FORTH

INSTITUTE OF COMPUTER SCIENCE

Cloud automation and interfacing to HPC – What happens with Data ?

Fotis Nikolaidis
Antony Chazapis
Manolis Marazakis
Angelos Bilas

Per3S
2023

About this presentation

- Objective:
  - Run Kubernetes applications on HPC infrastructure.

- Agenda
  - Background on Cloud/HPC Convergence.
  - Rootless architecture for deploying Kubernetes as Slurm job.
  - Show-case.
  - Key takeaways.

Slurm vs Kubernetes

INSTITUTE OF COMPUTER SCIENCE

- Slurm dominates HPC.
  - Specialized in launching MPI jobs at scale.
  - **Singularity containers is an option, but requires external management.**

- Kubernetes dominates Cloud.
  - Specialized in **container management** (i.e healing, scaling, replication).
  - Originally designed for DevOps, is now getting traction for **repeatable data science.**

Convergence of HPC and Cloud-native

INSTITUTE OF COMPUTER SCIENCE

Why should I care ?

Reasons for Convergence

INSTITUTE OF COMPUTER SCIENCE

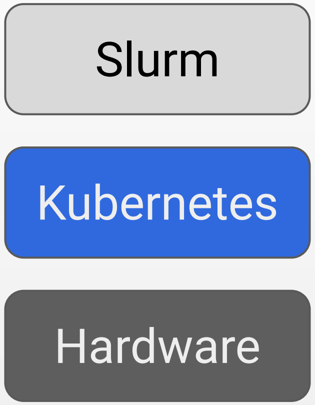
1. **Cloud-User PoV: Scale-out workflows written for Kubernetes.**
    a. Genomics and Bioinformatics
    b. ML Training

2. **HPC-user PoV: Exploit Cloud-native Data Science Tools.**
    a. Combine HPC codes with Cloud-native data analysis → Visualization, querying, ...
    b. Interactive code execution → Jupyter
    c. Workflow management → Argo Workflows, Apache Airflow, …

3. **HPC-center PoV: Increase utilization of data center.**
    a. Attract Cloud users

Architectures for Convergence

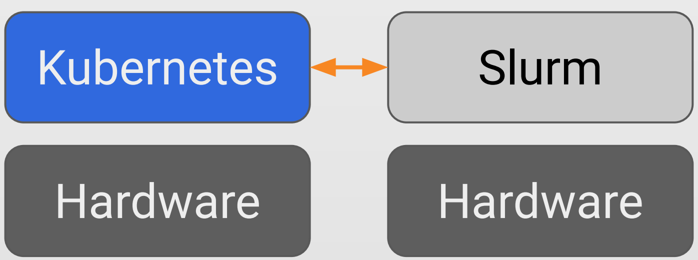
FORTH
INSTITUTE OF COMPUTER SCIENCE



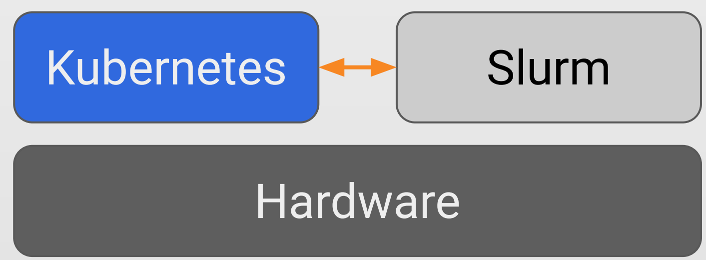
Case: Run MPI jobs on Cloud resources.

Today's Topic!!!

Case: Run Kubernetes on HPC cluster.



Case: Bridge different clusters.



Case: Support both, avoid data transfers.

Design Goals

Objective: Run K8s workloads within **unmodified** Slurm environment.

Requirements:

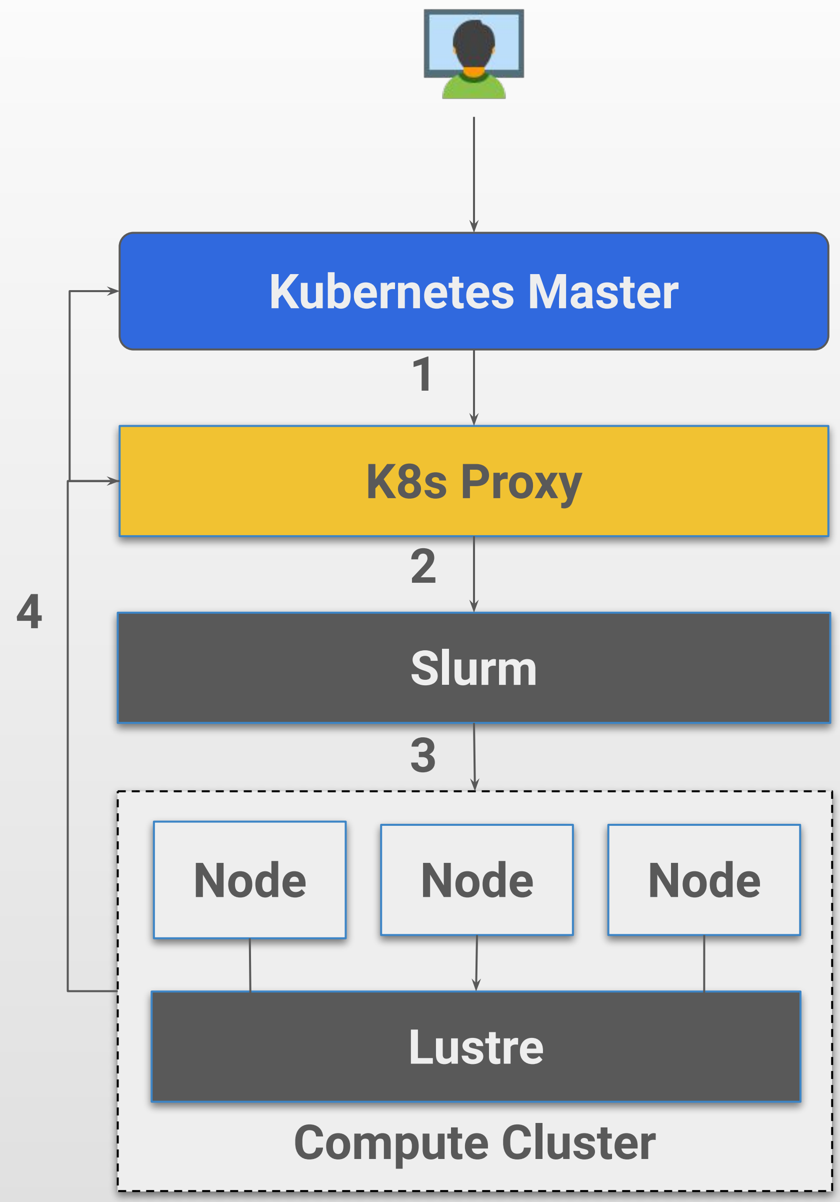
1. Create ephemeral k8s clusters as **user jobs.**
2. Support all Kubernetes abstractions, except privileged.
3. Scale across all nodes of the cluster.
4. Minimal pre-installed software.

Run Kubernetes On HPC Cluster

Top-level view

Four-step process.

1. Proxy receives Job Request from Kubernetes.
2. Proxy translates K8s Job to Slurm Job.
3. Slurms runs the job **(as the host user).**
4. Proxy keeps Slurm and Master in sync.



