Slurm and/or/vs Kubernetes

Tim Wickberg
CTO

slurm | SCHEDMD

Background

- This talk is meant to discuss the interplay between traditional HPC workload managers - Slurm - and cloud native orchestrators - Kubernetes
- "and/or/vs"... why not just pick a single conjunction?
  - Well... it's not that simple
  - Depending on your site, users, and systems, either Slurm or Kubernetes, or Slurm and Kubernetes combined, may be appropriate stacks

Warning!

- This is meant as a high-level, somewhat simplified, view of two complex products
  - Slurm and Kubernetes are both open-source
    - There are patches, plugins, and configurations that look radically different than what I've described
    - Both systems continue to evolve well beyond their original designs

Perspectives - Kubernetes

- Kubernetes was built to manage long-running processes
  - Designed to **orchestrate** multiple microservices
    - Usually in support of one or more web services
  - Core architecture permits scaling cluster size according to external demand
    - And managing availability and redundancy for the constituent services
- Cloud-native systems assume "**infinite**" resources are available
  - And the workload is **finite**
    - Albeit, with fluctuations in instantaneous demand
- Prioritization not a central aspect of cloud orchestration
  - All workload is expected to run concurrently by default

Perspectives - Kubernetes

- Kubernetes approaches scheduling at a different level - node centric
  - Scheduling API granularity is fixed at the node level
    - Extensions such as NVIDIA's DRA allows for GPU management
  - No model for CPU core affinity
    - Can't - centrally - ensure a pod won't share a core with other workloads
  - Scheduling semantics reflecting cloud workload demands, rather than HPC
    - E.g, **Affinity** and **Anti-Affinity** scheduling policies
      - Anti-Affinity is used to ensure pod instances don't share a node
        - Critical for architecting redundant systems
        - But doesn't translate into traditional HPC batch scheduling
- Services are containerized by default
- System use is generally programmatic, through tools like Terraform

Perspectives - HPC Batch Scheduling

- HPC systems assume system size is **fixed**
  - And the workload is **infinite**
  - Queue prioritization is thus critical
- "Slurm is a policy engine" - quote stolen from a colleague
- Slurm manages a number of intertwined HPC system management tasks
  - Job queuing and prioritization - **scheduling**
  - Job accounting
  - Control user access to compute resources (cgroups, pam_slurm_adopt)
  - Enable large-scale concurrent job launch (MPI, PMIx, nss_slurm, sbcast)
- Jobs assume access to a usable, fully-featured, default Linux environment
  - Containerization - including Slurm's built-in container support - is optional
- Jobs are usually ad-hoc scripts, submitted through the command line
  - Newer features such as Slurm's RESTful API can support more programmatic interaction, but are not yet as widely adopted

Current Kubernetes Batch Support

- Kubernetes has limited support for batch workflows
  - Modeled as either individual "pods", or as "jobs"
  - Most workflows use "pods" due to issues around the "jobs" model
- Prioritization models are limited
  - FIFO is most common

Current Kubernetes Batch Support

- MPI-style workload support is weak
  - Concurrent pod scheduling is not guaranteed by default Kubernetes components
    - Default behavior for HPC batch schedulers
- "MPI Operator" is the most commonly used component to ensure pods launch roughly simultaneously
  - But does not scale - struggles to launch above more than 80 ranks
    - Citation - https://doi.org/10.1109/CANOPIE-HPC56864.2022.00011

