Slinky: The Missing Link Between Slurm and Kubernetes

Tim Wickberg
Skyler Malinowski

CANOPIE-HPC 2024 @ SC24

Introduction

Survey questions:

- Who here is running Slurm in production?
- Who here is running Kubernetes in production?
- Who here is running **both**?
    - Who here has them isolated from one another?
    - Who here has them coordinating access... somehow?

What this talk is **not**

- An overview of Slurm's container integration
  - Please see separate presentations on Slurm's OCI container support
    - Nate Rini will be presenting on this at the Slurm booth this week
  - Or NVIDIA's Pyxis plugin for Slurm
    - Or NERSC's shifter plugin, or Apptainer, or Singularity, …
- Instead, I'm leaning on the "New Orchestration Paradigms" part of CANOPIE's backronym

Cloud Native vs HPC Perspectives

Warning!

- This is meant as a high-level - vastly simplified - view of two complex ecosystems
- Slurm and Kubernetes are both open-source
  - There are patches, plugins, and configurations for both that look radically different than what I've described here
  - 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 parts of the workload are expected to be **running**

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, Helm

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"
- 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

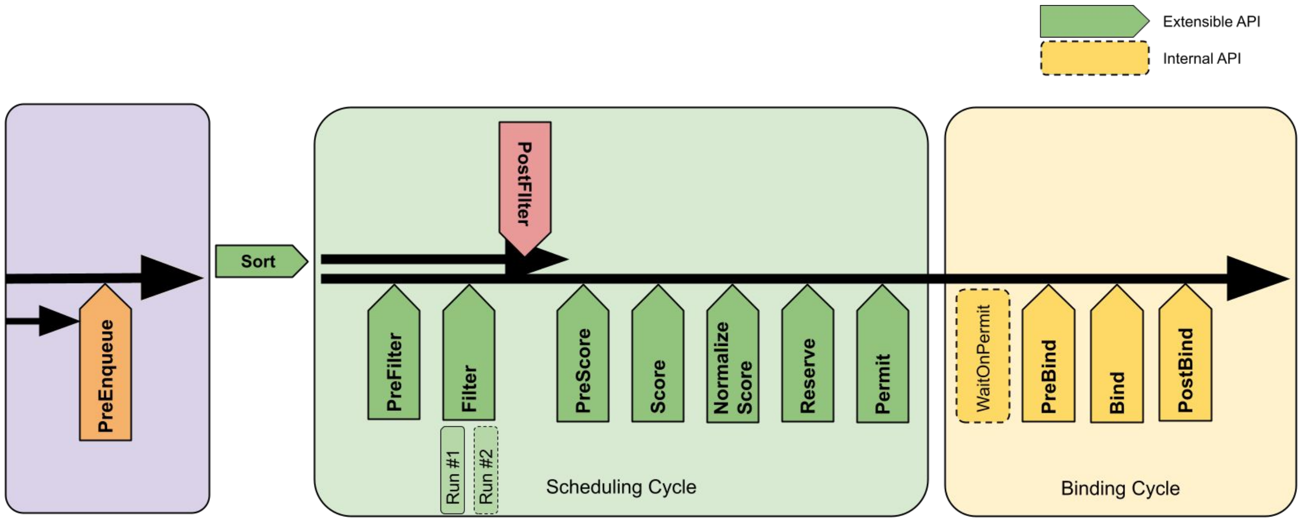
- Kubernetes has limited support for batch workflows
  - Modeled as either individual "pods", or as "jobs"
  - Most workflows use "pods" due to issues around the "jobs" model
- Prioritization models are limited
  - FIFO is most common

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
    - Kubernetes calls this "gang scheduling", HPC calls this… "scheduling"
      - Unrelated to Slurm's "gang scheduling", which manages time-slicing resources on a node
- "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
  - No native access to interconnects (Infiniband, NVLink, etc)
- The open-secret in the AI/ML space is that almost all training workloads run on Slurm

Kubernetes Scheduling Challenges

- The Kubernetes Scheduling API is... complex
  - Core parts of what the default scheduler implementation expects are exposed throughout the control plane, rather than being isolated internally



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
  - A lot of newer workload tools assuming Kubernetes by default
- 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

What is Slinky?

What is Slinky?

What is Slinky?

- A toolkit of components to enable Slurm integration with Kubernetes
  - Open-source, Apache 2.0 licensed
  - Initial components were released on November 8th
  - SlinkyProject on GitHub
  - Direct link to overview page - slinky.ai
- Uses Slurm's REST API for all core interaction
  - Wrapped into a client Golang library

What is Slinky?

