documentation." [Online]. Available: https://developer.hashicorp.com/nomad/docs. [Accessed: Jan. 20, 2024].

[5] Gitlab, "Gitlab Runners." [Online]. Available: https://docs.gitlab.com/runner/. [Accessed: Jan. 20, 2024].

[6] Hashicorp, "Introduction to Terraform." [Online]. Available: https://developer.hashicorp.com/terraform/intro. [Accessed: Jan. 20, 2024].

[7] Felipe A. Cruz and Alejandro J. Dabin, "Deploying Cloud-Native HPC Clusters on HPE Cray EX," in *Proceedings of the Cray User Group (CUG) Conference*, 2023.

[8] MUNGE Authentication Service. Available online: https://dun.github.io/munge/ (Accessed on May 8, 2024).

[9] JFrog - Software Supply Chain Platform for DevOps & Security. Available online: https://jfrog.com (Accessed on May 8, 2024).

[10] MariaDB Foundation. Available online: https://mariadb.org (Accessed on May 8, 2024).