Cnuas Software Stack, Datasheet

Software stack overview, runtime, daemons, control plane and tooling

Document DS-CNU-010 • **Revision** D • **Issued** 15 August 2026 • **Status** Published

Item	Value
Part	`cnuas software stack`
Type	Host and guest software components
Version	`9fbdddf-dirty`
Repo	PacketFive/cnuas

1. Overview

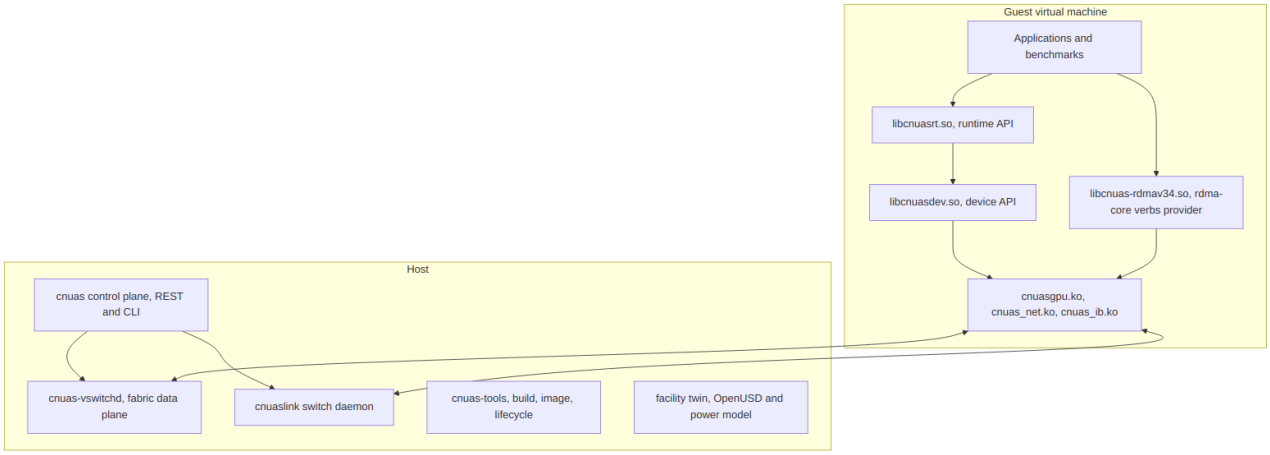
The other datasheets in this library describe emulated **devices**. This one describes the **software** that drives them: the GPU runtime and compute backends, the verbs provider, the switch and fabric daemons, the control plane, and the build and lifecycle tooling.

The stack is split into two clearly separated groups. Section 3 lists what ships and is tested today. Section 8 lists what is designed but not yet written. No component appears in both.

Read this with the validation matrix

Every availability claim below is backed by a named test in the Validation Matrix. If a component has no row there, it is not claimed as available here.

2. Stack layout



3. Available components

Component	Package	Role	Validated by
`libcnuasrt.so`	`cnuas-libcnuasrt`	GPU runtime API, CUDA-Runtime style	`cnuasrt_smoke`, `sgemm_smoke`

Component	Package	Role	Validated by
`libcnuasdev.so`	`cnuas-libcnuasrt`	Low level device access below the runtime	`compute_smoke`
Compute backends	`cnuas-libcnuasrt`	Scalar, AVX2, AVX-512 and CnuasIR kernels loaded at runtime	`compute_backend_smoke`, `compute_so_smoke`
`libcnuasblas.so`	`cnuas-libcnuasrt`	Dense linear algebra v0.1, cuBLAS style, over the runtime kernels	`cnuasblas_host_smoke`, `cnuasblas_smoke`
`libcnuasir.a`	`cnuas-libcnuasrt`	CnuasIR object format parser and validator	`cnuasir_object_smoke`
CnuasIR backend	`cnuas-libcnuasrt`	Interpreter for the CnuasIR v0.1 subset, selected by name	`cnuasir_interp_smoke`
`libcnuas-rdmav34.so`	`cnuas-libcnuas-provider`	rdma-core verbs provider for CnuasNIC	`test_rocev2_e2e.py`, `test_ib_e2e.py`
`cnuas-vswitchd`	`cnuas-vswitchd`	Switch data plane and management socket	35 cases in `tests/`
`cnuas-cli`	`cnuas-cli`	Switch command line client	`test_mgmt_api.py`
`cnuaslink-cli`	`cnuas-link-cli`	Fabric command line client	`cli/tests/test_cli_e2e.py`, 10 cases
`cnuassmi`	`cnuas-gpu-modules`	GPU inventory and telemetry, `nvidia-smi` equivalent	manual
`cnuas` control plane	source tree	REST API and command line over every component	52 cases
`cnuas-tools`	source tree	Build, package, image, virtual machine, release, deploy	122 cases
Facility twin	source tree	Campus layout, power model, OpenUSD stage	59 cases

