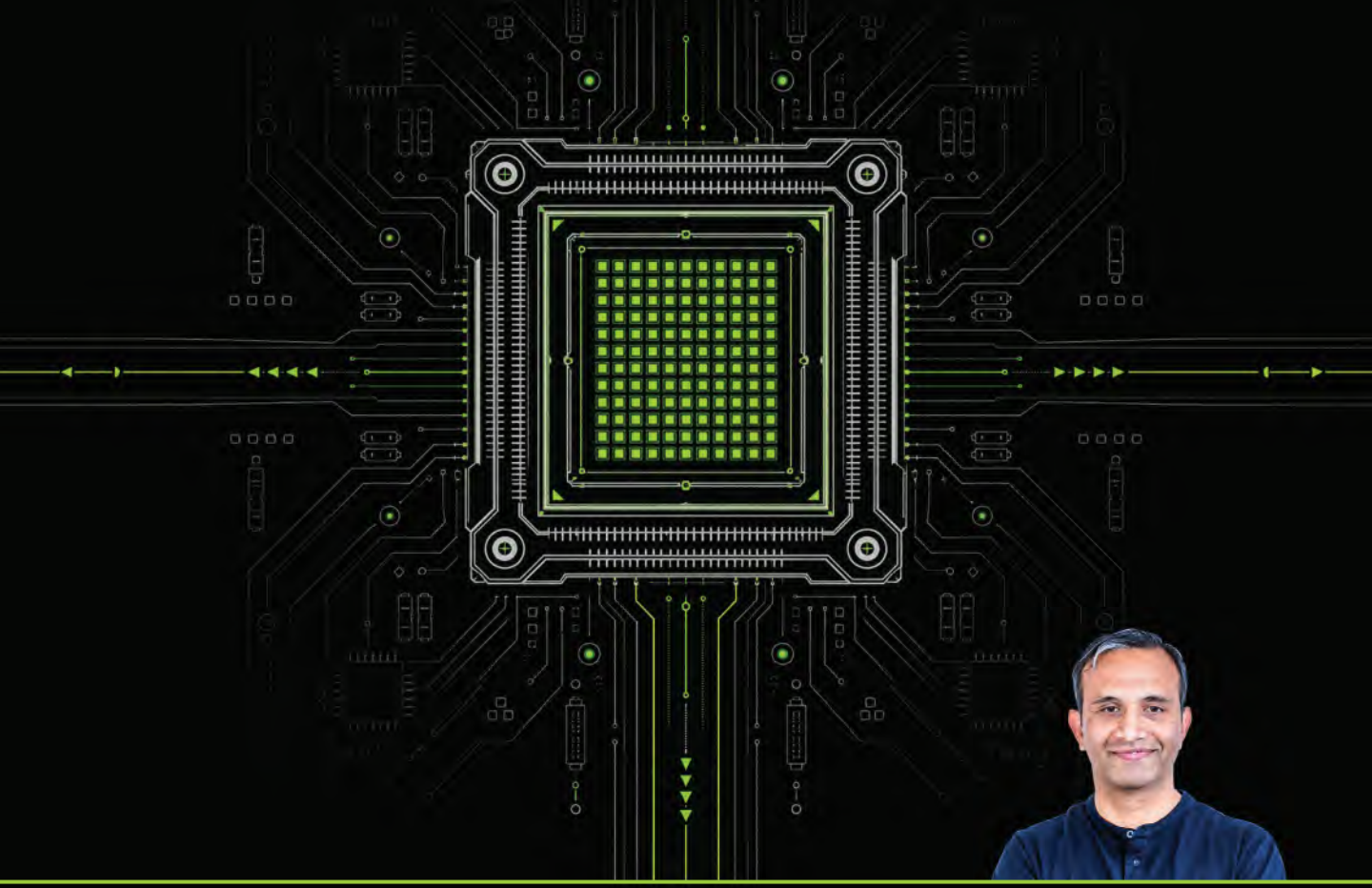
‹packt›

NVIDIA NCA-AIIO Certification Study Guide

Understand GPUs, NVIDIA AI infrastructure, and operations while preparing for the NCA-AIIO exam



ASHISH PRAJAPATI

NVIDIA NCA-AIIO Certification Study Guide

Understand GPUs, NVIDIA AI infrastructure, and operations while preparing for the NCA-AIIO exam

Ashish Prajapati

‹packt›

NVIDIA NCA-AIIO Certification Study Guide

Copyright © 2026 Packt Publishing
All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.
The author's original ideas were further refined by a professional publishing team. Cutting-edge AI was also used to shape content into this form. Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the author, nor Packt Publishing or its dealers and distributors, will be held liable for any damages caused or alleged to have been caused directly or indirectly by this book.
Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

Portfolio Director: Kartikey Pandey
Proposal Writer: Preet Ahuja
Project Manager: Sonam Pandey
Content Engineer: Arun Nadar
Technical Editor: Simran Ali
Copy Editor: Arun Nadar
Indexer: Hemangini Bari
Production Designer: Shantanu Zagade
Growth Lead: Abin Baiju

First published: October 2026

Production reference: 1271026

Published by Packt Publishing Ltd.
Grosvenor House
11 St Paul's Square
Birmingham
B3 1RB, UK.

ISBN 978-1-80881-847-9
www.packtpub.com

Contributors

About the author

Ashish Prajapati is a London-based Solutions Delivery Manager specializing in data center infrastructure, cloud computing, and AI. With extensive hands-on experience in large-scale cloud migrations, data center technologies, and modern AI architectures, he is passionate about making complex technical concepts accessible through practical insights and relatable analogies.

Ashish has trained more than 50,000 learners through his highly rated Udemy courses, helping professionals develop technical expertise and earn certifications in cloud and AI technologies. He is also the founder of BeSA Cloud Academy (Become a Solutions Architect), a global learning community that has supported more than 25,000 professionals in advancing their technical skills and careers.

About the reviewer

Gayathri Kolluru is an AI/ML Infrastructure Engineer specializing in architecting, deploying, and scaling enterprise AI systems on Google Cloud Platform (GCP). She has extensive experience optimizing training and inference workloads across distributed TPU and GPU clusters using JAX and PyTorch. An active open-source contributor to large-scale machine learning frameworks such as AxLearn, she specializes in model loading and fault-tolerant pipelines. She is passionate about building robust, cost-effective MLOps solutions that enable the reliable deployment of state-of-the-art large language models (LLMs) in production environments.

Table of Contents

Preface **xxi**

Free benefits with your book ..... **xxvii**

Chapter 1: Navigating NCA-AIIO and the NVIDIA Technology Stack **1**

Technical requirements **2**

Understanding what NCA-AIIO validates **3**

Checking the current exam snapshot • 3

Reading the three exam domains • 4

Essential AI Knowledge: understanding the workload • 5

AI Infrastructure: building the environment • 5

AI Operations: keeping the environment useful • 6

Using weights without creating blind spots • 6

Using the blueprint and study guide as the source of truth **6**

Using roles as operational context **7**

Keeping ownership separate from exam scope • 8

Reading the six-layer NVIDIA technology stack **8**

Physical layer • 9

Data movement and I/O acceleration • 10

OS, driver, and virtualization • 10

Core libraries • 10

Monitoring and management • 10

Applications and vertical solutions • 10

Supporting infrastructure around the stack • 11

Connecting the stack to the AI lifecycle • 11

Using the stack as an orientation model • 11

Separating AI knowledge, infrastructure, and operations concerns **11**

Working through a cross-domain scenario • 12

Distinguishing durable concepts from version-specific details **13**

Using a version-aware reading habit • 14

Using this book as a learning system **14**

Part 1: Establishing AI and workload foundations • 15
Part 2: Designing accelerated AI infrastructure • 15
Part 3: Building the NVIDIA software and virtualization stack • 15
Part 4: Operating and optimizing AI infrastructure • 16
Using the recurring chapter elements • 16
Completing a baseline knowledge check 16
Turning ratings into priorities • 18
Classifying tasks by domain and operational owner 18
Suggested classification • 19
Starting a realistic study and practice plan 20
Step 1: Define the time boundary • 20
Step 2: Convert domain gaps into chapter work • 20
Step 3: Produce evidence of learning • 20
Step 4: Review at measurable checkpoints • 21
Step 5: Reserve time for scenario integration • 21
Maintaining a weak-area tracker • 21
Approaching certification scenarios • 22
Review questions 22
Question 1 • 22
Question 2 • 23
Question 3 • 23
Question 4 • 23
Question 5 • 23
Question 6 • 24
Question 7 • 24
Question 8 • 24
Answers and explanations • 24
Summary 25

Chapter 2: Understanding AI Workloads, Training, and Inference 27

Technical requirements 28
Tracing the major drivers behind modern AI adoption 28
1. Growth in available data • 29
2. Growth in computational power • 29
3. Algorithmic breakthroughs • 29

Seeing the three drivers as one system • 30
Differentiating AI, machine learning, deep learning, and generative AI 30
Artificial intelligence: the broad field • 31
Machine learning: learning patterns from data • 31
Deep learning: using deep, multi-layer neural networks • 31
Generative AI: creating new content from learned patterns • 31
Classifying by behavior instead of by buzzword • 32
Understanding data, models, parameters, and lifecycle vocabulary 33
Starting with data • 34
Understanding the model and its parameters • 34
Comparing training and inference 34
Training: learning from repeated passes • 34
Optimization: preparing the trained model for use • 35
Inference: using the trained model on new input • 35
Recognizing transformers at the associate level 36
Following a next-word prediction example • 36
Understanding attention at a high level • 36
Repeating the prediction loop • 37
Recognizing infrastructure signals from workload scale 37
When datasets grow • 38
When models grow • 38
When inference demand grows • 38
Recognizing AI use cases across industries 39
Automotive and autonomous systems • 39
Healthcare and life-science workloads • 40
Video analytics • 40
Finance and banking • 40
Retail • 40
Manufacturing • 40
Avoiding common terminology traps in exam scenarios 41
Trap 1: treating AI and machine learning as synonyms • 41
Trap 2: treating deep learning as self-learning without neural networks • 41
Trap 3: treating all generative AI as a separate infrastructure category • 41
Trap 4: treating training and inference as the same workload • 42
Trap 5: assuming more data or compute automatically improves the result • 42

Using a five-question classification routine • 43
Practical activities **43**
Activity 1: classifying business scenarios • 43
Suggested classification • 43
Activity 2: explaining the transformer example in five steps • 44
Activity 3: comparing training and inference • 44
Review questions **45**
Question 1 • 45
Question 2 • 45
Question 3 • 45
Question 4 • 45
Question 5 • 46
Question 6 • 46
Question 7 • 46
Question 8 • 46
Answers and explanations • 47
Summary **47**

Chapter 3: Building the AI Data Center Foundation **49**

Technical requirements **50**
Inside an AI-centric data center **51**
Facility and support infrastructure • 51
Compute • 52
Networking • 52
Storage • 52
Identifying rack, power, cooling, and physical-space constraints **53**
Checking rack and physical-space constraints • 53
Checking electrical capacity • 54
Checking cooling and thermal limits • 54
Keeping resilience in scope • 55
Calculating and interpreting power usage effectiveness (PUE) **55**
Working through a PUE calculation • 56
Using PUE without overinterpreting it • 57
Comparing CPU and GPU architecture for AI workloads **57**
Recognizing a control-heavy pattern • 58

Recognizing a data-parallel pattern • 58
Comparing CPU, GPU, and DPU responsibilities 59
DPU responsibilities • 59
Recognizing CUDA Cores, Tensor Cores, and RT Cores 61
Recognizing storage roles for AI workloads 62
Separating capacity, throughput, and latency • 62
Using local NVMe for active data • 63
Using shared and parallel storage • 63
Using network file services for shared operational data • 63
Using object storage for large-scale retention • 63
Using tiering and locality selectively • 63
Recognizing how compute density affects facility limits 64
Practical activities 64
Activity 1: calculating PUE for three facility scenarios • 65
Activity 2: mapping work to the CPU, GPU, or DPU • 65
Activity 3: reviewing a dense rack for hidden limits • 66
Review questions 66
Question 1 • 66
Question 2 • 67
Question 3 • 67
Question 4 • 67
Question 5 • 67
Question 6 • 68
Question 7 • 68
Question 8 • 68
Answers and explanations 68
Summary 69

Chapter 4: Scaling with NVIDIA Accelerated Computing Platforms 71

Technical requirements 72
From GPU solution categories to integrated platforms 72
Moving from an accelerator to a platform • 73
Recognizing the platform around the accelerator • 74
Host processing and system memory • 74
Local accelerator communication • 74

Storage, networking, management, and software • 74
Understanding DGX as an integrated AI system 75
Reading the component map • 75
Separating stable roles from generation-specific details • 76
Scaling from one node to a cluster and DGX SuperPOD 77
What changes when systems become a cluster • 77
Positioning DGX SuperPOD • 78
Positioning ConnectX and BlueField within the platform 79
Using ConnectX for the high-performance network path • 80
Using BlueField for infrastructure offload and isolation • 80
Using NVIDIA reference architectures appropriately 81
Recognizing platform generation changes 81
Comparing on-premises, cloud, and combined deployment patterns 81
Using on-premises when control and locality matter • 82
Using cloud for rapid access and variable demand • 82
Combining environments when workload phases differ • 83
Practice scenarios 84
Activity 1: matching solution roles to scenarios • 84
Activity 2: choosing a scale level • 86
Review questions 86
Question 1 • 86
Question 2 • 87
Question 3 • 87
Question 4 • 87
Question 5 • 87
Question 6 • 88
Question 7 • 88
Question 8 • 88
Answers and explanations 88
Summary 89

Chapter 5: Moving Data Across Accelerated AI Infrastructure 91

Technical requirements 92
Separating traffic in an AI data center 93
Separating compute and storage traffic • 94

Recognizing management and external boundaries • 95
Understanding in-band and out-of-band management networks 96
Using in-band management for routine operations • 96
Preserving an independent out-of-band path • 96
Comparing Ethernet, converged Ethernet, RoCE, and InfiniBand 97
Starting with Ethernet • 97
Converging traffic on Ethernet • 98
Adding RDMA with RoCE • 98
Using InfiniBand as a high-performance fabric • 98
Reading throughput and latency in context 100
Distinguishing bandwidth, throughput, and latency • 101
Comparing PCIe, NVLink, and NVSwitch roles 101
Using PCIe as the general system I/O fabric • 101
Using NVLink for accelerator communication • 102
Using NVSwitch to build an NVLink fabric • 103
Understanding DMA and remote direct memory access (RDMA) 105
Moving local data with DMA • 105
Extending direct-memory transfer across a network • 105
Accelerating GPU-memory network transfers with GPUDirect RDMA 106
Checking the support chain • 107
Accelerating storage access with GPUDirect Storage 108
Managing the InfiniBand fabric with OpenSM 109
Practical activity: tracing an end-to-end data path 109
Review questions 110
Question 1 • 110
Question 2 • 111
Question 3 • 111
Question 4 • 111
Question 5 • 111
Question 6 • 112
Question 7 • 112
Question 8 • 112
Answers and explanations 112
Summary 113

Chapter 6: Understanding the NVIDIA GPU Software Stack 115

Technical requirements ... **116**
Reading the GPU software dependency chain ... **116**
Placing the operating system at the base • 118
Reading the software dependency diagram • 118
Understanding the NVIDIA driver role ... **119**
Connecting the operating system to the GPU • 119
Consulting current driver documentation • 119
Keeping driver success separate from application success • 119
Using nvidia-smi for software visibility ... **120**
Reading key nvidia-smi fields • 120
Understanding CUDA and core-library placement ... **121**
Placing CUDA between the driver and higher-level software • 121
Recognizing optimized libraries • 122
Placing frameworks and applications above the core stack • 122
Recognizing the official CUDA installation path ... **122**
Following the current installation guide • 122
Understanding NCCL collectives ... **123**
Separating NCCL from the data path • 123
Recognizing common collective operations • 123
Using topology awareness without overclaiming automation • 125
Keeping framework integration version-aware • 125
Review questions ... **125**
Question 1 • 126
Question 2 • 126
Question 3 • 126
Question 4 • 126
Question 5 • 127
Question 6 • 127
Question 7 • 127
Question 8 • 127
Answers and explanations ... **128**
Summary ... **128**

Chapter 7: Virtualizing Accelerated Infrastructure with vGPU and MIG 129

Technical requirements 130
Why virtualize or share GPUs 130
Starting from the workload requirement • 131
Recognizing the benefits and limits • 131
GPU virtualization foundations 131
Understanding NVIDIA vGPU 133
Following the host-to-guest path • 133
Understanding MIG 133
Reading a MIG profile • 134
Comparing vGPU and MIG 135
Choosing between the mechanisms • 135
Checking support and licensing • 136
Inspecting MIG support and profiles 136
Identifying the target GPU • 136
Comparing available profiles with existing instances • 137
Interpreting what the commands prove • 138
Review questions 139
Question 1 • 139
Question 2 • 139
Question 3 • 140
Question 4 • 140
Question 5 • 140
Question 6 • 141
Question 7 • 141
Question 8 • 141
Answers and explanations 141
Summary 142

Chapter 8: Mapping NVIDIA Software to the AI Lifecycle 143

Technical requirements 144
The four-stage AI workflow 144
Frameworks and accelerated libraries 145

Placing accelerated libraries underneath the workflow • 146
Data preparation and development tools 147
Using RAPIDS for data processing and analytics • 147
Using NeMo Curator for training-data curation • 148
Using NeMo Framework for model development • 148
Optimization and inference tools 148
Understanding TensorRT's build and runtime roles • 149
Using NVIDIA Dynamo-Triton for model serving • 149
Using NIM for a packaged inference capability • 150
Choosing between responsibilities rather than names • 150
NVIDIA AI Enterprise 151
Selected vertical solutions and NVIDIA GPU Cloud (NGC) 152
Using NGC as a catalog and distribution source • 152
Keeping distribution separate from support • 152
Recognizing selected domain-specific software • 152
Understanding the AI factory concept 153
Connecting the architecture to an outcome • 154
Review questions 154
Question 1 • 154
Question 2 • 155
Question 3 • 155
Question 4 • 155
Question 5 • 155
Question 6 • 156
Question 7 • 156
Question 8 • 156
Answers and explanations 156
Summary 157

Chapter 9: Monitoring and Managing GPU Infrastructure 159

Technical requirements 160
Monitoring goals and GPU measures 160
Choosing metrics that answer operational questions • 162
Reading GPU execution utilization • 163
Separating memory capacity from memory activity • 163

Interpreting temperature, power, and clocks together • 163
Using error and health evidence proportionally • 164
Using nvidia-smi for immediate inspection 165
Capturing a selective, script-friendly snapshot • 165
Repeating a local observation without mistaking it for history • 166
Turning the snapshot into a next action • 166
Understanding DCGM 167
Understanding entities, groups, fields, and watches • 167
Separating passive health monitoring from active diagnostics • 168
Using DCGM Exporter, Prometheus, and Grafana 170
Using Prometheus for collection and retained telemetry • 170
Using Grafana to visualize retained telemetry • 170
Positioning Base Command Manager 171
Choosing the right monitoring or management tool 171
Simple first-response reasoning 172
Review questions 173
Question 1 • 173
Question 2 • 173
Question 3 • 173
Question 4 • 173
Question 5 • 174
Question 6 • 174
Question 7 • 174
Question 8 • 174
Answers and explanations 175
Summary 175

Chapter 10: Scheduling and Orchestrating Accelerated Workloads 177

Technical requirements 178
Scheduling versus orchestration 179
Workload type is evidence, not a verdict • 180
Understanding Slurm essentials 181
Thinking in jobs, allocations, and partitions • 181
Priorities, reservations, and policy controls • 181
Exposing GPUs as schedulable resources • 182

Following the job lifecycle • 182
Keeping communication separate from scheduling • 182
Understanding Kubernetes essentials 183
Starting with Pods and desired state • 183
Services, health, and scaling • 183
Requesting GPU resources conceptually • 184
Comparing Slurm and Kubernetes 184
Recognizing NVIDIA integration points 186
Slurm and NVIDIA GPU resources • 186
Kubernetes and NVIDIA GPU enablement • 186
Operating mixed environments 187
Separating resource ownership from workload handoff • 188
Using monitoring without confusing it with control • 188
A responsibility-first scenario method • 188
Review questions 189
Question 1 • 189
Question 2 • 189
Question 3 • 189
Question 4 • 189
Question 5 • 190
Question 6 • 190
Question 7 • 190
Question 8 • 190
Answers and explanations 191
Summary 192

Chapter 11: Recognizing MLOps Practices in the AI Lifecycle 193

Technical requirements 194
Why MLOps matters 194
From a linear workflow to a managed loop • 194
Tracking data, code, models, and environments 195
Versioning and lineage answer different questions • 195
Understanding the model registry concept • 196
Recognizing CI/CD and event-driven retraining 196
Continuous integration in an ML workflow • 196

Continuous delivery and controlled promotion • 196
Event-driven retraining is a trigger pattern, not a rule • 197
Mapping NVIDIA tools to lifecycle operations 197
Connecting monitoring feedback to the lifecycle 198
Monitoring creates evidence, not automatic action • 198
A simple production scenario • 199
Review questions 199
Question 1 • 199
Question 2 • 199
Question 3 • 200
Question 4 • 200
Question 5 • 200
Question 6 • 200
Answers and explanations 201
Summary 201

Chapter 12: Preparing for NCA-AIIO 203

Technical requirements 204
Rechecking the current blueprint and exam logistics 204
Reviewing Essential AI Knowledge 206
Build relationships, not isolated definitions • 207
Use lifecycle responsibility to control product-name confusion • 208
Reviewing AI Infrastructure 208
Review the system path from facility to workload • 209
Keep network terms in the correct category • 210
Distinguish scale from completeness • 210
Reviewing AI Operations 210
Choose monitoring tools by responsibility • 211
Separate job scheduling from continuous orchestration • 211
Keep virtualization terms tied to the sharing goal • 211
Recognizing high-value comparisons and terminology traps 212
Reading scenario questions systematically 214
Step 1: Identify the requested outcome • 214
Step 2: Locate the layer and domain • 214
Step 3: Extract the constraint • 215

Step 4: Match the evidence to the responsibility • 215
Step 5: Eliminate wrong-layer and overclaim answers • 215
Step 6: Choose the answer that satisfies the scenario as written • 215
Readiness planning and targeted revision 216
Maintain a weak-area tracker • 216
Use measurable evidence of readiness • 216
Build the final two-week pass around gaps • 217
Use mock exams as measurement, not prediction • 217
Mixed-domain practice set 217
Question 1 • 218
Question 2 • 218
Question 3 • 218
Question 4 • 219
Question 5 • 219
Question 6 • 219
Question 7 • 220
Question 8 • 220
Answers and explanations 220
Summary 221

Chapter 13: Unlock Your Exclusive Benefits 223

Unlock this Book's Free Benefits in 3 Easy Steps 224

Appendix A: NCA-AIIO Objective Map and Mock Exams 227

Objective-to-chapter map 227
Mock-exam weighting and final review cues 230
How to attempt the mock exams 231
Mock Exam 1 232
Mock Exam 1 answers and explanations 242
Mock Exam 2 246
Mock Exam 2 answers and explanations 257
Post-exam review and weak-area retest 261

Appendix B: Referenced Resources and Further Reading 263

Using this appendix 263

Chapter 1: Navigating NCA-AIIO and the NVIDIA Technology Stack 264
Chapter 2: Understanding AI Workloads, Training, and Inference 264
Chapter 3: Building the AI Data Center Foundation 264
Chapter 4: Scaling with NVIDIA Accelerated Computing Platforms 265
Chapter 5: Moving Data Across Accelerated AI Infrastructure 265
Chapter 6: Understanding the NVIDIA GPU Software Stack 266
Chapter 7: Virtualizing Accelerated Infrastructure with vGPU and MIG 266
Chapter 8: Mapping NVIDIA Software to the AI Lifecycle 267
Chapter 9: Monitoring and Managing GPU Infrastructure 268
Chapter 10: Scheduling and Orchestrating Accelerated Workloads 268
Chapter 11: Recognizing MLOps Practices in the AI Lifecycle 269
Chapter 12 and Appendix A: Certification Preparation and Mock Exams 269
Keeping the resource list current 269

Other Books You May Enjoy 272

Index 275

Preface

Accelerated AI infrastructure is a full-stack discipline. A high-performance GPU can still underdeliver if power, cooling, storage, networking, software compatibility, resource allocation, or monitoring is wrong. For professionals preparing for NCA-AIIO, the challenge is to understand how these layers work together rather than memorize isolated product names.

NVIDIA NCA-AIIO Certification Study Guide is organized around that operational view. It starts with AI and workload fundamentals, then moves through data center foundations, NVIDIA accelerated computing platforms, data movement and networking, the GPU software stack, virtualization and resource sharing, lifecycle software, monitoring, scheduling, orchestration, and MLOps. A dedicated final chapter brings the material back to the NCA-AIIO objectives and exam strategy.

The book combines certification preparation with practical infrastructure reasoning. You will learn to translate workload behavior into compute, memory, storage, network, availability, and cost requirements; trace data across PCIe, NVLink, NVSwitch, Ethernet, InfiniBand, RDMA, and GPUDirect paths; validate the NVIDIA driver and CUDA layers; compare GPU-sharing approaches such as vGPU and MIG; map NVIDIA software to AI lifecycle tasks; and build monitoring and operational workflows.

Where product versions or certification details can change, the book treats them as version-sensitive and encourages you to verify current official documentation. GPU access can make some validation exercises more useful, but it is not required for the core learning path.

Who this book is for

This book is for candidates preparing for the NVIDIA Certified Associate: AI Infrastructure and Operations (NCA-AIIO) exam, as well as system, data center, cloud, and infrastructure administrators moving into GPU environments. Network engineers, operations professionals, DevOps and platform practitioners, MLOps practitioners, technical support professionals, and early-career solution architects who need practical NVIDIA infrastructure knowledge will also find it useful.

A foundational understanding of servers, operating systems, storage, and networking is assumed. Familiarity with data center or cloud concepts and basic Linux command-line use is helpful. You do not need CUDA programming, advanced model-development experience, or advanced mathematics to follow the book.

What this book covers

Chapter 1, Navigating NCA-AIIO and the NVIDIA Technology Stack, introduces what the certification validates and builds a mental model of the complete accelerated AI environment, from facilities and compute to storage, networking, system software, management tools, and workloads. It also shows how to map the exam domains to the book and build a focused study plan.

