FOSDEM 2026

Building Cloud Infrastructure for AI

Dave Hughes — CTO

Lukas Stockner — Principal Engineer

whoami

- Background: software engineer, sysadmin, and network wrangler
- Passionate technologist
- Open-source developer, user and enthusiast
- Problem solver

Dave Hughes
CTO
Stelia

whoami

- Background: rendering, HPC, everything infrastructure
- Got into cloud infra by coincidence, ended up staying
- Passionate about performance and the big picture

Lukas Stockner
Principal Engineer
Stelia

What is a cloud?

What is a cloud?

- Abstraction of resources and infrastructure
- Self-service
- On-demand
- Elastic
- API driven
- Multi-tenant

What is a 'GPU cloud'?

What is a 'GPU' cloud?

- GPUs (though other accelerators will be common in future)
- (typically) RDMA interconnect
- Fast storage
    - Read: not fastest
    - Not necessarily RDMA
- Dense/very big servers
- Three rough scales of consumption
    - 'a GPU' - basic PCIe passthrough
    - 'a GPU node' - typically requires P2P and potentially NVLink or equivalent
    - 'a GPU cluster' - typically requires cross-node interconnect

'GPU clouds': the good, the bad and the ugly

'GPU Clouds': the good, the bad and the ugly

- Bare-metal
    - Purchase the assets — image the assets — Handover
- OpenStack
    - Cloud (like) in nature, though quite complex / tightly coupled
    - Many problems arising from the above
- Kubernetes
    - Give people the ability to run containers
    - No strong isolation of tenants
    - Designed for containers first — shared kernel and drivers etc
    - Various other: network not quite right, ..., etc
- Other
    - Nomad, Triton

How to design a cloud 101

How to design a cloud 101

- Redundancy everywhere
    - Scheduling maintenance windows doesn't scale across many tenants
- Avoid manual steps e.g. bootstrapping, VLAN allocations, ..., etc
- Isolation is critical — for QoS as well as security
- Standardization is key
- Minimize state

Hardware selection

Hardware selection

- CPU compute, storage, control plane etc.
    - Standard OEM kit
- GPU compute
    - Standard 8-way server
        - Mostly identical across vendors
        - Typically overkill specs
    - Mainboard + PLX + GPUs
        - Originated from mining / rendering
    - 'Standard' server with some GPUs
        - Not great from a density perspective
- Note: OCP has been proven to be king at scale — but you need that scale to justify non OEM kit.

Firmware

Firmware

- Ideal world: full control — both for security and scalability
- Real world: coreboot + OpenBMC is … rare to say the least
- Room for improvement
    - Active work from vendors, e.g. Supermicro have contributed to getting Intel Sapphire/Emerald Rapids support into coreboot
    - Great work from 3mdeb + 9elements
- At least redfish tends to work
- Fun fact: each NVIDIA GPU tray has its own OpenBMC-based controller