Where the detail lives

This datasheet is the map. Each component has its own datasheet.

Component	Datasheet
Accelerator runtime	CnuasRT
Device library and compute backends	CnuasDev
Guest kernel modules	Kernel Modules
Verbs provider	Verbs Provider
Control plane	Control Plane
Build and deployment tooling	Cnuas Tools
Operator clients	Management Tools
Facility twin	Facility Twin
Management controllers	CnuasBMC
Compiler, instruction set, collectives, one sided messaging, numerical libraries	CnuasCC, CnuasIR, CnuasCCL, CnuasSHMEM, Math Libraries

4. GPU runtime API, as implemented

`libcnuasrt.so` version 0.2.0. This is the surface that exists in the tree today.

Group	Entry points
Lifecycle	`cnuasInit`, `cnuasShutdown`
Device	`cnuasGetDeviceCount`, `cnuasGetDevice`, `cnuasSetDevice`, `cnuasGetDeviceProperties`
Memory	`cnuasMalloc`, `cnuasFree`, `cnuasMemcpy` with host to device, device to host, and device to device kinds
Synchronisation	`cnuasDeviceSynchronize`
Internal	`cnuasInternalMapDevice`, and the backend helpers `cnuas_compute_get`, `cnuas_compute_active_name`, `cnuas_compute_reset`
Kernels	`cnuasSgemm`, `cnuasVectorAddF32`, `cnuasVectorDotF32`, `cnuasVectorScaleF32`
Activations	`cnuasReluF32`, `cnuasGeluF32`, `cnuasSiluF32`, `cnuasSigmoidF32`, `cnuasTanhF32`, `cnuasSoftmaxF32`
Modules	`cnuasModuleLoad`, `cnuasModuleUnload`, `cnuasLaunchKernel`
Errors	`cnuasError` codes and `cnuasGetErrorString`

Kernels compile, but only the scalar subset

`cnuasLaunchKernel` runs a CnuasIR object, and the interpreting backend executes it. `cnuascc` now compiles a device source file to such an object, so a kernel no longer has to be written as an instruction stream by hand. Two limits remain: code generation covers the scalar subset only, so a kernel that needs the vector unit is still hand written, and there is no triple angle bracket launch syntax, so the caller uses the module and launch entry points directly. Only the CnuasIR backend implements `execute_cnuasir`, so a launch on any other backend fails with `cnuasErrorNotSupported`. CnuasGPU Design describes the target design, not the current build. Full detail is in the CnuasRT datasheet.

5. Device API, as implemented

`libcnuasdev.so` version 0.1.0.

Group	Entry points
Handle	`cnuasdev_open`, `cnuasdev_close`
Discovery	`cnuasdev_get_device_count`, `cnuasdev_query_info`
Memory	`cnuasdev_alloc`, `cnuasdev_free`, `cnuasdev_memcpy`
Mapping	`cnuasdev_mmap`, `cnuasdev_munmap`
Diagnostics	`cnuasdev_status_str`, returning text for a `cnuasdev_status_t` code

6. Compute backends

Kernels are compiled once per instruction set and loaded through a small dispatching loader, so one runtime binary works across host generations.