Chapter 2, Understanding AI Workloads, Training, and Inference, establishes the AI vocabulary an infrastructure professional needs. It connects AI, machine learning, deep learning, and generative AI with transformers, training, inference, model scale, and common use cases, showing how workload behavior affects compute, memory, storage, and networking requirements.

Chapter 3, Building the AI Data Center Foundation, connects facility constraints with the compute and storage layers that depend on them. It covers rack power and cooling, power usage effectiveness, CPU, GPU, and DPU responsibilities, introductory GPU architecture, storage throughput and locality, and the effect of accelerator density on data center planning.

Chapter 4, Scaling with NVIDIA Accelerated Computing Platforms, moves from an individual GPU to integrated DGX systems, clusters, and DGX SuperPOD environments. It positions ConnectX adapters and BlueField DPUs, explains the role of NVIDIA reference architectures, and compares on-premises, cloud, and combined deployment models for accelerated workloads.

Chapter 5, Moving Data Across Accelerated AI Infrastructure, explains how data moves through accelerated systems and why a fast GPU can still wait on the surrounding infrastructure. It compares Ethernet, RoCE, and InfiniBand; in-band and out-of-band management; PCIe, NVLink, and NVSwitch; and the roles of DMA, RDMA, GPUDirect RDMA, GPUDirect Storage, and OpenSM in accelerated data paths.

Chapter 6, Understanding the NVIDIA GPU Software Stack, treats the operating system, NVIDIA driver, CUDA, libraries, frameworks, and applications as a dependency chain. It explains how to read `nvidia-smi` output, distinguish driver-supported CUDA from an installed Toolkit, recognize the official installation path, and understand how NCCL collectives support multi-GPU communication.

Chapter 7, Virtualizing Accelerated Infrastructure with vGPU and MIG, compares dedicated GPU access, NVIDIA vGPU, and Multi-Instance GPU (MIG). It focuses on isolation, flexibility, workload fit, hardware and software support, licensing, and operational control. It also shows how to inspect MIG support and available profiles before choosing a GPU-sharing approach.

Chapter 8, Mapping NVIDIA Software to the AI Lifecycle, organizes NVIDIA software by responsibility rather than as a product catalog. It places accelerated libraries, RAPIDS, NeMo, TensorRT, NVIDIA Dynamo-Triton, NIM, NVIDIA AI Enterprise, NGC, and selected vertical solutions across data preparation, training, optimization, inference, and operations, and explains the AI factory concept.

Chapter 9, Monitoring and Managing GPU Infrastructure, moves from immediate node inspection to continuous cluster telemetry. It covers GPU health and performance metrics, `nvidia-smi`, DCGM, DCGM Exporter, Prometheus, Grafana, and Base Command Manager, then brings them together in a first-response approach to common GPU infrastructure symptoms.

Chapter 10, Scheduling and Orchestrating Accelerated Workloads, compares Slurm and Kubernetes by workload and operating model. It explains how GPUs are exposed to jobs and Pods, how scheduling and orchestration differ, and where NVIDIA integration tools fit. It also examines shared environments where training and inference workloads must run without confusing monitoring with workload control.

Chapter 11, Recognizing MLOps Practices in the AI Lifecycle, introduces practices for making AI workflows repeatable and manageable. It covers tracking data, code, models, and environments; versioning and lineage; model registries; CI/CD and event-driven retraining; and how monitoring feedback and NVIDIA tools support day-to-day lifecycle operations.

Chapter 12, Preparing for NCA-AIIO, returns to the certification blueprint after the technical chapters. It reviews the three exam domains, high-value comparisons, and common terminology traps, then develops an approach to scenario questions, distractor elimination, weak-area tracking, targeted revision, and mixed-domain practice to assess exam readiness.

Appendix A, NCA-AIIO Objective Map and Mock Exams, provides a compact objective-to-chapter map, a readiness checklist, and two original 50-question mock exams with answers and explanations for structured practice.

Appendix B, Referenced Resources and Further Reading, organizes chapter-specific external resources, with an emphasis on current official NVIDIA documentation, product guides, compatibility information, certification resources, and other authoritative references for version-sensitive topics.

To get the most out of this book

The chapters are designed to build from workload fundamentals to infrastructure design and then to operations. Working through them in sequence will make the later software, monitoring, scheduling, and troubleshooting sections easier to follow. Use the chapter activities, review questions, and your own weak-area tracker to connect certification objectives with practical understanding.

Before you begin, make sure you have the following:

- A basic understanding of servers, operating systems, storage, and networking
- Familiarity with data center or cloud concepts
- Basic Linux command-line familiarity for the practical chapters
- A browser for checking current NVIDIA certification and product documentation
- A text editor, notebook, or spreadsheet for requirements, version, and study-tracking exercises
- Optional access to a non-production NVIDIA GPU host for hardware-dependent validation activities

If you do not have GPU access, you can still complete the core learning path by working through the architecture, analysis, and documentation-based activities. When you do run commands, use an authorized test environment and compare the results with the current vendor documentation for the hardware and software versions you are using.

Download the color images

Your purchase includes a color, DRM-free PDF copy of this book, ideal for viewing color images, screenshots, and diagrams. Refer to *Free benefits with your book section* at the end of the *Preface* to unlock your PDF copy.

Conventions used

There are a number of text conventions used throughout this book.

Code in text: Indicates code words in text, folder names, filenames, file extensions, pathnames, commands, user input, and tool names. For example: "The `--set` option configures health watches, `--fetch` displays their configuration, and `--check` evaluates the collected samples."

Any command-line input or output is written as follows:

```
dcgmi health --set a --group g
dcgmi health --fetch --group g
dcgmi health --check --group g
```

Bold: Indicates a new term, an important word, or words that you see on the screen. For instance, a new term appears like this: "The **CUDA Version** field in `nvidia-smi` reflects the latest CUDA version supported by the installed driver; it does not prove that a CUDA Toolkit is installed."

Note

Warnings or important notes appear like this.

Tip

Tips and tricks appear like this.

Get in touch

Feedback from our readers is always welcome.

General feedback: If you have questions about any aspect of this book or have any general feedback, please email us at `customercare@packt.com` and mention the book's title in the subject of your message.

Errata: Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you have found a mistake in this book, we would be grateful if you reported this to us. Please visit `http://www.packt.com/submit-errata`, click **Submit Errata**, and fill in the form.

Piracy: If you come across any illegal copies of our works in any form on the internet, we would be grateful if you would provide us with the location address or website name. Please contact us at `copyright@packt.com` with a link to the material.

If you are interested in becoming an author: If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, please visit `http://authors.packt.com/`.

Share your thoughts

Once you've read *NVIDIA NCA-AIIO Certification Study Guide*, we'd love to hear your thoughts! Scan the QR code below to go straight to the Amazon review page for this book and share your feedback.



`https://packt.link/r/1808818474`

Your review is important to us and the tech community and will help us make sure we're delivering excellent quality content.

Free benefits with your book

This book comes with free benefits to support your learning. Activate them now for instant access (see the "*How to Unlock*" section for instructions).

Here's a quick overview of what you can instantly unlock with your purchase:







How to Unlock

Scan the QR code (or go to `packtpub.com/unlock`). Search for this book by name, confirm the edition, and then follow the steps on the page.



Note: Keep your invoice handy. Purchases made directly from Packt don't require one

Stay Sharp in Cloud and DevOps – Join 44,000+ Subscribers of CloudPro

CloudPro is a weekly newsletter for cloud professionals who want to stay current on the fast-evolving world of cloud computing, DevOps, and infrastructure engineering.

Every issue delivers focused, high-signal content on topics like:

- AWS, GCP & multi-cloud architecture
- Containers, Kubernetes & orchestration
- Infrastructure as Code (IaC) with Terraform, Pulumi, etc.
- Platform engineering & automation workflows
- Observability, performance tuning, and reliability best practices

Whether you're a cloud engineer, SRE, DevOps practitioner, or platform lead, CloudPro helps you stay on top of what matters, without the noise.

Scan the QR code to join for free and get weekly insights straight to your inbox:



`https://packt.link/cloudpro`

1

Navigating NCA-AIIO and the NVIDIA Technology Stack

To get full value from an AI system, every layer of the infrastructure has to work together. Healthy GPUs are only one part of that equation. The rack needs enough power, cooling must sustain performance, storage has to feed data at the required rate, the network must move traffic efficiently between nodes, and the software layers must remain compatible. The workload experiences these pieces as one system, even when different teams and products manage them.

A useful starting point is to understand how the pieces fit together. Knowing the names CUDA, DGX, NVLink, MIG, Kubernetes, and DCGM becomes more valuable when you can place each item in the stack and explain the problem it solves. Before we examine the individual technologies, we need a reliable map of the environment they belong to and where each fits.

The NVIDIA-Certified Associate: AI Infrastructure and Operations certification, commonly shortened to NCA-AIIO, gives us useful coverage boundaries. This book uses those boundaries to guide coverage, but it follows an operational learning journey rather than reproducing the exam blueprint in order. We will move from AI and workload fundamentals through facilities, compute, storage, networking, NVIDIA platforms, system software, monitoring, scheduling, orchestration, and lifecycle operations before returning to certification preparation in a dedicated final chapter.

In this chapter, we will cover the following topics:

- Understanding what the NCA-AIIO certification validates
- Reading the current exam domains and objective weights
- Using the current NVIDIA blueprint and study guide as the source of truth

- Identifying the roles that deploy, operate, and support AI infrastructure
- Using the six-layer NVIDIA technology stack as the book's technical map
- Separating AI knowledge, infrastructure, and operations concerns
- Distinguishing durable concepts from version-specific product details
- Using chapter objectives, exam-focus notes, exercises, and review questions
- Creating a realistic study and practice plan

By the end of this chapter, you will be able to map the NCA-AIIO domains to the book, use the current blueprint and study guide as the source of truth, explain the course's six-layer NVIDIA technology stack, relate supporting infrastructure to that stack, classify common responsibilities by domain, and build a focused preparation plan with measurable review checkpoints.

> **Note**
>
> **Download the PDF version of this book**
>
> Your purchase includes a DRM-free PDF copy of this book and a range of exclusive benefits. To unlock everything, follow the *Free benefits with your book* section in the *Preface*.

Technical requirements

This chapter is conceptual and planning-oriented. You do not need a GPU, a cloud account, an NVIDIA software installation, or access to an AI cluster. To complete the activities, you need the following:

- A basic understanding of servers, operating systems, storage, and networking
- A browser for checking the current NVIDIA certification page and exam blueprint
- A text editor, notebook, or spreadsheet for the baseline assessment and study tracker
- A willingness to explain concepts in your own words rather than memorize product lists

Use the current NCA-AIIO page as the authoritative source for scheduling details and blueprint changes: `https://www.nvidia.com/en-us/learn/certification/ai-infrastructure-operations-associate/`

Note

Exam focus: This chapter spans all three NCA-AIIO domains. Its purpose is to help you read the blueprint, recognize the course's six-layer NVIDIA technology stack, and decide where later chapters fit. It is not a replacement for the current official exam guide.

Understanding what NCA-AIIO validates

NCA-AIIO is an associate-level credential for people who need a foundational understanding of AI computing infrastructure and operations. It is relevant to data center technicians, systems administrators, network engineers, DevOps and platform engineers, IT managers, delivery teams, and solution or systems architects. The common thread is responsibility for the environment that makes AI workloads usable, reliable, and supportable.

The certification does not ask an infrastructure professional to become a data scientist. It expects enough AI knowledge to understand what a workload is trying to do, enough infrastructure knowledge to identify the resources and data paths it needs, and enough operations knowledge to monitor, schedule, virtualize, and troubleshoot the environment. That division is central to the book.

Checking the current exam snapshot

At the time of writing, NVIDIA publishes the exam details summarized in *Table 1.1*. Use them to orient your preparation, but verify the current certification page before scheduling the exam:

Exam fact	Current detail	Planning implication
Credential level	Associate	Foundational coverage of AI infrastructure and operations
Delivery	Online and remotely proctored	Review the examination policy before scheduling
Time limit	60 minutes	Plan for concise reading and decision-making
Number of questions	50	Use the blueprint for coverage, not for predicting question order

Exam fact	Current detail	Planning implication
Price	USD 125	Confirm the current price before registering
Prerequisite	Basic data center infrastructure knowledge	The book develops the AI-specific layers from that base
Validity	Two years from issuance	Recertification may be achieved by retaking the exam

Table 1.1: Current NCA-AIIO exam snapshot

Note

The details in *Table 1.1* were rechecked against the official NVIDIA certification page in September 2026. Certification pages, prices, delivery platforms, validity periods, and blueprint objectives can change. Verify them again before you schedule the exam.

Reading the three exam domains

The current blueprint divides the exam into three domains: Essential AI Knowledge, AI Infrastructure, and AI Operations. The percentages show relative emphasis. They are useful for planning study time, but they should not be treated as isolated subject silos. A realistic scenario can begin with a workload requirement, move through infrastructure choices, and finish with an operational symptom.

Figure 1.1 shows the current domain weights and the broad focus of each domain:

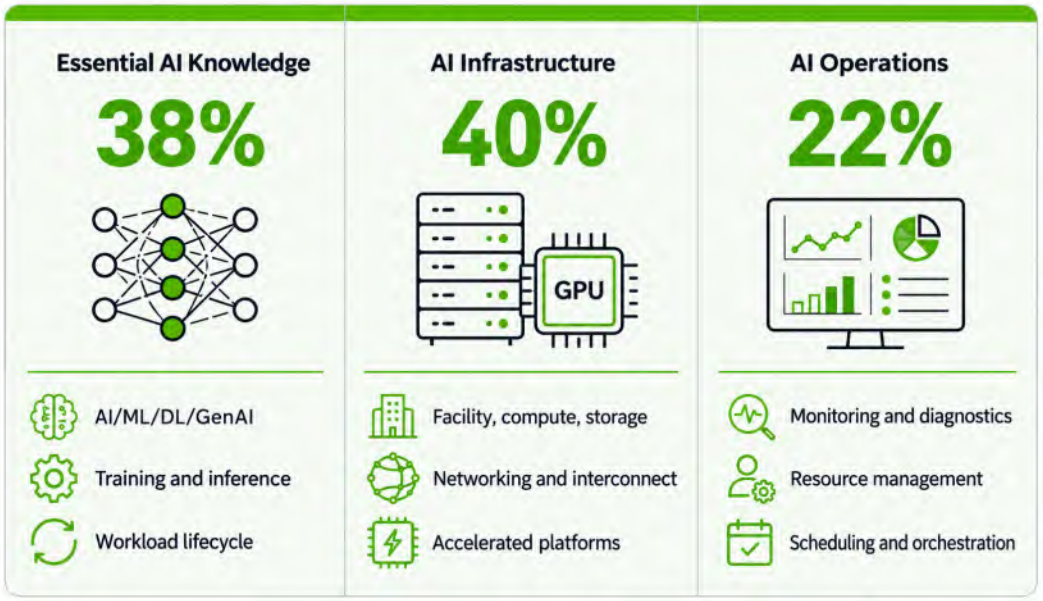


Figure 1.1: NCA-AIIO blueprint at a glance

Essential AI Knowledge: understanding the workload

Essential AI Knowledge currently represents 38% of the blueprint. It covers accelerated-computing use cases, the distinction among AI, machine learning, and deep learning, training and inference considerations, CPU and GPU architecture at a practical level, the NVIDIA software stack, and the software components used across the AI lifecycle.

For an infrastructure professional, this domain answers a simple question: what is the workload asking the platform to do? You do not need to derive neural-network mathematics, but you do need to recognize whether a scenario emphasizes training, inference, data preparation, optimization, or deployment. Identifying the stage helps you prioritize requirements: training often emphasizes compute, memory, and data throughput, while interactive inference puts more emphasis on response latency and availability. Storage, networking, and cost still matter, but their relative importance depends on the workload.

AI Infrastructure: building the environment

AI Infrastructure currently represents 40% of the blueprint, the largest single share. It covers hardware requirements, scaling GPU infrastructure, facility power and cooling, on-premises and cloud considerations, accelerated clusters, networking, high-speed data center options, and the purpose of DPUs.

This domain asks what the workload needs beneath the software. A powerful GPU is only one part of the answer. The environment must supply power, remove heat, move data, provide capacity, connect nodes, preserve availability, and fit the organization's deployment and support model. Chapters 3 through 7 develop the facility, platform, data-movement, software-stack, and resource-sharing topics, while Chapter 8 connects NVIDIA software to AI lifecycle responsibilities.

AI Operations: keeping the environment useful

AI Operations currently represents 22% of the certification exam blueprint. It covers data center management and monitoring, GPU utilization tracking, cluster orchestration, job scheduling, and virtualization considerations for accelerated infrastructure.

A smaller coverage percentage does not make operations optional. Infrastructure creates capability, while operations helps keep it healthy, visible, properly allocated, and recoverable. A design that cannot be monitored or supported is incomplete, even if its hardware specification looks impressive.

Using weights without creating blind spots

The weights help you decide where to spend time, but they do not justify ignoring any domain. Use them to prioritize revision without treating the three domains as separate subjects.

For each topic, connect the workload goal to the relevant technology, infrastructure dependency, and operational evidence. This system view helps you learn the relationships rather than collect definitions. It also explains why no single team owns the entire environment.

Using the blueprint and study guide as the source of truth

The course helps you learn the concepts, but the current NVIDIA certification page, exam blueprint, and study guide should control the exam scope. Use the course and this book to build understanding, then use the current blueprint to confirm the objectives, domain weights, and exam logistics that apply when you prepare.

If an older course slide, screenshot, or product detail conflicts with current NVIDIA certification material, treat the current official certification source as authoritative for objective wording and logistics. For product behavior, versions, interfaces, and implementation details, later chapters distinguish durable concepts from facts that need current technical verification.

Using roles as operational context

AI infrastructure is cross-functional by design. A single organization may use separate teams for facilities, servers, storage, networking, cloud, platform engineering, security, and machine learning. A smaller organization may assign several of those responsibilities to one person. These roles provide useful operational context, but the certification tests concepts and responsibilities rather than organization-specific job titles.

Table 1.2 summarizes the questions that common operational roles bring to the same AI platform:

Operational role	Primary question	Typical evidence or handoff
Facilities and data center operations	Can the site provide the required rack space, power, cooling, physical security, and resilience?	Rack capacity, power budget, cooling design, environmental and facility alarms
Systems and GPU platform administration	Are the servers, GPUs, firmware, operating system, drivers, and runtime layers supported and healthy?	Inventory, version records, hardware health, installation and validation results
Network engineering	Can management, storage, cluster, and external traffic meet requirements for throughput, latency, and resilience?	Topology, link state, counters, fabric configuration, path and congestion evidence
Storage engineering	Can datasets, checkpoints, and models be stored and delivered to GPU systems with enough capacity, throughput, and availability for the workload?	Capacity, I/O performance, data locality, protection, and recovery evidence
Cloud and solution architecture	Which deployment model satisfies control, speed, compliance, cost, availability, and skills constraints?	Requirements, decision records, service limits, cost and risk assumptions

Operational role	Primary question	Typical evidence or handoff
Platform, DevOps, and MLOps engineering	How are environments, containers, jobs, dependencies, artifacts, and deployment workflows made repeatable?	Build definitions, images, manifests, pipelines, registries, lineage, and policy records
Operations and technical support	Is the infrastructure healthy, what has changed in the workload or environment, and what evidence supports recovery or escalation?	Metrics, logs, alerts, diagnostics, incidents, runbooks, and escalation history
AI and application teams	What does the workload need, and how will success be measured?	Dataset and model characteristics, training or inference objectives, latency, throughput, and quality targets

Table 1.2: Roles and questions across an AI infrastructure environment

Responsibility boundaries vary by organization. Table 1.2 illustrates common handoffs, not job titles or exam objectives to memorize.

Keeping ownership separate from exam scope

A team may own a component without owning every symptom that appears around it. For certification preparation, focus first on the technology, dependency, and evidence in the scenario rather than trying to memorize an organization chart.

The role map provides context for the six-layer NVIDIA technology stack introduced next.

Reading the six-layer NVIDIA technology stack

The course's six-layer NVIDIA technology stack gives the book a consistent technical map. Facility, storage, networking, and scheduling concerns interact with the stack without becoming competing layers.

Figure 1.2 shows the model from physical hardware through applications and vertical solutions. Follow it upward to see the dependencies between layers; the integrations at the side connect to the stack without forming additional layers:

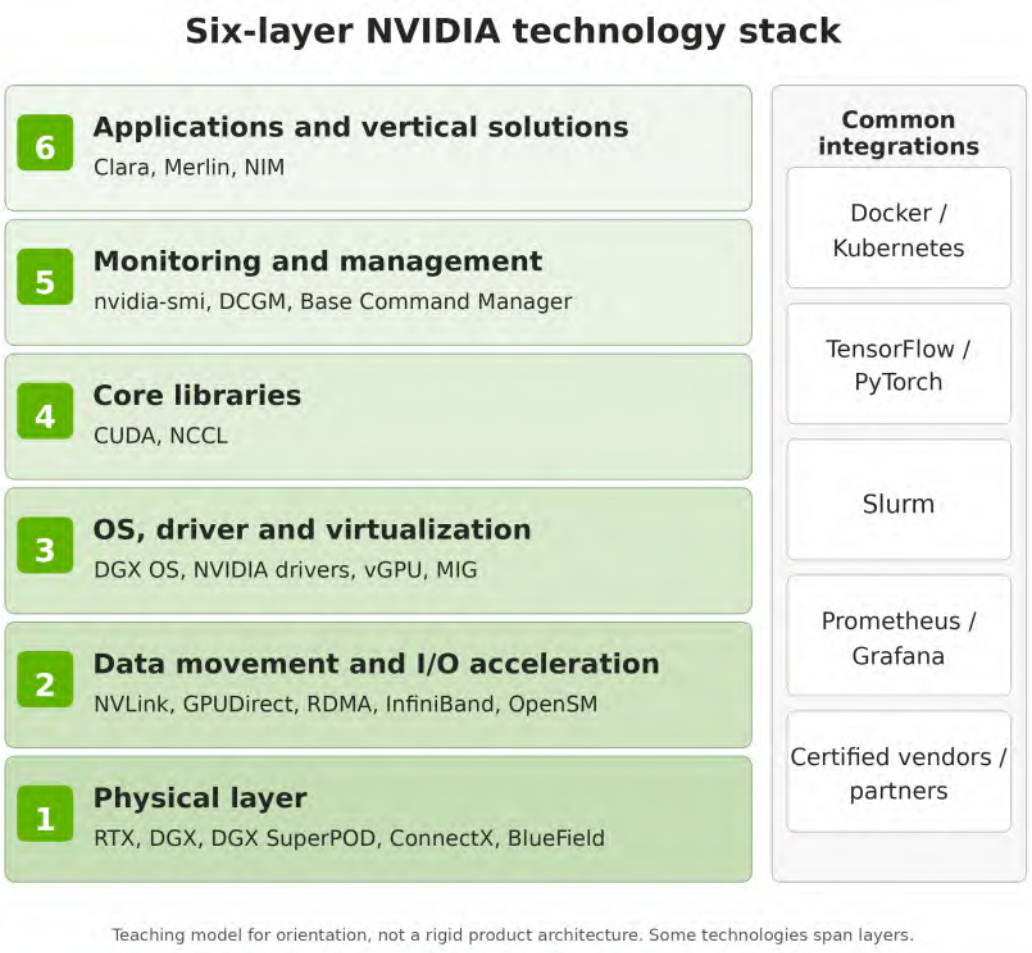


Figure 1.2: Course six-layer NVIDIA technology stack

The six layers are a teaching model rather than rigid product boundaries. Some technologies interact with more than one layer, but using the course's own model gives you one consistent way to place products and responsibilities while preparing for the certification.

Physical layer

The physical layer contains the accelerated systems and devices that provide the platform foundation. In the course, this layer includes systems and devices such as RTX, DGX, DGX SuperPOD, ConnectX, and BlueField. Chapter 3 develops the data center foundation around these components, while Chapter 4 examines NVIDIA accelerated computing platforms in more detail.

Data movement and I/O acceleration

The second layer focuses on moving data efficiently between accelerators, systems, storage, and nodes. The course groups technologies and tools such as NVLink, GPUDirect, RDMA, InfiniBand, and OpenSM in this part of the stack. Chapter 5 develops these data paths and clarifies the boundaries between local interconnects, network fabrics, RDMA, GPUDirect, and collective communication.

OS, driver, and virtualization

The third layer turns hardware into usable accelerated resources through the operating system, NVIDIA drivers, and virtualization capabilities. It is also where vGPU and Multi-Instance GPU (MIG) fit in the course model. Chapters 6 and 7 cover the GPU software foundation and the main virtualization approaches at the depth required for NCA-AIIO.

Core libraries

The fourth layer groups software components that expose and coordinate accelerated capabilities for higher-level software. CUDA provides the foundational programming platform, while NCCL provides collective communication for multi-GPU and multi-node workloads. Chapter 6 explains their roles without turning the certification guide into a programming manual.

Monitoring and management

The fifth layer provides visibility into and control over accelerated systems. The course introduces tools such as `nvidia-smi`, DCGM, and Base Command Manager here. Chapter 9 returns to these tools in a certification-focused monitoring and management context.

Other tools can integrate with the stack without creating new NVIDIA stack layers. For example, Prometheus and Grafana can participate in monitoring workflows, while Docker, Kubernetes, Slurm, TensorFlow, and PyTorch connect to different layers according to the job they perform.

Applications and vertical solutions

The sixth layer is where accelerated infrastructure supports applications and industry-focused solutions. The course uses examples such as Clara, Merlin, and NIM to illustrate this layer. Chapter 8 maps NVIDIA software and services to the AI lifecycle so you can recognize what each component contributes without memorizing a product catalog.

Supporting infrastructure around the stack

The stack organizes the NVIDIA technologies introduced above. Chapters 3 through 5 develop the surrounding facility, platform, storage, and data-movement requirements, while Chapters 9 and 10 cover monitoring, scheduling, and orchestration.

Connecting the stack to the AI lifecycle

The stack also needs to be read in the context of the workload. Training and inference place different emphasis on compute, memory, data movement, latency, and scale, while the broader AI lifecycle moves through data preparation, training, optimization, deployment or inference, and monitoring. Chapter 2 establishes the training and inference distinction, Chapter 8 maps software to lifecycle stages, and Chapter 11 introduces the course's MLOps material.