Orchestration

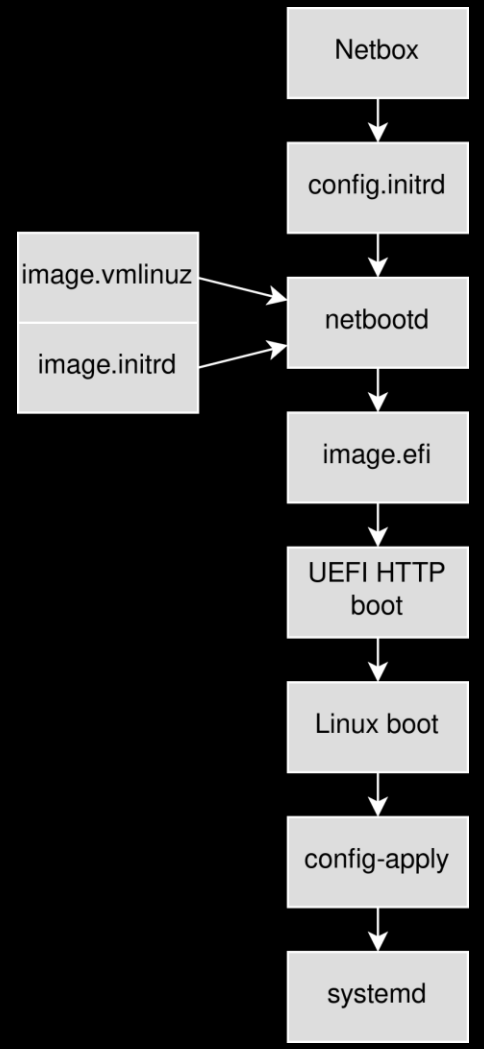
Orchestration

- Need way to run workloads (customer, internal and supporting)
- Kubernetes works very well here
    - "good enough", widely known and supported
    - Purely internal, all customer workloads run in containerized VMs
- Doubles as a platform for managing resources
    - Overlay networks, instances, IP allocations etc. as CRDs
    - Custom logic implemented as controllers
    - Solid API and tooling ecosystem

OS deployment

OS deployment

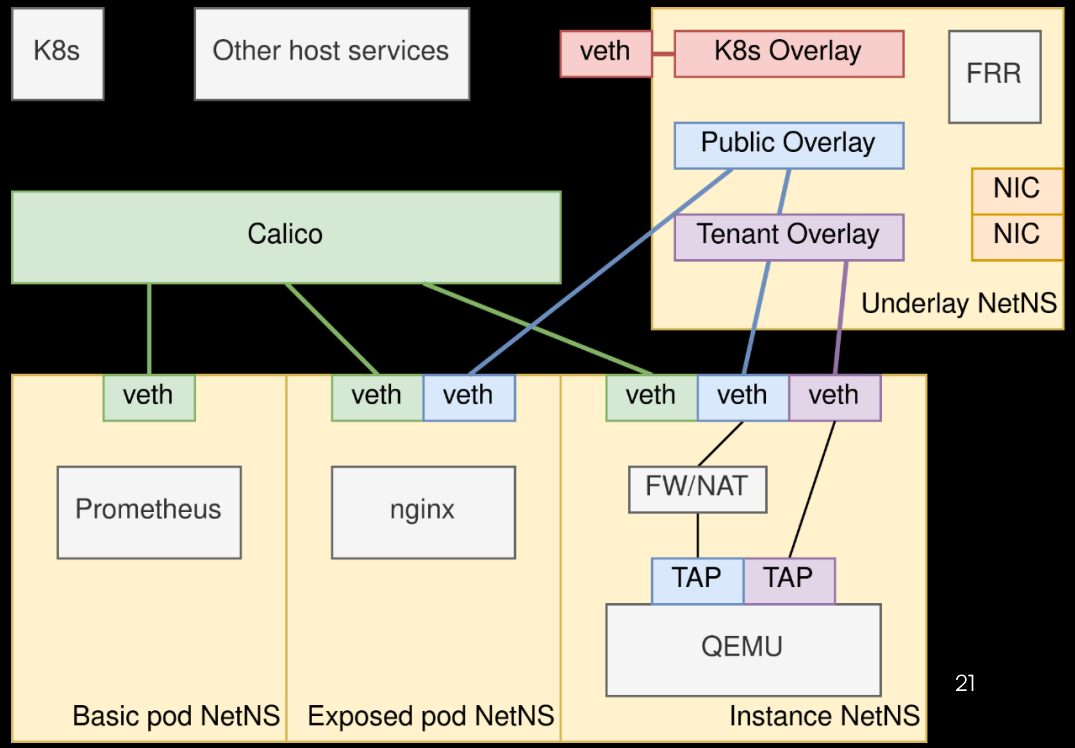
- Traditional mode: Image all servers
- Problem: State drift
- Solution — minimize state wherever possible
    - Nodes are ephemeral — just reboot for new state
    - Some state required for K8s and Ceph control plane
- Images build with mkosi — fully packed into the initramfs
- Delivered via UEFI HTTP netboot
- Config sourced from Netbox, image renders templates on boot



Networking

Networking

- Crucial for a cloud environment
- Scalability and flexibility are non-negotiable
- Separate underlay (connects servers) and overlays (connect workloads)
- Underlay:
    - Routing-to-the-host
    - SONiC on switches
- Overlays:
    - VXLAN / EVPN
    - Underlay isolated in its own network namespace
    - Kubernetes overlay for cluster traffic
    - Public overlay for internet connectivity
    - Tenant overlays for customer-internal traffic