- Current design has three main aspects:
  - The Slurm Operator
    - Managing Slurm running within Kubernetes
  - Assorted Tooling
    - Helm charts, Dockerfiles, Container Images, Slurm REST Client Library
  - The Slurm Bridge (Future)
    - Integration with Kubernete's scheduling API
      - Use Slurm's scheduling wherewithal to manage a converged pool of computing resources
        - Run K8s workloads through the Kubelet
        - Slurm workloads through slurmd

What is Slinky **not?**

What is Slinky **not**?

- Slinky is not a direct part of Slurm
    - Although Slinky's design has had - and will continue to have - influence on Slurm
    - Separate development team within SchedMD
    - Separate license - Apache 2.0
        - Slurm's license is "GPL v2 or later, with an OpenSSL exception"
- Slinky is not included in SchedMD's support for Slurm
- Slinky is not current intended as an out-of-the-box solution
    - Instead intended to provide flexibility in how it is adapted into an environment
    - Assumes some willingness to alter the various components as part of this adaptation
        - Changing the Dockerfiles, Helm charts, and even Golang code

Slinky Components - Today

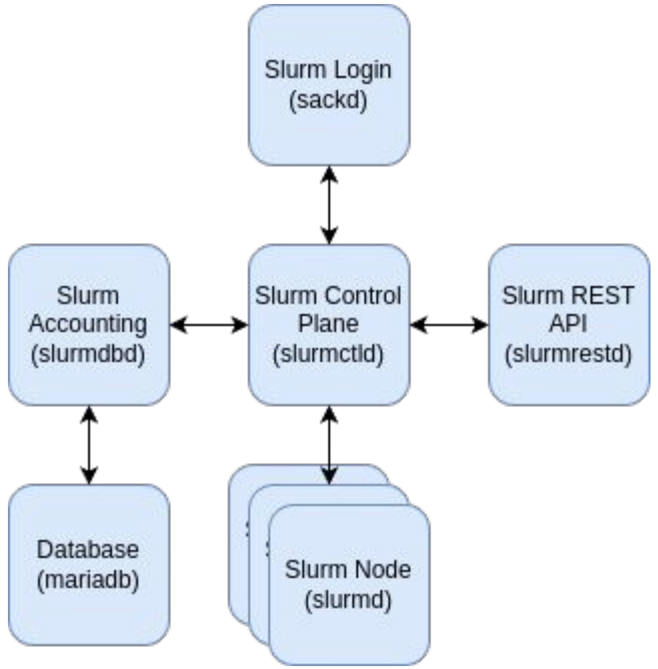
Slinky Components

- Slurm Operator
  - Kubernetes Operator for Slurm
- Slurm Exporter
  - Prometheus collector and exporter for metrics extracted from Slurm
- Slurm Client
  - Slurm versioned REST API endpoints are multiplexed for seamless request/response
- Helm charts
  - Slurm Cluster
  - Slurm Operator
  - Slurm Exporter
- Container images
  - Slurm Daemons
  - Slurm Operator
  - Slurm Exporter