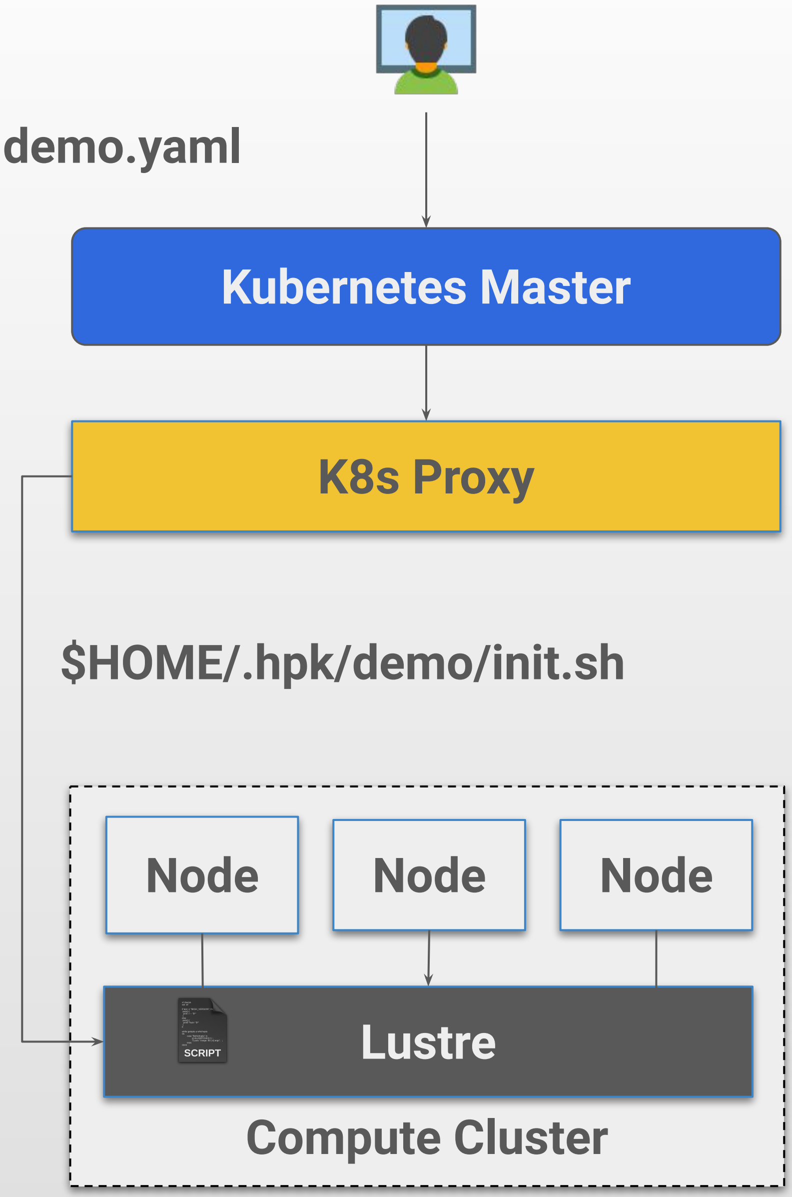
Job Translation

FORTH INSTITUTE OF COMPUTER SCIENCE

kubectl demo.yaml

K8s Proxy:
1. Parses YAML fields from k8s requests.
2. Generates equivalent sbatch scripts.
3. Stores the sbatch script (***init.sh***) in Lustre.

Supported Fields:
- Container image to run
- Resource Requirements
- Volumes to be mounted
- … and more …



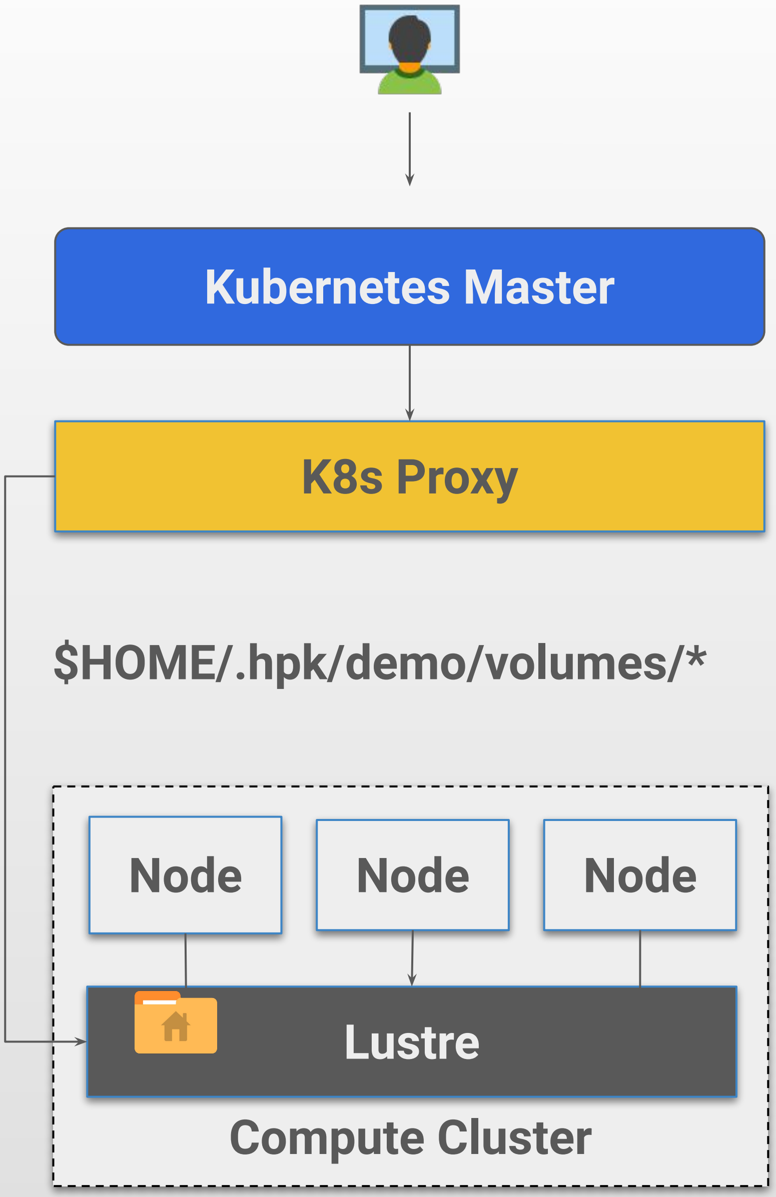
Volume Preparation

K8s requires certain data to be present.
- Credentials, Configs, …

Application needs scratch space for the runtime.
- Logs, temporary files, …

User needs access to persistent storage.
- Load dataset, write results.

K8s Proxy:
1. Convert volumes into Lustre files/dir.
   a. Downloads volumes data from master.
   b. Create temp dir for scratch.
   c. Create symlinks to other Lustre files.
2. Set the volume paths in **init.sh**.



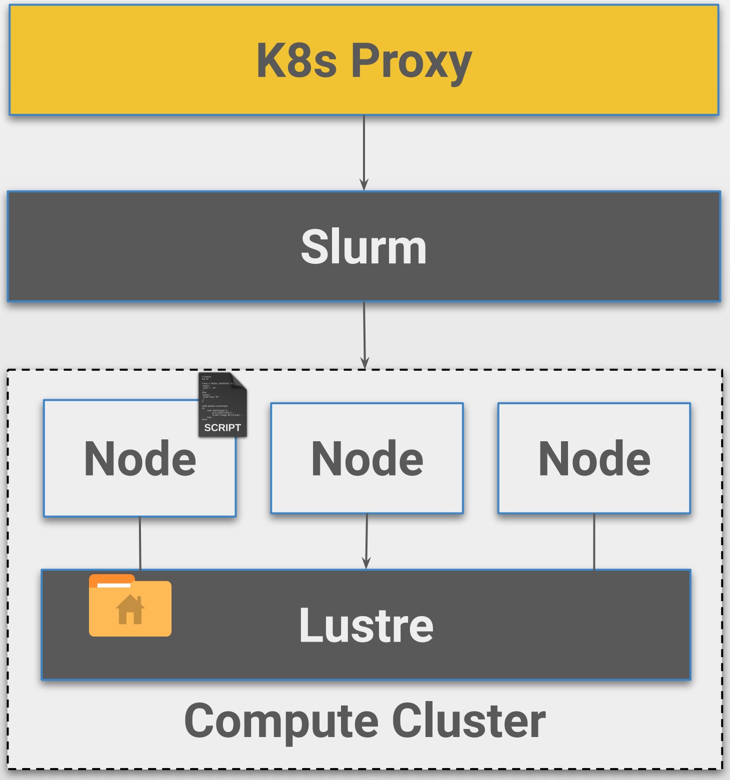
Job Execution

K8s Proxy: Submits ***/demo/init.sh*** to Slurm.

Slurm: Schedules job to the compute nodes.

Compute node: Runs the init.sh locally.

Init.sh: Mounts ***/demo/volumes/**** to the container.