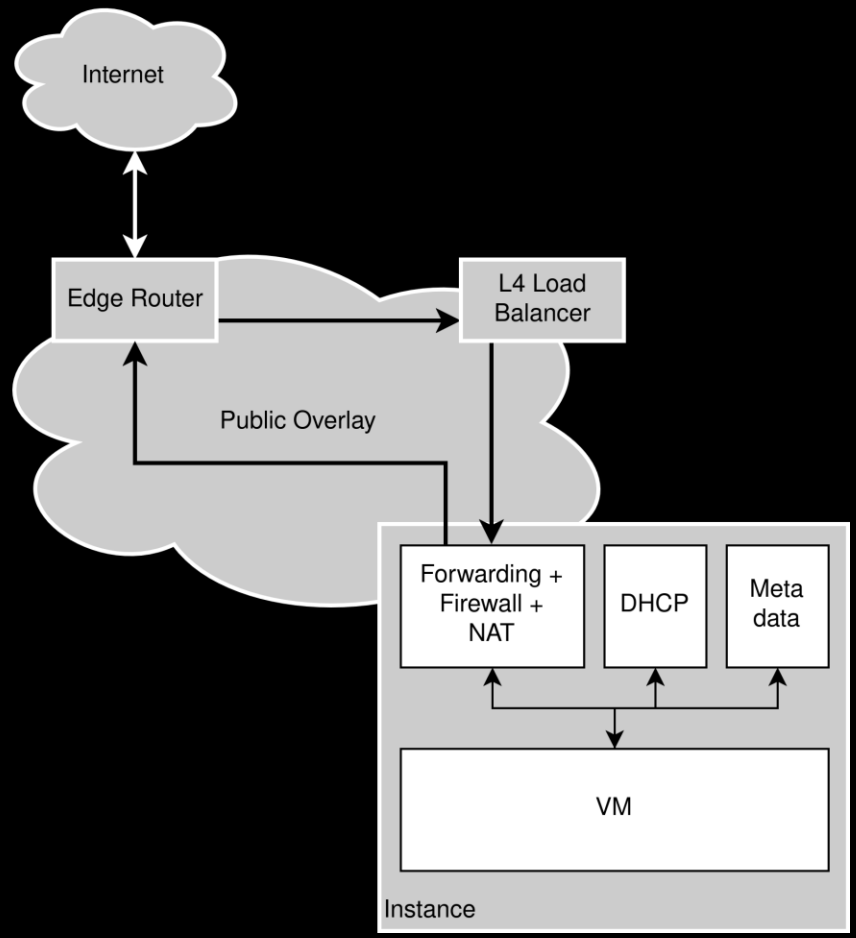
Networking: Public Overlay

IP: 2001:db8::f00/64
MAC: 12:34:56:78:9a:bc
Routes:
- 2001:db8:1:1::/64
- 2001:db8:2::/60
- 198.51.100.3, ports 1024-1280

- Basic custom SDN to have full flexibility
  - Endpoints (e.g. instances, load balancers, services, edge routes) sit in overlay
  - Endpoints announce reachability info (e.g. prefixes) via Kubernetes CRD
  - Public IPv6, but also routes IPv4
  - Daemon watches state and manages Linux network stack
  - Translated to BGP for hardware edge routers
- Stateless L4 load balancer
- Traffic offloading planned, two approaches
  - Userspace: Bind NICs to VPP, connect host and containers via TAP, connect VMs via vhost-user
  - Hardware: Move entire logic to DPU, connect workloads via VFs

Networking: Instances

- Public IPv6 /64 subnet, plus delegated /60
- Stateful firewall using netfilter + conntrack
    - in instance container to avoid centralization
- Public IPv4 address is optional
- Alternative: NAT
    - Challenge: Decentral, scalable, allow mapping traffic to customer
    - Solution: Assign port slice of a shared public IP to each instance
    - Instance container NATs to those ports
    - Return traffic is routed to load balancer, forwarded based on port



