Cnuas: A Software-Emulated Virtual Datacenter for AI/HPC Research and Education

PacketFive Networks Limited
connect@packetfive.com

August 2026

Abstract

The skills required to operate modern AI and high-performance computing (HPC) clusters are increasingly inaccessible: production-grade GPU servers, GPU peer-fabric switches and InfiniBand networks are priced beyond the reach of universities, independent researchers, and most engineering teams. This paper introduces Cnuas, a 100% software-emulated AI/HPC datacenter built on QEMU. Cnuas presents unmodified Linux guests with real PCIe virtual GPUs, virtual RDMA-capable NICs, a virtual top-of-rack InfiniBand/RoCEv2 switch, and a virtual GPU-to-GPU peer fabric. Above the compute and network layers, Cnuas additionally emulates the physical infrastructure of an Open Compute Project Open Rack v3 (ORv3) deployment — rack management controller, power shelf, battery backup and per-sled baseboard management controllers running real OpenBMC — and projects the whole into a procedural OpenUSD facility twin. The result is a complete multi-node GPU cluster, from silicon semantics to facility power, that runs on a single workstation while preserving the hardware behaviour — DCB, ECN, InfiniBand Local Identifier routing, GPU peer DMA, Redfish power control — that matters for production operations and skill transfer.

1 Introduction

The cost gap between the hardware required to deploy an AI/HPC system and the hardware available for learning has widened sharply since 2023. A single eight-GPU server with the matching peer-fabric switch and InfiniBand host channel adapters now exceeds the entire annual capital budget of most academic research groups. Existing virtualisation approaches available in Linux — bridges, `macvlan`, Soft-RoCE — do not preserve the hardware semantics that make production AI/HPC operations difficult: priority flow control, explicit congestion notification, Local Identifier routing, and GPU peer-to-peer DMA.

Cnuas fills this gap. Rather than hiding hardware behind host-side shortcuts, Cnuas builds custom QEMU PCIe device models and the corresponding Linux kernel drivers, so that the guest operating system sees real RDMA host channel adapters and real GPUs on its PCIe bus. Standard, unmodified userspace — `libibverbs`, `perftest`, UCX, MPI and NCCL's InfiniBand transport — runs against these devices exactly as it would on physical hardware.

Two properties distinguish Cnuas from a simulator. First, the guest is never aware it is emulated: drivers probe by PCI Vendor and Device ID, ring real doorbells, and receive real MSI-X interrupts. Second, the emulation is complete downwards as well as upwards. A learner can trace a collective operation from the application, through the runtime, the driver, the device model, the wire protocol and the switch pipeline, and then out to the power shelf that feeds the rack.

2 Design Philosophy

Cnuas is built on four commitments:

- **Preserve hardware semantics.** A learner who masters Cnuas must be able to transfer that knowledge to production hardware without re-learning fundamentals. Frame headers, DMA queues, doorbells and interrupts behave the way real silicon behaves.
- **Use unmodified guest stacks.** The guest kernel runs upstream Linux with a small additional module per device, and userspace runs the standard distribution-provided RDMA and GPU stacks. No patched libraries, no special compilers.
- **Emulate the facility, not just the server.** Power, thermal and out-of-band management

are first-class. Most cluster outages are infrastructure events, so an environment that omits them teaches an incomplete discipline.

- **Open, modifiable, and upstream-bound.** Every line of Cnuas is open source, and the generally useful parts are being submitted to QEMU, the Linux kernel and rdma-core rather than held privately.

3 System Architecture

Cnuas is composed of interlocking open-source components, each released as a stand-alone repository and integrated through the Cnuas superproject. The daemons and command-line tools are built with Bazel; the kernel modules and the QEMU fork are built with Make.