Slurm Deployment

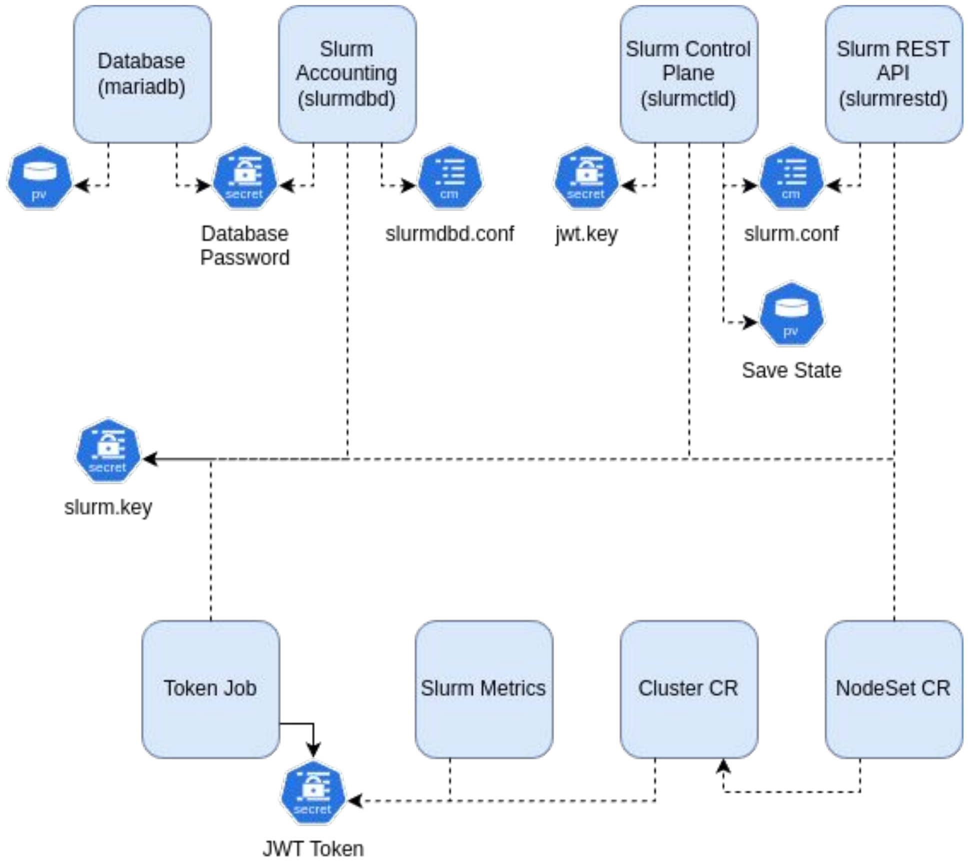
Slurm – Big Picture

- Slurmctld
  - Slurm Control Plane
- Slurmd
  - Slurm Worker Agent
- Slurmrestd
  - Slurm REST API Agent
- Slurmdbd
  - Slurm Database Agent
- Sackd
  - Slurm Auth/Cred Server
  - Typically runs on login nodes


Slurm Login (sackd)
Slurm Accounting (slurmdbd)
Slurm Control Plane (slurmctld)
Slurm REST API (slurmrestd)
Database (mariadb)
Slurm Node (slurmd)

Helm Chart

- Configless
  - Slurm configuration files are distributed by the Slurm Control Plane (slurmctld)
- auth/slurm
  - Instead of MUNGE
- auth/jwt
  - Alternative authentication mechanism
  - Issue token for Slurm REST API (slurmrestd) communication