Using the stack as an orientation model

The stack helps you place a technology before you study it in depth. If a scenario mentions low GPU utilization, storage behavior, an interconnect, virtualization, or a monitoring tool, first identify the relevant layer and exam domain. Do not treat the six layers as a troubleshooting runbook or assume that symptoms must be investigated in a fixed order.

Later chapters develop the evidence and tool boundaries for specific technologies. In Chapter 1, the purpose of the stack is narrower: it gives you a consistent map for recognizing where a concept belongs and where to study it next.

Separating AI knowledge, infrastructure, and operations concerns

The six-layer NVIDIA technology stack shows where technologies fit. The three exam domains describe what you need to understand about them. For example, GPUs are part of the stack's physical layer. Questions about CPU and GPU roles relate to Essential AI Knowledge; deploying GPU systems relates to AI Infrastructure; and checking GPU health relates to AI Operations. The same technology can therefore appear in more than one exam domain.

Table 1.3 shows the primary question and evidence associated with each domain:

Exam domain	Primary question	Typical evidence
Essential AI Knowledge	What is the workload doing, and why does the technology matter?	Use case, lifecycle stage, training or inference behavior, software responsibility, CPU or GPU fit
AI Infrastructure	What physical and logical resources are required to provide the capability?	Facility limits, hardware, topology, storage, network, platform scale, cloud or on-premises constraints
AI Operations	How is the environment allocated, observed, maintained, virtualized, and recovered?	Health and performance metrics, scheduler state, orchestration state, alerts, diagnostics, changes, and runbooks

Table 1.3: Three reasoning views of the same AI platform

Working through a cross-domain scenario

Imagine that a multi-node model-training job completes, but its throughput is much lower than expected. Essential AI Knowledge helps you recognize that distributed training depends on repeated computation and communication. AI Infrastructure directs attention to GPU topology, network fabric, storage throughput, and data paths. AI Operations asks whether the GPUs are healthy and utilized, whether the scheduler placed the job correctly, and whether errors or congestion appeared during the run.

A weak answer chooses a product name immediately. A stronger answer states the workload objective, identifies the likely boundaries, requests evidence, and then selects a technology or corrective action. That reasoning pattern is useful for the certification and for real operations, and it remains useful even when product names, interfaces, and supported versions change.

Distinguishing durable concepts from version-specific details

NVIDIA products, supported versions, console screens, certification details, and packaging change faster than the architectural problems they address. A useful technical book must teach both, but it should distinguish enduring principles from facts that require current verification. Durable concepts help you reason across generations. Version-specific facts help you implement or schedule something today.

Table 1.4 separates examples of the two:

Area	Durable concept to remember	Version-specific fact to verify
Compute role	Parallel workloads can benefit from GPU throughput; CPUs and DPUs retain different responsibilities	A particular GPU model, memory size, form factor, or benchmark result
Data movement	Performance depends on moving data efficiently between storage, memory, processors, and nodes	Supported link speeds, adapter models, firmware, topology limits, or feature names
Software compatibility	Drivers, toolkits, libraries, frameworks, and applications must form a supported chain	The exact supported version matrix and installation commands
Resource sharing	Sharing requires explicit decisions about isolation, scheduling, performance, ownership, and rollback	Supported vGPU releases, MIG-capable hardware, profiles, and licensing terms
Operations	Operators need observable evidence of GPU health, utilization, memory use, temperature, power, errors, jobs, and changes	Supported monitoring-tool versions, available command options, dashboard labels, or management interface layouts

Area	Durable concept to remember	Version-specific fact to verify
Certification	Understand workloads, infrastructure dependencies, and operational responsibilities	Price, delivery platform, question count, validity, domain weights, and objective wording

Table 1.4: Durable concepts and details that require current verification

Using a version-aware reading habit

When you encounter a product or procedure, record four things: its purpose, the layer it belongs to, the dependency it requires, and the date or version of the source. The first three help you transfer knowledge. The fourth helps you decide when to check for updates.

When comparing product versions, focus on the features that matter to the workload. Check which capabilities changed, whether the required hardware, driver, and software combinations are supported, and whether limits or licensing have changed. Use current NVIDIA release notes and support documentation to verify the comparison; a newer version is not automatically the best fit.

This book follows that habit. Product-specific procedures are presented where they add practical value, but version-sensitive details should be checked against current official documentation. A screenshot is evidence of one interface at one point in time, not a permanent definition of the service.

Using this book as a learning system

The NCA-AIIO blueprint spans workload knowledge, infrastructure, and operations, but the book organizes that coverage into four parts so that concepts, decisions, validation, monitoring, and troubleshooting develop in a practical sequence. *Figure 1.3* shows the progression and the main chapter dependencies:

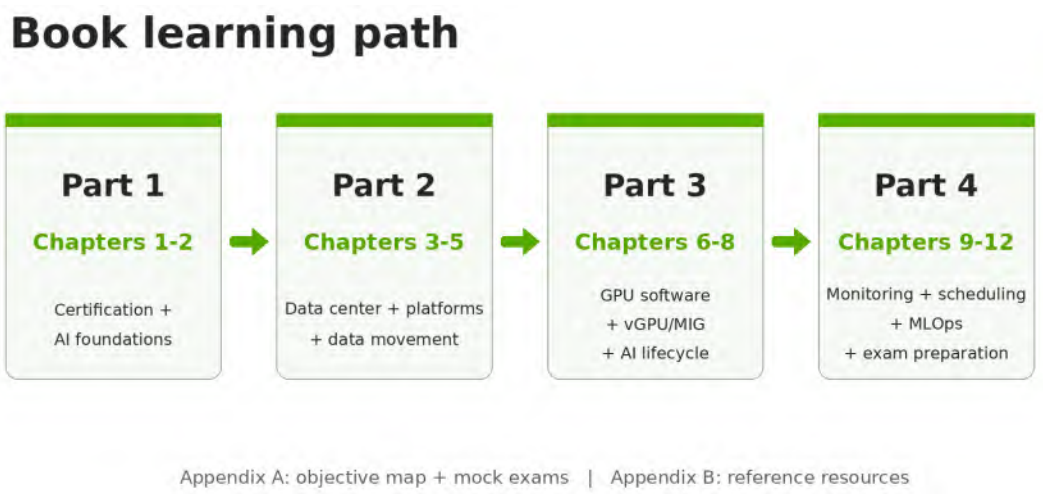


Figure 1.3: Book learning path and chapter dependencies

Read the four parts from left to right; the following sections explain each one. Appendix A provides the objective map and two original mock exams, while Appendix B lists chapter-organized resources and further reading.

Part 1: Establishing AI and workload foundations

Chapters 1 and 2 establish the certification context and the AI vocabulary used throughout the book. Chapter 2 distinguishes AI, machine learning, deep learning, generative AI, training, inference, and transformer concepts at the level an infrastructure professional needs. That comparison provides the workload context for the infrastructure chapters that follow.

Part 2: Designing accelerated AI infrastructure

Chapters 3 through 5 move into the accelerated infrastructure itself. Chapter 3 covers the AI data center foundation, including facility, compute, DPU, storage, and deployment considerations. Chapter 4 scales from components to NVIDIA accelerated computing platforms. Chapter 5 traces networking and accelerated data paths across the system and cluster.

Part 3: Building the NVIDIA software and virtualization stack

Chapters 6 through 8 turn the hardware platform into a usable software environment. Chapter 6 explains the GPU software stack and the roles of drivers, CUDA, and NCCL. Chapter 7 compares vGPU and MIG. Chapter 8 maps NVIDIA software categories to data preparation, training, optimization, inference, and enterprise operations.

Part 4: Operating and optimizing AI infrastructure

Chapters 9 through 12 complete the certification journey. Chapter 9 focuses on monitoring and management, Chapter 10 covers scheduling and orchestration, Chapter 11 introduces the course's MLOps concepts, and Chapter 12 returns to the complete blueprint for targeted revision and exam preparation.

Using the recurring chapter elements

Each chapter uses recurring elements for a different purpose:

- **Chapter objectives:** Define what you should be able to explain, decide, validate, or troubleshoot after completing the chapter.
- **Exam-focus notes:** Connect a topic to the blueprint without predicting live exam questions or replacing the official guide.
- **Figures and tables:** Build mental models, comparisons, data paths, and decision boundaries that are easier to recall than isolated terms.
- **Practical workflows:** Show how to gather evidence, validate a layer, or make a controlled decision when an environment is available.
- **Scenario exercises:** Require you to classify a problem, state assumptions, choose evidence, and justify a response.
- **Troubleshooting and production notes:** Separate a learning exercise from the additional controls required in a supported environment.
- **Review questions:** Test understanding with original questions and explanations rather than reproduce live exam content.

Read the book in sequence the first time. Later, use *Appendix A* for the objective map, readiness review, and two original mock exams, and use *Appendix B* for chapter-organized resources and further reading, prioritizing current official NVIDIA documentation. Next, identify your knowledge gaps.

Completing a baseline knowledge check

A study plan is only useful when it begins with evidence. Rate each checkpoint in *Table 1.5* before reading the later chapters. Use 0 for "new to me," 1 for "partly understood," and 2 for "able to explain or apply without notes." Be strict. Recognition is not the same as explanation.

Domain	Checkpoint	Rating
Essential AI Knowledge	Differentiate AI, machine learning, deep learning, and generative AI	0 / 1 / 2
	Explain the practical difference between training and inference	0 / 1 / 2
	Compare CPU and GPU roles for suitable workloads	0 / 1 / 2
	Place major software responsibilities within the AI lifecycle	0 / 1 / 2
AI Infrastructure	Explain why rack power, cooling, and physical space constrain dense GPU systems	0 / 1 / 2
	Explain how data reaches GPUs from storage through memory and, when needed, network paths	0 / 1 / 2
	Compare on-premises, cloud, and hybrid deployment considerations	0 / 1 / 2
	Explain the high-level roles of a GPU, DPU, interconnect, and cluster platform	0 / 1 / 2
AI Operations	Identify meaningful GPU health and performance evidence	0 / 1 / 2
	Distinguish monitoring, job scheduling, and container orchestration responsibilities	0 / 1 / 2
	Explain how dedicated GPU access, vGPU, MIG, and time slicing differ or can be combined	0 / 1 / 2
	Move from a workload symptom to the first useful layer and evidence to inspect	0 / 1 / 2

Table 1.5: Baseline knowledge check across the three exam domains

Turning ratings into priorities

Calculate the average rating for each domain, then compare the result with the published domain weights. Start with topics that have both a low self-rating and meaningful blueprint coverage, while keeping enough time for cross-domain scenarios. This avoids false precision: the purpose of the baseline is to direct practice, not to predict an exam score.

For example, if AI Infrastructure is your weakest domain and it also carries the largest published weight, it should receive the largest share of early study time. A stronger score in Essential AI Knowledge does not remove the need to review it, especially because realistic scenarios cross domain boundaries.

Keep the original ratings. Repeating the assessment after *Parts 1, 2, 3, and 4* gives you measurable checkpoints. The goal is not to make every number look perfect. The goal is to identify topics you still struggle to explain or apply. Once you know where the gaps are, test whether you can place real tasks in the correct domain and ownership boundary.

Treat this as a starting diagnostic rather than a final readiness score. Chapter 12 returns to the complete blueprint, mock exams, and remaining weak areas after you have worked through the technical chapters.

Classifying tasks by domain and operational owner

The following exercise connects the blueprint to real responsibilities. For each task, identify the primary exam domain and the operational owner most likely to lead it. Some tasks span more than one domain or team, so state the primary decision being made.

1. Explain why a large training job and an interactive inference service can require different infrastructure priorities.
2. Check whether a data center room can support the power and cooling needs of a proposed dense GPU deployment.
3. Select the storage and network path for moving training data to a multi-node cluster.
4. Install and validate a supported operating system, driver, CUDA, library, and framework chain.
5. Evaluate dedicated GPU access, vGPU, MIG, and time slicing for a shared platform, including where methods can be combined.
6. Investigate sustained high temperature, memory errors, and low utilization on a group of GPUs.

7. Choose how queued training jobs and containerized inference services will be scheduled or orchestrated.
8. Map a software component to data preparation, training, optimization, inference, monitoring, or enterprise support.

Suggested classification

One reasonable classification is shown in *Table 1.6*. Different organizations may use different job titles, but the domain and lead-owner choices should remain defensible:

Task	Primary domain	Likely lead owner
1. Training versus inference priorities	Essential AI Knowledge	AI/platform architect with the workload owner
2. Rack power and cooling capacity	AI Infrastructure	Facilities or data center operations
3. Storage and network data path	AI Infrastructure	Storage and network engineering with the platform architect
4. Software stack installation and validation	AI Infrastructure	Systems or GPU platform administration
5. GPU sharing model	AI Operations	Virtualization or platform administration with workload owners
6. Temperature, errors, and utilization	AI Operations	GPU operations or technical support, escalating to facilities or hardware when evidence points there
7. Scheduling and orchestration model	AI Operations	Platform, DevOps, MLOps, or cluster administration

Task	Primary domain	Likely lead owner
8. Software-to-lifecycle mapping	Essential AI Knowledge	Platform architect or MLOps engineer

Table 1.6: Suggested task, domain, and owner classification

Tasks 4 and 6 demonstrate why classifications are not walls. Installing the stack is an infrastructure activity, but validating and maintaining it is operational. A temperature alert begins in operations, yet its cause can lie in workload behavior, hardware, or facility cooling. State the primary concern, then acknowledge the handoffs. Use these classifications with your baseline ratings in the next section.

Starting a realistic study and practice plan

A useful plan turns a broad goal into observable work. Instead of "Study NVIDIA," set a task such as "Explain the stack's six layers without notes and classify five scenarios by domain." Record what you can demonstrate, not just the time spent reading.

Step 1: Define the time boundary

Choose a target exam date and plan backward from it. Set aside time for final revision, then divide the remaining weeks into reading, practice, and review goals. Work out how much time you can realistically study each day, even if it is only a short session, and use those hours to set achievable weekly targets. Reserve the final review period for consolidation rather than learning major topics for the first time.

Step 2: Convert domain gaps into chapter work

Use the baseline scores and the book map. Weak AI vocabulary or training-versus-inference knowledge points to Chapter 2. Weak facility, compute, platform, storage, or networking knowledge points to Chapters 3 through 5. Weak software compatibility, virtualization, or software-lifecycle knowledge points to Chapters 6 through 8. Weak monitoring, scheduling, orchestration, or MLOps knowledge points to Chapters 9 through 11. Use Chapter 12 after the technical journey to consolidate the current blueprint, exam strategy, and any remaining weak areas.

Step 3: Produce evidence of learning

For each study session, create one artifact: a diagram drawn from memory, a comparison table, a validated command output, a scenario explanation, a requirements brief, or a troubleshooting decision path. Reading can create familiarity. Producing an artifact exposes

whether the understanding is complete. Save each artifact so you can compare your progress at later checkpoints.

Step 4: Review at measurable checkpoints

At the end of each part, repeat the relevant baseline items and answer the chapter review questions without looking at the explanations. Mark a topic as ready only when you can explain what it is, why it matters, where it sits in the stack, what it depends on, and what evidence would show that it is working.

Step 5: Reserve time for scenario integration

The final stage should mix domains. Take a workload symptom or requirement and move from the workload to the relevant infrastructure or software layer, then back through monitoring and operational evidence. This prevents a common preparation problem: knowing every definition individually but not recognizing how the concepts interact in a scenario.

Maintaining a weak-area tracker

A tracker should describe the evidence of a gap and the next action. Avoid entries such as "networking weak" because they are too broad to resolve. *Table 1.7* shows a useful format:

Topic	Evidence of gap	Next action	Checkpoint
Training versus inference	Can define both but cannot compare infrastructure priorities	Review the Chapter 2 training-versus-inference comparison and explain one scenario aloud	After *Part 1*
Power and cooling	Cannot connect rack density to facility limits or interpret PUE	Complete the Chapter 3 facility exercises and redraw the energy flow	After Chapter 3
Monitoring tool boundaries	Confuses immediate inspection, metric collection, alerting, and visualization	Build the Chapter 9 tool-scope map from memory	After Chapter 9

Topic	Evidence of gap	Next action	Checkpoint
Job scheduling versus orchestration	Uses the terms interchangeably	Compare Slurm and Kubernetes responsibilities in three Chapter 10 scenarios	After Chapter 10

Table 1.7: Example weak-area tracker

Approaching certification scenarios

For each scenario, use the following five questions. Chapter 12 develops this approach further for exam preparation:

1. **Outcome:** What is the scenario trying to achieve or protect?
2. **Constraint:** Which requirement matters most, such as latency, throughput, capacity, availability, isolation, support, or cost?
3. **Layer**: Which stack component or supporting service is involved, such as the GPU hardware, data path, driver, monitoring tool, facility, or storage?
4. **Domain:** Is the question mainly about workload knowledge, infrastructure, or operations?
5. **Evidence:** Which observation or dependency would confirm the answer?

Now apply this routine to the questions below, identifying the relevant layer and evidence before choosing an answer.

Review questions

Note

The following questions are original practice items written for this chapter. They do not reproduce live NVIDIA exam questions. Use the explanations to review your reasoning, not only the correct option.

Question 1

Which statement best describes the purpose of the NCA-AIIO certification?

A. It validates advanced CUDA programming for AI model development.

B. It validates professional-level design of DGX SuperPOD environments.

C. It validates only NVIDIA GPU driver installation and maintenance.

D. It validates foundational AI infrastructure and operations knowledge.

Question 2

Which domain currently has the largest share of the NCA-AIIO blueprint?

A. Essential AI Knowledge, which accounts for 38%.

B. AI Infrastructure, which accounts for 40%.

C. AI Operations, which accounts for 22%.

D. All three domains, which are weighted equally.

Question 3

A proposed dense GPU system exceeds the available rack power and cooling capacity. Which concern should be addressed first?

A. AI framework and model optimization

B. Job scheduling and orchestration

C. Facility power and cooling capacity

D. GPU software and virtualization

Question 4

A training job shows low GPU utilization even though the GPUs report temperatures within expected limits and no hardware errors. What is the best next step?

A. Replace the GPUs before examining any other performance evidence.

B. Assume the workload cannot use GPU acceleration on this system.

C. Inspect workload state, job placement, storage, and network behavior.

D. Increase cooling capacity to raise the available GPU throughput.

Question 5

Which item is most clearly a version-specific fact that should be verified against current documentation?

A. GPUs benefit when workload data is delivered efficiently.

B. A supported stack requires compatible software layers.

C. Monitoring should separate observations from root causes.

D. The exact supported driver, toolkit, and framework versions.

Question 6

Which pair belongs to the monitoring and management layer in the course's six-layer NVIDIA technology stack?

A. CUDA and NCCL

B. `nvidia-smi` and DCGM

C. NVLink and GPUDirect

D. vGPU and MIG

Question 7

What is the most useful way to allocate study time from the exam weights?

A. Combine domain weights with baseline gaps and scenario practice.

B. Skip AI Operations because it has the lowest weight.

C. Study only AI Infrastructure because it has the highest weight.

D. Memorize domain percentages instead of reviewing the objectives.

Question 8

Which source should control the current exam objective wording if an older course slide or screenshot differs from it?

A. The current NVIDIA exam blueprint

B. The oldest available course recording

C. A third-party product comparison

D. A previous edition's chapter summary

Answers and explanations

1. **D:** NCA-AIIO is an entry-level associate credential that validates foundational AI computing concepts related to infrastructure and operations. It does not require advanced CUDA programming or professional-level architecture depth.
2. **B:** The current blueprint assigns 40% to AI Infrastructure, compared with 38% for Essential AI Knowledge and 22% for AI Operations.
3. **C:** Power and cooling are facility prerequisites for dense accelerated systems. Software, scheduling, or model changes cannot compensate for insufficient physical capacity.
4. **C:** Low utilization is an observation. The cause could involve data delivery, job placement, synchronization, or application behavior. Inspect the cross-layer evidence before changing hardware or facilities.

5. **D:** The need for compatibility is durable, but the exact supported version combination changes. Current official documentation should control the implementation decision.
6. **B:** In the course model, `nvidia-smi` and DCGM belong to monitoring and management. CUDA and NCCL are grouped under the course's core libraries layer; NVLink and GPUDirect are grouped with data movement; and vGPU and MIG belong with OS, driver, and virtualization.
7. **A:** Weights help prioritize, but baseline evidence identifies personal gaps and scenarios can cross domains. Ignoring a domain creates blind spots.
8. **A:** The current NVIDIA exam blueprint should control objective wording and exam scope. The course and this book support learning, but dated slides or screenshots should not override the current official certification source.

Summary

In this chapter, you established the map used throughout the book. NCA-AIIO currently divides its coverage among Essential AI Knowledge, AI Infrastructure, and AI Operations. Those domains describe different reasoning views of one environment rather than three disconnected systems. For exam objectives and logistics, the current NVIDIA certification page, blueprint, and study guide remain the source of truth.

You used the course's six-layer NVIDIA technology stack as the main technical map, then related facility, storage, networking, scheduling, orchestration, and lifecycle concerns to that map. You also identified the teams that commonly support these environments and saw why operational symptoms can cross ownership boundaries.

Finally, you distinguished durable concepts from version-specific facts, assessed your starting knowledge, classified practical tasks, and built a study plan. Chapter 2 introduces the AI vocabulary and training-versus-inference distinctions expected in NCA-AIIO.

2

Understanding AI Workloads, Training, and Inference

AI is often treated as a single workload label, but the systems behind that label can behave very differently. A retail recommendation service, a fraud-detection model, an autonomous-driving perception system, and a generative model can all be described as AI, yet the way they use data, compute, memory, storage, and network resources is not identical. For an infrastructure professional, vocabulary is therefore more than terminology. It is an early signal about what the platform may be asked to support.

Before looking at individual workloads, we need two foundations: why modern AI has scaled so quickly, and how AI, machine learning, deep learning, and generative AI relate. From there, this chapter moves into lifecycle vocabulary and the training-versus-inference comparison that appears directly in the certification blueprint.

The goal is not to turn you into a model developer. It is to help you recognize what a workload is doing, where it sits in the AI lifecycle, and how training and inference differ from an infrastructure perspective. Later chapters build on these workload signals when they introduce data center, platform, networking, software, and operations decisions.

In this chapter, we will cover the following topics:

- Tracing the major drivers behind modern AI adoption
- Differentiating AI, machine learning, deep learning, and generative AI
- Understanding data, models, parameters, and lifecycle vocabulary
- Comparing model training and inference in terms of objectives, compute, memory, and scalability
- Recognizing transformer attention and next-token prediction at the associate level

- Recognizing how workload scale changes resource pressure
- Recognizing AI use cases across industries
- Avoiding common terminology traps in exam scenarios

By the end of this chapter, you will be able to distinguish AI, machine learning, deep learning, and generative AI; compare training and inference in terms of objectives, compute, memory, and scalability; recognize transformer attention and next-token prediction at the associate level; and classify common industry scenarios by workload behavior rather than product names.

Technical requirements

This chapter is conceptual. You do not need a GPU, a cloud account, a machine-learning framework, or access to an AI cluster. To complete the activities, you need the following:

- The infrastructure fundamentals established in *Chapter 1*
- A notebook, text editor, or spreadsheet for the classification exercises
- Enough familiarity with servers, storage, and networking to recognize basic resource constraints
- A willingness to explain each term in operational language instead of memorizing a definition

Note

Exam focus: *Chapter 2* concentrates on Essential AI Knowledge objectives 1.2 to 1.5. The purpose is to give you enough workload vocabulary to compare training and inference, distinguish AI, machine learning, and deep learning, recognize the drivers behind AI adoption, and interpret common industry use cases. Detailed model development and advanced mathematics remain outside the scope of this book.

Tracing the major drivers behind modern AI adoption

Three reinforcing drivers help explain modern AI adoption: the growth of available data, greater computational power, and algorithmic breakthroughs. They work together rather than independently. For infrastructure teams, the useful takeaway is that AI growth increases demand across data, compute, and software, not only for accelerators. Here, an accelerator means specialized hardware such as a GPU that is designed to execute compute-intensive work efficiently.

1. Growth in available data

Human activity on websites and mobile apps, along with digital images, documents, and transaction records, creates substantial amounts of data. Sensors and connected devices add machine-generated data, further increasing the volume available for AI. ImageNet is one example of a large dataset associated with advances in computer vision. The infrastructure point is broader: learning systems depend on examples, and larger datasets increase the amount of information that must be collected, prepared, stored, moved, and repeatedly accessed.

More data does not remove the need for good data preparation. This preparation creates storage and I/O work before the main model-training calculations begin. It also explains why the path from storage to compute becomes important as dataset scale grows.

2. Growth in computational power

The second driver is the availability of much more computational power. GPUs, TPUs, and cloud computing can provide large amounts of parallel processing. That growth makes it practical to train models that would otherwise take too long or require too many general-purpose compute resources.

For this book, the important idea is not a particular processor brand or cloud service. Many modern AI workloads involve substantial parallel computation. When that work is mapped efficiently to accelerators, training and inference can scale beyond what is practical on small general-purpose systems. Later chapters examine the hardware and software that make this possible.

3. Algorithmic breakthroughs

The third driver is the evolution of algorithms and model architectures, including neural-network architectures, new training methods, reinforcement learning, transformers, and diffusion models. These developments changed the kinds of tasks machines could perform and how efficiently large amounts of data and compute could be used.

Transformers are especially relevant to modern generative AI. Attention mechanisms allow models to account for relationships in the input, while the architecture can make effective use of parallel computation. We will return to that process later in the chapter. *Table 2.1* brings these three drivers together and shows the infrastructure signal associated with each one.

Driver	What changed	Infrastructure signal
Explosion of data	More digital sources and much larger datasets are available for learning and analysis.	Data preparation, storage capacity, storage throughput, and movement of data toward compute become more important.
Computational power growth	GPUs, TPUs, and cloud resources make large amounts of parallel processing available.	Workloads can scale to more accelerators and nodes, which increases the importance of compute density and efficient data movement.
Algorithmic breakthroughs	Neural-network architectures, transformers, reinforcement-learning methods, and diffusion models expand what AI systems can do.	The workload pattern changes with the model and task, so infrastructure decisions must follow the workload rather than the AI label alone.