3.1 CnuasSwitch — Virtual TOR Fabric

`cnuas-vswitchd` is a standalone C daemon implemented around a single-threaded `epoll` event loop with a 1000 ms tick. Each switch port is a `SOCK_SEQPACKET` UNIX-domain socket, which preserves message boundaries and therefore frame framing without a length-prefix protocol of its own. Frames arriving on a port are classified into one of two forwarding pipelines:

- An **Ethernet pipeline** parses the MAC header and forwards using a standard L2 forwarding database with VLAN tagging and DCB classification.
- An **InfiniBand pipeline** parses the Local Routing Header and forwards on the Local Identifier using a linear forwarding table, with subnet management agent support.

Datacenter Bridging primitives are modelled end-to-end across the fabric: priority flow control (802.1Qbb), enhanced transmission selection (802.1Qaz) and explicit congestion notification (RFC 3168). This allows realistic experiments in congestion control and lossless transport, which is precisely the class of problem that host-side shortcuts such as Soft-RoCE cannot reproduce.

A control plane listens on a dedicated management socket at `/var/run/cnuas/mgmt.sock`, speaking newline-delimited JSON over `SOCK_STREAM`. It accepts commands to bring ports up and down, set per-port mode, manipulate the forwarding and linear forwarding tables, and dump or clear telemetry. Telemetry is additionally exported in OpenTelemetry form on the console port.

3.2 CnuasNIC — Virtual RoCE-IB NIC

CnuasNIC is a custom QEMU PCIe device, `cnuas-vnic`, presented to the guest as a Gen5 x16 endpoint under PCI vendor `0x1AF4` and device `0x10F0`. BAR0 exposes a 4 KB register space containing the doorbell and status registers. When the guest rings the transmit doorbell, the device model reads the descriptor and the associated buffer directly from guest memory, frames the bytes into the appropriate protocol, and writes them to the UNIX-domain socket that represents the "cable" to CnuasSwitch. Completions are signalled back to the guest with MSI-X.

Two protocols are carried. RoCEv2 traffic is encapsulated in UDP to the standard destination port 4791, and native InfiniBand traffic carries a Local Routing Header followed by a Base Transport Header. Because both are the real wire formats, a frame captured inside Cnuas is a valid frame outside it.

Two Linux kernel modules accompany the device. `cnuas_net.ko` probes by Vendor and Device ID and registers a `netdev`; `cnuas_ib.ko` registers an RDMA-capable device with the kernel's `ib_core` subsystem. A userspace verbs provider, `libcnuas-rdmav34.so`, plugs into rdma-core under the provider name `cnuas`. Together these expose the standard Verbs API, so `ibv_devinfo`, `ibstat`, `ethtool`, `perftest`, `ibv_rc_pingpong`, UCX, MPI and NCCL operate without modification.

3.3 CnuasGPU — Virtual GPU

CnuasGPU is a QEMU PCIe device implementing a Single-Instruction Multiple-Thread execution model. The default configuration presents 16 streaming multiprocessors with a warp size of 32 threads and one tensor MAC unit per SM. The memory hierarchy is modelled explicitly, and each level carries a representative access latency so that the performance consequences of a memory access pattern remain visible to the learner:

Level	Capacity	Latency
Register file	64 KB per SM	∼1 cycle
L1 / shared	128 KB per SM	∼30 cycles
L2 cache	8 MB	∼200 cycles
Device memory	8 GB	∼500 cycles

The runtime as it stands today is fixed-function: kernels are selected by name from a built-in catalogue and dispatched onto scalar, AVX2 or AVX-512 host backends. The programmable path is specified but not yet written. In the design, device code is compiled by `cnuascc`, an LLVM-based device

compiler analogous to `nvcc`, which lowers kernels to CnuasIR, an intermediate representation defined as a subset of the RISC-V Vector extension 1.0 plus a small set of Cnuas tensor opcodes. Grounding the ISA in RVV rather than inventing one is deliberate: it keeps the generated code intelligible to anyone with RISC-V knowledge, and it allows the execution backend to be dispatched at runtime onto host SIMD instructions rather than interpreted.

