Clustering Four DGX Sparks in a 200Gb/s Ring

November 5, 2026 / Gavin Jackson

AI DGX Spark vLLM



Back in October I finished a post with a cliffhanger: once the 200Gb/s switch arrived, four DGX Sparks would become a single 512GB inference engine. The switch never happened. We went a better way: no switch at all.

Each DGX Spark has a ConnectX-7 with two 200Gb/s QSFP ports. With four nodes that is all you need for a closed ring. Each Spark cables directly to its two neighbours, one port each way, four cables in total, and every node gets 200Gb/s of full-duplex bandwidth in both directions around the loop. No switch to buy, rack, configure or cool. Total cluster memory: 512GB of unified memory across the four nodes.

The catch with any multi-node build is the configuration grind: static IPs on the high-speed interfaces, SSH trust between every pair of nodes, matching MTUs, NCCL settings that actually match the physical cabling. That is the part that usually eats a weekend. Here it took one command.

The setup helper script

The tooling comes from eugr/spark-vllm-docker, the same project that builds the B12X vLLM containers we have been running since the Ollama days. Its automated cluster setup is a single script run once from the head node over the ordinary management network:

```
./setup-cluster.sh 192.0.2.11 192.0.2.12 192.0.2.13 192.0.2.14 --topology ring
```

For a ring you list the nodes in physical cable order. The script is upfront about preferring evidence over instructions: if LLDP is running it maps the ring itself and rejects a node order that conflicts with what the cables actually say. Given how ring cabling goes in practice, that check earns its keep.

From there it does the two jobs that matter.

Network configuration. It migrates the CX7 interface definitions on every node into a dedicated Netplan file, backing up the originals first, and validates the combined configuration in a temporary directory before touching anything live. Each cable in the ring gets its own /24 from an RFC1918 pool, two distinct subnets per link, static addresses, DHCP off, jumbo frames at MTU 9000 by default. It applies the config to the workers first and the head last, then waits for every interface to come up with the planned address, carrier, MTU and routes before probing anything. Verification is real verification: pings on both rails, in both directions, at the full MTU with fragmentation disabled, so you find out about a marginal cable or a MTU mismatch before your model weights do.

SSH configuration. Each node generates a dedicated Ed25519 keypair in `~/.ssh/spark-vllm-cluster/`, the script exchanges public keys only, and a block is prepended to `~/.ssh/config` selecting that key and a dedicated host-key file for the cluster addresses. The password bootstrap happens once, through a local askpass helper over a private Unix socket, with no `sshpass` package and nothing written to disk. Afterwards it proves the result: passwordless SSH from every node to every other node's management and CX7 addresses. If that matrix passes, anything the cluster does later, Ray included, inherits working trust.

It also makes sure your user is in the `docker` group on every node, and writes an autodiscovery-compatible `.env` on the head with the topology's NCCL settings baked in. For a ring of four or more, that means `NCCL_ALGO=Ring`, because letting NCCL auto-select on a ring leads it to attempt connections between non-neighbours and time out. The whole thing is reversible: setup keeps backups and journals, `--doctor` checks and repairs a saved setup, and a restore puts every interface and file back the way it was.

Life on a ring

A closed ring has one honest limitation, and the tooling enforces it rather than letting you discover it at 2am: inference runs on the full ring, on two directly connected nodes, or solo. You cannot run a three-node job on a four-node ring, because removing any one node breaks the closing link and turns it into an open chain. Ask for it and the launch is rejected before any containers start.

Model and container distribution is cleverer than I expected. Only direct neighbours can reach each other over the ring, so getting a 510GB checkpoint from the head to the far side of the loop uses SSH `ProxyJump` through the intermediate nodes, streamed, without staging copies on the machines in between. Copy failures fail loudly rather than silently falling back to the management network, which is exactly the failure mode you want.