Table 2.1: Drivers of AI evolution and the infrastructure signals they create

Seeing the three drivers as one system

The three drivers reinforce each other. A model may need a large dataset, many calculations, and an architecture that can make productive use of both. If any one side becomes a bottleneck, the others can be underused. For example, even a large GPU cluster can be underused if storage cannot supply data fast enough. Likewise, a larger dataset does not guarantee better results if the model cannot learn from it effectively.

This systems view gives us the right starting point for the terminology that follows. AI, machine learning, deep learning, and generative AI describe different levels or capabilities, but the infrastructure impact comes from what the workload actually does with data and compute.

Differentiating AI, machine learning, deep learning, and generative AI

AI is the broadest field. Machine learning is an approach within AI, and deep learning is a machine-learning approach that uses neural networks with many layers. Generative AI describes systems that create new content from learned patterns and is commonly implemented using deep learning. For certification scenarios, classify the behavior first: is the

system following programmed logic, learning from data, using a deep neural network, or generating new content?

Artificial intelligence: the broad field

Artificial intelligence describes machines performing tasks that simulate aspects of human intelligence and decision-making. A system can be described as AI because it makes decisions or performs intelligent behavior, even when the mechanism is not the same as modern machine learning. A rule-driven chess system provides an intuitive example: the machine observes the chessboard state and chooses an action according to the logic it has been given.

For infrastructure teams, the label AI by itself is too broad to plan a platform. First identify the technique being used and whether the workload is preparing data, training, or performing inference.

Machine learning: learning patterns from data

Machine learning shifts the emphasis from explicitly programming every decision to learning from examples or past data. A fraud-detection system, for example, can learn from historical legitimate and fraudulent transactions instead of relying only on a handwritten rule for every possible fraud pattern.

That shift matters operationally because data becomes a core system dependency. Alongside code and compute, the platform needs data preparation, training data, model artifacts, and a repeatable process for updating learned behavior as new data becomes available.

Deep learning: using deep, multi-layer neural networks

Deep learning is a machine-learning approach that uses neural networks with many layers. It is called deep learning because those layers can learn increasingly complex representations from data. For NCA-AIIO, recognizing these characteristics is sufficient; you do not need to derive neural-network equations. Deep learning supports workloads such as computer vision, speech processing, and large-scale model training.

The infrastructure signal is the amount and pattern of computation. Neural-network training can involve repeated passes over data and large numbers of mathematical operations. That is one reason accelerated computing has become so closely associated with modern AI.

Generative AI: creating new content from learned patterns

Generative AI focuses on producing new output from learned patterns, including text, images, audio, and code. A generative model can therefore be asked to produce something that was not stored as a fixed response, while still drawing on patterns learned during training.

For classification, look at the requested behavior. If a scenario is primarily about creating new text, an image, audio, code, or another generated artifact from a prompt or learned pattern, generative AI is the most useful capability label. Most current generative AI systems are also machine-learning and deep-learning systems, so these labels can describe different aspects of the same implementation rather than mutually exclusive categories. The diagram that follows illustrates a common deep-learning-based generative approach, not an exhaustive taxonomy of every generative method.

Figure 2.1 summarizes the relationships among these terms for certification-level recognition. This is a simplified hierarchy for recognition, not a universal classification of every generative method. For another visual perspective, see the SAS Communities article "Where does GenAI fit within the AI landscape?" at [`https://communities.sas.com/t5/SAS-Communities-Library/Where-does-GenAI-fit-within-the-AI-landscape/ta-p/915000`].

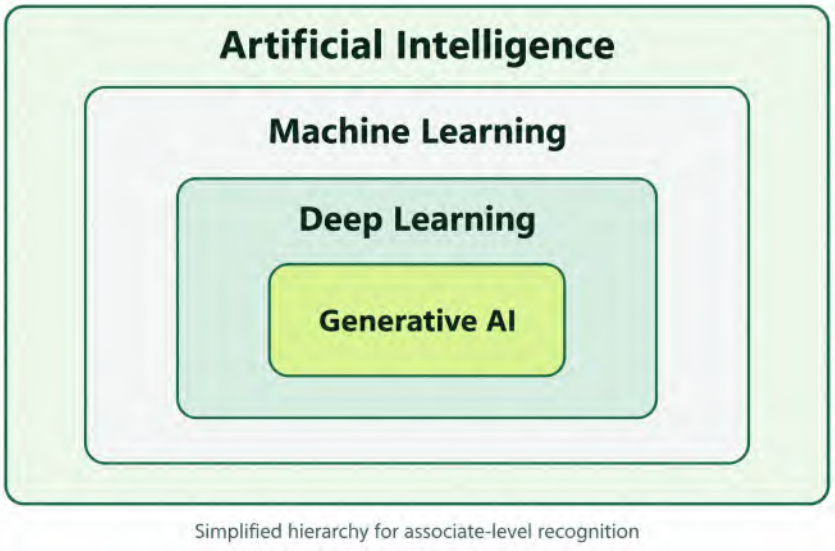


Figure 2.1: Relationship among AI, machine learning, deep learning, and generative AI

Classifying by behavior instead of by buzzword

Table 2.2 turns these distinctions into behavior-first questions you can use in exam scenarios.

Term	Distinguishing question	Example	Operational signal
Artificial intelligence	Is the machine performing an intelligent decision or task?	A machine plays chess by evaluating the board and choosing a move.	The AI label alone is broad; identify the actual processing pattern before making an infrastructure decision.
Machine learning	Is the system learning patterns from examples or historical data?	A fraud model learns from past valid and fraudulent transactions.	Training data, data preparation, model artifacts, and retraining become part of the platform.
Deep learning	Is learning based on deep, multi-layer neural networks?	Image-analysis and other deep neural-network workloads are typical examples.	Repeated parallel calculations make accelerated compute important.
Generative AI	Is the system creating new content from learned patterns?	Generating text, images, audio, or code from a prompt.	Model training can be compute-intensive, while production use introduces repeated inference requests.

Table 2.2: Classifying AI terminology by observable behavior

With the terminology separated by behavior, the next step is to follow what happens to data and the model over time. That lifecycle vocabulary connects the labels above to concrete workload phases such as data preparation, training, optimization, deployment, and inference.

Understanding data, models, parameters, and lifecycle vocabulary

A useful high-level AI lifecycle includes data preparation, model training, model optimization, deployment, and inference. Deployment makes a trained model available in a target

environment; inference processes new inputs once the model is available. *Chapter 8* maps NVIDIA software to these lifecycle stages, while *Chapter 11* revisits the operational practices around them. Here, we establish the vocabulary needed to compare training and inference.

Starting with data

Raw data is rarely ready for training exactly as it arrives. Data preparation can involve cleaning, transforming, and augmenting raw data to make it suitable for the model. Missing values, inconsistent formats, low-quality images, or too few examples can complicate data preparation.

From the infrastructure side, this means an AI platform may perform significant work before model training begins. Data can be read, transformed, written, and reread. Large datasets therefore create both capacity and throughput requirements, and those requirements can be visible even when GPU utilization is low.

Understanding the model and its parameters

A trained model combines a model architecture with learned parameter values that are used to produce a prediction or other output. Training adjusts those parameters through repeated learning iterations. In many cases, a larger parameter count increases a model's capacity to represent more complex patterns, while also increasing the amount of model state that must be stored and processed.

Parameters matter to infrastructure because the model must be stored, loaded, and processed. As model scale grows, memory and compute requirements can grow as well. The exact requirement depends on the model and workload, so parameter count is a signal rather than a complete sizing rule.

Comparing training and inference

Training: learning from repeated passes

During training, prepared data is processed repeatedly so the model can adjust its parameters. The distinction is straightforward: training is the learning phase, and larger training jobs often require substantial compute, memory, and communication resources. For neural networks, gradients indicate how model weights should change, while an optimizer applies those updates and may keep additional values called optimizer state. This state, along with intermediate activations created during training, can increase memory demand beyond the model weights alone.

Training can also scale across multiple GPUs or servers when one device is not enough. For this certification, you do not need the mathematics of gradients or optimizer internals; focus on the workload pattern and the infrastructure implications.

Optimization: preparing the trained model for use

After training, a model may be optimized for its target environment. The model used in production can therefore have different runtime characteristics from the training workload. Detailed optimization techniques belong later in the book.

The important point is that training infrastructure and inference infrastructure do not have to be identical. The objective changes from learning model parameters to serving predictions or generated output.

Inference: using the trained model on new input

Inference uses a trained model to process new input and return a prediction or generated result. Unlike training, it does not repeat the learning loop to update model parameters for every request.

A deployed model may later be retrained as new data becomes available, but inference and training remain distinct workload phases. *Table 2.3* compares their objectives, compute and memory needs, optimization goals, and scalability.

Feature	Model training	Model inference
Objective	Learn model parameters from prepared data through repeated iterations.	Use a trained model to produce a prediction or output for new input.
Compute requirements	Repeated, compute-intensive learning iterations; larger jobs may use multiple GPUs or nodes.	Compute demand varies with the model and serving pattern; online services often emphasize response time and throughput.
Memory requirements	Training may keep model weights, gradients, optimizer state, and intermediate activations in memory; requirements vary with the training setup.	Inference uses memory for model weights and runtime state; needs vary with model size, input length, and concurrent requests.
Optimization goals	Improve model quality while completing training efficiently.	Meet response-time, throughput, and efficiency goals for the target use case.

Feature	Model training	Model inference
Scalability	May scale across accelerators or nodes when the training workload requires it.	May scale by increasing concurrency or replica count; the pattern depends on the workload.

Table 2.3: Comparing model training and model inference

Table 2.3 is the core comparison to remember for objective 1.2. Treat the rows as common tendencies rather than fixed rules: demanding inference workloads can also require substantial compute and memory, and both training and inference can scale in different ways.

Recognizing transformers at the associate level

Transformers are widely used in modern generative AI and other sequence-processing tasks. For NCA-AIIO, you do not need to learn detailed encoder-decoder architecture. Two ideas matter here: attention lets surrounding context influence processing, and autoregressive generation repeatedly selects a plausible next token or word.

Following a next-word prediction example

A simple example begins with the incomplete sentence 'The quick brown fox jumps over the lazy ...'. The model considers the surrounding context and scores possible continuations. In the example, candidates such as person, rabbit, and dog are compared, and dog becomes the most plausible continuation.

This example uses whole words for simplicity. In practice, language models operate on tokens, which may be whole words or parts of words. The exam-level takeaway is the repeated prediction pattern, not tokenizer or embedding internals.

Understanding attention at a high level

Attention lets the model use relationships in the input when scoring a continuation. In the fox example, words such as quick, brown, fox, jumps, and lazy provide context that makes dog more plausible than the other candidates. The exact example scores are illustrative; the important point is that context influences the ranking.

This matters to infrastructure teams because transformer processing can involve substantial parallel computation. Later chapters cover the hardware and software that support this workload.

Repeating the prediction loop

After a candidate is selected, it is appended to the sequence and the model predicts again. Repeating that loop produces the output one step at a time.

Longer generated outputs therefore require repeated inference steps.

Figure 2.2 summarizes the example in five steps: input context, attention to relationships, candidate continuations, selection, and repetition.

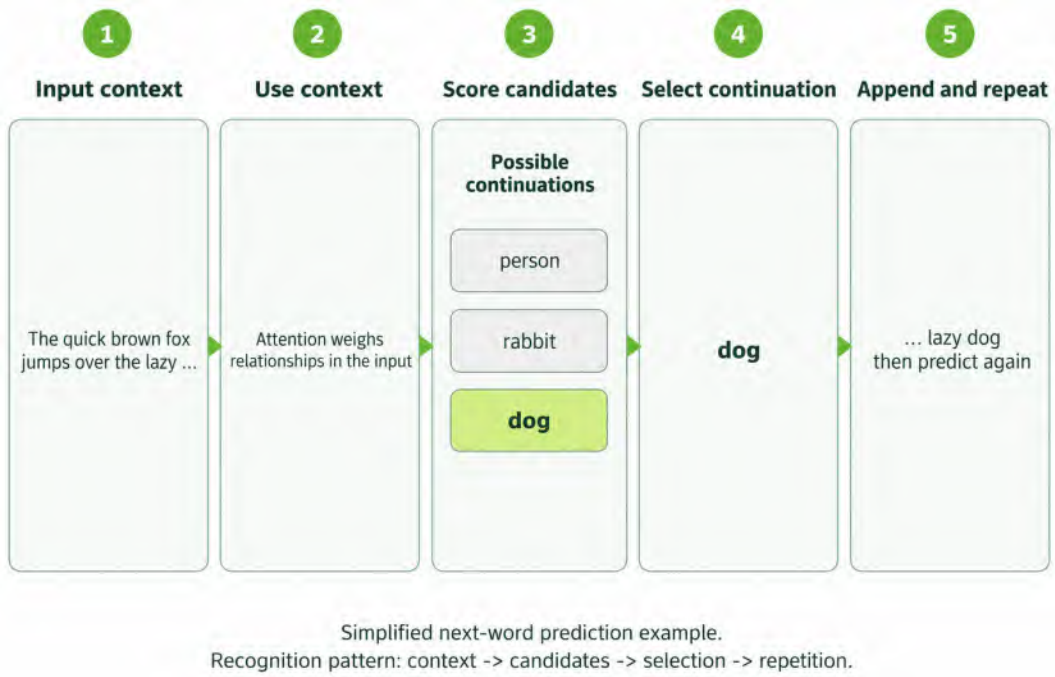


Figure 2.2: Simplified transformer inference flow

The prediction loop illustrates inference in practice. The next section shows how workload scale affects resource pressure.

Recognizing infrastructure signals from workload scale

Different changes to an AI workload place pressure on different infrastructure resources. A larger dataset must be prepared, stored, and delivered to the processors. A larger model may need more compute and memory, and training spread across multiple accelerators also

requires communication between them. Identifying what has grown helps you recognize which resources are most likely to become constraints.

When datasets grow

Larger datasets mean more data must be prepared, stored, and delivered to training. The detailed choice of storage technology is covered later; here, remember that dataset growth increases work on the storage and data-movement path.

If data cannot be delivered fast enough, accelerators can spend time waiting even when they are healthy. *Chapter 3* introduces the data center foundation, and *Chapter 5* examines the network and accelerated data paths that move data between systems and GPUs.

When models grow

Larger models can require more computation and more memory. If one accelerator or server is not enough, training may be distributed across multiple devices or nodes.

That introduces communication between GPUs and systems. *Chapter 4* covers NVIDIA accelerated platforms, and *Chapter 5* covers the interconnects and data paths that support multi-GPU and multi-node work.

When inference demand grows

Inference pressure depends on how results are consumed. Online services may prioritize consistent response time as request volume rises, while offline or batch inference may emphasize total throughput and efficiency.

Table 2.4 summarizes the resource signals to recognize as workloads scale.

Workload change	Likely effect	Recognition cue
Dataset becomes much larger	More data preparation, storage, and movement.	Check whether data preparation and delivery can keep pace with training.
Training uses a larger model or more parameters	More compute and memory demand.	Check available compute and memory rather than sizing from parameter count alone.
Training expands across GPUs or servers	More communication between accelerators or nodes.	Check how efficiently the accelerators can exchange data during training.

Workload change	Likely effect	Recognition cue
Inference must serve many requests	Higher serving throughput demand.	Check whether the service can handle the increased number of requests.
Inference is latency-sensitive	Faster response becomes the priority.	Check whether each request can meet its required response time.

Table 2.4: Recognizing workload-scale cues

Figure 2.3 introduces six representative AI use-case areas. The next section explains the workload behavior associated with each.

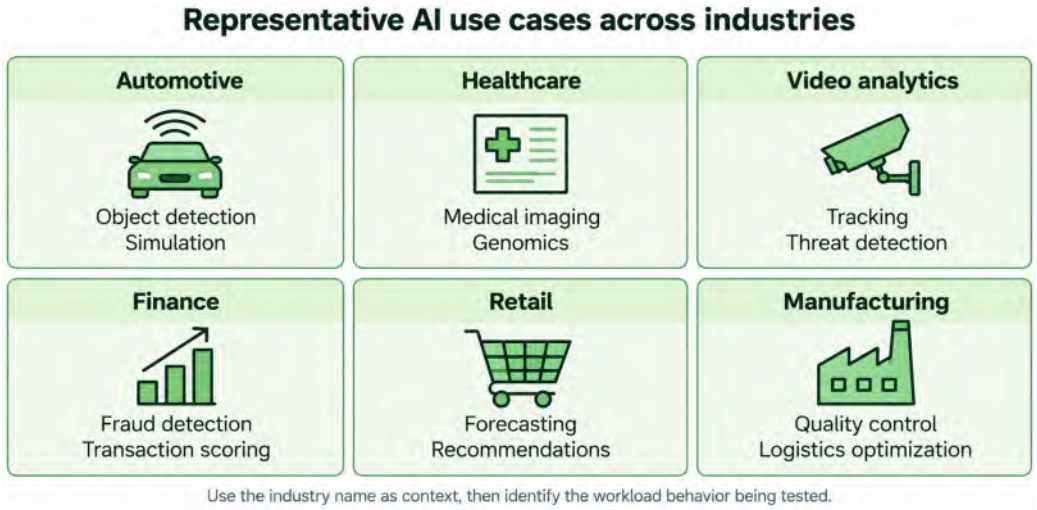


Figure 2.3: Representative AI use cases across industries

The figure is an orientation map rather than a sizing guide.

Recognizing AI use cases across industries

The following examples illustrate AI applications across six industries without prescribing a fixed infrastructure pattern for any sector.

Automotive and autonomous systems

Automotive applications include real-time object detection and classification, autonomous decision-making, and simulation for design and testing.

Healthcare and life-science workloads

Healthcare applications include automated medical-image analysis, genomics pipelines, and anomaly detection.

Video analytics

Video analytics use cases include real-time video-stream analysis, object detection and tracking, threat detection, and multi-camera processing.

Finance and banking

In finance, examples include real-time fraud detection and transaction scoring at scale.

Retail

Retail use cases include demand forecasting, supply-chain optimization, inventory management, and personalized recommendations.

Manufacturing

Manufacturing applications include automated quality control through image recognition, predictive simulation, and supply-chain or logistics optimization. *Table 2.5* summarizes these six domains and their distinguishing workload cues.

Industry	Representative use cases	What to recognize
Automotive	Object detection and classification, autonomous decision-making, simulation.	Perception, decision, and simulation use cases.
Healthcare	Medical-image analysis, genomics, anomaly detection.	Image or sequence analysis and anomaly detection.
Video analytics	Video-stream analysis, object detection and tracking, threat detection.	Streaming analysis and tracking.
Finance	Fraud detection, transaction scoring.	Prediction and inference on financial events.
Retail	Demand forecasting, supply-chain optimization, inventory management, recommendations.	Forecasting, optimization, and recommendation use cases.

Industry	Representative use cases	What to recognize
Manufacturing	Quality control, predictive simulation, logistics optimization.	Image analysis, simulation, and optimization.

Table 2.5: Industry examples and the workload signals behind them

The same technique can appear in several industries, and one industry can use several AI techniques. The industry label provides context, but the workload behavior determines which AI concept is being tested. The next section removes a few terminology traps that can distort that classification.

Avoiding common terminology traps in exam scenarios

Foundational questions are often difficult because familiar terms are used too broadly. A reliable approach is to separate the technique, lifecycle stage, workload objective, and infrastructure signal. The following traps are especially important in this book.

Trap 1: treating AI and machine learning as synonyms

AI is the broader field. Machine learning specifically introduces learning from examples or data. If the scenario says a system follows programmed decision logic, that can still fit the broad AI category. If it emphasizes learning patterns from historical data, machine learning is the stronger description.

Trap 2: treating deep learning as self-learning without neural networks

The defining signal here is the use of deep, multi-layer neural networks. Do not classify a system as deep learning merely because it improves over time or because it evaluates many possible actions. Look for an explicitly described deep neural-network method in the scenario.

Trap 3: treating all generative AI as a separate infrastructure category

Generative AI tells you that the system creates new content from learned patterns. It does not, by itself, tell you how large the model is, how many requests it serves, or how many accelerators it needs. Those requirements must be derived from the actual workload characteristics.

Trap 4: treating training and inference as the same workload

Training learns model parameters through repeated iterations, while inference uses the trained model on new input. If a question describes learning from training data or updating parameters, it points toward training. If it describes serving predictions, response time, or request throughput, it points toward inference.

Trap 5: assuming more data or compute automatically improves the result

The three drivers of AI evolution reinforce one another. More data is useful only when it can be processed effectively, and more compute helps only when the model and algorithm can use it. In scenario questions, avoid treating any single driver as a guarantee of better outcomes. *Table 2.6* summarizes the scenario clues and assumptions to avoid.

Scenario clue	Most useful interpretation	What not to assume
Learns from historical examples	Machine learning is involved.	Do not call every learned system generative AI.
Uses deep, multi-layer neural networks	Deep learning is involved.	Do not define deep learning only as a system learning by itself.
Creates new text, images, audio, or code	Generative AI is the strongest capability label.	Do not infer exact model size or hardware requirements from the label alone.
Repeated learning iterations or parameter updates	The scenario describes model training.	Do not treat it as an inference-serving scenario.
New production input, predictions, response latency, request throughput	The scenario describes inference.	Do not assume the training infrastructure pattern must be identical.

Table 2.6: Common terminology clues for scenario questions

With those clues separated, the next step is to apply them consistently. The five-question routine below turns the terminology into a repeatable scenario-reading method.

Using a five-question classification routine

When you read a foundational AI scenario, ask these five questions before choosing an answer:

1. **Technique**: Is this broad AI behavior, learning from data, neural-network learning, or content generation?
2. **Lifecycle stage**: Is the system preparing data, training, optimizing, or performing inference?
3. **Objective**: Is the priority learning quality, compute throughput, response latency, serving throughput, or another stated goal?
4. **Data pattern**: Is the workload processing a large static dataset, repeated production requests, or a continuous stream?
5. **Infrastructure signal**: Which resource is most likely to matter next: compute, memory, storage, network, or a combination?

The fifth question does not ask you to design the platform. It only connects the AI vocabulary to the infrastructure map from *Chapter 1*. Use the activities below to practice the classification before later chapters move into specific infrastructure technologies.

Practical activities

Activity 1: classifying business scenarios

For each scenario, choose the most useful primary label: AI, machine learning, deep learning, or generative AI. More than one label may technically apply, so choose the label that best matches the behavior being tested.

1. A rules-based system evaluates the current chessboard and selects a legal move from programmed logic.
2. A fraud-detection system learns patterns from historical legitimate and fraudulent transactions.
3. A medical-imaging system uses a deep, multi-layer neural network to classify images.
4. A service creates new product descriptions from a prompt and learned language patterns.
5. A recommendation system learns from previous user activity to rank products.
6. A model generates an image from a text prompt.

Suggested classification

One reasonable classification is shown in *Table 2.7*. The important part is the reasoning, not the label in isolation.

Scenario	Primary label	Reason
Rules-based chess decision	AI	The system follows programmed decision logic; the scenario does not describe learning from examples.
Fraud detection from historical transactions	Machine learning	The system learns patterns from past data.
Medical imaging with a deep neural network	Deep learning	The scenario explicitly identifies deep, multi-layer neural-network processing.
Generated product description	Generative AI	The system creates new text from learned patterns.
Recommendation from user history	Machine learning	The system learns from historical behavior to make a prediction or ranking.
Image generated from text	Generative AI	The requested behavior is creation of new image content.

Table 2.7: Suggested classification for the business scenarios

Once you can classify the technique from its behavior, the next activity shifts from labels to the processing flow inside a generative request.

Activity 2: explaining the transformer example in five steps

Without looking back at *Figure 2.2*, explain the simplified example using these five ideas: input context, attention to relationships, candidate continuations, selection, and repetition. Keep the explanation at the recognition level rather than describing transformer internals.

With the transformer example clear, the next activity returns to the chapter's central training-versus-inference comparison.

Activity 3: comparing training and inference

Create a two-column comparison for training and inference using the five rows from *Table 2.3*: objective, compute requirements, memory requirements, optimization goals, and scalability. Then explain one reason the same organization might use different infrastructure patterns for the two phases.

Finish by classifying one interactive inference example and one distributed training example. State which operating priority matters most in each case and which later chapter you would consult for the underlying infrastructure.

Review questions

The following questions are original practice items written for this chapter. They do not reproduce live NVIDIA exam questions. Use the explanations to check your reasoning as well as the selected option.

Question 1

Which combination matches the three drivers of AI evolution described in this chapter?

A. More data, greater compute, and algorithmic breakthroughs

B. Smaller datasets, lower compute, and manual rule creation

C. Virtualization, containers, and software licensing

D. Cooling, rack space, and power distribution

Question 2

Which statement best distinguishes machine learning from the broader AI category in this chapter?

A. It always creates new content from prompts.

B. It learns patterns from data rather than relying only on fixed rules.

C. It is limited to rule-based decision systems.

D. It is another name for accelerated computing.

Question 3

Which scenario most directly signals deep learning?

A. A rules engine uses a fixed decision table.

B. A service fills a document template.

C. An image system uses a deep neural network.

D. A storage system archives model checkpoints.

Question 4

What is the primary distinguishing behavior of generative AI?

A. It creates new content from learned patterns.

B. It always runs on one accelerator.

C. It removes the need for model training.

D. It is limited to image classification.

Question 5

Which statement most clearly describes model training?

A. It uses learned parameters to answer new production requests.

B. It repeatedly processes training data to adjust model parameters.

C. It only runs after a model has been deployed.

D. It focuses only on minimizing response time for each request.

Question 6

In the simplified transformer explanation, what does attention help the model do?

A. Replace storage with accelerator memory.

B. Use relationships in the input as context.

C. Eliminate the need for training data.

D. Guarantee that every output is correct.

Question 7

A training dataset becomes much larger while the model and training method remain the same. Which infrastructure signal should you expect first from the material in this chapter?

A. More data preparation, storage, and movement

B. Less need for any network connectivity

C. Automatically lower inference latency

D. No need to store model parameters

Question 8

Why should an infrastructure professional distinguish training from inference before selecting a platform design?

A. They use identical loops and priorities.

B. They have different objectives and resource patterns.

C. Only inference workloads can use GPUs.

D. Only training workloads process data.

After answering all eight questions, use the explanations below to check the reasoning behind each choice, not only the letter you selected.