The userspace mirrors the incumbent GPU stack one-to-one, with cleanly named Cnuas equivalents. The mapping is the point: a learner who knows CUDA recognises every API, but no vendor trademark is used. The status column separates what is implemented and covered by tests today from what is specified in the design documents but not yet written, so a design is never mistaken for a shipped feature.

Cnuas	Incumbent equivalent	Status
Cnuas Compute, `libcnuasrt.so`	CUDA Runtime	Shipping
`libcnuasdev.so`	CUDA Driver API	Shipping
`cnuassmi`	`nvidia-smi`	Shipping
`cnuascc`	`nvcc`	Designed
CnuasIR	PTX	Designed
CnuasCCL	NCCL	Designed
CnuasSHMEM	NVSHMEM	Designed
CnuasBLAS, CnuasDNN	cuBLAS, cuDNN	Designed
CnuasFFT, CnuasSPARSE	cuFFT, cuSPARSE	Designed
CnuasSOLVER	cuSOLVER	Designed
`cnuas-dcgmi`	DCGM	Designed
`cnuas-prof`	Nsight Compute	Designed

3.4 CnuasLink — GPU Peer Fabric

CnuasLink is an independent fabric carrying only GPU-to-GPU peer traffic, equivalent in role to NVLink, with `cnuasgpu-link-switchd` playing the role of NVSwitch. The reference switch presents eight fabric ports, with port 8 reserved as an inter-switch link and port 9 as the console and Open-Telemetry endpoint.

For GPUs resident on the same host, the fast path is `ivshmem`: POSIX shared memory for the payload with UNIX-domain socket signalling for doorbells. Peer transfers therefore proceed at memory-copy speed rather than network speed, which is the property that makes the NVLink analogy behave correctly under collective workloads. CnuasSHMEM one-sided put and get operations are specified to ride this transport directly.

Critically, the two fabrics are functionally orthogonal. A VM may have a CnuasGPU but no CnuasNIC, in which case it still participates in GPU peer DMA over CnuasLink, or the reverse. This mirrors a real GPU server, where the peer fabric and the host channel adapters are separate devices with separate failure domains, and it lets students reason about each independently.

3.5 Control Plane and Programmability

Every component is reachable from a single control plane. The `cnuas` command-line tool drives the platform interactively. The same operations are exposed by `cnuas-api`, a REST service with a generated OpenAPI schema and both Swagger UI and ReDoc documentation, so that laboratory exercises, CI pipelines and external orchestration can be written against a stable contract rather than against screen scraping. A Python package provides the underlying implementation and can be imported directly, and `cnuas-webui` presents the fabric, the rack and live telemetry in a browser.

4 Rack and Facility Management

A cluster is not only its servers. Cnuas therefore emulates the physical envelope as well, which is unusual for a teaching environment and is the component most directly relevant to operations staff.

The reference rack follows the Open Compute Project Open Rack v3 specification. Cnuas models a 24 OU half-height rack; the full ORv3 specification ships in 41 OU and 44 OU heights. Rack-level power is provided by an ORv3 power shelf carrying six 5.5 kW power supply units for 33 kW total, backed by a battery backup unit shelf rated for 16.5 kW of ride-through. A populated Cnuas rack draws approximately 6 kW.

Out-of-band management is not simulated at the protocol surface but genuinely executed. A rack management controller runs `cnuas-rackmond`, reading the power shelf over PMBus and I^2C and publishing readings on D-Bus and Redfish. Per-sled baseboard management controllers run real OpenBMC images, built from a dedicated `meta-cnuas` layer and executed on QEMU's ASPEED AST2600 machine model. Because this is genuine OpenBMC on a genuine BMC SoC model, Redfish power control, sensor enumeration and serial-over-LAN behave as they do on production hardware, and firmware work done in Cnuas transfers directly.