Job Monitoring

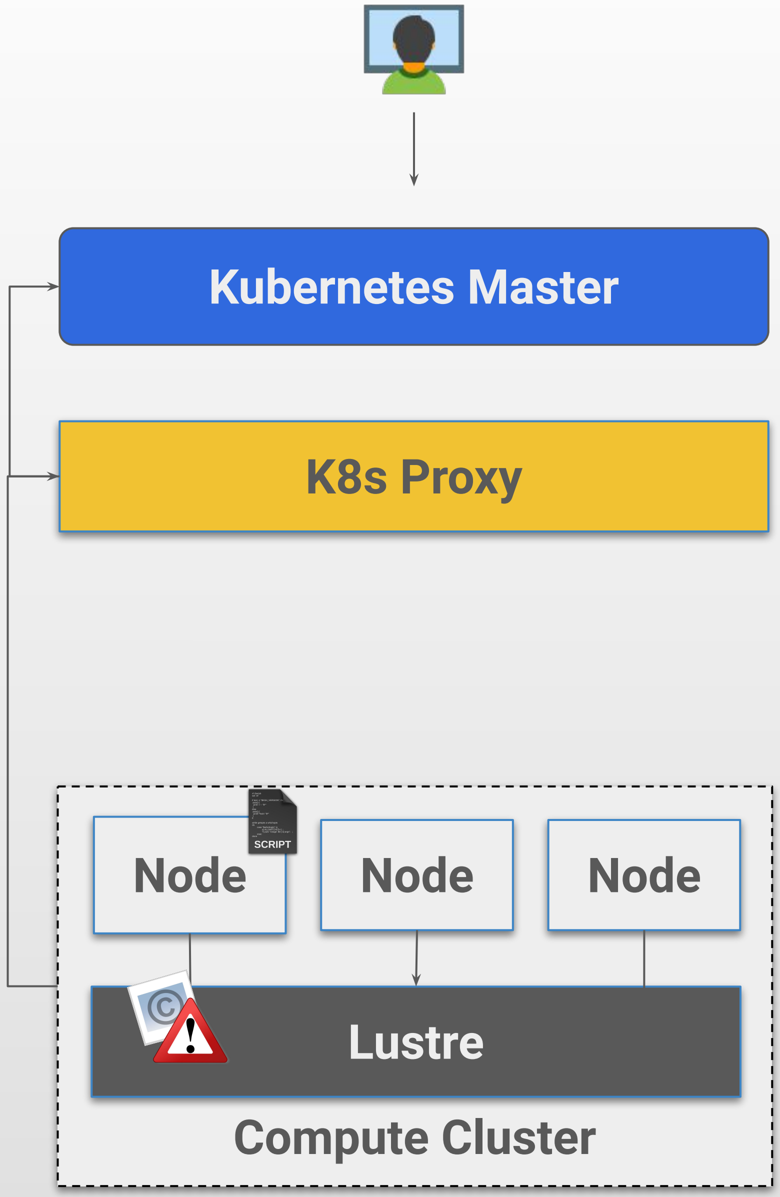
FORTH
INSTITUTE OF COMPUTER SCIENCE

How does Kubernetes now the status of the job ?

Init.sh
Through its execution, it creates *Control Files* on Lustre to indicate the state of execution.

K8s Proxy:

- Tracks changes on *Control Files* by using an Inotify-like mechanism.
- Updates the job's status according to defined semantics.
- Updates Kubernetes master about the new status.

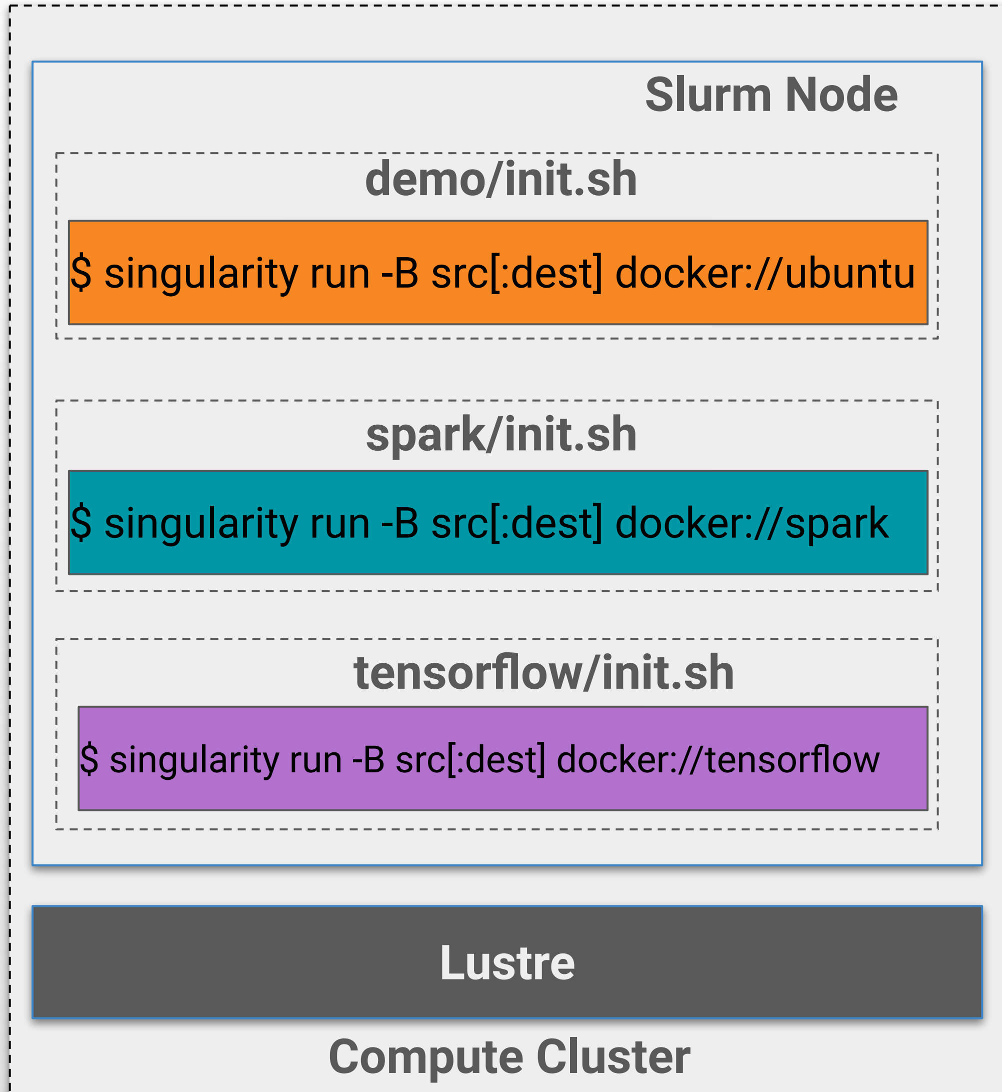


How a Job looks like

INSTITUTE OF COMPUTER SCIENCE

Singularity is the reference container technology for HPC.

- Daemonless: just a binary.
- Rootless: runs as a simple user.
- Integrated: inherits user's env and mountpoints.
- Backward-compatible with Docker images.