Convergence of HPC and Cloud-Native

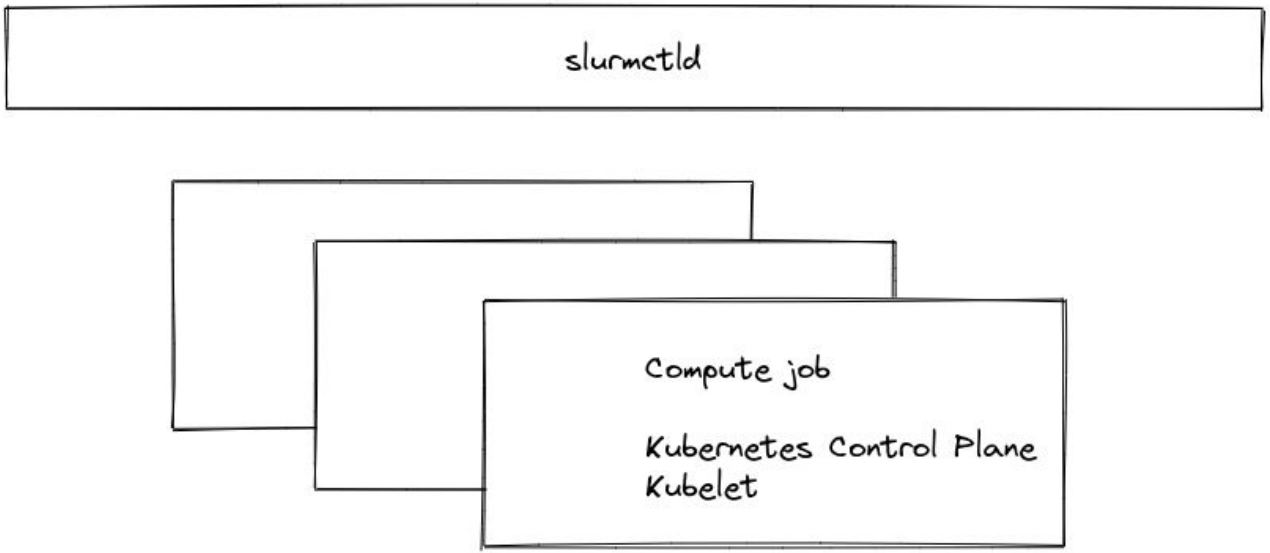
- So... why am I talking about this?
- There's an opportunity to bridge the gap between HPC and Cloud-Native workloads
  - Find a way to bring familiar commands, tooling, prioritization models into newer architectures
  - Clusters will continue to evolve - users are interested in access to new tools and technologies
  - Both ecosystems stand to benefit from each other
    - Kubernetes from increased throughput, different approaches to job scheduling and prioritization
    - Slurm from newer cloud native technologies and tools, and increased focus on flexibility in support of new user workflows

Converged Environments

Models of Converged Environments

- Four high-level models for a converged Slurm + Kubernetes environment:
  - Over
  - Distant
  - Adjacent
  - Under
- These are from Slurm's perspective... flip the Over/Under terms for Kubernetes' viewpoint