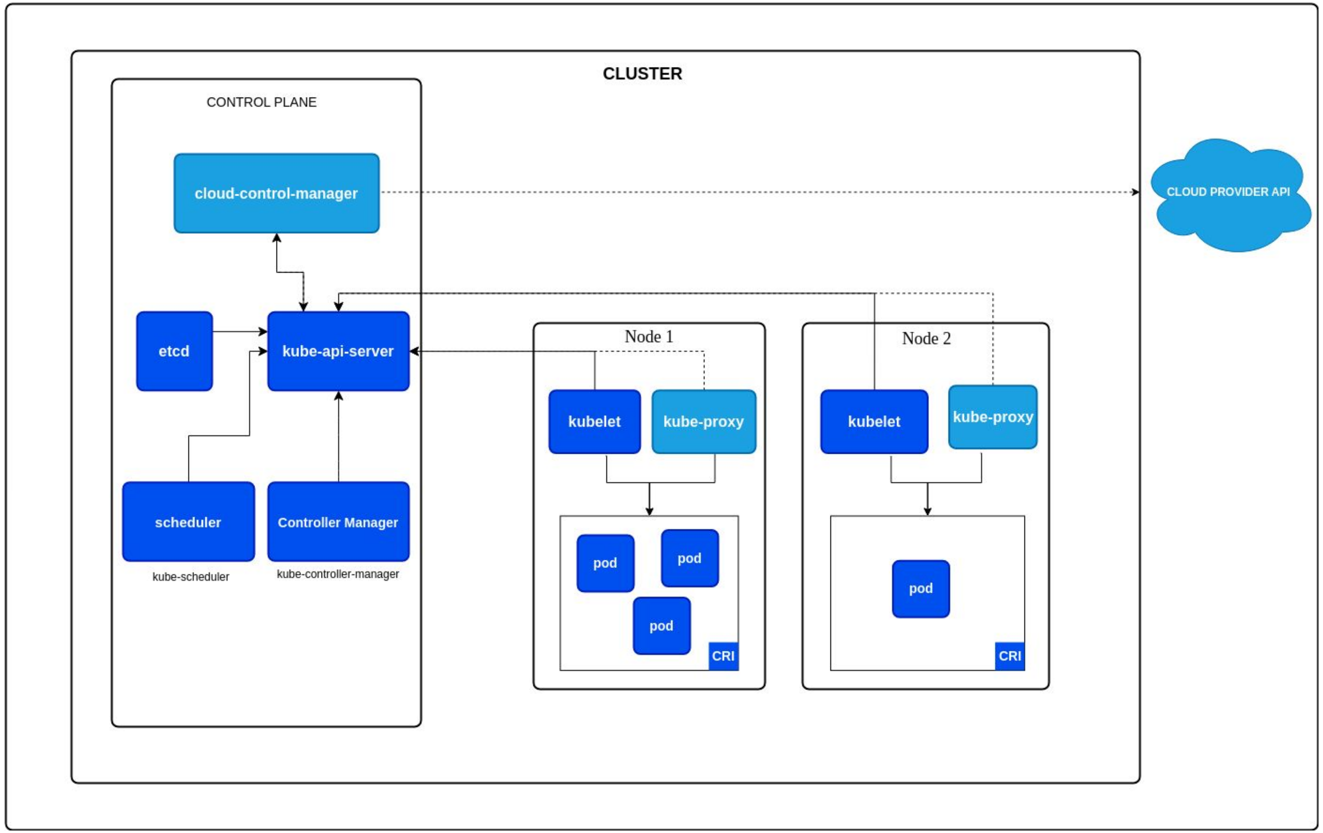


Slurm Exporter

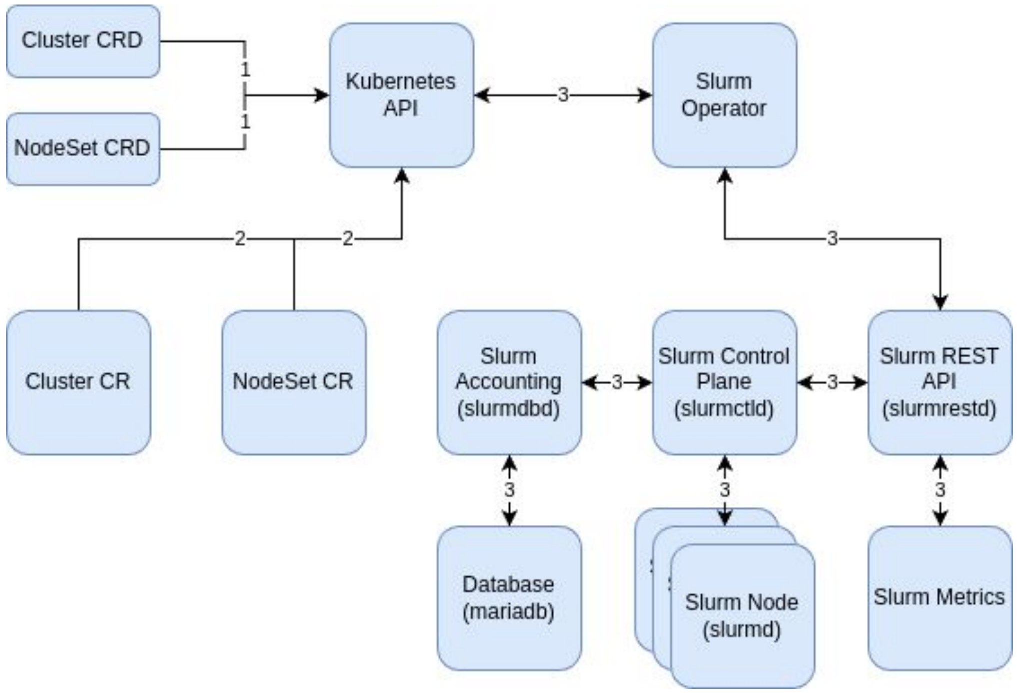
- Metrics collected from Slurm cluster
  - Uses the Slurm client for communication
- Prometheus collector and exporter
- Consumable by:
  - Horizontal Pod Autoscaler (HPA)
  - Grafana

Slurm Operator

Kubernetes – Big Picture



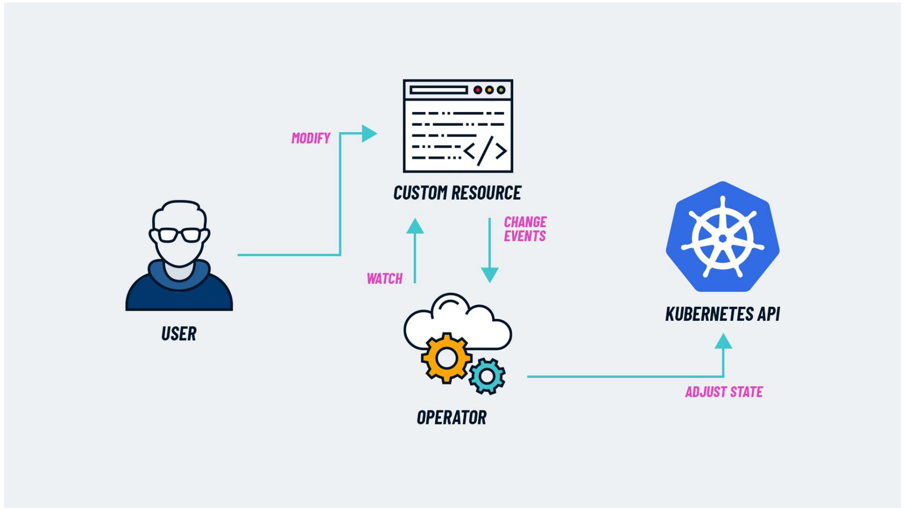
Big Picture



1. Install Slinky Custom Resource Definitions (CRDs)
2. Add/Delete/Update Slinky Custom Resource (CR)
3. Network Communication

Operator Pattern

- Operators are software extensions to Kubernetes that make use of *Custom Resources (CRs)* to manage applications and their components.
  - *CRs* are extensions of the Kubernetes API.
- Operators follow Kubernetes principles, notably the *control loop*.
  - In robotics and automation, a *control loop* is a non-terminating loop that regulates the state of a system.