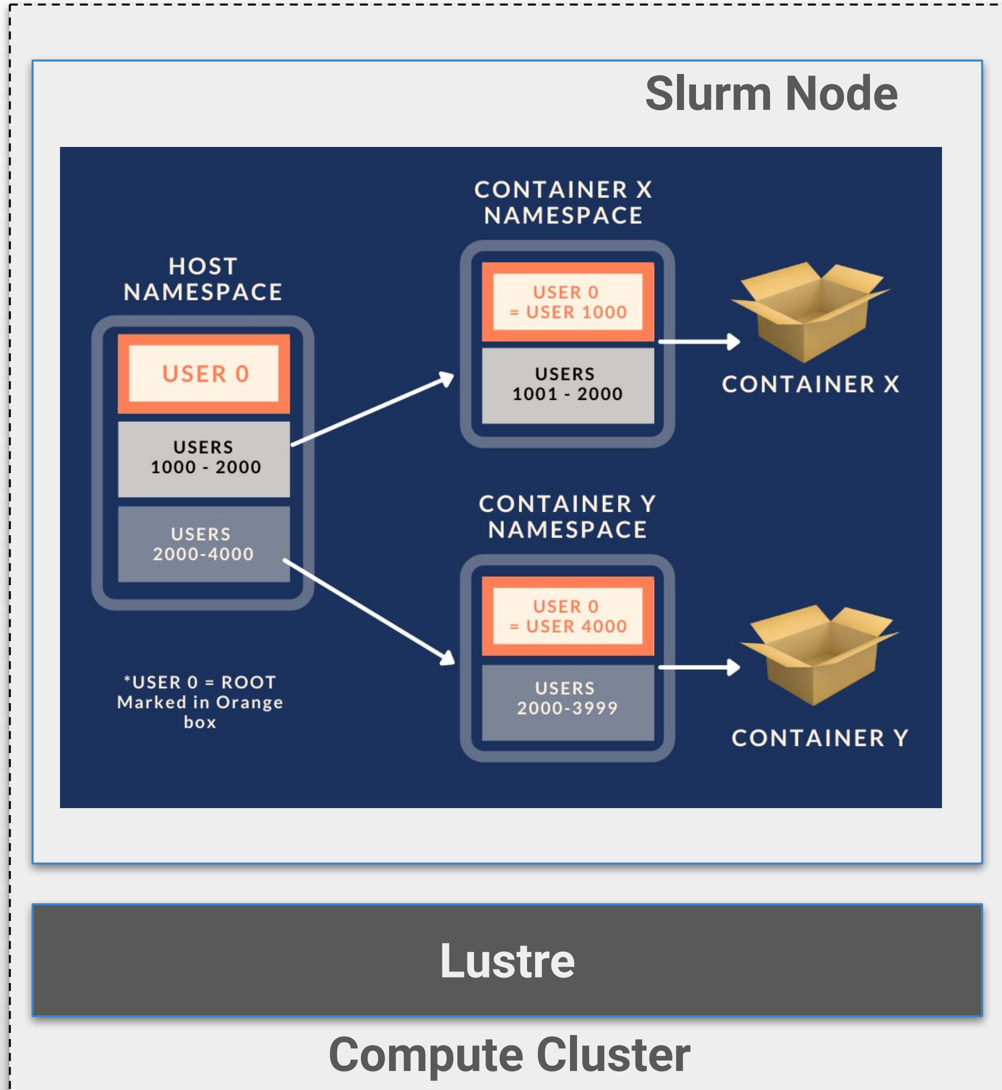
Rootless containers

Execute container runtimes as an unprivileged user, by using Linux User Namespaces.

User Mapping: map UIDs/GIDs in the container namespace to unprivileged range in the host namespace.

- Within container: *root:root*
- Outside container: *user:group*

Fakeroot Capabilities: Full capabilities, except for inserting kernel modules, rebooting, …



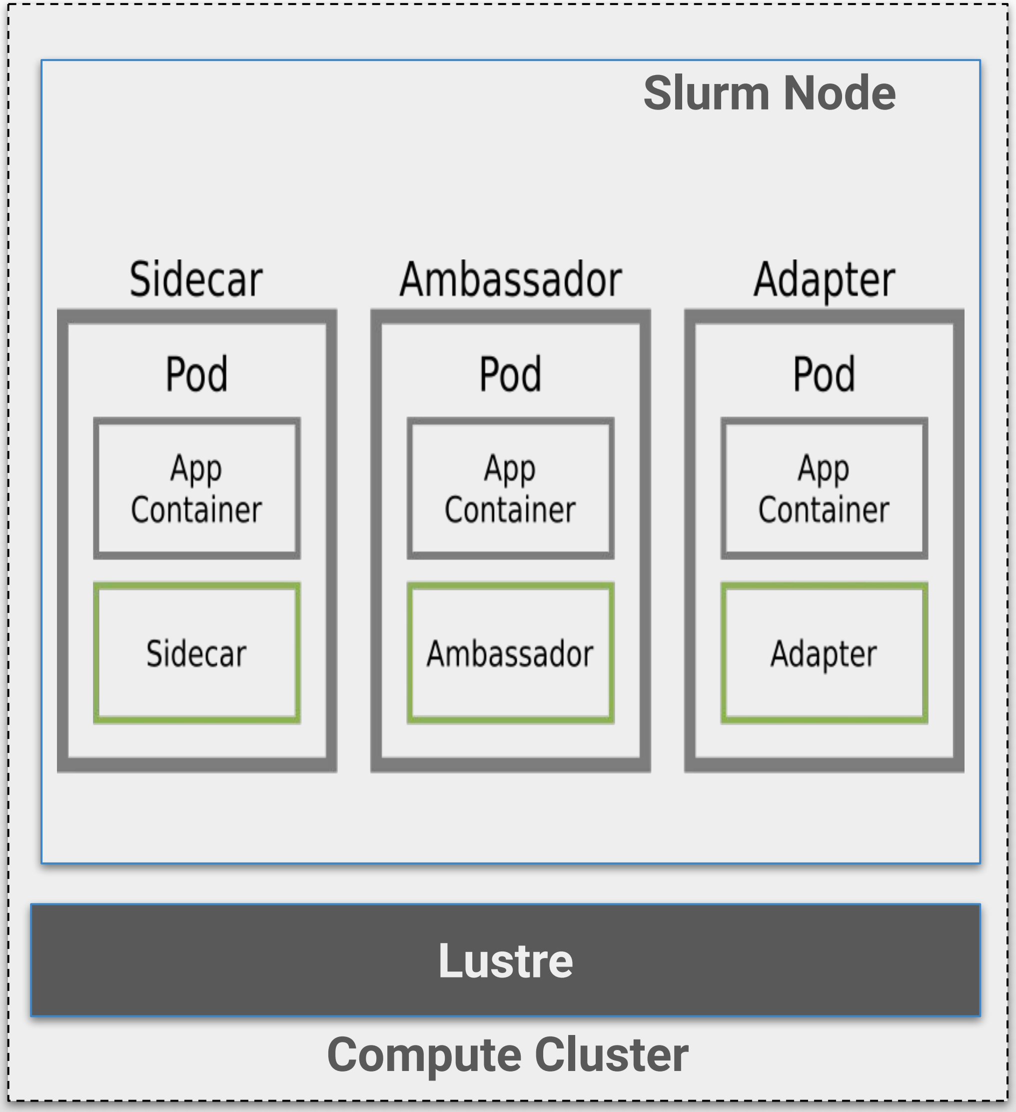
… But not exactly what we want

INSTITUTE OF COMPUTER SCIENCE

Kubernetes attributes its success to the concept of Multi-Container Pods.

The Pod is Logical group of containers with shared storage and network resources.

Issue: How can we support Pods with Singularity ?

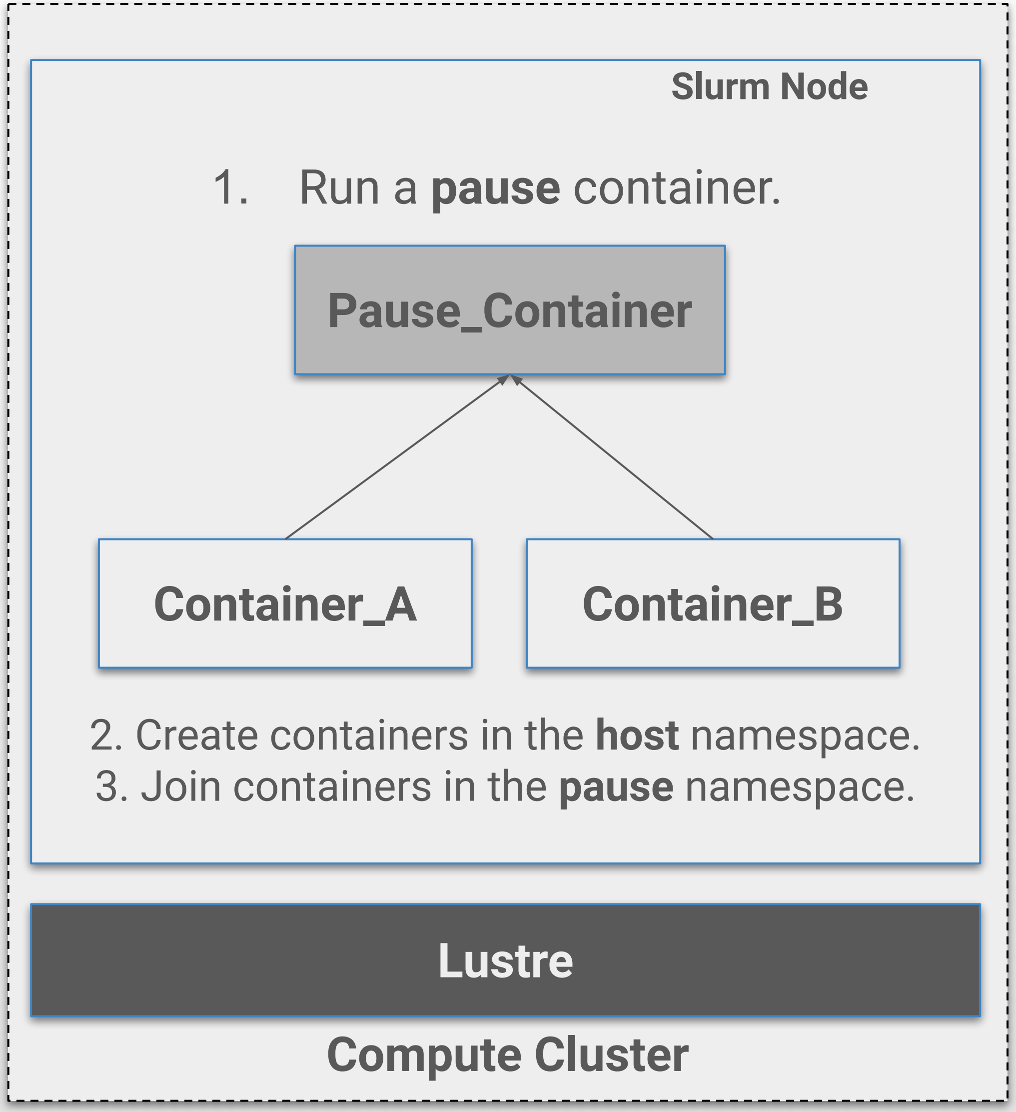


Rootful Pods (Docker)

Pause container:
Empty container which establishes namespaces and reservations before individual containers are created.