Answers and explanations

1. **A:** The three drivers are the explosion of data, growth in computational power, and algorithmic breakthroughs.
2. **B:** The machine-learning distinction is learning from data rather than explicitly programming every possible decision. AI is the broader category.
3. **C:** Deep learning is identified here by the use of a deep, multi-layer neural network. The other choices do not identify that learning method.
4. **A:** Generative AI is described as creating new content such as text, images, audio, or code based on learned data and patterns.
5. **B:** Training repeatedly processes training data so the model can adjust its parameters. Inference uses the trained parameters to process new input without repeating that learning loop for every request.
6. **B:** Attention lets the model use relationships in the input as context when scoring plausible continuations.
7. **A:** A larger dataset increases the amount of data that must be prepared, stored, and delivered to the training workload. Later chapters determine how the storage and network should be designed.
8. **B:** Training and inference have different objectives and resource patterns. Training learns parameters through repeated iterations, while inference uses the trained model to process new inputs.

Taken together, the answers reinforce the chapter's central habit: classify the workload behavior first, then connect it to the infrastructure signals that matter.

Summary

In this chapter, you learned the workload vocabulary covered in the NCA-AIIO Essential AI Knowledge domain. You saw how data growth, computational power, and algorithmic breakthroughs reinforced one another, and you differentiated AI, machine learning, deep learning, and generative AI.

You then followed the lifecycle vocabulary from data preparation through training, optimization, deployment, and inference. Most importantly, you compared training and inference across their objective, compute and memory behavior, operating goals, and scaling patterns.

Finally, you used a simplified transformer example and representative industry use cases to practice scenario recognition without drifting into model-development detail. In *Chapter 3*, you will move from workload recognition to the physical foundations of AI data centers, including facility requirements, compute roles, storage, and DPUs.

3

Building the AI Data Center Foundation

Chapter 2 established the AI workload concepts that infrastructure teams need to recognize, including the difference between training and inference. This chapter now moves from those workloads into the physical and computational foundation that supports them: the data center facility, processors, and storage systems.

The building blocks of a data center are familiar: compute, networking, storage, and support infrastructure. AI workloads can increase the pressure on each of them, especially where dense accelerator systems concentrate power demand, heat, and data movement into a relatively small physical footprint. An accelerator is a specialized processor, such as a **graphics processing unit** (**GPU**), that speeds up a particular kind of computation. The important skill is to recognize how those blocks work together without turning the chapter into a generic data-center design framework.

We will begin with the four fundamental blocks of an AI-centric data center and the power, cooling, and physical-space constraints that surround them. We will then compare the roles of **central processing units** (**CPUs**), GPUs, and **data processing units** (**DPUs**), distinguish the roles of CUDA Cores, Tensor Cores, and RT Cores at the depth required here, and finish with the storage categories introduced in the course. *Chapter 4* then builds on this foundation with integrated NVIDIA accelerated computing platforms.

In this chapter, we will cover the following topics:

- Recognizing the four blocks of an AI-centric data center
- Identifying rack, power, cooling, and physical-space constraints
- Calculating and interpreting power usage effectiveness

- Comparing CPU and GPU architecture for AI workloads
- Comparing CPU, GPU, and DPU responsibilities
- Recognizing CUDA Cores, Tensor Cores, and RT Cores
- Recognizing storage roles for AI workloads
- Recognizing how compute density affects facility limits

By the end of this chapter, you will be able to assess the main facility and compute constraints in an AI-centric data center, distinguish the roles of CPUs, GPUs, and DPUs, interpret PUE, recognize key GPU execution resources, and match common storage approaches to AI workload needs.

Note

Download the PDF version of this book

Your purchase includes a DRM-free PDF copy of this book and a range of exclusive benefits. To unlock everything, follow the *Free benefits with your book* section in the *Preface*.

Technical requirements

This chapter focuses on architecture and does not require access to a GPU server or data center. To complete the activities, you need the following:

- The AI workload concepts established in *Chapter 2*
- Basic familiarity with servers, racks, storage, and networking
- A standard calculator, either software-based or physical, for the **power usage effectiveness** (**PUE**) exercises
- A notebook, text editor, or spreadsheet for the role-mapping and rack-review activities
- A willingness to reason from workload and infrastructure roles instead of memorizing product specifications

Note

Exam focus: *Chapter 3* concentrates on the facility, processor, DPU, and storage concepts behind objectives 1.8, 2.1, 2.3, 2.6, and 2.10, with supporting context for accelerated-cluster components. Focus on recognizing roles, constraints, and relationships at the depth taught in the course rather than building a generic facility-design method.

Inside an AI-centric data center

An AI-centric data center is not a completely different type of data center. It still depends on compute, networking, storage, and support infrastructure. The difference is that AI workloads can intensify each dependency at the same time. A distributed training job may need sustained accelerated compute, repeated access to a large dataset, high-speed communication between nodes, and enough power and cooling capacity for the entire cluster for long periods.

The course presents an AI-centric data center through four fundamental blocks: compute, network, storage, and support infrastructure. Support infrastructure provides the physical conditions that allow the other three blocks to operate, while compute, network, and storage work together to keep AI workloads supplied with processing power and data. We will use those four blocks as the chapter map.

Figure 3.1 shows the four blocks used throughout this chapter. Compute, network, and storage support the workload directly, while support infrastructure provides the power, cooling, security, and physical conditions that allow the environment to operate:

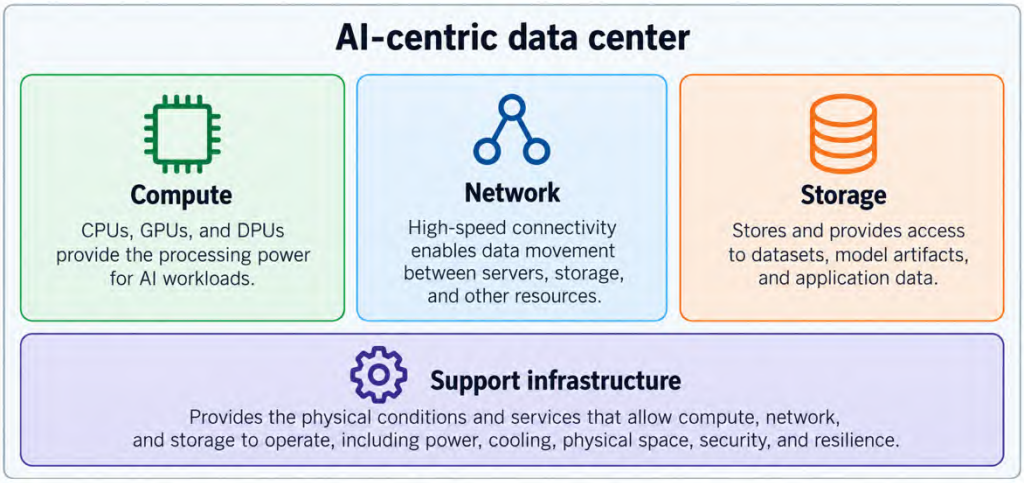


Figure 3.1: The four fundamental blocks of an AI-centric data center

These blocks are interdependent, but support infrastructure sets the physical operating envelope for the others. We will start there.

Facility and support infrastructure

The facility provides the conditions under which the other three blocks can operate. Rack space, electrical capacity, power distribution, cooling, fire protection, physical security, and resilience all belong here. For dense GPU systems, the most visible constraints are often power,

heat removal, and available physical space, but serviceability and recovery paths matter as well.

Even if a server fits in a rack and has compatible power connections, it may not be practical to deploy if the rack cannot supply sufficient power continuously, the room cannot remove the heat, or there is too little space for cabling and maintenance. Facility checks therefore belong at the beginning of design, not after hardware has been selected.

Compute

Compute includes CPUs, GPUs, DPUs, system memory, accelerator memory, and the server architecture that connects them. CPUs remain essential for operating-system work, control flow, general-purpose processing, and coordination. GPUs provide high-throughput parallel acceleration for suitable workloads. DPUs can offload selected infrastructure functions so that CPUs and GPUs spend less time on networking, storage, or security processing.

Networking

Networking connects compute nodes to each other, to storage, to management systems, and to users or external services. Different traffic types can have different latency, throughput, reliability, and isolation needs. *Chapter 5* develops those fabrics and data-movement technologies in depth. At this stage, the important point is that the network is part of the compute path whenever workload progress depends on data arriving from another device or node.

Storage

AI workloads consume and create raw datasets, prepared datasets, checkpoints, model artifacts, container images, logs, and inference results. Storage therefore has to be evaluated for more than capacity. Throughput, latency, concurrency, data locality, security, and retention can determine whether accelerators stay productive or spend time waiting for data.

Table 3.1 summarizes the four course-defined blocks and the first recognition question associated with each one:

Block	Primary role	First recognition question
Support infrastructure	Provide power, cooling, physical space, and site services	Can the site support dense IT equipment within its power, cooling, and space limits?
Compute	Execute general-purpose and accelerated processing	Which processor role best matches the work: CPU, GPU, or DPU?

Block	Primary role	First recognition question
Network	Connect compute, storage, management, and external services	What data must move, and what performance or isolation does it require?
Storage	Retain and deliver datasets, checkpoints, artifacts, and results	Can the required data be delivered with the needed capacity and performance?

Table 3.1: First questions across an AI-centric data center

We now examine support infrastructure in more detail. Dense accelerator deployments can encounter power, cooling, and physical-space limits before the available compute is fully useful.

Identifying rack, power, cooling, and physical-space constraints

Power, cooling, and physical space are tightly coupled in a dense deployment. Adding more compute equipment to a rack increases the electrical load and usually increases the amount of heat that must be removed from a smaller area. The result is that rack density is not simply a question of whether equipment can physically fit into the available rack units.

Checking rack and physical-space constraints

A rack review begins with available rack units, equipment depth, weight, cable routing, service clearance, and the location of power and network connections. A dense server can consume relatively few rack units while still imposing large power and cooling demands, so physical fit is only the first check.

Floor-level planning matters when multiple dense racks are placed together. The room must support not only the equipment footprint but also distribution equipment, cooling infrastructure, cable paths, maintenance access, and any local safety requirements. A layout that maximizes equipment density but prevents safe service or blocks airflow is not an efficient design.

Checking electrical capacity

Electrical planning asks how much power the rack and room can deliver continuously, how that power is distributed, and what margin is required for safe operation and growth. Work from the equipment estimate toward the facility limit rather than assuming that a free rack has enough electrical capacity.

Power and energy are related but not interchangeable. Power describes the instantaneous rate at which equipment consumes electricity, commonly expressed in watts or kilowatts. Energy describes consumption over time, such as kilowatt-hours. Rack design is usually constrained by available power, while operating-cost and efficiency discussions often consider energy over a period.

For early planning, sum the expected power demand of compute systems, networking, storage, and other rack-level equipment. For a quick estimate, *rack IT load (kW) = server power + network power + storage power + other rack equipment power* (all in kW). Then compare that total with the deliverable rack capacity and any operating headroom required by the organization. Detailed electrical engineering belongs to qualified facility teams, but infrastructure architects still need to recognize when the proposed load exceeds the available envelope.

Checking cooling and thermal limits

Virtually all electrical power consumed by IT equipment eventually appears as heat that the cooling system must remove. Dense accelerator systems can therefore change the thermal profile of a rack even when the room previously supported conventional servers without difficulty.

Air cooling removes heat by moving cooler supply air through equipment and carrying away warmer exhaust air. Liquid cooling uses circulating coolant to absorb heat near heat-generating components and carry it away. At this stage, you only need to recognize these as two approaches to removing heat; detailed cooling-system design is outside this chapter's scope.

Note

The correct question is not simply whether the room is air-cooled or liquid-cooled. It is whether the deployed cooling method can remove the expected heat at the required density and environmental conditions. Airflow, supply temperature, return temperature, containment, liquid-cooling capacity where applicable, and monitoring all contribute to that answer. Exact limits depend on the facility and equipment being used.

Keeping resilience in scope

Support infrastructure should also include resilience, but the course does not develop a detailed facility-resilience design method. At this level, recognize that a loss of power or cooling can remove available AI capacity and that resilient facilities may use redundant power or cooling paths according to workload and site requirements.

Power and cooling tell us whether the site can support the proposed IT load. Power usage effectiveness adds a second question: how much total facility energy is consumed relative to the energy used by IT equipment?

Calculating and interpreting power usage effectiveness (PUE)

Power usage effectiveness (**PUE**) is a data-center efficiency metric. It compares total facility energy with the energy used by IT equipment such as servers, storage, and networking. It does not measure how useful the AI workload is, how efficient a model is, or how much work a GPU completes per watt. It measures the facility overhead around the IT load.

The basic formula is:

PUE = Total facility energy / IT equipment energy

The same relationship can be calculated using power measurements when the numerator and denominator are measured consistently over the same operating condition. A theoretical PUE of 1.0 would mean that all facility energy goes to IT equipment and none is consumed by cooling, power conversion, lighting, or other support systems. Real facilities have overhead, so practical PUE values are greater than 1.0.

Figure 3.2 shows the energy relationship. Total facility energy includes the energy consumed by IT equipment plus the overhead required to deliver and support that IT load:

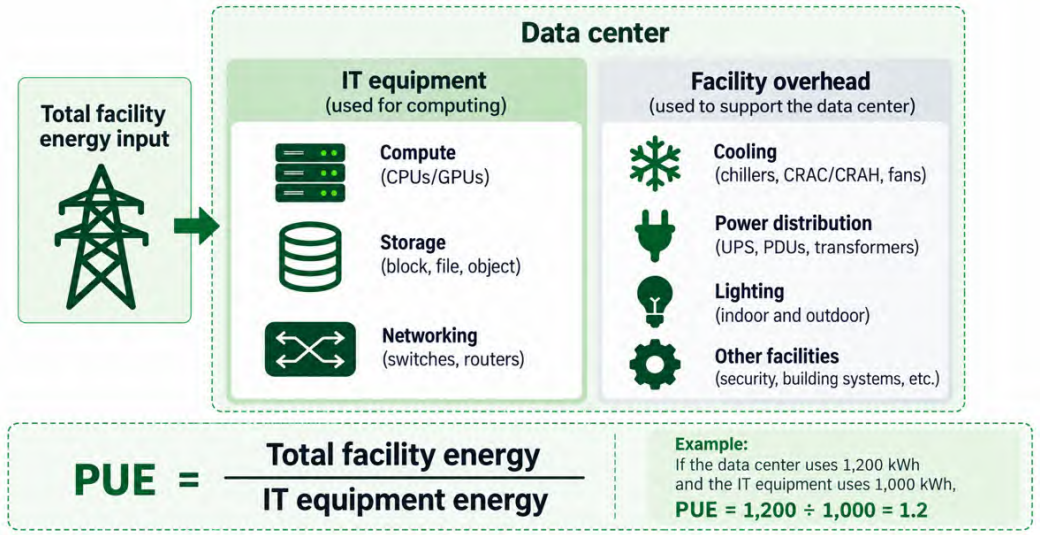


Figure 3.2: Facility energy breakdown and PUE calculation

The diagram separates energy consumed by IT equipment from the facility overhead needed to support it. The next example applies that relationship to a simple calculation.

Working through a PUE calculation

Suppose a facility consumes 1.2 MWh of total energy during a measurement period and the IT equipment consumes 1.0 MWh during the same period. The calculation is 1.2 / 1.0 = 1.2, so the PUE is 1.2.

Interpret that result carefully. A PUE of 1.2 means the facility consumed 20% more energy than the IT equipment. It does not mean exactly 20% of total facility energy was overhead. In this example, the overhead is 0.2 MWh out of 1.2 MWh total, or about 16.7% of total facility energy.

The same method applies to other scenarios. *Table 3.2* provides three examples with interpretations of the results:

Scenario	Total facility energy	IT equipment energy	PUE	Interpretation
A	1.2 MWh	1.0 MWh	1.20	Low facility overhead relative to the IT load
B	1.5 MWh	1.0 MWh	1.50	More energy is being consumed outside the IT equipment

Scenario	Total facility energy	IT equipment energy	PUE	Interpretation
C	2.0 MWh	1.0 MWh	2.00	Total facility energy is twice the IT equipment energy

Table 3.2: Example PUE calculations

The arithmetic is straightforward; the more important skill is interpreting what the resulting PUE value does and does not tell you.

Using PUE without overinterpreting it

Lower PUE generally indicates less facility overhead for a given amount of energy consumed by IT equipment, but PUE should not be used as the only measure of data-center effectiveness. Its value also depends on operating conditions. It also cannot tell you whether the IT equipment itself is doing useful work efficiently.

The course uses a PUE of 1.2 as an example of low facility overhead. Treat that value as an illustration rather than a universal optimum because PUE varies with climate, facility design, utilization, and measurement boundary. For an infrastructure scenario, use PUE for the question it actually answers: how much facility energy is required relative to energy consumed by IT equipment?

Power, cooling, and space define the environmental envelope. The next step is compute: the processing work that turns data and instructions into results. In an AI system, that work is shared across processor types with different strengths. We will first compare CPUs and GPUs, then extend the model to DPUs.

Comparing CPU and GPU architecture for AI workloads

CPU and GPU comparisons are often reduced to a slogan: CPUs are serial and GPUs are parallel. That is useful as an initial intuition, but it is technically incomplete. Modern CPUs have multiple cores and can execute work in parallel, while GPUs can still perform poorly when a workload has little parallelism, irregular control flow, or frequent dependencies between operations.

A better distinction is that CPUs are optimized for strong per-thread performance, complex control, low-latency decision making, and a wide range of general-purpose work. GPUs dedicate much more of their silicon to executing large numbers of similar operations

concurrently. That makes them effective when a problem can be divided into many independent or lightly dependent pieces of work.

The course uses a private jet and a commercial airliner as a memory aid. A private jet emphasizes flexibility and responsiveness for a smaller number of passengers, much as a CPU is optimized for strong general-purpose and control-heavy work. A commercial airliner is designed to move many passengers together efficiently, which mirrors the GPU emphasis on high-throughput parallel work. The analogy is not literal, but it makes the architectural trade-off easier to remember.

Recognizing a control-heavy pattern

Consider a task that repeatedly makes decisions based on the result of the previous step, follows many branches, and handles operating-system or application control logic. Such work benefits from the flexible execution resources and cache hierarchy of a CPU. Adding thousands of GPU execution lanes does not automatically help if the work cannot be exposed as parallel operations.

Recognizing a data-parallel pattern

Now consider work that applies the same calculation across many values, such as every pixel in an image or every element in a matrix. Many of those operations can be performed concurrently. This is where GPUs provide their main advantage: they can schedule large amounts of parallel work and sustain high arithmetic throughput when software and data access patterns keep the execution resources busy.

Training and inference frequently contain large matrix and tensor operations, where a tensor is a multidimensional array used to represent model data. That is one reason GPUs are central to modern AI. The GPU still depends on the CPU for system coordination and on memory and data paths for inputs, weights, and intermediate results. Parallel compute does not remove those dependencies; it makes stalls in those dependencies more visible because execution resources can sit idle while data or synchronization is delayed.

Table 3.3 compares CPU- and GPU-oriented strengths without suggesting that either processor supports only one execution pattern:

Attribute	CPU-oriented strength	GPU-oriented strength
Execution style	Complex control flow and strong per-thread performance	Large numbers of similar operations executed concurrently

Attribute	CPU-oriented strength	GPU-oriented strength
Typical role	Operating system, application control, preprocessing, coordination	Matrix, tensor, vector, image, simulation, and AI acceleration
Parallelism	Fewer powerful cores with substantial control and caching resources	Many execution resources organized for high-throughput parallel work
Main caution	May be inefficient for very large data-parallel workloads	Acceleration depends on sufficient parallel work and effective data movement

Table 3.3: Comparing CPU-oriented and GPU-oriented compute patterns

The practical conclusion is not CPU versus GPU. An AI server needs both, and increasingly it may also use a DPU. The design question is which processor should carry which class of work.

Comparing CPU, GPU, and DPU responsibilities

The course extends its air-travel analogy here: moving passengers depends not only on the aircraft but also on specialist operational services around the journey. Likewise, an AI system relies on the CPU to run the operating system and coordinate tasks, the GPU for parallel computation, and networking, storage, and security services to keep the workload running. A DPU provides a place to offload selected infrastructure work so that host processors can focus more of their resources on their primary roles.

DPU responsibilities

A data processing unit, or DPU, is designed to offload, accelerate, and isolate selected infrastructure services. Depending on the platform and software, those services can include networking, storage processing, security functions, telemetry, and virtualized infrastructure tasks. The goal is to reduce the infrastructure work that would otherwise consume host CPU cycles and to create clearer control and isolation boundaries.

A DPU does not replace the GPU for model training or inference, and it does not replace the CPU as the server's general-purpose processor. Think of it as an infrastructure processor that handles data-centric services around the workload so the CPU and GPU can focus on their primary roles.

Typical infrastructure work that a DPU can offload includes the following:

- **Process network traffic**, such as moving packets, routing traffic between virtual machines, and handling high-speed network protocols.
- **Encrypt and decrypt data** as it moves in and out of the server, and enforce firewall and security rules.
- **Manage access to remote storage**, including supported configurations in which remote block storage is presented to the host as a local drive.

Figure 3.3 maps the three processor categories by responsibility. The DPU examples illustrate possible capabilities; not every configuration supports every listed function:

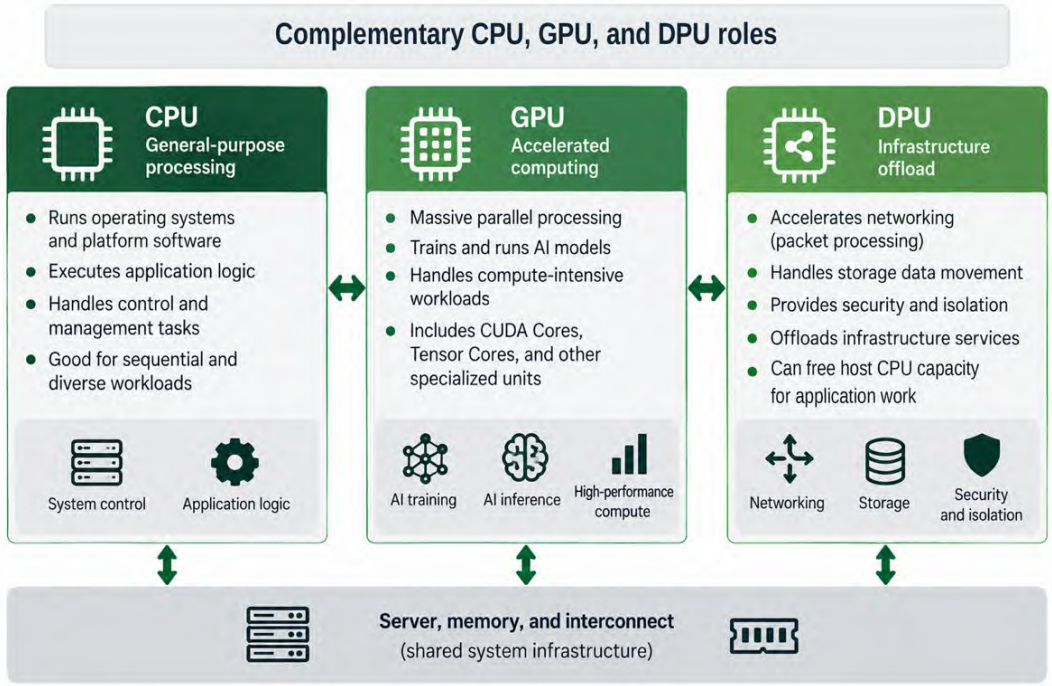


Figure 3.3: CPU, GPU, and DPU responsibility map

Table 3.4 provides a scenario-oriented comparison. Use each processor's primary role to guide your choice, but remember that a workload can involve all three:

Type	Primary responsibility	Example work in an AI system	Common design question
CPU	General-purpose control and host processing	Operating system, application control, preprocessing, job coordination	Is CPU processing limiting the ability to keep the accelerators busy?
GPU	High-throughput parallel acceleration	Training, inference, matrix and tensor operations	Does the workload have enough GPU compute and memory capacity?
DPU	Infrastructure offload and isolation	Networking, storage, security, and telemetry offload where supported	Can infrastructure processing be offloaded from the CPU and isolated from the host?

Table 3.4: CPU, GPU, and DPU responsibilities in an AI system

This chapter covers only the general DPU role. *Chapter 4* introduces NVIDIA BlueField after the discussion moves into NVIDIA accelerated platforms.

With the processor roles separated, the next step is to look one level deeper inside the GPU. The course introduces CUDA Cores, Tensor Cores, and RT Cores to distinguish different GPU execution roles. For NCA-AIIO, keep this at recognition depth rather than expanding into deeper GPU microarchitecture.

Recognizing CUDA Cores, Tensor Cores, and RT Cores

GPUs provide parallel processing resources and, depending on the architecture, may include specialized cores for particular kinds of computation.

The course uses a school analogy as a memory aid. CUDA Cores are like versatile regular teachers, Tensor Cores are like mathematics specialists for matrix-oriented AI work, and RT Cores are like specialists for rendering and ray-tracing tasks. The mapping is intentionally simplified; it helps distinguish roles without implying that GPU software literally assigns work the way a school principal assigns classes.

At the level required here, focus on the role of each resource rather than a generation-specific block diagram or core-count table.

Table 3.5 keeps the GPU-core discussion at recognition depth and includes RT Cores as a brief contrast:

Execution resource	Course-level role	Certification takeaway
CUDA Cores	General parallel arithmetic execution resources	Recognize them as general parallel execution resources used across suitable GPU workloads.
Tensor Cores	Specialized matrix and tensor execution resources	Recognize their role in accelerating many AI training and inference operations on supported GPUs.
RT Cores	Ray-tracing and graphics-focused acceleration	Recognize them as specialized ray-tracing resources; they are supporting context for this certification.

Table 3.5: CUDA Core, Tensor Core, and RT Core roles at a glance

Raw CUDA Core or Tensor Core counts should not be used as a universal performance comparison across different GPU architectures or generations. Architecture, memory performance, supported data types and operations, power limits, and software optimization can all influence delivered performance. GPU capabilities should therefore be evaluated in the context of the target workload.

GPU compute must also be supplied with data efficiently, which brings us to storage.