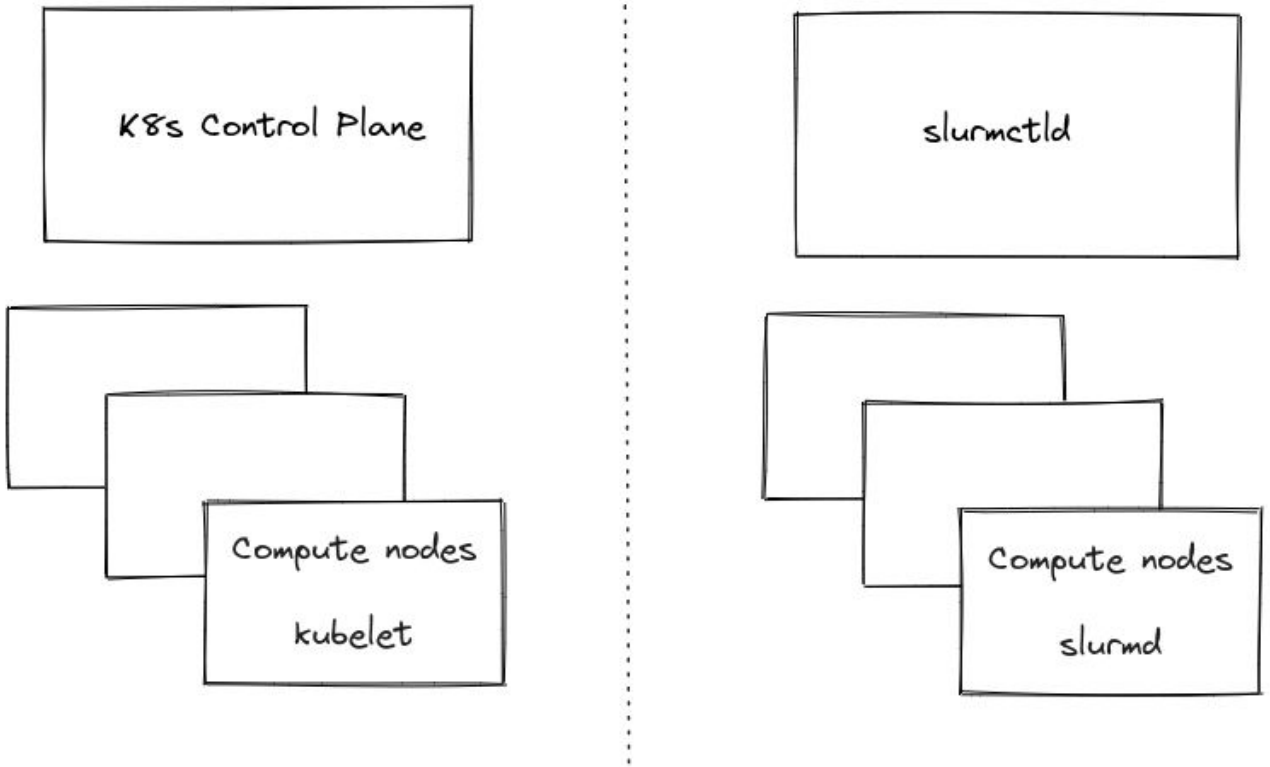
Over



Over

- Slurm manages all cluster resources
- Kubernetes clusters are created ephemerally within Slurm batch jobs
- Kubernetes control plane unavailable until job launches...
  - Or needs to be hosted outside of the traditional cluster
- Not especially useful beyond test / development environments IMNSHO

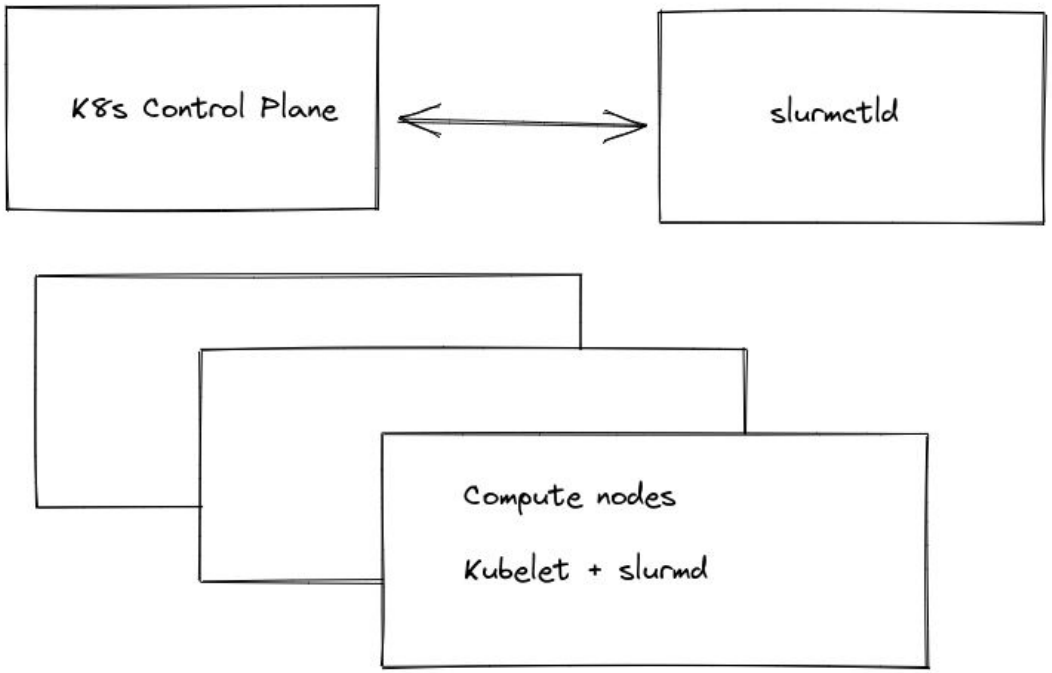
Distant


K8s Control Plane
Compute nodes
kubelet
slurmctld
Compute nodes
slurmd

Distant

- Run both Slurm and Kubernetes within the cluster environment
- Potential to enlist an additional management tool to shift nodes between the two sides
- Neither Slurm nor Kubernetes are aware of the current resources and demand for the other environment
    - Management tool needs to handle assignment of resources between environments
- Approach taken today by tools such as Dell's Omnia toolkit