Main Containers:
Containers are created on the host namespace, and then join the namespace of the pause container using **nsenter** command.

Issue: nsenter requires root.



Rootless Pods (nested containers)

Step 1:

Run a pause container.

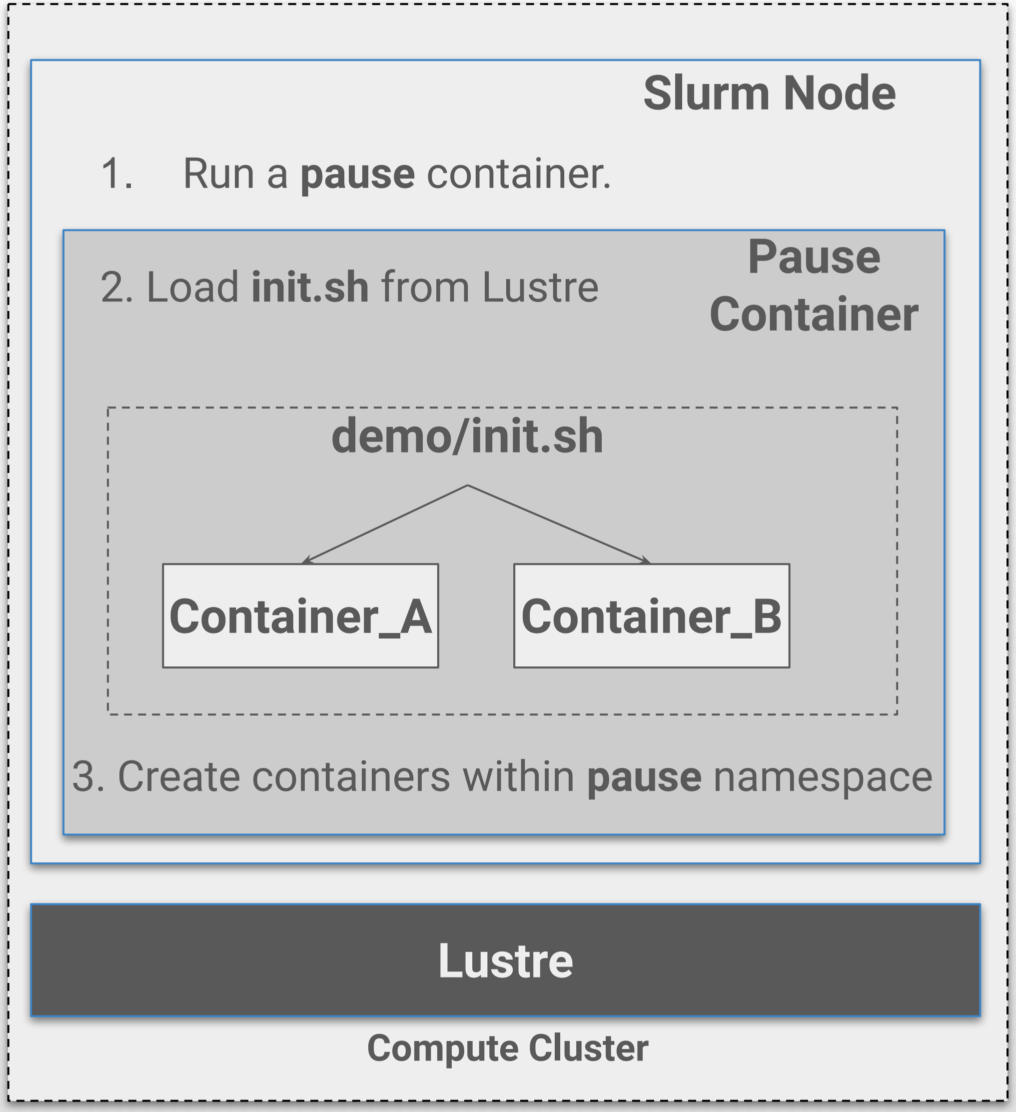
Step 2:

Within the pause container, load and run **init.sh**.

Step 3:

The script creates the containers.

Limitations: all containers must be known in advance.



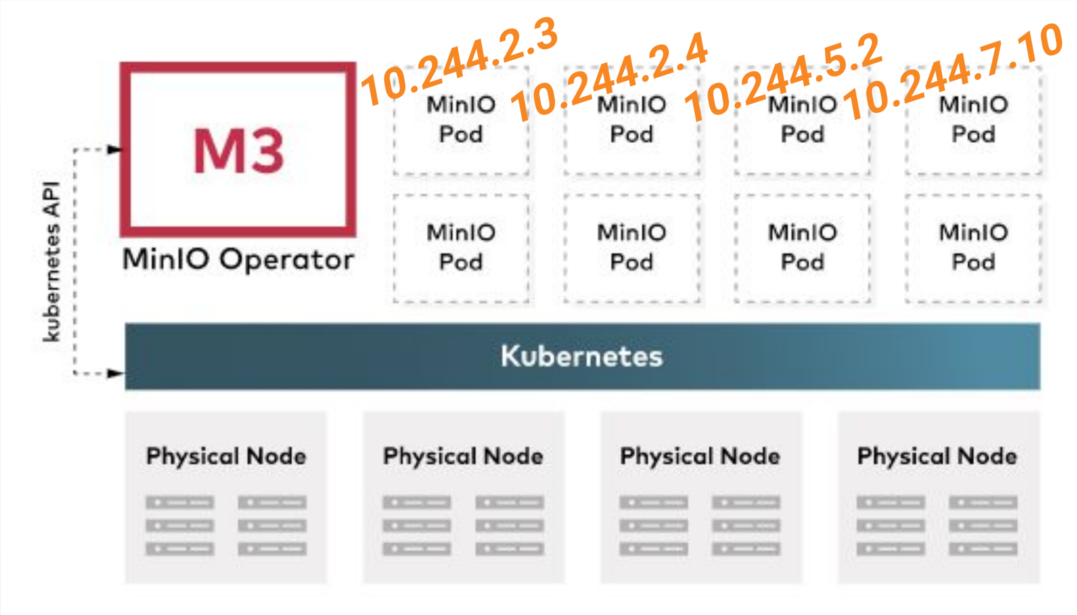
What about object storage ?

Minio is an S3-compatible implementation that is already Kubernetes-native.

Issue #1: Minio, like most other tools, is web-based and requires a routable IP.

Issue #2: You can't pollute the host IP range.