Recognizing storage roles for AI workloads

Storage supplies datasets, checkpoints, model artifacts, logs, and other data used across AI workloads. The course emphasizes high-throughput, low-latency, scalable access and introduces four useful storage categories: local **Non-Volatile Memory Express (NVMe)**, parallel or high-performance file systems, network file services, and object storage.

Separating capacity, throughput, and latency

Capacity describes how much data can be retained, throughput describes how much data can be moved over time, and latency describes how long an access takes. Shared access and data

locality also matter. The workload stage and data-access pattern determine which characteristics deserve attention.

A large-capacity system is not automatically the right active-data path if it cannot deliver data at the required rate. Likewise, fast local storage may not satisfy a workload that needs many nodes to access the same dataset.

Using local NVMe for active data

Local NVMe can provide high-bandwidth, low-latency access close to the server and is commonly used for staging, caches, and active working data. These are common uses, but NVMe need not be local or limited to these roles.

Using shared and parallel storage

Parallel or high-performance shared file systems are useful when multiple workers need concurrent access to large active datasets. Their value comes from shared access and aggregate throughput; suitability still depends on the workload and the network path between storage and compute.

Using network file services for shared operational data

Network file services provide shared file access and may support scripts, user data, operational files, and other workloads where their performance is sufficient. This does not mean network file services are limited to small files or unsuitable for all demanding workloads.

Using object storage for large-scale retention

Object storage provides durable, scalable storage capacity and is commonly used for datasets, artifacts, checkpoints, logs, and retention. Active data can be staged to faster shared or local tiers when the workload benefits from doing so.

Using tiering and locality selectively

A common storage-tiering pattern is to keep frequently accessed hot data on faster storage and move warmer or colder data to capacity-oriented tiers when appropriate. Treat locality and tiering as workload-dependent patterns, not as one mandatory architecture.

The four storage categories are complementary roles rather than one mandatory pipeline. *Table 3.6* summarizes their typical strengths and uses:

Storage role	Typical strength	Typical AI use
Local NVMe	Low-latency, high-bandwidth local access	Staging, caches, active working data, and temporary local datasets
Parallel/shared high-performance storage	Concurrent shared access and aggregate throughput	Large active datasets accessed by multiple training nodes
Network file service	Shared file access with familiar semantics	Shared scripts, user data, operational files, and workloads where its performance is sufficient
Object storage	Durable, scalable object capacity and lifecycle management	Datasets, artifacts, checkpoints, logs, and retention; active data may be staged to faster tiers

Table 3.6: Storage roles introduced in the course

Storage planning completes the component-level view. Networking has only been introduced at a high level here; *Chapter 5* develops the network fabrics and data-movement paths in depth. We can now return to the facility and see how the concentration of compute, memory, networking, and storage inside a rack changes the planning problem.

Recognizing how compute density affects facility limits

A rack can run out of deliverable power or cooling capacity while rack units remain available. That makes density a question of deployable infrastructure, not just physical space.

Use rack space, power, and cooling limits together when deciding whether a dense deployment is viable.

Practical activities

You can complete the following activities without a live GPU environment. Use the stated values, show your reasoning, and identify any assumption that would need validation in a real deployment.

Activity 1: calculating PUE for three facility scenarios

Calculate PUE for each scenario, then rank the facilities from lowest to highest facility overhead. Do not infer GPU or workload efficiency from the PUE value alone. *Table 3.7* provides the values for the three scenarios:

Scenario	Total facility energy	IT equipment energy
A	2.3 MWh	2.0 MWh
B	2.7 MWh	2.0 MWh
C	3.2 MWh	2.0 MWh

Table 3.7: PUE calculation activity

For each scenario, write the formula, the calculated PUE, and one sentence explaining what the result says about facility overhead. Then explain why the result does not measure AI workload performance.

With facility efficiency separated from workload efficiency, the next activity shifts from energy accounting to processor responsibilities.

Activity 2: mapping work to the CPU, GPU, or DPU

For each task below, identify the processor that is most directly associated with the work. If another processor is still involved, note that dependency rather than treating the task as exclusive to one processor:

1. Run operating-system services and coordinate an application process.
2. Execute large matrix operations during model training.
3. Offload packet processing and selected security functions where the platform supports it.
4. Prepare a training batch and submit accelerator work.
5. Execute tensor operations used by a deep-learning inference workload.
6. Handle infrastructure networking or storage services without consuming as much host CPU time.

After classifying the tasks, explain why the statement "the GPU does the AI workload" is useful but incomplete from an infrastructure perspective.

The final activity brings those processor roles back together with the facility constraints introduced earlier in the chapter.

Activity 3: reviewing a dense rack for hidden limits

A proposed rack contains six accelerator servers estimated at 7 kW each, 3 kW of network equipment, and 2 kW of local storage and management equipment. The rack can deliver 52 kW of IT power. The cooling design for that rack position is rated for 45 kW under the expected operating conditions.

Complete the following review:

1. Calculate the planned rack IT load.
2. Compare the result with the electrical capacity.
3. Compare the result with the cooling capacity.
4. Identify the first constraint that prevents the configuration from being accepted as written.
5. Suggest two categories of change that could resolve the problem without naming a specific product.
6. List two additional checks you would perform before approving a revised design.

The purpose is not to design the final facility. It is to recognize that the limiting constraint may sit outside the compute specification and must be resolved before platform selection can be considered complete.

With the three activities complete, use the review questions to test whether you can recognize the same constraints in shorter scenarios.

Review questions

The following questions are original practice items written for this chapter. They do not reproduce live NVIDIA exam questions. Use the explanations to assess your reasoning, not just your chosen answer.

Question 1

A proposed accelerator rack fits physically, but its expected load exceeds the rack's deliverable electrical capacity. What should be addressed first?

A. Increase the model batch size to improve GPU utilization.

B. Address the rack power constraint before deployment.

C. Replace the shared storage tier with object storage.

D. Change the monitoring dashboard to show power in watts.

Question 2

A facility uses 1.8 MWh of total energy while its IT equipment uses 1.0 MWh during the same period. What is the PUE?

A. 0.56

B. 1.00

C. 1.80

D. 2.80

Question 3

Which statement best describes a GPU-oriented compute advantage for AI workloads?

A. It replaces CPU coordination and operating-system services during AI processing.

B. It executes many suitable arithmetic operations concurrently at high throughput.

C. It guarantees lower latency for every control-heavy application path.

D. It stores complete training datasets permanently in accelerator memory.

Question 4

Which processor is primarily designed to offload selected networking, storage, and security infrastructure functions from the host?

A. CPU

B. GPU

C. DPU

D. L2 cache

Question 5

Which statement best distinguishes CUDA Cores from Tensor Cores at the level required in this chapter?

A. CUDA Cores manage host control flow; Tensor Cores manage device memory.

B. CUDA Cores handle parallel arithmetic; Tensor Cores accelerate matrix and tensor operations.

C. CUDA Cores offload high-speed networking; Tensor Cores manage shared-storage services.

D. CUDA Cores manage rack power; Tensor Cores calculate facility energy efficiency.

Question 6

Why is raw core count alone a weak way to compare unrelated GPU generations?

A. Architecture and workload context also affect delivered performance.

B. Core counts include only graphics resources and exclude AI-capable GPUs.

C. Every GPU generation implements those execution resources in exactly the same way.

D. Host RAM capacity determines GPU performance more strongly than core architecture.

Question 7

A distributed training job repeatedly reads a large shared dataset from many nodes. Which storage characteristic is most directly relevant?

A. Sustained throughput and concurrent shared access.

B. The drive-activity indicator color during the training job.

C. The number of user accounts configured on each compute node.

D. The facility PUE measured during normal office hours.

Question 8

A rack is within its power limit but very close to its cooling limit under sustained workload. What is the strongest conclusion?

A. Available rack power guarantees that the design can operate safely.

B. Cooling may be the limiting sustained-load constraint and needs validation.

C. GPU workloads should move to CPUs whenever the rack runs warm.

D. Storage capacity must be reduced before the rack can continue operating.

Answers and explanations

Use the explanations below to check the reasoning behind each answer and revisit the relevant section if a distinction is still unclear.

1. **B:** Facility power is a hard deployment constraint. If the rack cannot deliver the planned electrical load safely and continuously, the equipment plan must change before workload tuning or component-level choices matter.
2. **C:** PUE equals total facility energy divided by IT equipment energy. 1.8 / 1.0 = 1.8.
3. **B:** GPUs are effective when workloads expose substantial parallel arithmetic. They complement rather than eliminate the CPU and still depend on memory, storage, networking, and software.

4. **C:** A DPU is designed to offload and isolate selected infrastructure services such as networking, storage, and security processing where the platform supports those functions.
5. **B:** CUDA Cores support broad parallel arithmetic, while Tensor Cores specialize in supported matrix and tensor operations used heavily in AI workloads.
6. **A:** Raw core count is not a universal cross-generation performance measure because architecture and workload context affect delivered performance.
7. **A:** A multi-node training job that repeatedly reads a common dataset needs a storage path that can support concurrent shared access and the required aggregate throughput.
8. **B:** Power and cooling are separate constraints. Remaining electrical capacity does not compensate for insufficient heat-removal capacity under sustained load.

Summary

In this chapter, you moved from the workload concepts in *Chapter 2* into the foundations of an AI-centric data center. You identified its four main blocks and the facility constraints that shape dense accelerator deployments.

You calculated PUE and kept facility efficiency separate from workload performance. You compared CPU- and GPU-oriented compute behavior, assigned complementary responsibilities to CPUs, GPUs, and DPUs, and recognized the roles of CUDA Cores, Tensor Cores, and RT Cores without delving into detailed GPU microarchitecture.

Finally, you matched storage approaches to AI data needs and related dense compute to facility limits. *Chapter 4* builds on this foundation by moving from individual components to integrated NVIDIA accelerated computing platforms and deployment options.

4

Scaling with NVIDIA Accelerated Computing Platforms

Chapter 3 established the physical foundation of an **artificial intelligence** (**AI**) data center: facility constraints, processor roles, storage, and the generic **data processing unit** (**DPU**) concept. This chapter moves one level higher and explains how NVIDIA solutions combine those building blocks into usable accelerated systems and larger platforms.

That distinction matters because a **graphics processing unit** (**GPU**) is not a complete platform. A production AI environment also depends on host processing, accelerator and system memory, local and shared data paths, networking, management access, and a supported software environment. For NCA-AIIO, the useful skill is to recognize both the role of a solution and the scale at which it is intended to operate.

We will start with physical solution categories, move into DGX as an integrated AI system, scale from one system to DGX SuperPOD, place ConnectX and BlueField in the platform, and then consider reference architectures, platform generations, and deployment options. The emphasis is on role and workload fit rather than on memorizing product specifications.

In this chapter, we will cover the following topics:

- Moving from NVIDIA GPU solution categories to integrated platforms
- Understanding DGX as an integrated AI system
- Scaling from DGX systems to DGX SuperPOD
- Positioning ConnectX and BlueField within the platform
- Using NVIDIA reference architectures appropriately
- Recognizing what changes across DGX platform generations
- Comparing on-premises, cloud, and combined deployment patterns

By the end of this chapter, you will be able to distinguish NVIDIA solution categories by role, explain what makes DGX an integrated AI system, recognize what changes when infrastructure scales to DGX SuperPOD, separate ConnectX connectivity from BlueField offload, use reference architectures appropriately, and compare on-premises, cloud, and combined deployment patterns.

Technical requirements

This chapter is architecture-focused. You do not need access to a DGX system, a GPU cluster, or a cloud account. To complete the activities, you need the following:

- The workload characteristics established in *Chapter 2*
- The facility, processor, storage, and DPU concepts established in *Chapter 3*
- Basic familiarity with servers, storage, and networking
- A notebook, text editor, or spreadsheet for the comparison exercises
- A willingness to separate durable platform roles from generation-specific specifications

> **Note**
>
> **Exam focus**: *Chapter 4* concentrates on AI Infrastructure and solution recognition. Focus on DGX and DGX SuperPOD roles, scaling choices, accelerated-cluster components, ConnectX versus BlueField, the purpose of reference architectures, and the trade-offs between on-premises and cloud infrastructure.

From GPU solution categories to integrated platforms

NVIDIA solutions span different workload contexts. RTX-class solutions appear in workstation, visualization, and selected accelerated-computing scenarios, while Jetson provides an edge-focused option. DGX then moves the discussion into an integrated data-center AI system, and DGX SuperPOD extends that idea to large multi-system infrastructure.

For NCA-AIIO, the useful distinction is the role and scale of each solution, not a catalog of current models. Product names, generations, and exact specifications change, but the boundary between an accelerator-oriented solution, an integrated AI system, and a larger coordinated platform is durable.

Table 4.1 summarizes the solution categories by role and recognition cue:

Solution level	Typical role	Recognition cue
NVIDIA RTX-class solution	Accelerated graphics, visualization, and professional workstation workloads, with selected GPU compute use cases	Recognize a workstation or individual accelerated-system role rather than a multi-node AI platform.
Jetson edge platform	Compact accelerated computing for edge and embedded AI	Recognize an edge role when AI processing needs to run close to devices or sensors.
DGX system	Integrated data-center AI system combining GPUs with host, interconnect, storage, networking, management, and system software	Recognize an integrated multi-GPU platform rather than a standalone accelerator.
DGX SuperPOD	Large-scale infrastructure built from interconnected DGX systems and supporting cluster components	Recognize a large coordinated AI and **high-performance computing** (**HPC**) platform when multi-system scale is required.

Table 4.1: NVIDIA solution categories at role and use-case level

The categories in the table are not a mandatory upgrade path. A workload should remain at the smallest level that meets its compute, memory, locality, and operational requirements. The next step is to see what has to surround the accelerator when the requirement moves from a component to an integrated platform.

Moving from an accelerator to a platform

A GPU supplies accelerated compute, but it does not operate in isolation. The host has to prepare and coordinate work, memory has to hold active data, local and external **input/output** (**I/O**) paths have to feed the workload, and management interfaces support system operation. As a result, platform-level reasoning asks how these roles work together rather than treating the GPU specification as the whole system.

For example, a GPU may have enough compute capacity for a model, but the complete system must still load the model, supply data, coordinate processing, and provide management and recovery paths. That is the difference between selecting an accelerator and evaluating an integrated platform. Multi-system scaling is discussed later in this chapter.

Recognizing the platform around the accelerator

These component groups provide the vocabulary for reading DGX system diagrams. Detailed network, storage, and operational mechanisms are covered in later chapters.

Host processing and system memory

The host CPU runs the operating system and services, coordinates application work, and participates in I/O and control. System memory supports that host-side work. The key point is that accelerator performance depends on the host being able to prepare, move, and coordinate work efficiently.

Local accelerator communication

Multi-GPU systems need a fast way for accelerators to exchange data. NVIDIA platforms can use technologies such as NVLink and NVSwitch for this local accelerator communication. For now, focus on this role; *Chapter 5* explains where these technologies sit in the broader data-movement path.

Storage, networking, management, and software

Non-Volatile Memory Express (**NVMe**) storage can provide fast system-local capacity, while external network interfaces connect the system to peers, shared storage, users, and services. Management interfaces support administration and recovery, and the system software provides the supported environment in which the accelerators are used.

Together, these roles explain why an integrated AI system is more than a collection of GPUs. *Table 4.2* summarizes each component's responsibility within an accelerated platform.

Platform component	Primary role	Why it matters at platform level
Host processing and system memory	Run the operating environment, services, control work, and host-side data preparation	The accelerator depends on the host being able to coordinate and feed work.
Accelerators	Provide parallel AI and HPC compute with accelerator-attached memory	Compute capability and active-memory capacity define what can run efficiently.
Internal interconnect	Move data among local accelerators and related components	Multi-GPU work depends on efficient local communication.

Platform component	Primary role	Why it matters at platform level
Local storage	Provide fast system-local capacity for active data, caches, checkpoints, and temporary artifacts	Local I/O can support staging and working data close to the system.
External networking	Connect peers, shared storage, users, and services	Multi-system and shared-data workloads depend on paths beyond one server.
Management and system software	Provide administration, recovery, drivers, libraries, and the supported operating environment	The integrated hardware still needs to be manageable and usable as a platform.

Table 4.2: Roles that make an accelerated system a platform

Understanding DGX as an integrated AI system

Treat DGX as the name of NVIDIA's integrated AI system family rather than as an acronym to memorize. The useful distinction is the role of the system: DGX combines accelerated compute with the host, interconnect, storage, networking, management, and software needed to use it as a data-center AI platform.

Reading the component map

Start with the accelerators. Multiple GPUs provide the main parallel compute capacity, while accelerator-attached memory holds the active model and runtime data. Host CPUs and system memory run the operating environment and coordinate the work that reaches those accelerators.

Figure 4.1 brings these component roles together in a generation-neutral map:

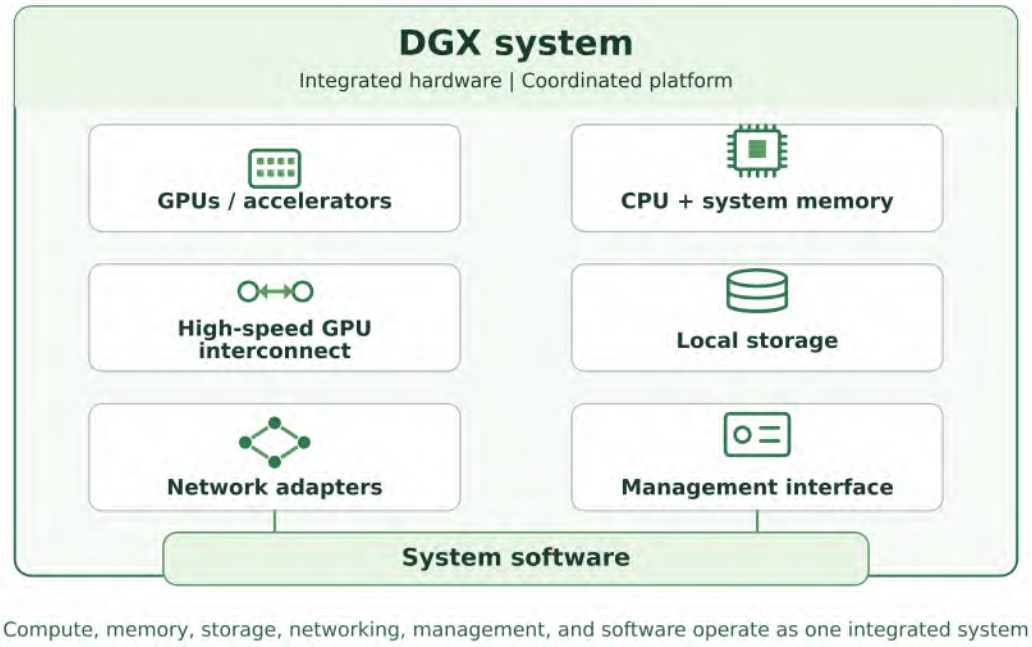


Figure 4.1: Generation-neutral DGX system component map

Figure 4.1 shows functional groups rather than a literal chassis layout. The physical arrangement and specifications vary by DGX generation.

Separating stable roles from generation-specific details

DGX component roles remain consistent even as their specifications change across generations.

Table 4.3 lists what to verify for each platform component group in *Table 4.2*:

Platform component	Generation-specific details to verify
Host processing and system memory	CPU model and count; system-memory capacity and configuration
Accelerators	GPU generation and count; supported numeric formats; performance; accelerator-memory capacity, bandwidth, and technology
Internal interconnect	NVLink/NVSwitch generation, topology, and supported bandwidth

Platform component	Generation-specific details to verify
Local storage	NVMe device count, capacity, and configuration
External networking	Adapter types, interfaces, port speeds, and network connectivity
Management and system software	Management interfaces; supported operating system, drivers, CUDA, libraries, firmware, and management software

Table 4.3: Generation-specific details to verify for DGX components

This avoids the assumption that newer hardware is always better. A generation change matters when it changes workload fit, capacity, communication, facility impact, or software support in a way that is relevant to the requirement.

Scaling from one node to a cluster and DGX SuperPOD

A DGX system has finite compute, memory, storage, and I/O capacity. When a workload needs more aggregate compute, distributed execution, greater concurrency, or a larger service footprint than one system can provide, the architecture moves from a single-system view to a cluster and then, at larger scale, to DGX SuperPOD.

What changes when systems become a cluster

The first major change is that local resources are no longer enough. Multiple systems need a high-performance communication path, coordinated access to data, and cluster-wide management. These are cluster-level dependencies even though the detailed network, storage, scheduling, and monitoring mechanisms are covered later in the book.

Table 4.4 summarizes the progression from an individual accelerated system to larger coordinated infrastructure:

Scale view	What is added	What to recognize
Workstation or single accelerated host	One local accelerated system	Useful when the workload fits one host and does not need cluster-scale coordination.
Integrated DGX system	A complete NVIDIA multi-GPU AI system with host, interconnect, storage, networking, management, and software	The system boundary is larger than the GPU itself.
Multi-system cluster	Communication, shared data access, and coordinated management across systems	Once work spans systems, cluster-level dependencies become part of workload fit.
DGX SuperPOD	Large-scale coordinated DGX infrastructure with compute, storage, networking, management, and software	Use the SuperPOD concept to recognize large AI/HPC scale, not a fixed node count.

Table 4.4: Recognizing the scale path from system to SuperPOD

Positioning DGX SuperPOD

DGX SuperPOD combines multiple DGX systems with compute and storage fabrics, management, and software for coordinated large-scale AI and HPC workloads.

A logical SuperPOD topology makes those relationships easier to see. *Figure 4.2* illustrates the management, compute, storage, and network paths without prescribing a fixed deployment blueprint:

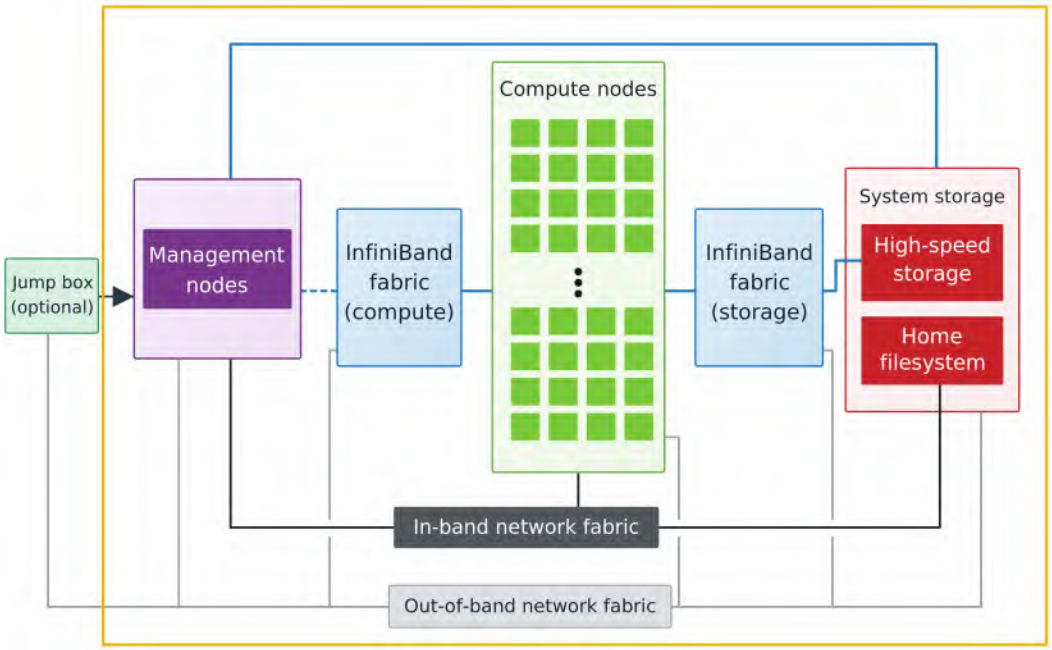


Figure 4.2: Logical view of DGX SuperPOD components and network paths (adapted from the NVIDIA DGX SuperPOD Reference Architecture)

Figure 4.2 shows DGX compute nodes, compute and storage fabrics, system storage, the management domain, and in-band and out-of-band network paths. Specific components and connections vary by generation and architecture. The InfiniBand fabric labels illustrate one supported configuration; other designs may use different supported network fabrics.

SuperPOD-class infrastructure is appropriate when distributed training, HPC scale, or multi-user demands justify cluster-level coordination.

At cluster scale, network connectivity and infrastructure offload become more prominent. ConnectX and BlueField serve different roles in these paths.

Positioning ConnectX and BlueField within the platform

Chapter 3 introduced DPUs at a general level. Here, that concept moves into NVIDIA's platform context alongside ConnectX networking. Both participate in accelerated infrastructure, but their primary responsibilities are different.

Using ConnectX for the high-performance network path

ConnectX provides the server-side interface to a high-performance network. *Chapter 5* explains InfiniBand, Ethernet, **remote direct memory access (RDMA)**, **RDMA over Converged Ethernet (RoCE)**, and the accelerated data-movement mechanisms that use those paths.

Using BlueField for infrastructure offload and isolation

BlueField is NVIDIA's DPU family for programmable infrastructure offload. It can offload selected networking, storage, and security operations from the host CPU and support isolation. NVIDIA **DOCA (Data Center Infrastructure-on-a-Chip Architecture)** provides the software framework used with BlueField DPUs. The important point here is the platform role, not a catalog of BlueField variants or DOCA features.

A BlueField DPU is therefore not required simply because a server contains GPUs. It becomes relevant when the platform needs infrastructure offload, isolation, or programmable data-path processing that the host would otherwise handle.

Table 4.5 compares their roles and indicates where the underlying mechanisms are covered:

Product family	Primary platform role	Scope boundary
ConnectX	Server-side high-performance network interface for accelerated network paths	Network protocol and data-path mechanics are covered in *Chapter 5*; RDMA is a capability, not another name for ConnectX.
BlueField	DPU family for programmable infrastructure offload and isolation	Extends the generic DPU concept from *Chapter 3* and is relevant when offload or isolation is required, not simply because GPUs are present.

Table 4.5: Positioning ConnectX and BlueField in an accelerated platform

Using NVIDIA reference architectures appropriately

NVIDIA reference architectures document how DGX platform components fit together in generation-specific designs. They provide an implementation starting point rather than a fixed topology for every deployment.

