Faysal Aziz

Building On-Premises AI with NVIDIA DGX Spark

Know the hardware. Run your own models. Build a service your team can use.

FA-0831 · 3 days

Agentic & Generative AI · DevOps, Cloud & Infrastructure · Embedded Systems and Hardware

Deploy, measure and adapt local language models on DGX Spark, then test two-node inference and hand over a bounded internal AI service.

Introduction

Getting a model to answer its first question is the easy part. The engineering starts when a second user connects, a longer prompt consumes the available memory, or a colleague asks whether the model's licence permits the team to use it. This hands-on course takes developers and engineers through those decisions on real DGX Spark hardware, from inspecting the machine over SSH to operating an internal AI service.

Participants build an engineering support assistant that can look up an approved internal runbook and use it to answer a question. Along the way, they serve an open-weight model, measure its performance, run a short fine-tuning job and distribute inference across two Sparks. The same project gives the team a practical way to decide when to retrieve a document, when to change the model's behaviour and when a second machine is worth using.

The instructor demonstrates development with Claude and Claude Code. Participants can use their preferred AI development tool and also work with an open-source harness connected to a local model.

Learning outcomes

By the end of the course, participants will be able to:

- Inspect a Spark and explain how its processor architecture, unified memory, storage and network interfaces affect a deployment.

- Select a model from Hugging Face, check its licence and estimate the memory needed for weights, context and concurrent requests.
- Deploy a model behind a local API, call it from Python and measure time to first token and generation speed.
- Run a QLoRA fine-tuning job and compare the adapter with the base model on held-out examples.
- Validate a two-Spark connection, run distributed inference and interpret the results against a single-node baseline.
- Build and test a tool-using agent, restart its service and hand the deployment to another engineer.

Prerequisites

- **Python proficiency.** Independently write and debug a Python application that calls an HTTP API, validates JSON, handles errors and runs automated tests.
- **Linux proficiency.** Independently use SSH, manage permissions and processes, inspect logs and diagnose a failed service or network connection.
- **DevOps proficiency.** Independently use Git, build and run a container, configure ports and volumes, and reproduce a deployment from its repository.
- **Participant equipment.** An SSH-capable laptop or workstation and access to an AI development tool of the participant's choice.
- Internet access permitted for the selected training tools and model downloads.
- Permission and connectivity to use SSH with the designated training systems.
- Access to DGX Spark hardware, including a compatible two-node lab for distributed inference.

For engineers currently working at a basic Python level, a practical readiness check is to containerise a small Python API, deploy it over SSH and diagnose a failed request from its logs. Participants who need assistance with those tasks should complete preparatory Python, Linux and DevOps training before attending. Machine learning concepts are introduced during the workshop.

Training outline

1. DGX Spark hardware and system selection

1. NVIDIA Founders Edition, NVIDIA configurations and partner-built GB10 systems
2. Comparing actual specifications by SKU: 64 GB and 128 GB memory options, NVMe capacity, networking and vendor support
3. GB10 Grace Blackwell architecture: Arm CPU, Blackwell GPU, Tensor Cores and coherent unified memory
4. Model capacity versus generation speed: memory bandwidth, numerical precision and the meaning of peak FP4 performance

2. Commissioning a Spark and running the first model

1. DGX OS, headless access and host inventory
2. Arm64 container images, GPU drivers, CUDA runtime and GPU visibility
3. Hugging Face repositories, model cards, tokenizers, pinned revisions and licence review
4. Open weights and open-source AI: permissions to use, modify and redistribute
5. Single-node vLLM deployment, Python API client and repeatable inference baseline

3. Inference, memory and performance

1. Tokens, context windows, prompt processing and token generation
2. Memory budgeting for model weights, runtime overhead and the attention key-value (KV) cache
3. On-disk model caches, NVMe storage and in-memory KV-cache use
4. Measuring first-token latency, tokens per second and memory pressure under longer prompts and concurrent requests

4. Adapting a model to an internal task

1. Choosing among prompting, retrieval, continued pretraining and supervised fine-tuning
2. Full-weight fine-tuning, low-rank adaptation (LoRA) and quantised low-rank adaptation (QLoRA)
3. Preparing training and held-out data, training a small adapter and comparing outputs against the baseline
4. Saving, loading and rolling back an adapter

5. Clustering and distributed inference

1. Management Ethernet, ConnectX-7 networking and two-node topology
2. Link validation, interface selection and NCCL communication checks
3. Splitting one model across two nodes versus running independent replicas
4. Two-node vLLM serving, single-node comparison and the role of Kubernetes in orchestration

6. AI development harnesses, agents and service handover

1. Model versus harness: project context, tool access, permissions and review of generated changes
2. Proprietary and open-source harnesses: Claude Code workflow and Aider with a local model endpoint
3. Python service development: request validation, model calls and a read-only runbook lookup tool
4. Agent tool selection, argument validation, execution limits and missing-document handling
5. End-to-end acceptance testing, request logs, restart recovery and deployment handover

Practical assessment

The practical assessment is a live handover: demonstrate a working request, trace a tool call, recover the service after a restart and explain a deployment decision using measurements.

Disclaimer

This outline serves as a guideline for delivery. The trainer may amend the sequence, scope, tools or practical activities at his discretion, without prior notice, to account for participant readiness, hardware availability and changes to the software platform.

© 2026 Faysal Aziz · Kuala Lumpur