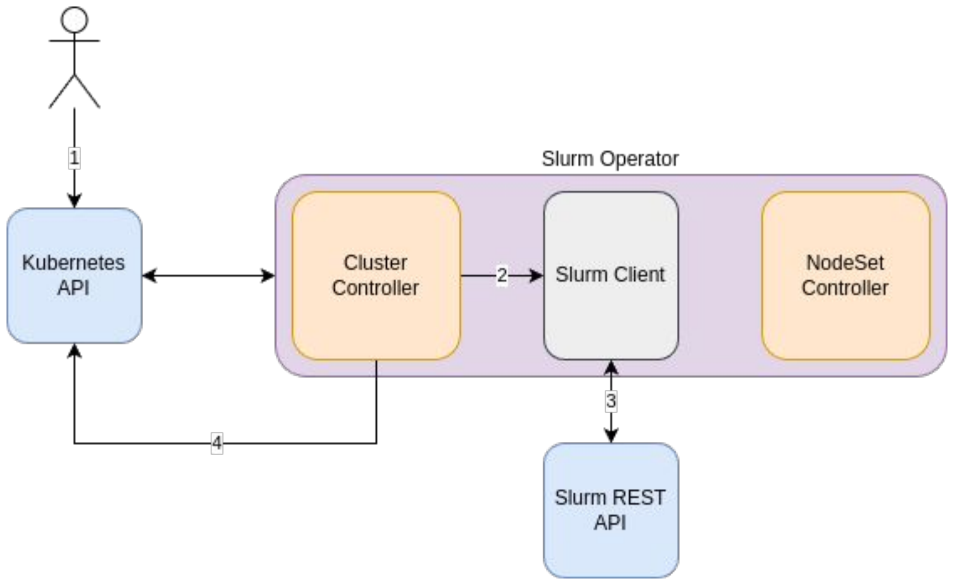


Custom Resources

- Cluster CR
  - Represents a Slurm cluster, by Slurm REST API (slurmrestd)
  - Define server URL and JWT auth token secret
  - Reconciles to internal Slurm client
- NodeSet CR
  - Represents a set of Slurm nodes (slurmd)
  - Define pod spec, Slurm specific options
  - Reconciles to Kubernetes pods

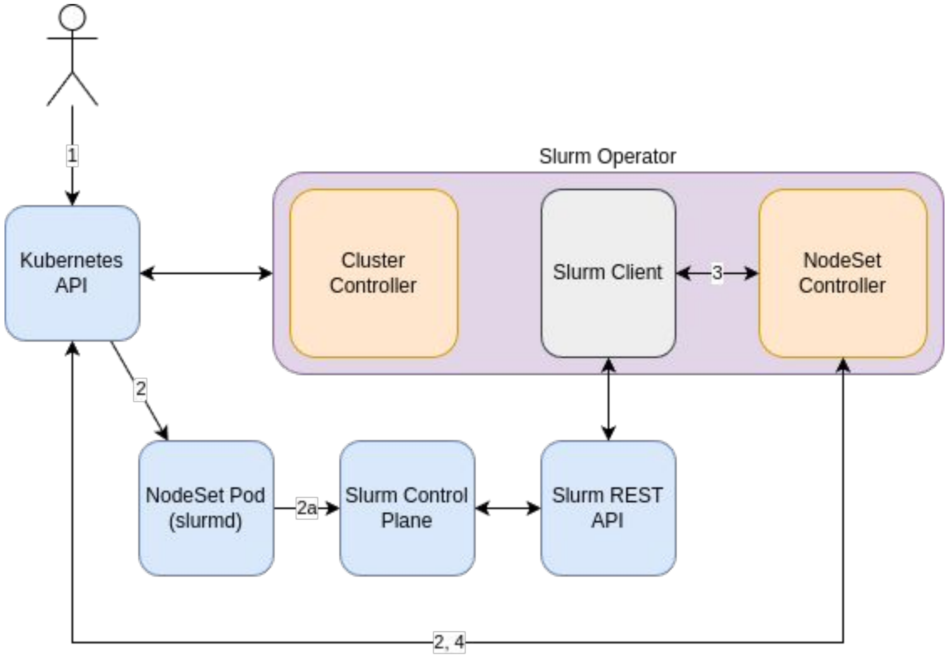
Slurm Operator – Cluster Client

1. User installs a Cluster CR
2. Cluster Controller creates Slurm Client from Cluster CR
3. Slurm Client polls Slurm resources (e.g. Nodes, Jobs)
4. Update Cluster CR Status