Adjacent



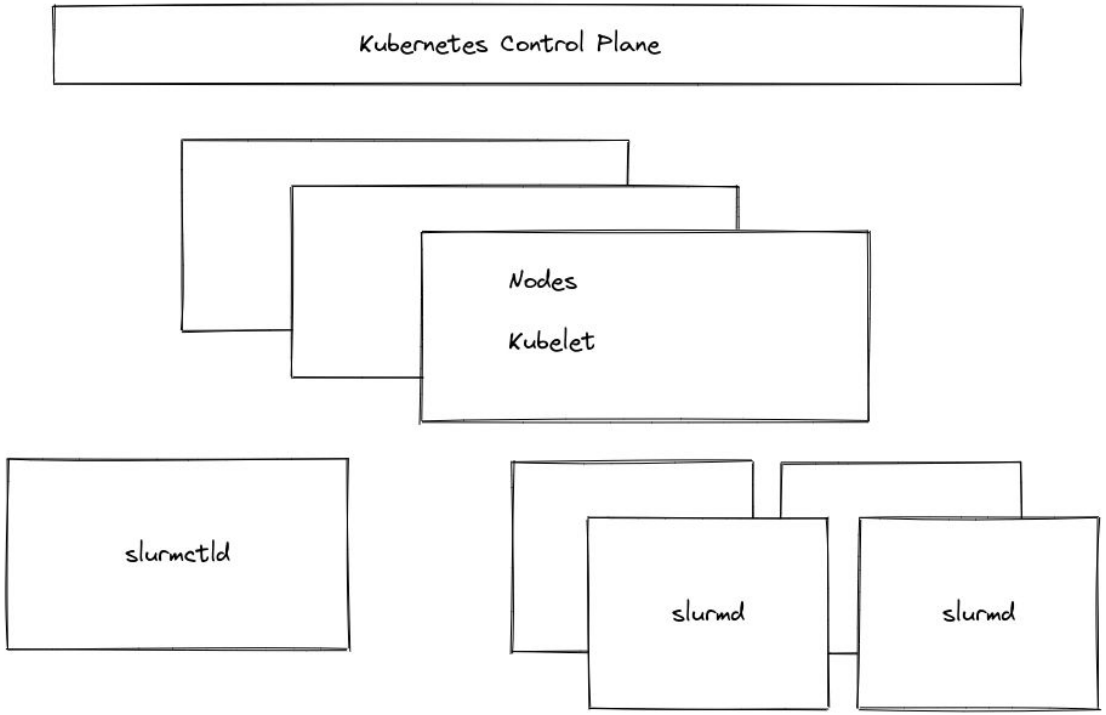
Adjacent

- Overlap both control planes
- Install Slurm Kubernetes scheduler plugin
  - Have Slurm prioritize and schedule both Slurm and Kubernetes workloads
- Kubernetes jobs managed by the kubelet
  - Full access to Kubernetes capabilities - sidecars, operators
- Slurm jobs run through Slurm
  - Manage high-throughput workloads and large-scale MPI workloads
  - Provides traditional CLI interfaces that HPC users expect
    - Alongside RESTful API

Adjacent

- Current known limitations
  - Kubernetes scheduling is still at node-level granularity
    - DRA driver provides some support for GPU management
    - No further granularity available currently
      - But changes are difficult to push upstream
  - Some Kubernetes scheduling primitives - e.g., affinity/anti-affinity - are difficult to model in Slurm's internals

Under


Kubernetes Control Plane
Nodes
Kubelet
slurmctld
slurmd
slurmd

Under

- Run Slurm cluster(s) within a Kubernetes environment
- Kubernetes-native cloud providers are already emerging
  - And all mainstream cloud environments have a managed Kubernetes offering
- Long-lived "login" nodes (Kubernetes pods) provide for traditional user experience
  - While allowing for increased user-to-user isolation
- Auto-scaling - best implemented through a Kubernetes Operator - can be used to shift resources to/from Slurm's control
  - The dynamic nodes feature in Slurm 22.05+ makes this simple
  - Auto-scaling here can also be a bit more nuanced than the existing Slurm power-saving-based cloud bursting model

Under

- Pros
  - Traditional experience for Slurm users
  - Allows for higher throughput, and full MPI support for those workloads
- Cons
  - Kubernetes workloads run outside of Slurm's view
  - Prioritization between Slurm and Kubernetes workloads difficult
    - All limitations of Kubernetes scheduling apply

Current Directions

SUNK

- SchedMD is working with CoreWeave on "SUNK" - "[S]l[u]rm o[n] [K]ubernetes"
  - CoreWeave is a specialized GPU cloud provider, and uses Kubernetes to manage their bare metal
    - Use Slurm on Kubernetes for customer workloads, including large-scale AI training work
      - Including their recent record-setting MLPerf run on 3,584 H100 GPUs

SUNK

- Kubernetes used to manage and deploy the Slurm cluster on bare metal
- Kubernetes Operator deployed to monitor Slurm cluster state through the REST API
    - Scale nodes (pods) up-and-down automatically by adding/removing dynamic nodes from the cluster
- Kubernetes scheduling plugin also allows for Kubernetes workloads to be tracked and managed through that same Slurm cluster
- Combination of the "Under" and "Adjacent" models

SUNK

- … where is it?
  - CoreWeave is working on open-sourcing SUNK, planned for early 2024
    - SchedMD is working with them to extend it to additional K8s environments

Questions?


SCHEDMD
The Slurm Company