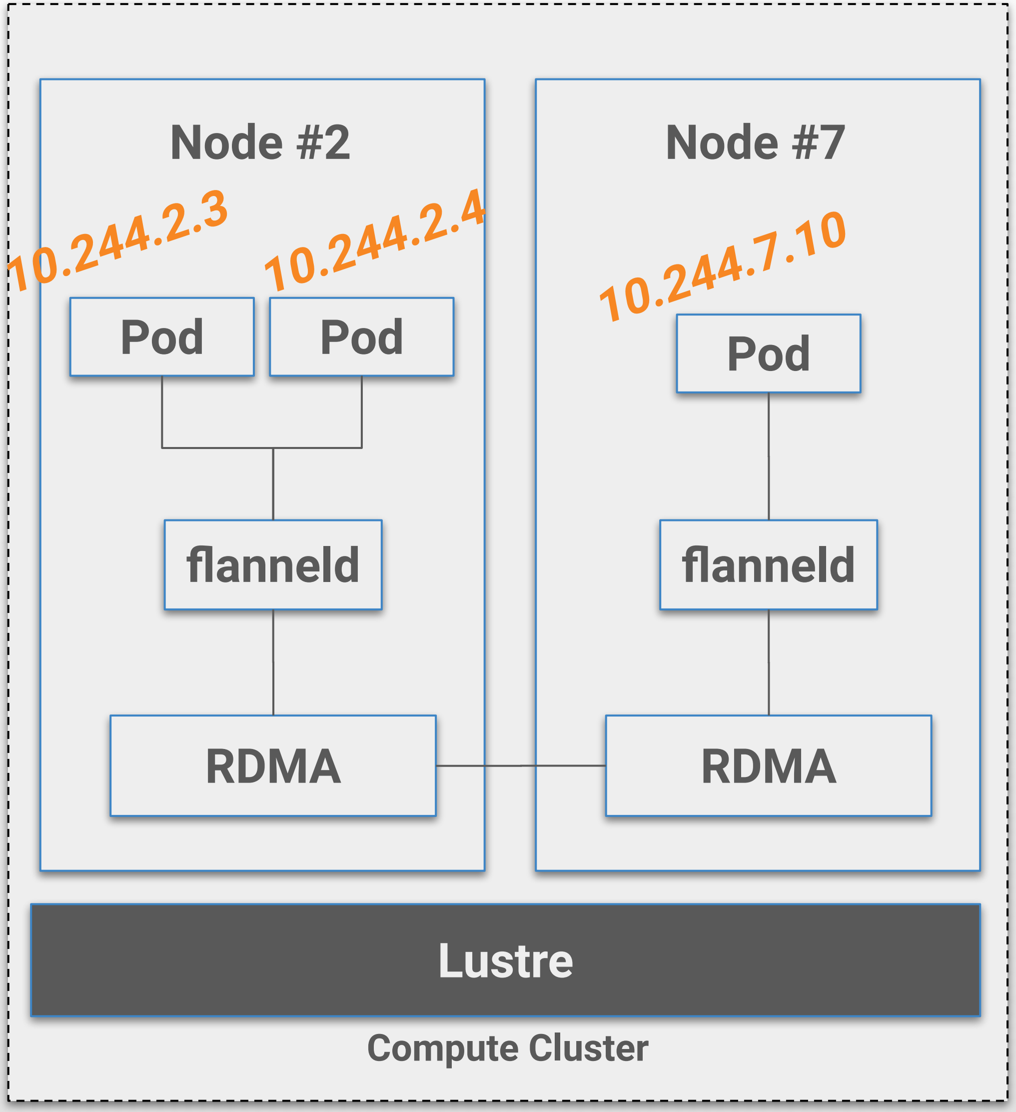
Issue #3: How can Minio servers become discoverable?



Container Networking

INSTITUTE OF COMPUTER SCIENCE

- Within the Pod, we create a network namespace.
- The namespace takes IPs from flanneld that runs on the host.
- Flannel implements container-to-host and host-to-host by modifying the routing tables of the host.

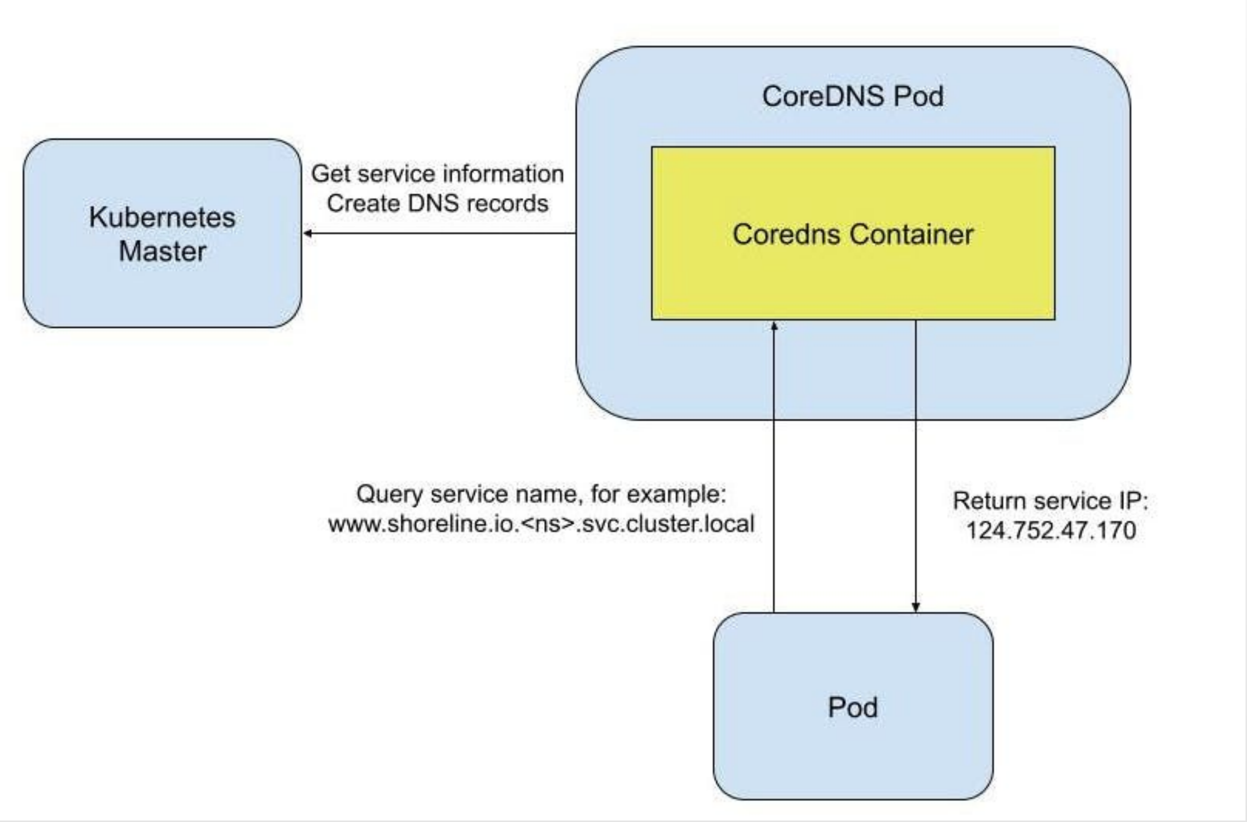


Service Discovery

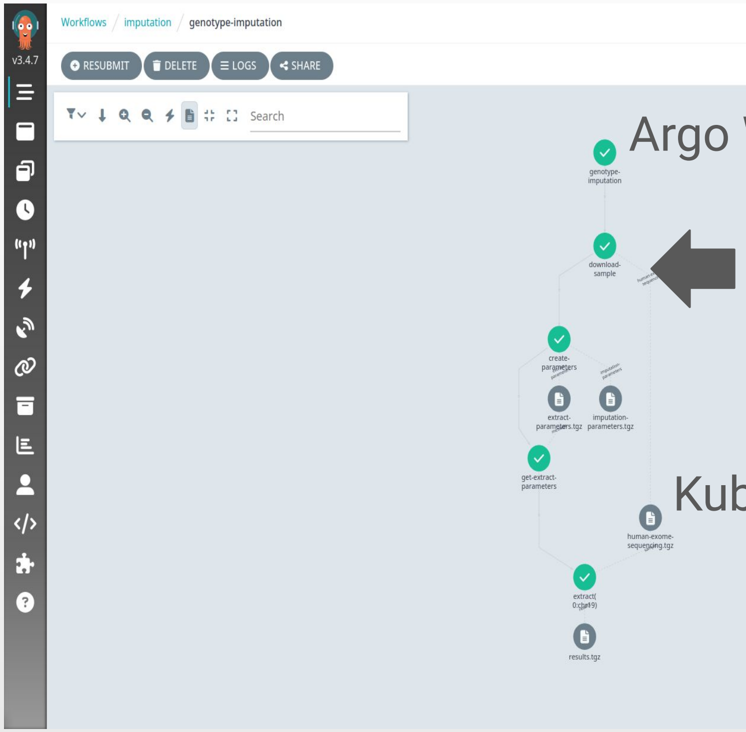
CoreDNS: DNS records for Kubernetes.

- Exclude load balancing as it requires changes on iptables.