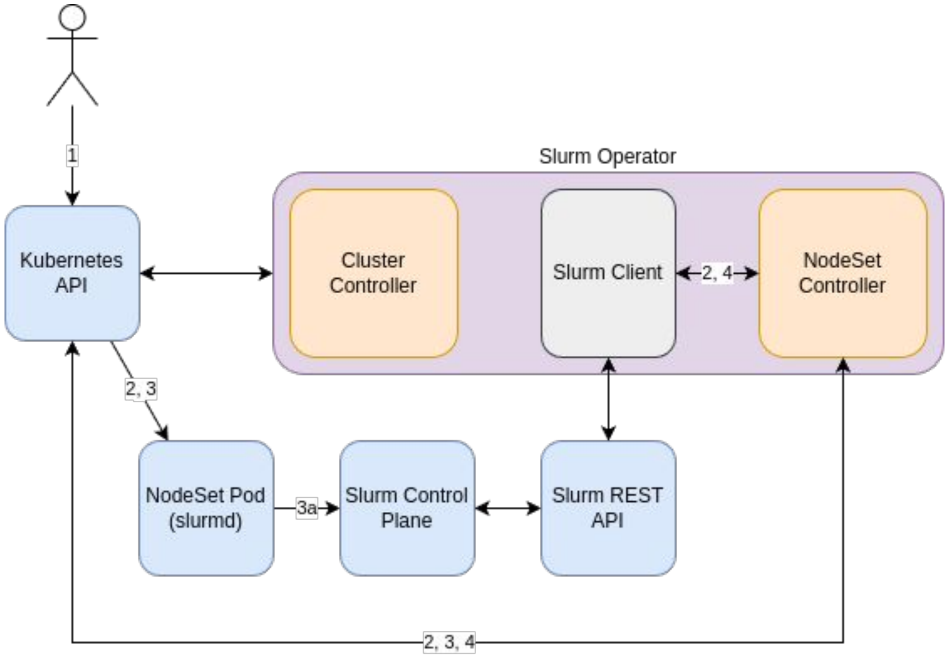
Slurm Operator – NodeSet Scale-Out

1. User installs NodeSet CR
2. NodeSet Controller creates NodeSet Pods from NodeSet CR pod spec
   a. On process startup: the `slurmd` registers to `slurmctld`
3. Update NodeSet CR Status
   a. Kubernetes NodeSet Pod Status
   b. Slurm Node Status



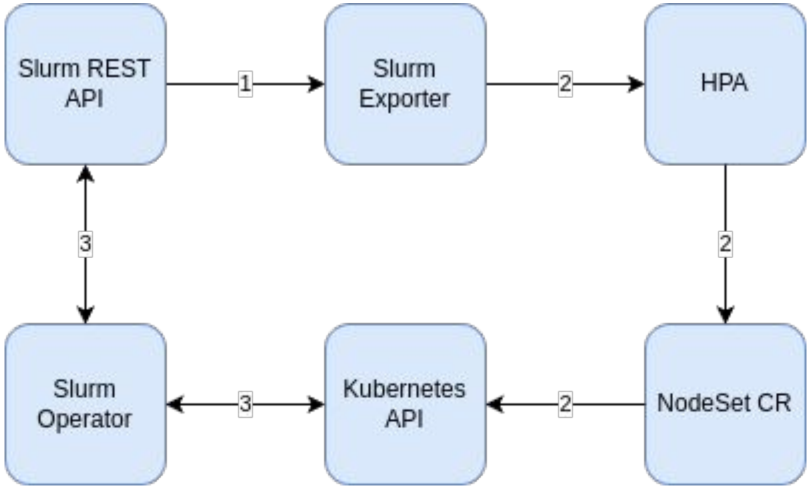
Slurm Operator – NodeSet Scale-In

1. User updates NodeSet CR replicas
2. NodeSet Controller cordons NodeSet pod scale-in candidates:
   a. Candidates are determined based on Slurm node and job information
   b. Cordoned pods will be drained in Slurm, in preparation for safe termination and deletion
3. NodeSet Controller terminates NodeSet pod after fully draining a candidate
   a. On pod preStop: Slurm node deletes itself from Slurm
4. Update NodeSet CR Status
   a. Kubernetes NodeSet Pod Status
   b. Slurm Node Status