Above the rack, the Cnuas facility twin renders the deployment as a procedural OpenUSD scene: halls, rows, racks and the power and thermal model that connects them. Geometry and telemetry are carried on the same primitives, so a rack's live

power draw is a property of the object a user can select, rather than a number in a separate dashboard. The twin places the Cnuas racks in the context of a 132 kW liquid-cooled AI row, which lets a student reason about a small experimental deployment inside a realistic facility power budget.

5 Reference Deployment Topology

The reference rack is populated as follows:

Element	Role	Height
CnuasSwitch	TOR, RoCEv2 + IB	1 OU
CnuasLink switch	GPU peer fabric	1 OU
Management host	Daemons, RMC	1 OU
VM blades (×8)	CnuasNIC + CnuasGPU	2 OU each
Power shelf	Six 5.5 kW PSUs	2 OU
Total		**24 OU**

While the OU heights are visual rather than physical in the emulator, the layout maps one-to-one onto physical ORv3 racks, so the same diagrams serve teaching, design review and real laboratory build-outs. The documented topology scales to two racks, which is sufficient to exercise inter-switch links and multi-rack collectives.

6 Build, Deployment and Verification

Cnuas runs on a conventional Linux workstation. The published laboratory reference is an Ubuntu 24.04 host on a Xeon W-2133 with 128 GB of RAM; hardware virtualisation via `/dev/kvm` is required for the end-to-end gate. The default laboratory brings up two guests, `vm-a` and `vm-b`, from a golden `qcow2` image using `cnuas-tools vm lab`.

Verification is enforced rather than asserted. The RDMA datapath is functionally complete on the wire, with device enumeration, queue pair creation and modification, completion queue polling, memory region registration and RoCEv2 transmit and receive all exercised end to end. The gate test boots the two-VM laboratory and runs `ibv_rc_pingpong` between the guests over the emulated fabric; it is skipped automatically when the environment lacks KVM, the golden image or the tooling, so that a developer without a suitable host still gets a meaningful build. Known gaps are published alongside the passing results, currently reliable-connection reliability semantics, native InfiniBand Local Routing Header framing, and a fast path in the userspace provider.

7 Upstreaming

Components of Cnuas that are useful beyond the project are being prepared for upstream submission, in a deliberate order. A fix to the ASPEED GPIO model and a new CMIS optical module model are queued for QEMU. An application for official PCI device identifiers is sequenced ahead of the device submissions themselves, so that the QEMU device models, the Linux kernel drivers and the rdma-core provider can be proposed with permanent identifiers rather than squatted ones. The kernel work includes a `RDMA_DRIVER_CNUAS` identifier in the RDMA UAPI, and the rdma-core work a standard provider registration.

This ordering matters. Requesting identifiers first is slower in the short term, but it avoids asking three separate upstream communities to accept a device whose identity is provisional.

8 Educational and Research Applications

Cnuas is the laboratory environment for the HiCAIN training programme. Students run real distributed-training workloads, RDMA microbenchmarks and MPI collective operations against the emulated fabric, then take the same commands to production hardware unchanged. Coursework that previously required a hyperscale laboratory now runs on a single workstation, and an entire cohort can be given a private cluster each.

For research, the open and modifiable architecture enables experiments that are impractical on commercial silicon: fabric scheduling and congestion control, in-network computing, collective algorithm design, firmware and out-of-band security, and power-aware scheduling that spans the compute and facility layers simultaneously. Because the device models are source-available, a researcher can instrument any layer of the stack without vendor cooperation — the property that black-box commercial hardware structurally cannot offer.

9 Conclusion

Cnuas removes the hardware barrier that has limited AI/HPC infrastructure education and research to those with hyperscaler-grade budgets. By emulating the full datacenter — compute, peer fabric, NIC, switch, rack management and facility — in software, while preserving the hardware semantics that matter, PacketFive makes the next generation of AI cluster operators, researchers and engineers possible.

Cnuas is open source and available at `https://github.com/PacketFive/cnuas`, with documentation at `https://cnuas.io` and companion training material at `https://hicain.io`.