Backend	Shared object	Selected when
AVX-512	`libcnuasrt-avx512.so`	Host advertises AVX-512
AVX2	`libcnuasrt-avx2.so`	Host advertises AVX2
Scalar	`libcnuasrt-scalar.so`	Fallback, always usable

Selection happens at first use and can be overridden with an environment variable, which is what `compute_backend_smoke` exercises. Both backend tests run on any host, with no emulated device present, which makes them the fastest regression signal in the GPU stack.

7. Control plane and tooling

Surface	Command	Notes
Health across all components	`cnuas system health`	Reachability probe
Inventory	`cnuas system inventory`	Snapshot of reachable components
Versions	`cnuas system versions`	Resolved from git tags per repository
Switch control	`cnuas switch ...`	Ports, forwarding database, priority flow control, telemetry
Build and package	`cnuas-tools pkg build`	Produces the Debian packages named in section 3
Golden image	`cnuas-tools image build`	Ubuntu 24.04 base plus pinned kernel
Virtual machines	`cnuas-tools vm ...`	Lifecycle for the lab guests
Release and deploy	`cnuas-tools release`, `cnuas-tools deploy`	Bundle name is `cnuas`

8. State of the upper software stack

Named here so the boundary between plan and product stays visible, and so nobody mistakes a design document for a shipped feature. A component is listed as partial only when a test in the validation matrix covers what is claimed.

8.1 Partially implemented

Component	Models	What ships	What is missing
CnuasBLAS	cuBLAS	v0.1, level 1 to level 3 over the shipped kernels	Kernels the caller writes, which need `cnuascc`
CnuasCCL	NCCL	v0.1 bootstrap, membership, broadcast, AllReduce, AllGather	Carriage over CnuasLink, reduce scatter, all to all
CnuasIR toolchain	PTX	Object format, packer, validator, disassembler, interpreting backend, and the module and launch API	A compiler that emits it, the tensor and synchronisation instruction classes, and the instruction set freeze at version 1

8.2 No code in the tree

Component	Models	Blocking dependency	Preview datasheet
`cnuascc`	`nvcc`	LLVM backend for CnuasIR	CnuasCC
CnuasSHMEM	NVSHMEM	One-sided put, get, and atomics over CnuasLink	CnuasSHMEM
CnuasDNN, CnuasFFT, CnuasSPARSE	cuDNN, cuFFT, cuSPARSE	`cnuascc`	Math Libraries
CnuasSOLVER	cuSOLVER	CnuasBLAS v0.2	Math Libraries
`cnuas-dcgmi`, `cnuas-dcgm`	dcgmi, dcgm	Telemetry schema	Management Tools
`cnuas-prof`	Nsight Compute	Performance counter interrupt path	Management Tools

9. Integration information

Item	Value
Superproject	`PacketFive/cnuas`
Guest operating system	Ubuntu 24.04
Kernel	6.19.0-cnuas, from `PacketFive/linux`
Emulator	`PacketFive/qemu` fork
Python	3.12
Kernel modules	`cnuas_net.ko`, `cnuas_ib.ko`, `cnuasgpu.ko`
Golden image	`cnuas-vm-vX.Y.Z.qcow2`

Version coupling between the kernel, the emulator, and the modules is enforced by the `compat.json` document that ships with every release, using its `cnuas_min` field. See Deployment.

10. Revision history

Version	Change
0.4.0	Records roadmap 6.4c. Adds the CnuasIR interpreting backend to section 3 and the module and launch API to section 4.
0.3.0	Section 8 split into partially implemented and no code in the tree, placing CnuasCCL, CnuasBLAS and the CnuasIR tools under partially implemented.
0.2.0	Adds CnuasBLAS v0.1 and the CnuasIR object parser to section 3.
0.1.0	First publication. Records the implemented runtime, device API, compute backends, control plane, and tooling, and separates them from the designed but unwritten compiler and numerical libraries.

PacketFive, Packet Five Networks Ltd., Dublin, Ireland.