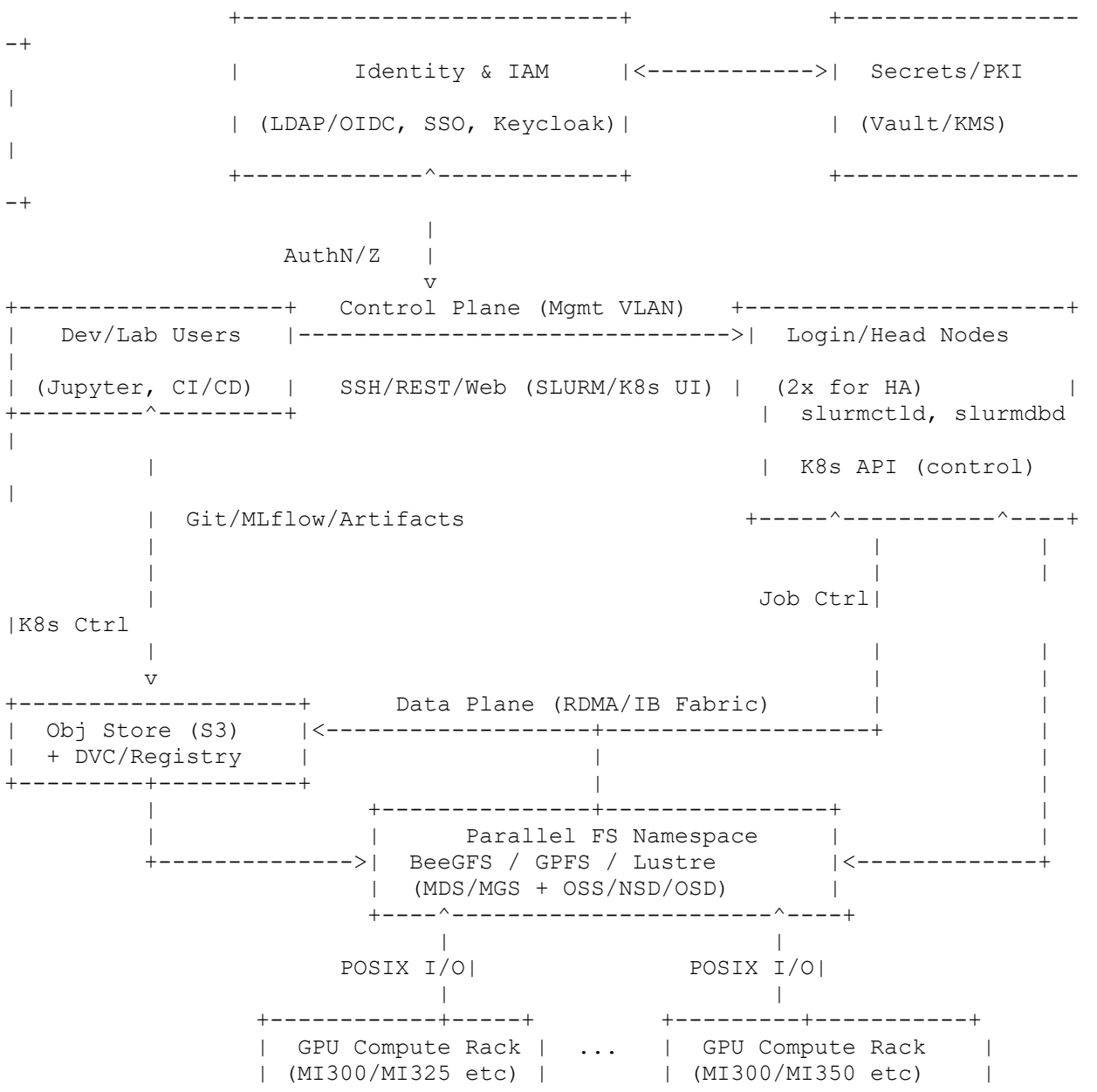
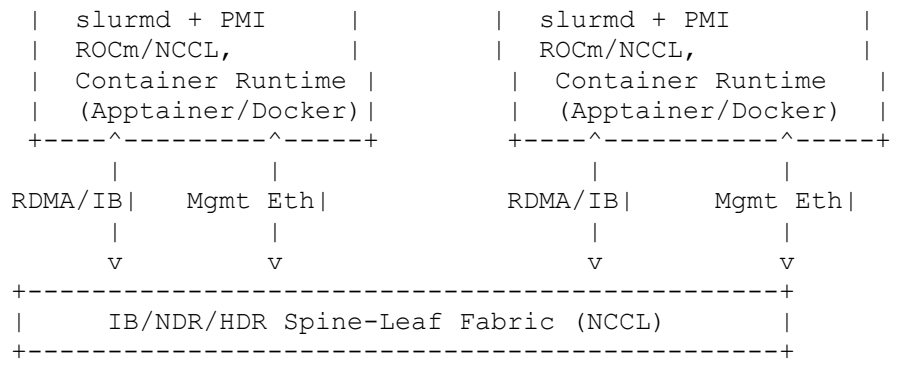
AI Cluster Architecture (SLURM + Kubernetes + RDMA + Parallel FS)

Use this as a quick, whiteboard-ready guide you can sketch in 60–120 seconds during the interview. Keep it simple, label clearly, and narrate the data/control flows.

---

1) High-Level Diagram (ASCII you can replicate fast)





Legend: solid arrows = control/API; bold blocks = data paths; RDMA fabric for NCCL collectives; Mgmt Ethernet for SLURM/K8s control.