Consult the current official reference architecture to:

- Understand how compute, networking, storage, management, and software are arranged in a documented design
- Verify generation-specific components, topology, and software support
- Inform, rather than replace, site-specific design and validation

Recognizing platform generation changes

The component details in *Table 4.3* matter when they change workload fit or deployment constraints. Accelerator memory can affect model fit, interconnects can affect distributed workloads, and power and cooling demands can limit facility compatibility.

Verify that the required drivers, libraries, firmware, and management software are supported for the generation being evaluated.

An existing system may be the better choice when it already meets the workload, facility, and support requirements.

With scale and generation separated, the final platform decision is where the accelerated capacity should run.

Comparing on-premises, cloud, and combined deployment patterns

On-premises and cloud infrastructure differ in control, capacity, cost, and operational responsibilities. A combined deployment can also place different workload phases in different environments, such as using cloud capacity for training and on-premises infrastructure for inference.

The deployment choices form part of the broader platform view. *Figure 4.3* summarizes the platform components and deployment options before we examine their trade-offs:

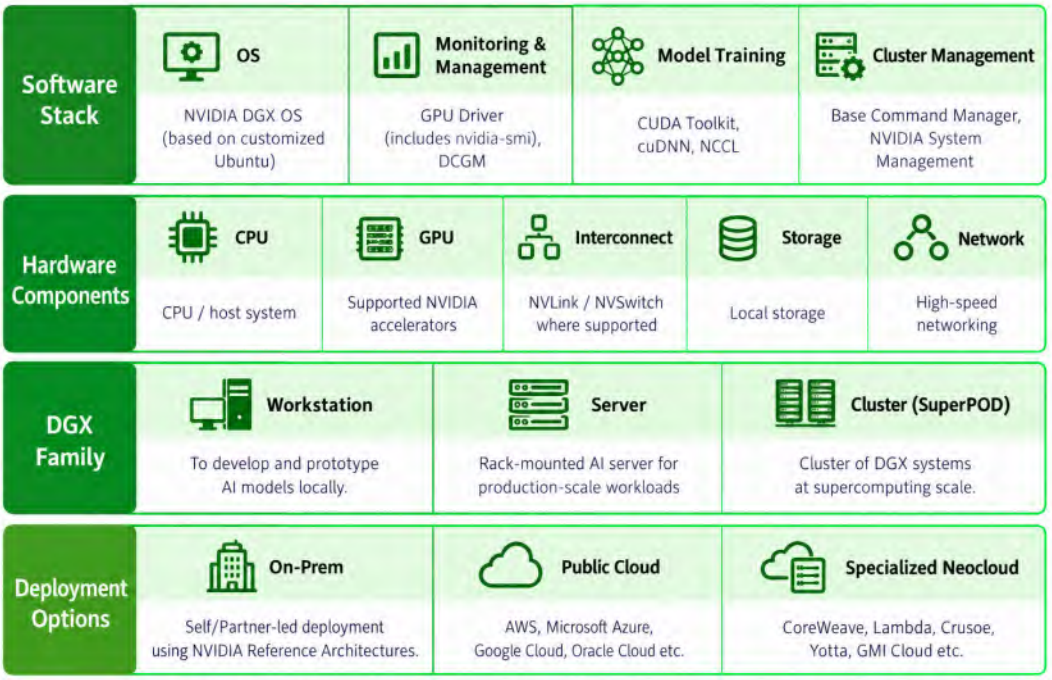


Figure 4.3: NVIDIA accelerated platforms and deployment options at a glance

Figure 4.3 is a general platform overview with illustrative hardware, software, and provider examples, not a generation-specific architecture. Consult current NVIDIA documentation for implementation details. CUDA and NCCL are discussed in *Chapter 6*, while monitoring and management tooling is revisited in *Chapter 9*.

Using on-premises when control and locality matter

On-premises infrastructure gives the organization direct control over its physical platform, local integration, and data-placement choices. That can be valuable when locality, sovereignty, existing systems, or organizational control are important requirements.

The trade-off is that the organization owns the physical capacity and its supporting facility responsibilities. Expansion depends on available rack space, power, cooling, network and storage capacity, procurement timelines, and operational resources rather than on requesting additional provider capacity.

Using cloud for rapid access and variable demand

Cloud infrastructure can provide faster access to accelerated capacity without first purchasing and installing the physical systems. This can suit experimentation, temporary training

demand, or workloads whose capacity requirements change over time, subject to provider availability, quotas, service limits, and pricing.

Cloud is not automatically cheaper, less secure, or more scalable in every situation. The organization still has to evaluate cost behavior, data location, access controls, compliance, network transfer, provider boundaries, and the services available in the required region or environment.

Combining environments when workload phases differ

One useful combined pattern is to use cloud capacity for compute-intensive training, bring the trained model into an on-premises environment for production inference, and return to cloud capacity when retraining is required.

The value of the example is workload placement. Training and inference can have different scale, locality, latency, cost, and control requirements, so they do not always need to run in the same environment. Moving data and model artifacts between environments still requires appropriate transfer, security, and validation.

Table 4.6 compares these decision dimensions and keeps the combined column tied to that specific workload-placement pattern:

Decision area	On-premises	Cloud	Combined pattern
Control and data location	Direct organizational control of the physical environment and local placement	Placement follows provider regions, services, and policy boundaries	Different workload phases can use different environments when requirements differ
Access and cost model	Requires owned capacity and supporting facility investment	Provides consumption-based access without first installing the physical systems	Cloud capacity can be used only for the phase that needs temporary scale
Scaling	Grows within installed capacity and procurement timelines	Can add capacity more flexibly, subject to availability, quotas, and service limits	Training can burst to cloud while steady inference remains on-premises

Decision area	On-premises	Cloud	Combined pattern
Compliance and sovereignty	Can keep selected workloads within organization-managed infrastructure	Requirements must be matched to provider capabilities, regions, and service boundaries	Placement can differ by phase when data and control requirements allow it
Operational responsibility	Organization operates the physical platform and supporting facility	Provider operates the physical infrastructure while the customer still owns workload governance	Organization manages the boundary, transfer, and consistency between the two environments
Typical fit	Steady demand, locality, or strong control requirements	Experimentation, variable demand, or rapid access to temporary capacity	Train in cloud, deploy for on-premises inference, and return to cloud for retraining

Table 4.6: Comparing on-premises, cloud, and a combined deployment pattern

The deployment label is therefore less important than the requirement behind it. Use the comparison to explain why a workload belongs in one environment, or why different phases may reasonably use different environments.

Practice scenarios

Complete the following scenarios using the solution roles and scale boundaries from this chapter. Where a scenario does not provide enough information for a final selection, identify the missing evidence instead of inventing it.

Activity 1: matching solution roles to scenarios

For each scenario, identify the NVIDIA solution or platform that best matches the requirement described:

1. A professional visualization team needs a workstation-class accelerated solution for interactive graphics and design workloads.
2. An embedded application needs AI inference close to sensors at an edge location with limited space and power.

3. A data-center team wants one integrated multi-GPU AI system with host resources, local interconnect, storage, networking, management, and supported system software.
4. A research environment needs large-scale distributed training across many coordinated accelerated systems with high-performance networking, shared storage, and centralized management.
5. An accelerated server needs a high-performance network interface for demanding cluster or storage traffic.
6. A platform needs programmable infrastructure offload and isolation for selected networking, storage, and security functions.

Table 4.7 shows the role or solution that best matches each cue. The purpose is recognition, not model-level product selection:

Scenario cue	**Closest role or solution and why**
Professional visualization or workstation-class accelerated workload	NVIDIA RTX-class solution: suited to workstation and visualization-oriented roles.
Embedded or edge AI inference near devices and sensors	Jetson edge platform: suited to edge deployments where data-center scale is not required.
Integrated multi-GPU data-center AI system	DGX system: DGX combines accelerators with host resources, interconnect, storage, networking, management, and software.
Large distributed AI/HPC environment across many coordinated systems	DGX SuperPOD consideration: suited to large-scale coordinated DGX infrastructure.
High-performance server network-interface requirement	ConnectX: it provides the server-side high-performance network interface.
Programmable infrastructure offload and isolation requirement	BlueField DPU: it extends the DPU role into NVIDIA's platform with infrastructure processing and offload.

Table 4.7: Matching platform cues to NVIDIA solution roles

These matches identify the closest role from the information provided. The next activity removes the product names and asks you to reason about scale itself.

Activity 2: choosing a scale level

Consider three workloads and select the smallest scale level that can reasonably satisfy each stated requirement. Do not choose a specific DGX model.

1. A research team has one model that fits comfortably on one integrated system and is used by a small number of users. The team values operational simplicity.
2. A training workload no longer completes within the required window on one node and can distribute efficiently across several nodes. It also needs shared access to a large active dataset.
3. An enterprise needs a large, multi-user AI and HPC environment with centralized operations, distributed training, shared storage, high-performance networking, and planned growth across many accelerated systems.

For each scenario, state the scale level, explain why more or less scale is appropriate, and identify the new platform dependencies that appear if the workload spans systems. A large integrated architecture such as DGX SuperPOD is a consideration only when the stated scale and operational requirements justify it.

These activities move from solution recognition to scale decisions. Use the review questions to confirm that you can keep system boundaries, product roles, and deployment choices separate before *Chapter 5* introduces the detailed network and data-movement mechanisms.

Review questions

The following questions test solution roles, scaling decisions, and deployment choices. Select the best answer for each.

Question 1

A team needs an integrated data-center AI system that combines GPUs with host resources, local interconnect, storage, networking, management, and system software. Which solution best matches that boundary?

A. A Jetson edge platform for embedded AI processing
B. An RTX-class workstation solution for professional visualization
C. A DGX system for integrated data-center AI workloads
D. A DGX SuperPOD for large multi-system AI infrastructure

Question 2

One integrated DGX system already meets a workload's capacity, performance, and service requirements. What is the most appropriate scaling conclusion?

A. Keep the workload on one system until a requirement justifies more scale.

B. Add a second system so every workload starts with cluster-level redundancy.

C. Move to SuperPOD because multi-node platforms are always easier to operate.

D. Move to cloud because a single on-premises system cannot run sustained inference.

Question 3

Which statement best describes the ConnectX role in this chapter?

A. It provides the local GPU-to-GPU interconnect inside one multi-GPU system.

B. It provides programmable infrastructure offload for host networking and storage services.

C. It provides local NVMe storage for checkpoints and active working data.

D. It provides the server-side high-performance network interface for accelerated data paths.

Question 4

Which requirement most directly points to BlueField rather than a conventional high-performance network-interface role?

A. A need for additional local accelerator memory inside one GPU

B. A need for programmable infrastructure offload and isolation from the host CPU

C. A need for faster local NVMe storage inside the integrated system

D. A need for more facility power and cooling capacity for the rack

Question 5

What should a candidate remember about an NVIDIA reference architecture?

A. It is a fixed topology that every current DGX deployment must reproduce.

B. It is a product specification table that should be memorized for the exam.

C. It is an implementation resource and documented starting point that can be consulted.

D. It removes the need to check current support and site-specific requirements.

Question 6

What is the strongest way to compare two DGX platform generations?

A. Choose the generation with the highest single published specification, regardless of the rest of the platform.

B. Choose the newest generation because newer hardware automatically provides the best workload and facility fit.

C. Compare only the GPU model and memory because host, network, and software differences are secondary.

D. Compare the system categories that affect workload fit, facility impact, and software support.

Question 7

Which scenario most directly matches the cloud advantage taught in this chapter?

A. A steady local workload must remain close to organization-managed data and systems.

B. A short training experiment needs temporary accelerated capacity without new hardware first.

C. A deployed system already has enough compute but needs additional facility cooling.

D. A single integrated system already meets the workload's capacity and service targets.

Question 8

Which scenario matches the combined deployment pattern described in this chapter?

A. Every workload runs simultaneously in cloud and on-premises environments by default.

B. Production stays in cloud while all training is permanently restricted to local hardware.

C. Training uses cloud capacity, inference runs on-premises, and retraining can return to cloud.

D. Workloads move between environments automatically without data or artifact validation.

Answers and explanations

1. **C:** A DGX system is the integrated data-center AI system in this comparison. Jetson targets edge use, RTX-class solutions can serve workstation-oriented roles, and SuperPOD represents larger multi-system infrastructure.

2. **A:** Scale should solve a requirement. If one integrated system already meets the workload and service objectives, adding nodes introduces cluster dependencies without solving a stated need.
3. **D:** ConnectX supplies the server-side high-performance network-interface role. The detailed InfiniBand, Ethernet, RoCE, and RDMA mechanisms are covered in *Chapter 5*.
4. **B:** BlueField is the DPU role for programmable infrastructure offload and isolation. It does not add GPU memory, local storage capacity, or facility power.
5. **C:** Consult reference architectures when exact platform details matter. You do not need to memorize a detailed topology or treat one example as a universal deployment design.
6. **D:** Compare the categories that can change workload fit and deployment constraints, such as accelerator and memory capability, communication, host resources, facility impact, and supported software.
7. **B:** Temporary demand and rapid access to accelerated capacity are reasons to consider cloud infrastructure, subject to provider availability, quotas, service limits, and pricing.
8. **C:** In the combined example, cloud capacity is used for training, on-premises infrastructure is used for inference, and cloud capacity can be used again when retraining is needed.

Summary

In this chapter, you moved from the component-level foundation of *Chapter 3* to NVIDIA solution and platform roles. You distinguished RTX-class and edge examples from integrated DGX systems, then treated DGX as a complete AI system rather than as a GPU model or an acronym to memorize.

You followed a generation-neutral DGX component map, scaled that view from one system toward clusters and DGX SuperPOD, and separated ConnectX network connectivity from BlueField infrastructure offload. You also kept reference architectures at the awareness level needed here rather than turning them into a separate design methodology.

Finally, you compared platform generations by durable categories and evaluated on-premises, cloud, and the combined deployment pattern described in this chapter. *Chapter 5* will build on these platform roles by explaining the network fabrics and accelerated data paths that connect them.

5

Moving Data Across Accelerated AI Infrastructure

Chapter 4 showed how individual accelerators become integrated systems and larger NVIDIA platforms. Once those systems start working together, another question becomes just as important as the GPUs themselves: can data move between the components fast enough to keep the workload moving? A cluster can have excellent compute on paper and still leave GPUs waiting because data cannot move fast enough across the network, over accelerator interconnects, or from storage.

That is the problem we will work through in this chapter. We will begin by separating the different kinds of traffic in an AI data center, because a management session, a checkpoint write, and a distributed training exchange should not be treated as the same workload. From there, we will compare Ethernet, converged Ethernet, **RDMA over Converged Ethernet** (**RoCE**), and InfiniBand, then move inside the server to place **PCI Express** (**PCIe**), NVLink, and NVSwitch in the right part of the path. Once the physical paths are clear, direct memory access (DMA), remote direct memory access (RDMA), GPUDirect RDMA, and GPUDirect Storage become much easier to understand.

The habit to build is simple: start by asking where the data begins and where it needs to end up. Engineers often call those two points the endpoints. The transfer may pass through a local GPU interconnect, a data-center network, or a direct-memory mechanism. Those pieces can work together in one end-to-end path, but each solves a different problem. Keeping those roles separate will make the technology choices in this chapter easier to follow.

In this chapter, we will cover the following topics:

- Separating compute, storage, in-band management, and out-of-band management traffic

- Understanding in-band and out-of-band management networks
- Comparing Ethernet, converged Ethernet, RoCE, and InfiniBand
- Understanding latency and throughput in accelerated networks
- Comparing PCIe, NVLink, and NVSwitch roles
- Understanding DMA and remote direct memory access
- Accelerating GPU-memory network transfers with GPUDirect RDMA
- Accelerating storage access with GPUDirect Storage
- Positioning OpenSM within an InfiniBand environment
- Tracing end-to-end data paths in a practical cluster scenario

By the end of this chapter, you will be able to map a workload's data path, identify the roles of network and accelerator technologies in that path, and explain how fabric management differs from payload movement.

Technical requirements

This chapter is mainly about architecture and certification-level recognition, so you do not need access to a live GPU cluster or a high-performance fabric. To work through the examples and activity, you need the following:

- The training and inference concepts established in *Chapter 2*
- The processor and storage concepts established in *Chapter 3*
- The accelerated platform concepts established in *Chapter 4*
- Basic familiarity with Ethernet and **Internet Protocol** (**IP**) networking
- A notebook, text editor, spreadsheet, or diagramming tool for the data-path exercises

Note

Exam focus: *Chapter 5* concentrates on AI Infrastructure, especially networking requirements, data center network options, and accelerated data paths. When a scenario names several technologies at once, first ask where the data starts, where it needs to go, and which layer is responsible for that part of the journey. PCIe, NVLink, NVSwitch, Ethernet, InfiniBand, RDMA, and GPUDirect technologies can appear in one design, but they solve different parts of the path.

Separating traffic in an AI data center

When engineers say "the network" in an AI environment, they can be referring to several very different flows. A training job exchanging data between nodes has different priorities from a checkpoint write, a **Secure Shell** (**SSH**) session, a **Baseboard Management Controller** (**BMC**) connection, or a user sending an inference request. If we group all of those flows together too early, we lose the information that actually drives the design.

A better starting point is to separate the traffic by what it is trying to accomplish:

- **Compute traffic** contributes directly to distributed workload progress.
- **Storage traffic** moves datasets, checkpoints, model artifacts, images, logs, and results.
- **In-band management traffic** reaches the running operating environment for administration, telemetry, scheduling, and software deployment.

Out-of-band management traffic reaches a hardware management controller independently of the host operating system and its operational state.

Client and external traffic can also cross the cluster boundary. These flows are relevant to routing and security but are outside the four fabric roles emphasized in this chapter.

These categories do not mean that every environment needs each role on a physically separate network. They give us a way to decide which flows can safely share infrastructure and which ones need stronger isolation, capacity, or recovery guarantees. *Table 5.1* turns that classification into a first design question for each traffic type.

Traffic category	Typical purpose	First design question
Compute	Distributed GPU communication, synchronization, and workload exchange	Which exchanges are latency-sensitive, and how do they grow as the workload scales?
Storage	Dataset reads, checkpoint writes, shared files, and artifact movement	What aggregate throughput and concurrency must the complete storage path sustain?
In-band management	Administration, telemetry, scheduling, software access, and control services	Which routine services must remain reliable without competing with workload traffic?

Traffic category	Typical purpose	First design question
Out-of-band management	Power control, hardware monitoring, remote console, and recovery	Can administrators reach the hardware when the host OS or production network is unavailable?
External/client boundary (context)	Inference requests, user access, data ingress and egress, and external dependencies	How will routing, security, availability, and application performance be protected at the boundary?

Table 5.1: Traffic categories in an accelerated environment

With those traffic classes separated, we can now look at the two flows that most directly affect workload progress: compute communication and storage movement.

Separating compute and storage traffic

Compute traffic becomes important the moment workload progress depends on another accelerator. Inside a single system, GPUs may exchange data through local interconnects. As soon as the job spans servers, the network between those servers becomes part of the workload itself because every synchronization step now depends on data crossing that network.

Distributed training makes this easy to see. Workers may exchange model data such as gradients, parameters, activations, or other state many times during a run. From an infrastructure perspective, what matters is that the exchanges happen repeatedly, so a fast worker can still sit idle while it waits for another worker or for the network between them.

Storage traffic follows a different rhythm. Nodes may stream samples, read many shards at once, stage an active dataset, write checkpoints, or push artifacts back to durable storage. In each case, the useful number is the performance of the complete path, not the specification of one storage device or one network port. The data has to travel from the storage system, through the filesystem and network, through host input/output (I/O), and finally to the workload.

Shared infrastructure becomes a concern when those traffic peaks overlap. A large checkpoint write can compete with a communication-heavy training phase, while synchronized GPU traffic can leave less capacity for data reads. The answer is not always a dedicated network. It may be physical separation, enough shared capacity, traffic classification, or a combination of those controls. The important point is to make that choice deliberately rather than discovering the challenge during a production run.

Recognizing management and external boundaries

Management traffic is usually small compared with training or storage traffic, but its importance is much larger than its bandwidth suggests. It carries the access, telemetry, and control that operators need during normal operation and when problems arise.

External traffic sits at another boundary. Inference requests, data ingestion, package and container retrieval, and movement between cloud and on-premises environments may all cross into or out of the cluster. Those flows can share Ethernet infrastructure, but they should not inherit the same trust assumptions as BMC access or the same performance assumptions as GPU synchronization.

Figure 5.1 brings those traffic zones together so you can see which paths serve workload progress, which ones support operations, and which ones cross the cluster boundary.

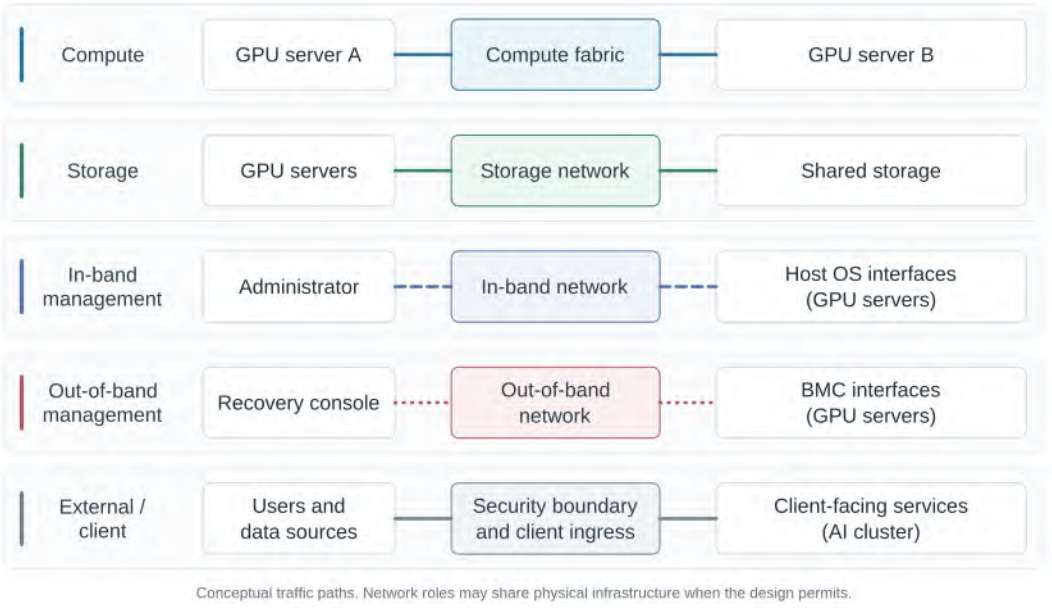


Figure 5.1: AI data center network zones

Notice that the figure does not force every flow onto its own physical network. Instead, it shows the responsibilities that need to remain visible in the design. Once those responsibilities are separated, the next question is how administrators reach the systems during normal operation and how they recover them when the normal path is no longer available.

Understanding in-band and out-of-band management networks

In-band and out-of-band management are often described as the normal path and the backup path. That is close, but the more useful distinction is dependency: what has to be working before the administrator can reach the system?

Using in-band management for routine operations

In-band management uses the interfaces and services of the running host. This is where routine operating-system administration, telemetry collection, scheduler communication, name resolution, repository access, and configuration management usually happen. When the host is healthy, it is the natural place to work because the administrator is interacting with the system in its normal operating context.

The limitation is that the path depends on the same operating environment you may be trying to troubleshoot. If the OS is hung, the host network is misconfigured, or the management service is unavailable, the normal administrative route can disappear with the failure. That is exactly why an independent path is valuable.

Preserving an independent out-of-band path

Out-of-band management uses a hardware management controller and a network path that does not depend on the host operating system. A BMC can provide remote power control, hardware sensor data, event logs, firmware management, and console access even when the host itself is not responding normally.

The goal here is recovery and visibility, not high throughput. The bandwidth requirement is modest, but reachability and access control matter a great deal because the same interface that lets you inspect a failed server may also let you reset it or change firmware. That makes the out-of-band network a privileged operational path, not just another management virtual local area network (VLAN).

Independence also has to be real, not just a label. If the in-band and out-of-band paths depend on the same upstream switch, power source, or identity service, one failure can remove both. A good recovery design looks at the complete failure domain rather than assuming that a separate port automatically creates a separate path.

Table 5.2 compares the two management paths from that dependency and recovery perspective.

Area	In-band management	Out-of-band management
Dependency	Relies on the host operating environment and its network path	Uses a management controller that operates independently of the host OS
Typical work	Routine administration, telemetry, scheduling, configuration, and software access	Power control, console access, sensor inspection, event review, and recovery
Performance need	Reliable, moderate bandwidth without disruption from workload congestion	Low bandwidth with high availability and predictable reachability
Main failure risk	Host or network configuration failure removes normal access	Shared power, switching, identity, or access design removes the recovery path

Table 5.2: Comparing in-band and out-of-band management

With management access separated, we can turn to the traffic that carries the workload itself. This is where Ethernet, RoCE, and InfiniBand enter the design, and where it helps to compare them by role rather than by a single speed number.

Comparing Ethernet, converged Ethernet, RoCE, and InfiniBand

Ethernet and InfiniBand are often placed side by side as if the design is choosing one or the other. A useful analogy is a highway and a bullet train: both move traffic, but one is a general-purpose network designed to support many kinds of journeys, while the other is designed around a more specialized high-performance path. The analogy is only about role and design intent. It does not mean that every InfiniBand path is always faster or that Ethernet cannot support high-performance communication. **RDMA over Converged Ethernet** (**RoCE**), explained shortly, illustrates that Ethernet can also carry RDMA traffic.

Starting with Ethernet

Ethernet is the familiar foundation for a large part of enterprise and cloud networking. It supports established switching and routing designs, broad interoperability, and a mature operational ecosystem. In an AI environment, management, client, storage, and even compute

traffic can all use Ethernet when the topology and service behavior meet the workload requirements.

It also helps to avoid one shortcut that causes confusion: Ethernet is not the same thing as TCP/IP. Ethernet operates at the link layer and can carry IP, RoCE, storage protocols, and other traffic. In the same way, saying that Ethernet is simply "slow" tells us very little. Workload performance depends on the complete path, including the endpoints, switches, shared capacity, and traffic mix.

Converging traffic on Ethernet

A converged Ethernet design uses one Ethernet infrastructure for traffic that might otherwise have lived on separate physical networks. For example, application traffic, storage traffic, and selected high-performance communication can share the same switches while remaining separated through VLANs or other logical networks, traffic classes, and policy.