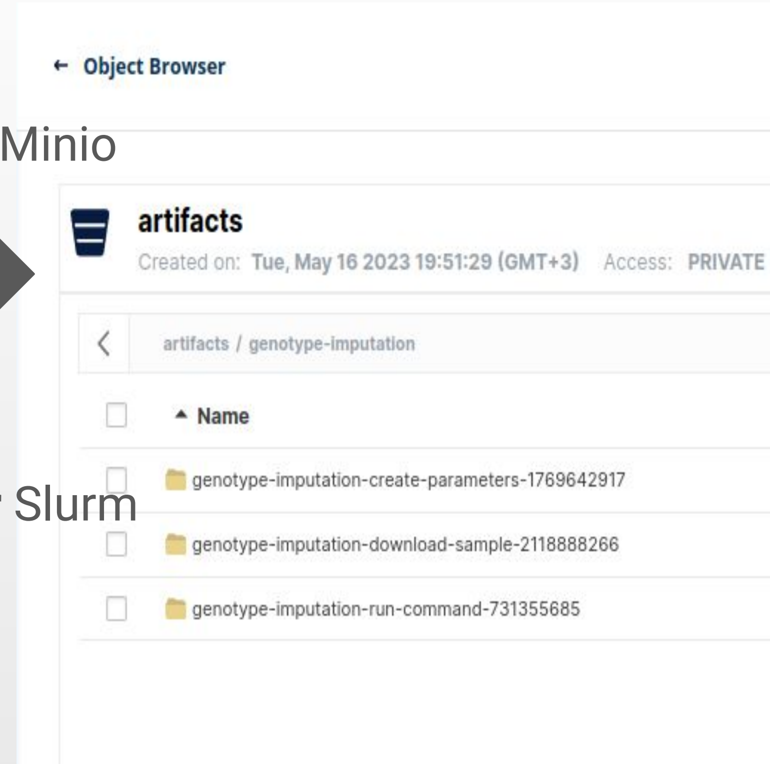
- **Support direct Service-To-Pod mappings.**



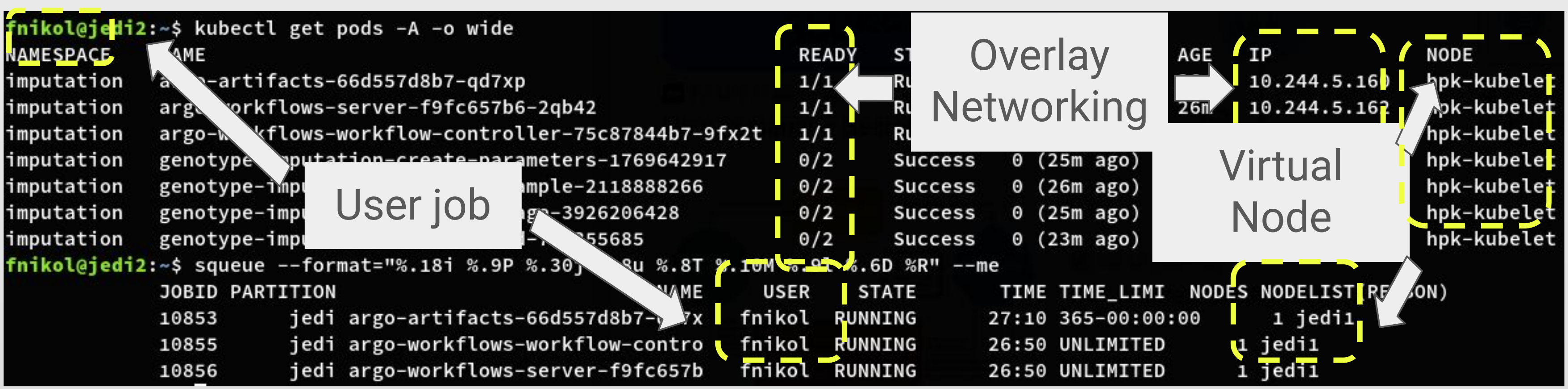
Demo: Genotype Analysis Workflow



Argo Workflows + Minio



Kubernetes **over** Slurm



User job

Overlay Networking

Virtual Node

Key Takeaways

- **Objective:** Run Kubernetes on Slurm.

- **Challenges**
  - Easy Deployment
    - All key components (API server, etcd, CoreDNS, …) are packaged in a container.
  - Rootless execution
    - Implemented using Singularity containers.
  - Pod Support
    - Implemented using nested containers.
  - Network Services
    - Implemented using flanneld.

- **Available at Github:** https://github.com/CARV-ICS-FORTH/HPK

- **System requirements:**
  - Singularity should allow running as fakeroot.
  - Singularity configured with Flannel (or other CNI) for assigning cluster-wide IPs.

Thank you !

We thankfully acknowledge the support of the European Commission and the Greek General Secretariat for Research and Innovation under the EuroHPC Programme through project EUPEX (GA-101033975). National contributions from the involved state members (including the Greek General Secretariat for Research and Innovation) match the EuroHPC funding.

FORTH

User Namespaces

- To allow multi-user mappings, shadow-utils provides `newuidmap` and `newgidmap` (packaged by most distributions).
  - SETUID binaries writing mappings configured in `/etc/sub[ug]id`

```
/etc/subuid:
 1000:420000:65536
```

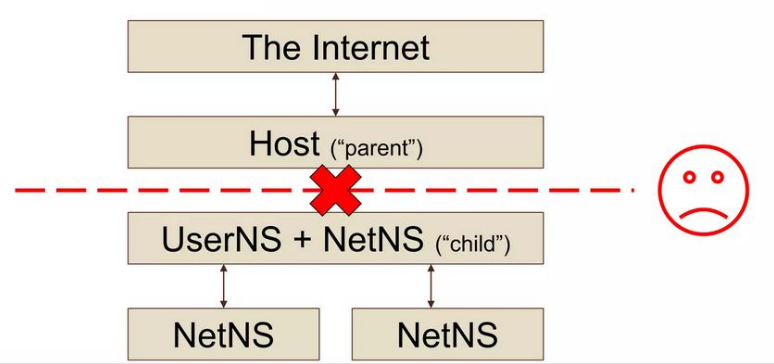
Provided by the admin (real root)

```
/proc/42/uid_map:
   0    1000      1
   1  420000  65536
```

User can configure map UIDs after unsharing a user namespace

Challenge: Networking

- An unprivileged user can create network namespaces along with user namespaces
  - For iptables, VXLAN, abstract socket isolation…
- But an unprivileged user cannot set up `veth` pairs across the host and namespaces, i.e. No internet connection
  - User-mode network stack ("Slirp") can be used instead



Challenge: Networking

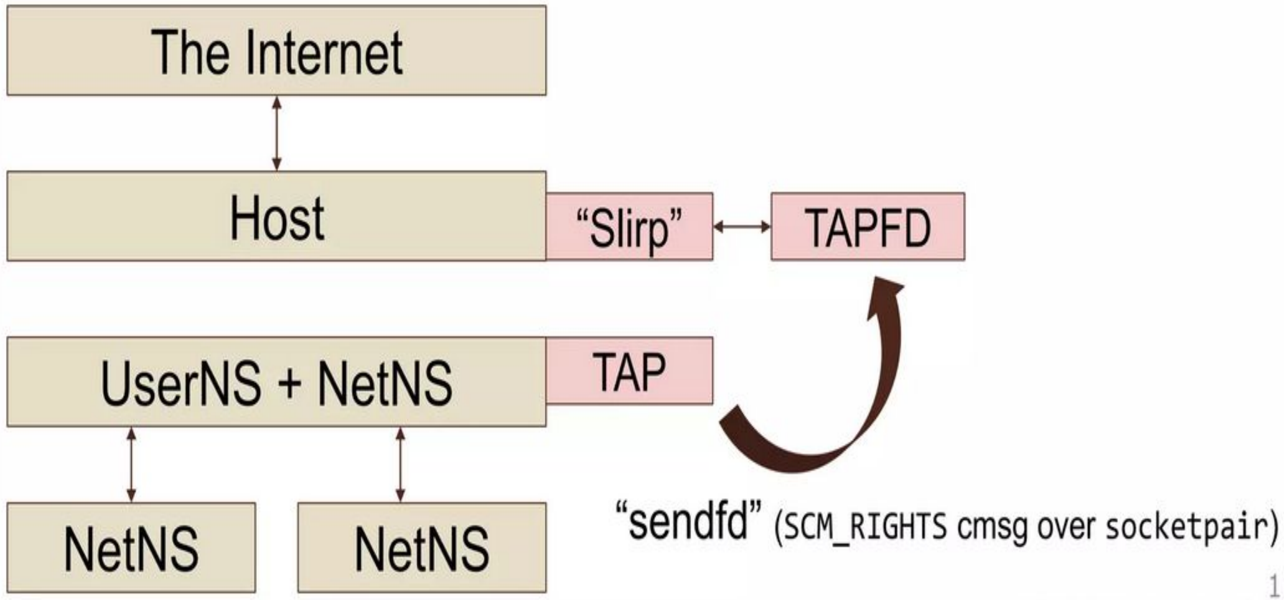
INSTITUTE OF COMPUTER SCIENCE

- **Prior work: LXC uses SETUID binary (`lxc-user-nic`) for setting up the VETH pair across the parent and the child namespaces**