---

2) 60-Second Walkthrough Script

- **Users → Head Nodes:** "Researchers submit via SLURM or K8s portals on HA head nodes running `slurmctld`/`slurmdbd` and K8s control plane. IAM handles SSO; Vault manages secrets."
- **Scheduling:** "SLURM owns GPUs using `select/cons_tres` and GRES; K8s is used for pipelines/operators. We integrate via SLURM Operator or run K8s workloads inside SLURM allocations."
- **Data:** "Cold/curated data in S3; hot POSIX via BeeGFS/GPFS/Lustre. Jobs optionally stage to node-local NVMe scratch."
- **Network:** "NDR/HDR InfiniBand spine–leaf for NCCL; GPU Direct RDMA enabled; topology-aware scheduling keeps jobs within a pod/leaf."
- **Observability:** "We track GPU/IB/FS KPIs with Prometheus + Grafana; SLURM accounting informs fair-share and capacity planning."

---

3) SLURM Config Highlights (sound bites)

- `SelectType=select/cons_tres`, `GresTypes=gpu` with per-node `gres.conf` entries (e.g., `Name=gpu Type=mi300 File=/dev/dri/renderD128 Count=8`).
- Partitions per workload (ai-train, ai-infer, hpc), QoS with preemption for urgent jobs.
- Topology plugin + `switches` to prefer single-leaf placement; `--gpu-bind=closest`.
- Accounting: `slurmdbd` + MySQL/Postgres; enforce usage caps and fair-share multifactor.
- Container: `Prolog`/`Epilog` to mount datasets, set NCCL env, and collect job metrics.

---

4) Kubernetes Integration Patterns

Pattern A (K8s-primary): SLURM Operator/Volcano; K8s schedules workflow DAGs, offloads GPU allocations to SLURM.
Pattern B (SLURM-primary): Launch ephemeral K8s namespaces inside SLURM jobs for ML pipelines; preserves HPC env, good for multi-tenant clusters.
GPU mgmt: Expose GPUs to K8s via device plugins; align with SLURM GRES to avoid overcommit. Prefer Apptainer for MPI/HPC compatibility.

---

5) Storage Layout & Data Flow

- **Tiers:** S3/object (cold) → Parallel FS (warm) → Node-local NVMe (hot scratch).
- **Throughput:** Stripe large training data across OSTs/targets (e.g., 8–16); enable metadata striping (Lustre) for small-file directories.
- **Data lifecycle:** Pre-stage with Prolog; purge scratch on Epilog; use ILM (GPFS) or policies (BeeGFS) for tiering.

Training Flow: Loader pulls from POSIX → optional cache on NVMe → GPUs compute → checkpoints to POSIX/S3.

---

6) RDMA & NCCL Tuning Pointers

- IB NDR/HDR, PFC/ECN tuned; enable GPU Direct RDMA.
- NCCL env: `NCCL_IB_HCA`, `NCCL_IB_GID_INDEX`, `NCCL_NET_GDR_LEVEL`, `NCCL_TOPO_FILE` for custom topo.
- Pin ranks to GPUs/NUMA; ensure `hwloc` awareness.

---

7) Security & Multi-Tenancy

- SSO via OIDC; per-tenant accounts, partitions, and storage quotas.
- Network isolation (VRFs/VLANs) between mgmt and data fabrics.
- Image attestation for containers; secrets via Vault; read-only dataset mounts.

---

8) Observability & SLOs (what to say if asked)

- **GPU:** utilization, memory, SM occupancy.
- **Network:** IB port errors, retransmits, fabric load.
- **Storage:** metadata ops/s, read/write BW, client cache hits.

- **Scheduler:** queue wait vs run time, backfill efficiency.

Prometheus exporters (SLURM, node, IB, FS) + Grafana dashboards; alerts on underutilization and tail latencies.

---

9) Capacity & Cost Cheatsheet

- Rule of thumb: plan POSIX BW at ~2–3 GB/s per training node for large models; metadata IOPS sized for loaders.
- At least 1:1 NVMe scratch to active dataset working set per node.
- IB oversubscription ≤ 2:1 for heavy All-Reduce; keep training jobs leaf-local when possible.

---

10) Example Command Snippets (verbal anchors)

- Submit 8-GPU job pinned within one node:
  ```
  srun -N1 --gres=gpu:mi300:8 --gpu-bind=closest --ntasks-per-node=8
  python train.py
  ```
- Multi-node PyTorch:
  ```
  srun -N8 --gres=gpu:mi300:8 --cpus-per-task=8 bash -c 'torchrun --
  nproc_per_node=8 --nnodes=8 --rdzv_backend=c10d --rdzv_id=$SLURM_JOB_ID
  --rdzv_endpoint=$SLURM_NODELIST train.py'
  ```
- Pre-stage dataset (Prolog):
  ```
  rsync /fs/datasets/$JOB_USER/$JOB_ID /local_nvme/datasets
  ```

---

11) Common Pitfalls to Call Out

- Fragmentation from overly specific constraints (features/GPUs) → fix with flexible requests and backfill.
- Storage metadata hot spots → directory sharding, metadata striping.
- K8s and SLURM double-scheduling GPUs → single source of truth for allocation.
- NCCL hanging due to mixed GID or miswired topo → validate with `nccl-tests`.

---

12) 30-Second Closing

“Design keeps SLURM as the GPU source of truth, couples RDMA-aware placement with hot POSIX and node-local scratch, and integrates Kubernetes where it shines: pipelines and

automation. Observability and fair-share policies ensure high utilization without sacrificing researcher velocity.”