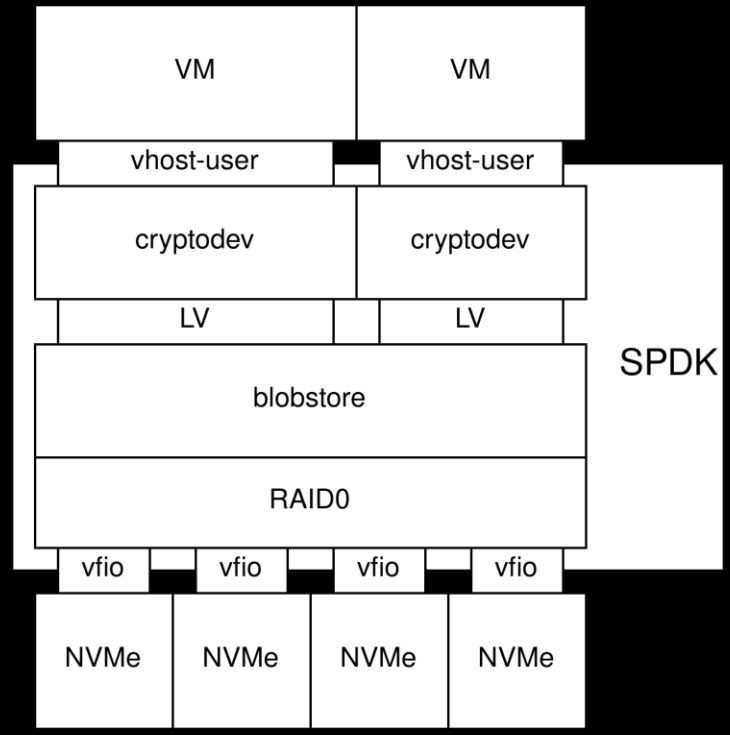
Networking: RDMA

- Needs to be hardware-accelerated (3.6 TBit/s!)
    - Luckily one NIC per GPU, so no need to share
- InfiniBand: Partitioning via PKeys
    - Create VF on NIC, assign GUID
    - Associate GUID with PKey on subnet manager
    - Pass VF to VM
- RoCE
    - NICs can do basic VLAN isolation
    - Better to do it on the switches

Storage

Storage

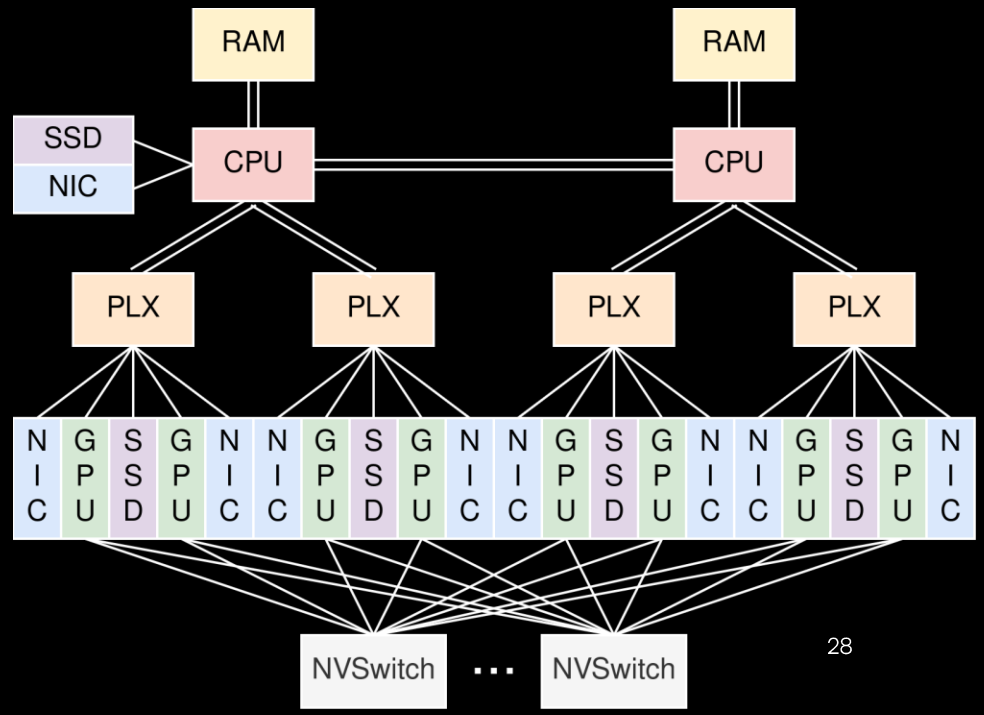
- Want: fast *and* reliable … unfortunately not exactly a feasible combination
- Split storage is an option
    - Local: NVMe — fast but ephemeral
    - Network: Robust primarily — fast secondarily
- Local:
    - Easy to bottleneck on software (here: RAID0 + LVM + dm-crypt + virtio-blk)
    - Solution: SPDK — exposes 4 Gen4 NVMe drives at 98% of raw host throughput (26GB/s)
- Network:
    - Ceph provides a robust, reliable and reasonably fast unified storage platform
    - File system via virtio-fs, Block via SPDK, Object via RadosGW, Kubernetes via Rook