NodeSet Auto-Scale

1. Metrics are collected and exported
2. Horizontal Pod Autoscaler (HPA) scales NodeSet CR replicas, based on:
    a. Current metrics data
    b. User defined scaling policy
3. The Slurm Operator reconciles the adjusted NodeSet CR replicas value:
    a. Scale-in (replicas reduced)
    b. Scale-out (replicas increased)



Slurm Client

OpenAPI Specification (OAS) – Background

- The OpenAPI Specification (OAS) was donated to the Linux Foundation under the OpenAPI Initiative (originally known as Swagger) in 2015.
- The OpenAPI Specification (OAS) defines a standard, language-agnostic interface to HTTP APIs which allows both humans and computers to discover and understand the capabilities of the service without access to source code, documentation, or through network traffic inspection.
- An OpenAPI definition can then be used by documentation generation tools to display the API, code generation tools to generate servers and clients in various programming languages, testing tools, and many other use cases.

OpenAPI Client Generator

De Facto:

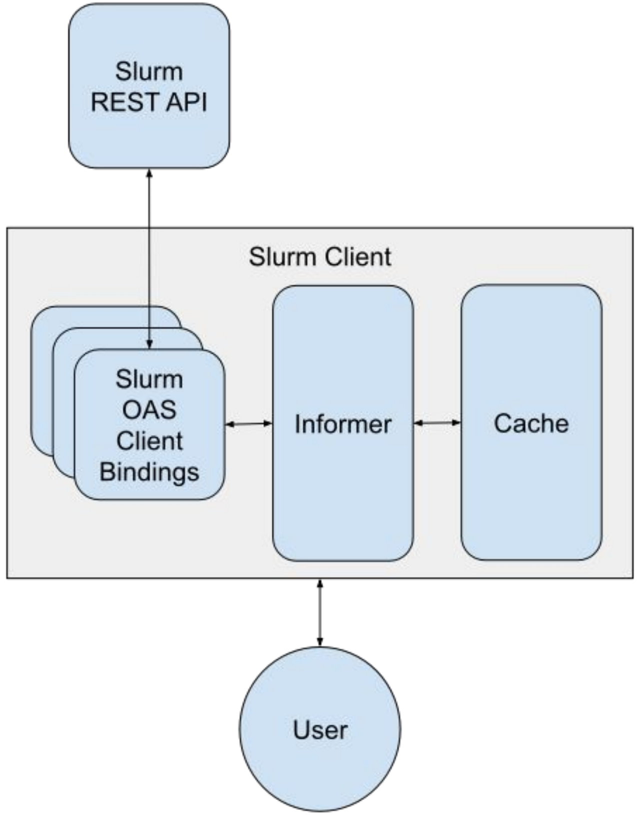
- OpenAPITools/openapi-generator
  - Supports multiple languages
  - Written in Java
- OpenAPITools/openapi-generator-cli
  - Supports multiple languages
  - Written in TypeScript, wraps OpenAPITools/openapi-generator

Community:

- oapi-codegen/oapi-codegen
  - Supports only Golang
  - Written in Golang
  - *slurm-client* uses this

Slurm Client – Architecture

- Generate OpenAPI Spec (OAS) bindings for each Slurm versioned endpoint
  - v0.0.40
  - v0.0.41
  - v0.0.42
- Informer polls the Slurm REST API and stores Slurm object data in Cache
  - Mitigates overloading Slurm REST API when querying the same data
  - Similar to how the Kubernetes client library functions
- Can avoid Cache
  - POST, DELETE
  - GET + client option