- **Problem: SETUID binary can be dangerous!**
  - CVE-2017-5985: netns privilege escalation
  - CVE-2018-6556: arbitrary file `open(2)`

Challenge: Networking

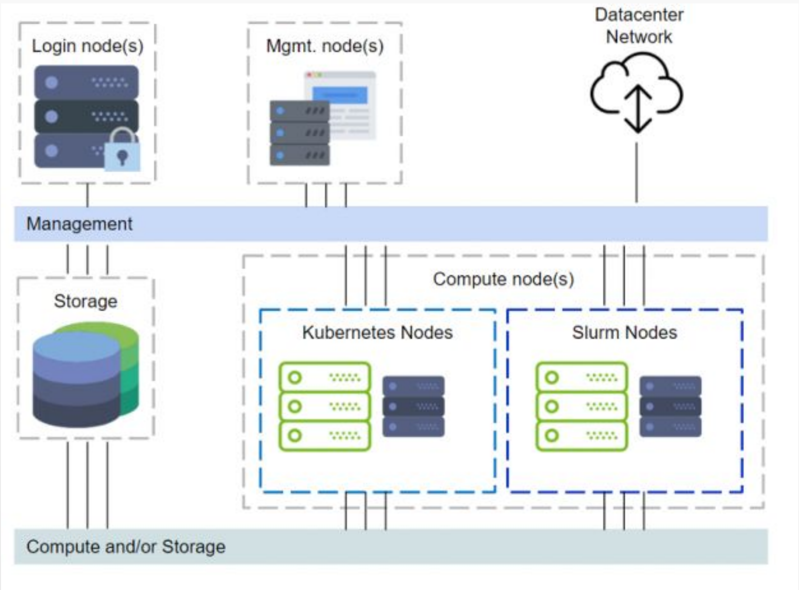
- **Our approach: use usermode network (“Slirp”) with a TAP device (https://github.com/rootless-containers/slirp4netns)**
  - Similar to `qemu -netdev user`
  - Completely unprivileged
  - iperf3 benchmark on Travis: 9.21 Gbps



NTT

11

“Side”

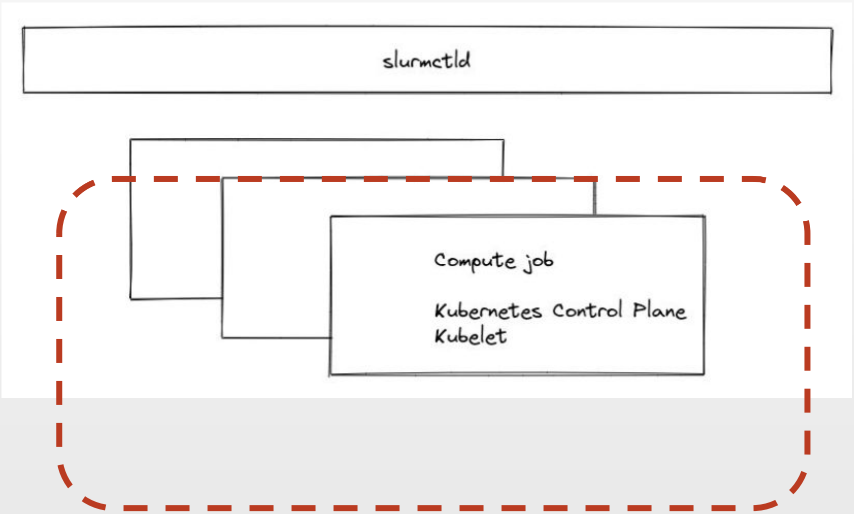


Setup: Partition K8s and Slurm nodes.[1]

- Pros:
  - Full access to k8s capabilities
  - Full access to Slurm capabilities
- Cons:
  - Poor Interfacing between k8s and Slurm.
  - Data transfers from one partition to another.
  - Doubles the maintenance cost.

[1] **KNoC is a Kubernetes node to manage container lifecycle on HPC clusters** (InteractiveHPC 2022) https://github.com/CARV-ICS-FORTH/knoc

“Under”



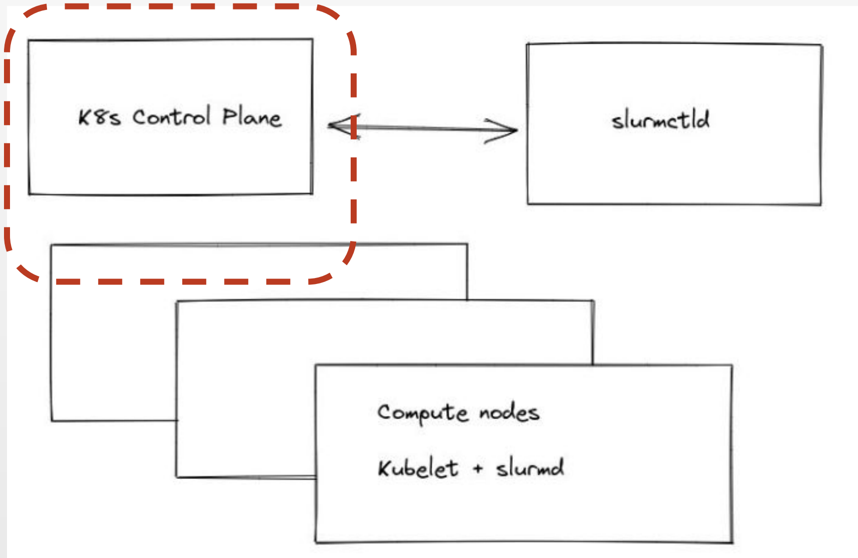
Setup: Run Slurm cluster(s) within a K8s environment.[2]

- Pros:
  - Elastic use of Cloud resources.
  - Portability of HPC solutions across Cloud.
  - Traditional experience for Slurm users.

- Cons:
  - Does not address the site underutilization issue.

[2] **Genisys is a Kubernetes scheduler for running HPC jobs inside Virtual Clusters alongside other services** (VHPC’22) https://github.com/CARV-ICS-FORTH/genisys

“Adjacent”



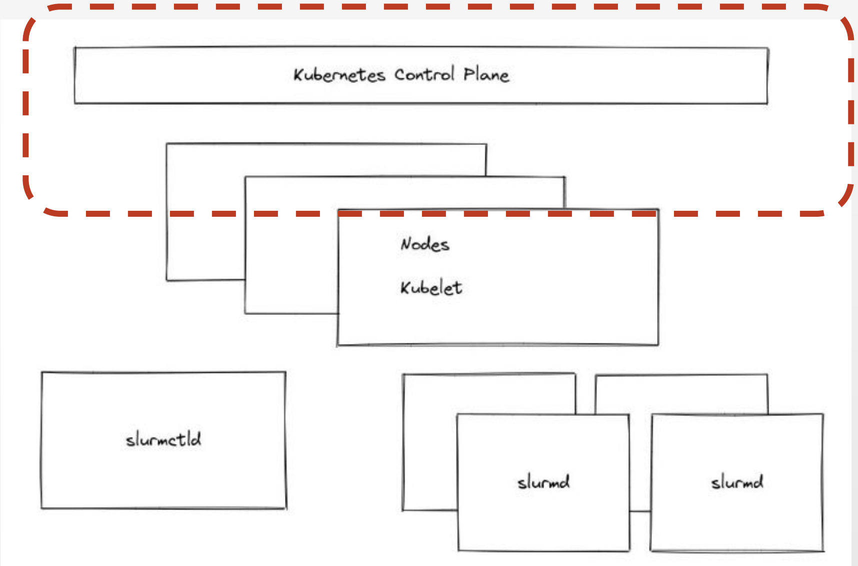
Setup: Run both Slurm and k8s on the same nodes.

- Pros:
  - Full access to Slurm capabilities
  - Full access to k8s capabilities
  - No data transfers required.

- Cons:
  - Security concerns (k8s runs as root)
  - Resource conflicts (nodes vs pods)
  - Increased Maintenance costs

Our vision - “Over”

INSTITUTE OF COMPUTER SCIENCE



Setup: Run K8s cluster(s) within unmodified Slurm environment.

Design Goals:

1. Create ephemeral k8s clusters as **user jobs.**
2. Support all Kubernetes abstractions, except privileged.
3. Scale across all nodes of the cluster.
4. Minimal pre-installed software.