Virtualization

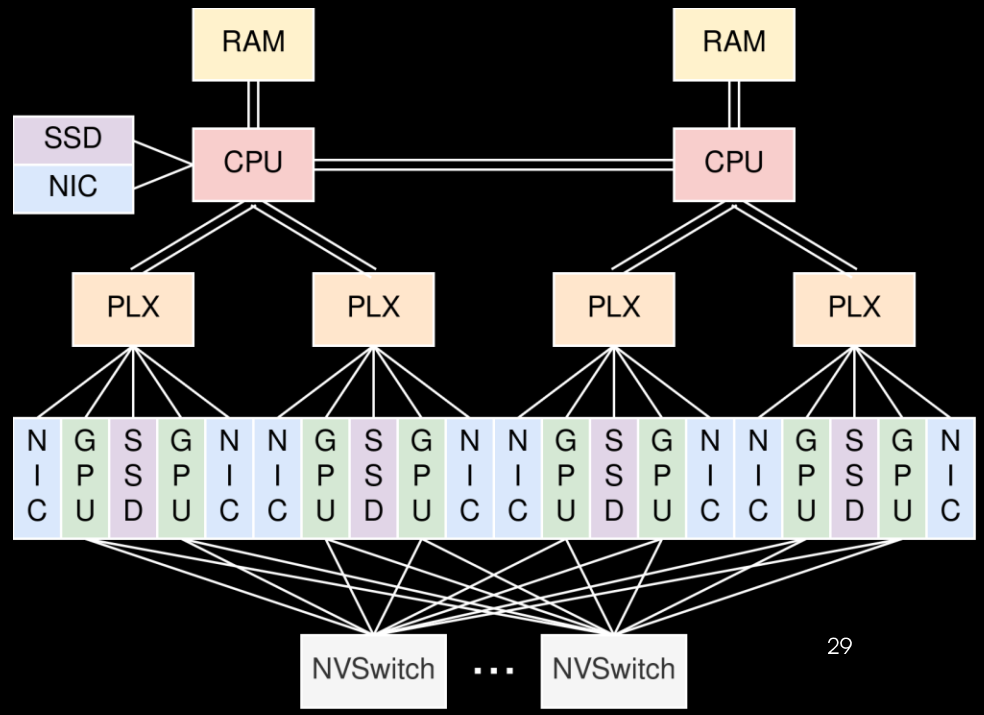
Virtualization

- Obvious option at first glance: KubeVirt
  - Designed (on last check) for seamless integration of legacy applications into Kubernetes
  - We want the opposite — full isolation
  - vhost-user, custom networking, NUMA, PCIe logic etc. requires *a lot* of tricks
- Simple: Just QEMU via small wrapper
- NUMA
  - Select GPU(s) based on locality, e.g. same PLX
  - Select CPU core(s) based on GPU locality, e.g. same socket
  - Pin vCPU(s) to physical core(s)
  - Communicate NUMA topology to guest



Virtualization: GPUs

- PCIe topology
  - GPU-GPU and GPU-NIC traffic is PCIe P2P, would overwhelm CPU root complex
  - Workloads need to know topology
    - Can be configured manually, but easier to just model through QEMU
  - Pitfall: IOMMU forces P2P through the CPU!
    - Avoid via ATS on NIC and ACS Direct Translation on PLX
- GPU-GPU interconnect (NVLink / UALink)
  - 1x GPU is easy — just skip interconnect
  - 8x GPU(s) is easy — just passthrough entirely
  - Mixed is difficult — need to configure lanes from the host



Questions?

Dave Hughes
david.hughes@stelia.ai

Lukas Stockner
lukas.stockner@stelia.ai