Slinky Roadmap

Future Directions

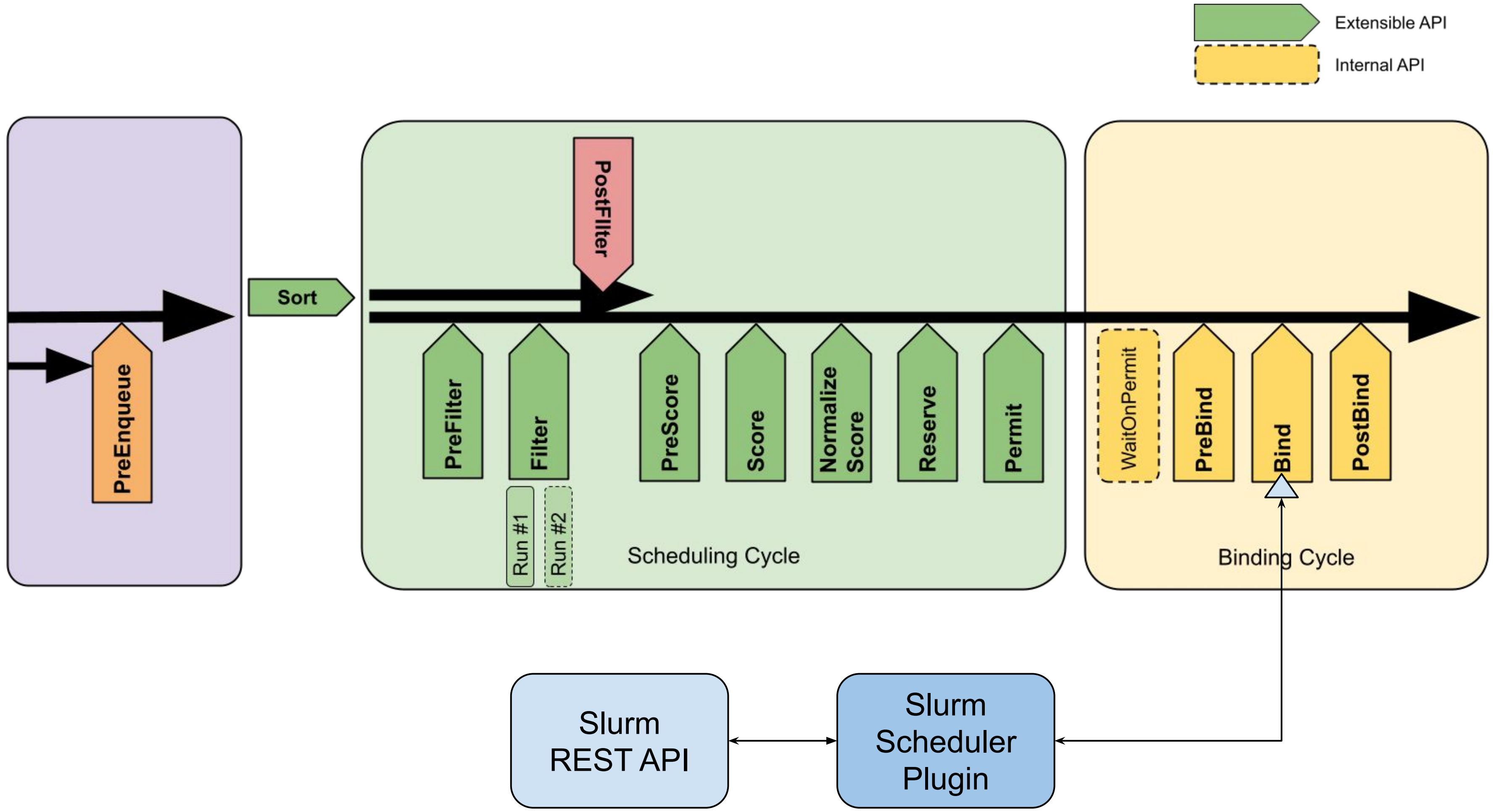
- Next major development goal is the Slurm Bridge
    - Slinky's Kubernetes Scheduler plugin
        - Schedule both Slurm jobs as well as Kubernetes jobs on the same hardware
    - Target is Spring '25, ahead of KubeCon Europe
    - Will require building a DRA CPU Core management plugin
        - Can operate without, but requires limiting nodes to running either K8s pods or Slurm jobs, not both simultaneously

DRA for Cores

- "Dynamic Resource Allocation" (DRA) is an API to request and reserve specific resources within a Kubernetes node
    - Used to manage access to GPUs on the node
    - Plugins can be added to control additional resources
        - Intent is to add Core management through this interface
            - Giving the Slurm Bridge a way to communicate core allocations for the Kubernetes jobs
                - And avoid contention between Slurm vs Kubernetes jobs sharing a compute node
        - SchedMD working with partners to get this built as an out-of-tree driver
            - Want to get this added as a central capability in a future K8s release

Slurm Bridge Design

- Translate K8s pods into "placeholder" Slurm jobs
  - Automatically translate resource requests
    - Core count, memory amount, GPUs
  - Support custom annotations for Slurm-specific settings
    - Such as the partition, account, QoS, time limit
- When the placeholder job is scheduled, inform the Kubernetes scheduler API of the node placement
  - Inject resource claims for DRA
    - For GPUs
    - And - once developed - for Cores


Extensible API
Internal API
PreEnqueue
Sort
PostFilter
PreFilter
Filter
Run #1
Run #2
PreScore
Score
Normalize Score
Reserve
Permit
Scheduling Cycle
WaitOnPermit
PreBind
Bind
PostBind
Binding Cycle
Slurm REST API
Slurm Scheduler Plugin

Questions?

Shameless Plugs

Slurm at SC'24

- Slurm Booth - #2809
- Slurm Community Birds-of-a-Feather
  - Thursday, 12:15pm, Room B203
  - Slurm Roadmap, Community Survey

SchedMD is hiring

- https://www.schedmd.com/careers/

SCHEDMD

The Slurm Company