The recipes

The repo ships ready-made recipes per cluster shape. For our four-node ring the interesting folder is `recipes/4x-spark-cluster/`:

- `deepseek-v4.1-flash.yaml`
- `qwen3.5-397b-a17b-fp8.yaml`
- `qwen3.5-397b-int4-autoround.yaml`
- `nemotron-3-ultra-nvfp4.yaml`
- `minimax-m2.5.yaml`

One command builds the container, downloads the model, distributes image and weights to every worker, brings up the Ray cluster and starts serving an OpenAI-compatible endpoint on the head node:

```
./run-recipe.sh recipes/4x-spark-cluster/deepseek-v4.1-flash.yaml --setup
```

What we are running

The current resident is DeepSeek-V4.1-Flash, and it is a serious piece of engineering. It is a multimodal mixture-of-experts model with a 552-billion-parameter backbone, built on what DeepSeek calls a Causal Encoder-Decoder architecture: a twenty-layer causal encoder followed by a twenty-layer decoder, with the decoder's global KV cache projected from the encoder's final hidden states. Only 8 billion parameters activate per token during prefill and 16 billion during decode. Context is up to one million tokens, and thanks to FP4 KV caching the persistent KV footprint is around 890 bytes per token, roughly a quarter of its predecessor.

The checkpoint is about 510GB of weights. The cluster has 512GB of unified memory in total. It fits by a whisker, and only because the recipe knows exactly what it is doing: tensor parallelism splits the weights four ways, the engram memory tables are pinned to NVMe rather than GPU memory, and the KV cache compression means there is still room left over for actual context. The recipe also enables DSpark speculative decoding with seven speculative tokens, which takes some of the latency sting out of running something this size on desktop hardware.

Read those numbers again. A model in the half-trillion-parameter class, one that natively sees images, handles million-token contexts, does tool calling and reasoning, served entirely offline, from four small boxes cabled into a loop, on hardware that came in under AUD $50,000 all up. Two years ago this was a datacentre workload. It now runs next to the gardening supplies.

> ***Breakout: DeepSeek and Australian Government use***
>
> *Running DeepSeek weights on our own infrastructure is a materially different arrangement from sending prompts and documents to DeepSeek's hosted service. That distinction matters when reading the Australian Government Department of Home Affairs' PSPF Direction 001-2025, issued*

> *on **4 February 2025**. It requires non-corporate Commonwealth entities to remove covered DeepSeek products and prevent their access, use or installation, identifying DeepSeek's data collection and exposure of that data to foreign government directions as risks.*
>
> ***There is an explicit exclusion for certain locally deployed models.** Page 1, footnote 2 of the directive (PDF) excludes open-sourced LLMs when all three conditions hold:*
>
> - *The entire codebase is available for inspection.*
> - *The model is deployed locally on a government system.*
> - *Appropriate mitigations are in place as outlined in **Policy Explanation Note 001-25 - DeepSeek Products, Applications and Web Services**.*
>
> ***My reading is that the directive leaves a deliberate path for local evaluation**, subject to those conditions. In our constrained test environment, that gives me a reasonable basis to explore the model. With inference running locally and outbound access controlled, prompts and responses can stay within the environment without being transmitted to DeepSeek. That directly addresses the data-exposure concern described in the directive. We still need to verify the behaviour of the runtime and connected tools, including telemetry and external API calls.*
>
> *This is a conditional exclusion, not blanket approval of every open-weight deployment. For government use, the agency security team needs to confirm the three conditions against the accompanying note. My position here is that a constrained local trial is a sensible way to evaluate capability and controls; approval to handle operational government information is a separate decision.*
>
> *The agency also needs to check the current Commonwealth Technology Standard under PSPF Direction 004-2025 - Commonwealth Technology Management, including its Deny List and Applications Policy.*
>
> *This is also why the harness is my next focus. Keeping inference local gives us control over where the model runs. Now I want to establish what OpenCode, VS Code, Hermes and their integrations can do with the information and permissions we give them.*

Where this leaves us

The cluster gives us room to keep comparing models, but my next major focus is **the security of the harnesses that put those models to work**: OpenCode, VS Code with its AI extensions, Hermes and similar tools. I want to understand what each configured setup can read, what it can execute, which credentials it can reach, and where it can send data. My testing will cover permission boundaries, sandboxing, prompt injection through repository content and tool results, extension and MCP server trust, outbound connections, and whether approvals and audit logs give us meaningful control.

It is one thing to ship a big model. Users are now asking me, "What can I do with it, and can you vouch that it is secure?" That is a big ask. A useful answer needs a defined workflow, tested controls and evidence about the system we are actually giving people. Getting the model running has brought me to

the point where I can start that work; now I want to show what users can safely do with it, under which conditions, and where the limits are.

Downloaded from https://www.gavinj.net/post/dgx-spark-200gbe-ring-cluster
Generated November 14, 2026. Copyright Gavin Jackson. All rights reserved.