That can simplify the physical design and let a team reuse existing Ethernet skills, but it also means the team has to carefully control how different types of traffic share the network. A fast uplink does not guarantee that a storage burst will leave latency-sensitive compute traffic untouched. Shared capacity, prioritization, congestion handling, redundancy, and failure domains still have to be designed with the workload in mind.

Adding RDMA with RoCE

RoCE adds another capability to that Ethernet environment. RDMA over Converged Ethernet allows compatible adapters to move data between memory areas that the application has prepared for RDMA, with less host-CPU involvement in moving the bulk data than a conventional buffered exchange. In other words, the network is still Ethernet, but the way the data is transferred changes.

For this chapter, the important point is that RoCE depends on the complete Ethernet environment. Compatible endpoints, drivers, switches, and software must work together; the exact congestion-control settings are implementation-specific and belong in the validated platform design.

Using InfiniBand as a high-performance fabric

InfiniBand approaches the problem from a different direction. It is a dedicated high-performance network designed for fast communication among servers, storage systems, and accelerators. A complete InfiniBand environment includes host adapters, switches, links, and a service called Subnet Manager that manages the InfiniBand subnet. The important point is that these pieces are designed to operate as one coordinated fabric.

That makes InfiniBand a natural fit for large distributed training and **high-performance computing** (**HPC**) workloads where communication between servers is a significant part of each iteration. The benefit does not come from a port-speed label alone. It comes from how the server adapters, switches, network layout, traffic management, and software work together under load.

That still does not make InfiniBand the automatic answer for every AI deployment. A smaller environment may already meet its objectives on Ethernet, while another organization may prefer routed integration, existing operational tooling, or a common network standard. The decision should follow the communication pattern, required predictability, scale, topology, support model, and the skills of the team that will operate the fabric.

Table 5.3 summarizes those roles without turning the comparison into a fastest-to-slowest ranking.

Technology or approach	Architectural role	Main dependency or caution
Ethernet	General-purpose link and switching foundation for management, application, storage, and compute traffic	Performance depends on the complete topology, congestion design, and traffic mix, not only link rate
Converged Ethernet	Shared Ethernet fabric carrying several logically separated traffic classes	Shared capacity, prioritization, isolation, and failure domains must be designed deliberately
RoCE	RDMA transport over Ethernet	Requires compatible endpoints and a validated congestion-control configuration
InfiniBand	Dedicated high-performance fabric with native RDMA and subnet management	Requires InfiniBand components, operational skills, and a workload that benefits from the specialized fabric

Table 5.3: Comparing Ethernet, converged Ethernet, RoCE, and InfiniBand

Figure 5.2 illustrates the different architectural roles and deployment dependencies of Ethernet, RoCE, and InfiniBand.

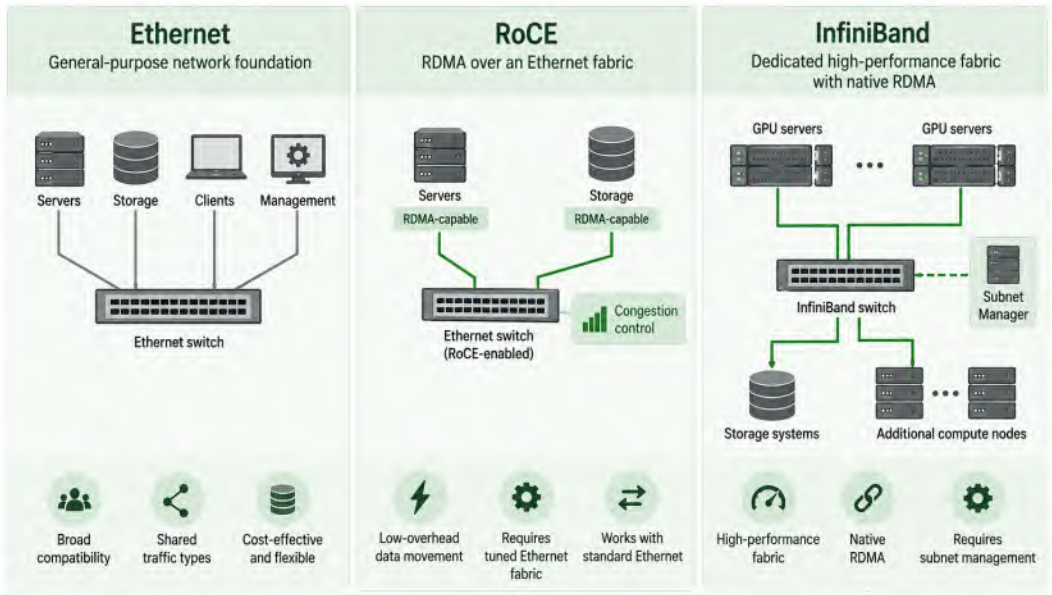


Figure 5.2: Ethernet, RoCE, and InfiniBand comparison

The comparison establishes the fabric roles. The next step is to examine the performance behaviors that determine whether any of those fabrics can keep the workload moving.

Reading throughput and latency in context

A high link rate does not guarantee good workload performance if the complete path cannot sustain the necessary data rate or if repeated communication delays stall synchronized workers.

Distinguishing bandwidth, throughput, and latency

Bandwidth describes the nominal capacity of a link or path. Throughput is the useful rate the workload actually achieves. Latency describes how long a communication event takes. Large transfers often emphasize sustained throughput, while tightly synchronized training can be sensitive to delays across repeated exchanges.

For certification scenarios, avoid judging a network by one specification. First identify the workload communication pattern, then decide whether the design needs general-purpose Ethernet, RDMA over Ethernet, or a dedicated high-performance fabric. With those performance terms in place, we can now move inside the host and examine PCIe, NVLink, and NVSwitch.

Comparing PCIe, NVLink, and NVSwitch roles

Ethernet and InfiniBand carry traffic between systems. PCIe, NVLink, and NVSwitch operate closer to the accelerator and solve a different part of the journey. A single distributed transfer can use both sets of technologies, so the goal is not to choose one family over the other. It is to understand where each one sits in the path.

Using PCIe as the general system I/O fabric

PCI Express, or PCIe, is the general-purpose I/O path inside the host. It connects GPUs, network adapters, storage devices, and other peripherals to the platform. For accelerated workloads, the key point is topology: devices inside the same server can still have different paths to one another, so placement can affect whether a preferred direct data path is available.

That placement matters when two devices need to exchange data directly. The practical lesson is to use the validated platform topology rather than assuming that devices are equally connected simply because they are installed in the same server.

Figure 5.3 shows a conceptual PCIe host I/O topology with a CPU, system memory, two GPUs, a **network interface card** (**NIC**), and a **Non-Volatile Memory Express** (**NVMe**) **solid-state drive** (**SSD**). The diagram illustrates device connectivity rather than the layout of a particular NVIDIA platform.

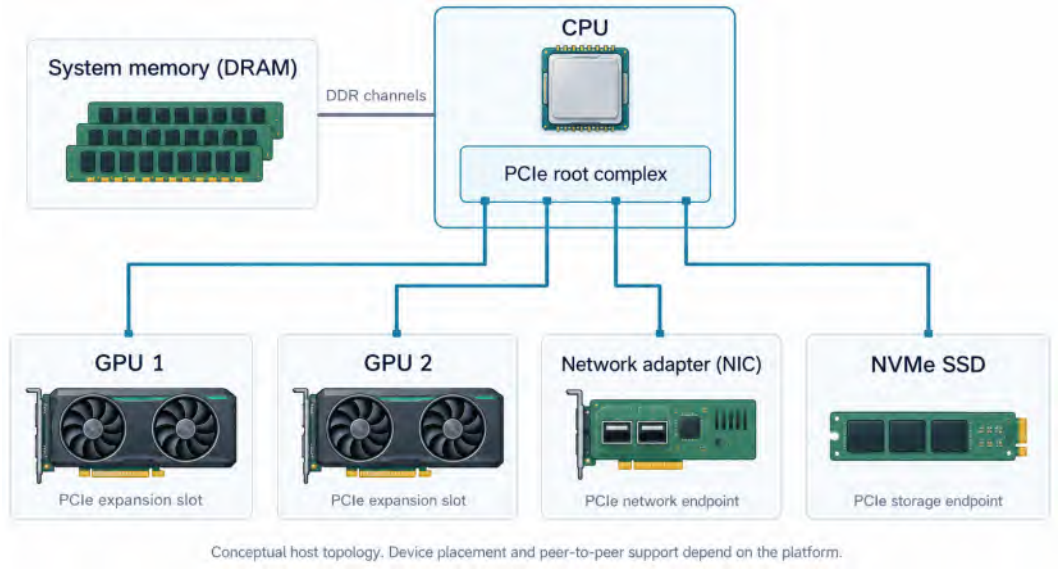


Figure 5.3: Conceptual PCIe host I/O topology with two GPUs

The figure gives us the host-side baseline. PCIe is the broad I/O fabric that connects devices; the next section narrows the focus to NVLink, which is designed for high-bandwidth communication among supported NVIDIA processors.

Using NVLink for accelerator communication

NVLink is more specialized. It is NVIDIA's high-bandwidth interconnect for communication among supported NVIDIA processors, most commonly GPUs and, on selected platforms, CPUs and GPUs. Where PCIe provides the broad I/O structure of the system, NVLink is designed for tightly coupled accelerator communication.

A simple way to visualize that difference is the skybridge analogy in *Figure 5.4*. Two buildings can each have their own internal routes, while a bridge creates a direct connection between selected endpoints.

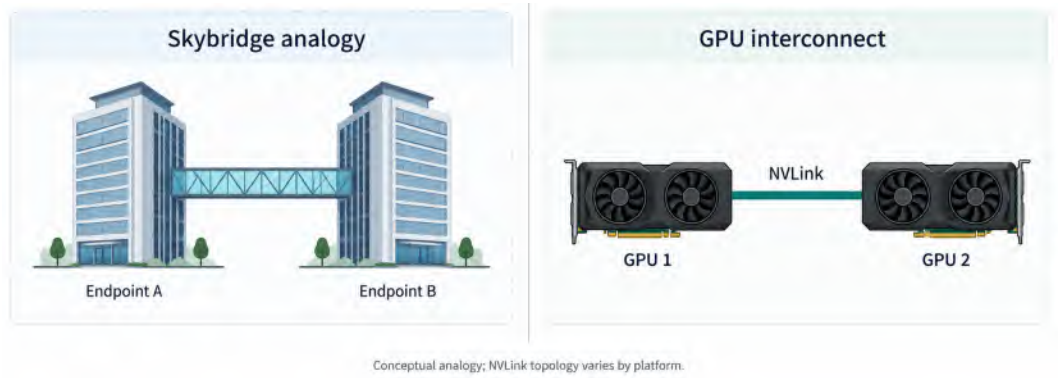


Figure 5.4: Skybridge analogy for a supported GPU-to-GPU NVLink connection

The towers represent endpoints in a visual analogy, not physical GPU hardware. The diagram on the right illustrates one supported GPU-to-GPU NVLink connection; exact processor endpoints, links, and bandwidth vary by platform and generation.

Using NVSwitch to build an NVLink fabric

As the number of GPUs grows, a collection of direct point-to-point links may no longer provide the connectivity the platform needs. NVSwitch adds switching to the NVLink fabric so that supported GPUs can communicate through a broader switched topology rather than relying only on individual direct links.

A network engineer may find the link-and-switch analogy useful, but the boundary matters: NVSwitch is an accelerator-fabric component, not a replacement for Ethernet or InfiniBand. The relationship is easier to remember in three parts:

- **NVLink** provides the high-speed connection.
- **NVSwitch** switches traffic across the supported NVLink domain.
- **PCIe** remains the general I/O path for GPUs, NICs, storage, and other devices.

Table 5.4 summarizes the roles and, more importantly, the first question to ask when one of these technologies appears in a design.

Technology	Primary role	Question to ask
PCIe	General system I/O fabric connecting GPUs, NICs, storage, and other devices	Where are the devices placed in the host topology, and is the required peer path supported?

Technology	Primary role	Question to ask
NVLink	High-bandwidth communication among supported NVIDIA processors	Does the workload need tightly coupled accelerator communication beyond the ordinary PCIe path?
NVSwitch	Switching fabric for NVLink-connected processors in supported platforms	Does the platform need broad switched connectivity across several participating GPUs?

Table 5.4: Comparing PCIe, NVLink, and NVSwitch

Table 5.4 separates the three technologies by role. *Figure 5.5* then focuses on the scaling problem NVSwitch addresses: adding a switch layer inside a supported NVLink domain so that multiple GPUs do not depend only on a collection of direct point-to-point links.

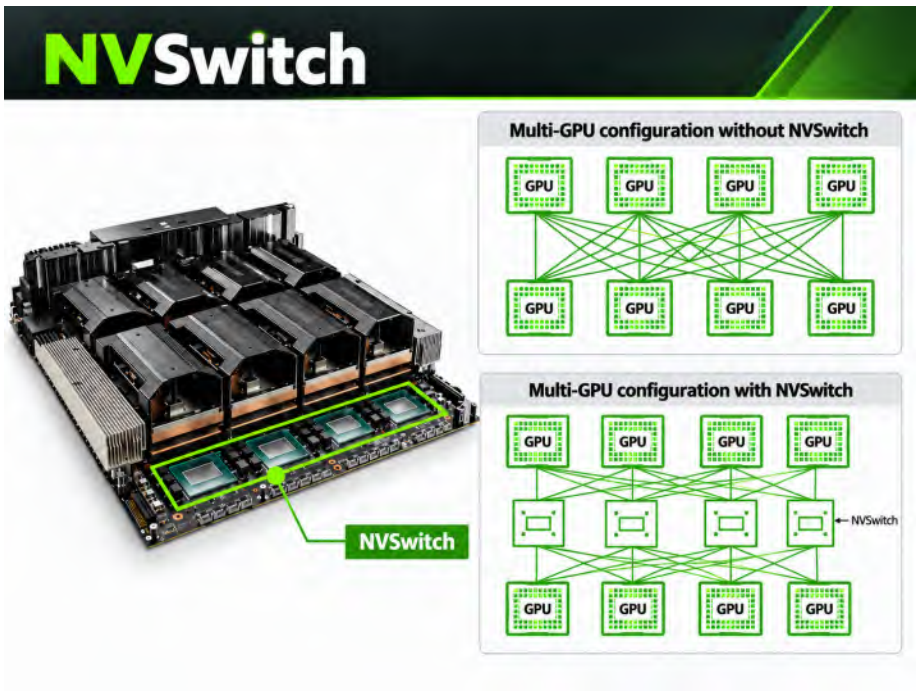


Figure 5.5: NVSwitch providing switched connectivity within a supported multi-GPU NVLink domain

These local paths define where data can travel inside the system. The NVSwitch comparison is conceptual, and the exact direct-link or GPU-to-NVSwitch connectivity varies by platform. The role boundary is what matters: NVLink provides the high-speed interconnect, while NVSwitch adds switching within the supported NVLink domain. With the local accelerator path established, the next section turns to DMA and RDMA, which describe how devices move data without making the CPU copy every byte.

Understanding DMA and remote direct memory access (RDMA)

The word direct appears several times in the rest of this chapter, and it can sound more dramatic than it really is. DMA, RDMA, GPUDirect RDMA, and GPUDirect Storage do not make the CPU or operating system disappear. They mainly remove particular copying or staging steps for the actual data being transferred, which networking documentation often calls the payload. If we keep asking "which copy is being avoided?", the terminology stays manageable.

Moving local data with DMA

Direct memory access (**DMA**) lets a capable device move data to or from memory after the CPU and operating system have prepared the transfer. The CPU still allocates buffers, configures the device, starts the operation, and handles completion or errors. What changes is that the CPU no longer has to copy every unit of payload itself.

That matters because high-volume I/O would otherwise spend valuable CPU cycles moving data rather than doing application or control work. GPUs, network adapters, and storage controllers all rely on DMA-capable paths, so this is a basic building block for the accelerated data paths that follow.

Extending direct-memory transfer across a network

Remote direct memory access (**RDMA**) carries the same idea across a network. Compatible interfaces can transfer data between memory areas that software has explicitly prepared for RDMA on different hosts, with less involvement from the remote CPU and the usual operating-system networking path while the bulk data is moving.

The setup work still exists. Applications and drivers create the required resources, prepare memory for RDMA, queue transfer requests, and handle completion events. RDMA improves the transfer path; it does not automatically fix an inefficient application, a poor device layout, or a congested network.

RDMA can run over technologies such as InfiniBand and RoCE. *Table 5.5* summarizes the distinction between local DMA and RDMA across hosts.

Technology	Endpoints or scope	What changes in the data path
DMA	A local device and memory	The device moves payload after software prepares the transfer, rather than the CPU copying each unit of data
RDMA	Registered memory regions across hosts	RDMA-capable interfaces move data with reduced remote-CPU and kernel data-path involvement

Table 5.5: Comparing DMA and RDMA

That progression from DMA to RDMA provides the foundation for the two GPU-direct paths that follow, starting with communication between GPU memory and a network peer.

Accelerating GPU-memory network transfers with GPUDirect RDMA

RDMA itself is an industry-standard networking capability, used by technologies such as InfiniBand and RoCE to move data directly between memory locations with minimal CPU involvement. GPUDirect RDMA, however, is an NVIDIA-specific technology. It extends this idea to NVIDIA GPU memory, allowing a compatible PCIe peer device, such as an RDMA-capable network interface, to access GPU memory directly. NVIDIA describes GPUDirect RDMA as providing a direct path between the GPU and a third-party peer device using standard PCI Express features.

GPUDirect RDMA can reduce host-memory staging in network transfers involving GPU data. Where the hardware, device layout, drivers, and software support it, a compatible RDMA-capable peer device can exchange payload data directly with NVIDIA GPU memory.

One common multi-node example looks like this:

GPU memory -> RDMA-capable network interface -> RoCE or InfiniBand fabric -> RDMA-capable network interface -> GPU memory

The important word here is payload: the actual data being transferred. That data can avoid the traditional host-memory bounce buffer, which is an intermediate copy in system memory, but the CPU and operating system still create, authorize, and manage the communication. *Figure 5.6* makes that distinction easier to see.

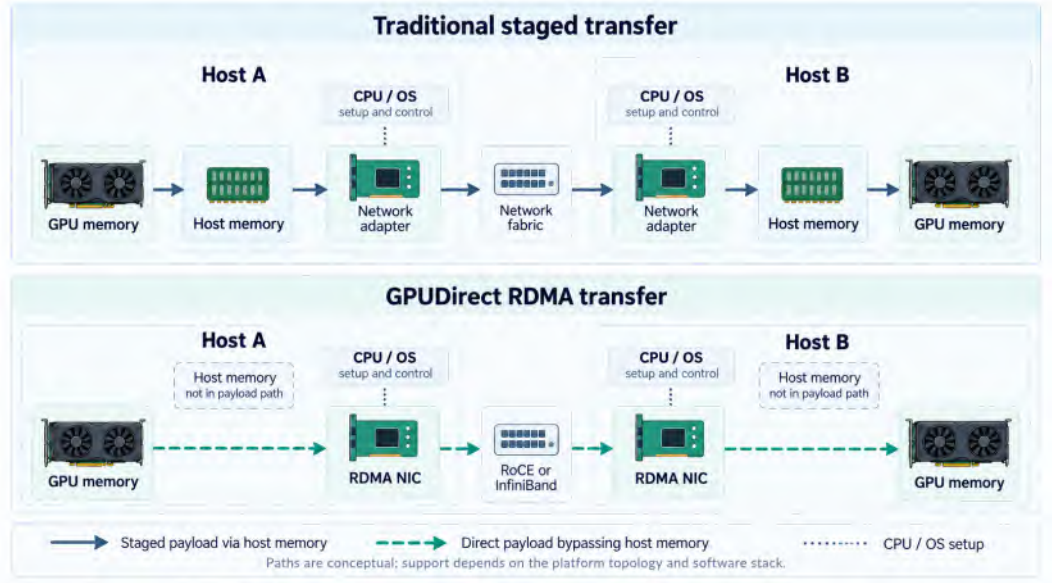


Figure 5.6: Traditional transfer and GPUDirect RDMA flow

The shorter payload path is useful only when the platform supports it end to end, so the next check is the complete hardware and software support chain.

Checking the support chain

For the certification, remember these core support requirements:

- A supported NVIDIA GPU and driver and software stack
- A compatible RDMA-capable network adapter or **data processing unit** (**DPU**)
- A supported platform topology and RDMA-capable network fabric

Exact support depends on the platform and software release, so use the validated topology and support documentation rather than treating a simplified diagram as proof that the direct path exists.

GPUDirect RDMA is most relevant when network transfers involving GPU memory are significant enough that avoiding host-memory staging can improve the data path. It does not replace the network fabric itself.

Accelerating storage access with GPUDirect Storage

Chapter 3 followed data from durable storage toward the accelerator. Even when the storage system itself is fast, the final movement into GPU memory can still create extra host-memory copies and CPU work. GPUDirect Storage addresses that part of the path.

Like GPUDirect RDMA, GPUDirect Storage is an NVIDIA-specific technology. A conventional transfer can look like this:

Storage -> host memory -> GPU memory

GPUDirect Storage, or GDS, allows supported storage I/O to use a DMA path to or from GPU memory without first staging the bulk data in host memory. As with GPUDirect RDMA, the benefit comes from removing an extra copy, not from making the rest of the storage system or software disappear.

Figure 5.7 uses a remote-storage RDMA example to compare traditional host-memory staging with a supported GDS direct path.

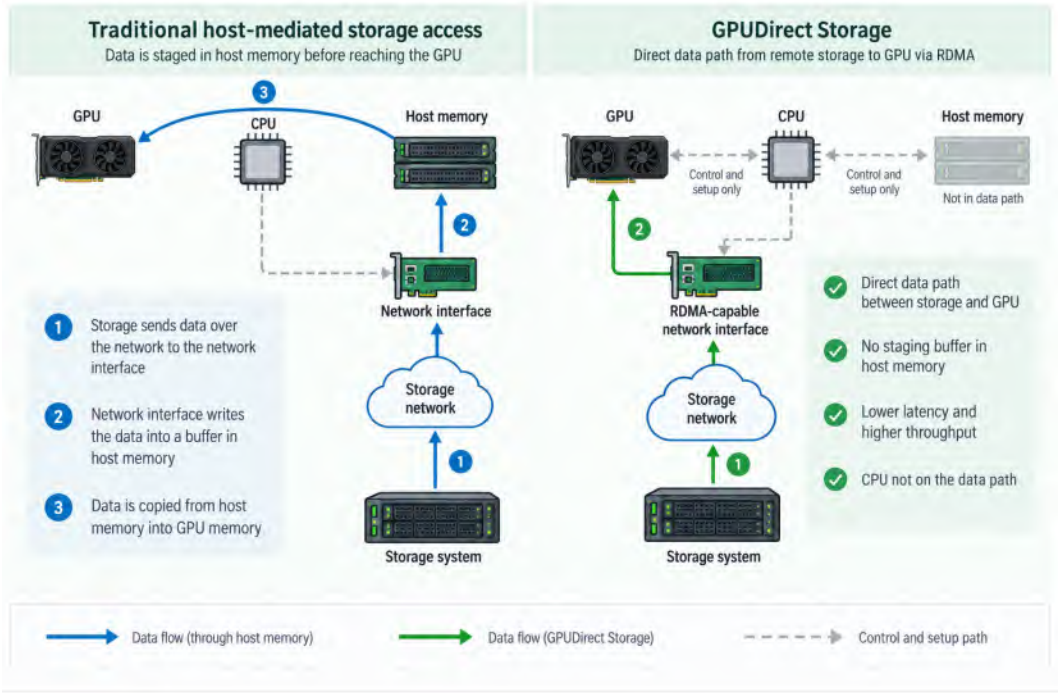


Figure 5.7: GPUDirect Storage data path

The storage source does not have to be an SSD sitting beside the GPU. Depending on the supported platform and software stack, GDS can support direct transfers involving local NVMe, networked file storage, networked NVMe storage accessed through **NVMe over Fabrics** (**NVMe-oF**), or parallel filesystems. The durable idea is the supported storage-to-GPU DMA path, not one particular storage product.

GDS is most relevant when storage-to-GPU transfers matter to the workload and the GPU, driver, filesystem, and storage components support the direct path. It does not replace the underlying storage system or make a slow storage source fast. Performance benefits vary with the platform and workload; unsupported direct paths may fall back to host-memory staging. Avoiding a CPU-memory bounce buffer does not necessarily remove CPU setup or PCIe root-complex traversal.

GPUDirect RDMA and GPUDirect Storage solve adjacent but different problems. *Table 5.6* puts their roles side by side before we move to the InfiniBand fabric-management layer.

Technology	Path it accelerates	Recognition cue
GPUDirect RDMA	GPU memory and a compatible RDMA-capable network peer	Network data movement involving GPU memory; requires an RDMA-capable network path
GPUDirect Storage	GPU memory and supported storage	Storage I/O to or from GPU memory; does not mean local storage only

Table 5.6: Comparing GPUDirect RDMA and GPUDirect Storage

With the two GPU-direct paths distinguished, the next section turns to InfiniBand fabric management.

Managing the InfiniBand fabric with OpenSM

OpenSM is an implementation of an InfiniBand Subnet Manager. It discovers and configures the InfiniBand subnet so the fabric can operate. It manages fabric connectivity rather than carrying GPU workload payloads. With that management role established, the practical activity below brings the data paths together.

Practical activity: tracing an end-to-end data path

Use one diagram to connect the layers from this chapter. Draw a simplified four-node training environment with GPUs, system memory, network adapters, a data-center fabric, shared

storage, and a separate out-of-band management path. Then work through the following steps:

1. Mark the local PCIe path inside each server and add NVLink or NVSwitch only where the platform supports those accelerator paths.
2. Choose either Ethernet with RoCE or InfiniBand for the inter-node communication path and write one sentence explaining why it fits the scenario.
3. Mark where generic RDMA operates, then show how GPUDirect RDMA changes the payload path when GPU memory participates directly.
4. Add shared storage and draw both the conventional staging path through host memory and the supported GPUDirect Storage path.
5. If the design uses InfiniBand, place OpenSM at the fabric-management layer and identify the subnet it manages.
6. Choose one pair that is easy to confuse, such as PCIe and NVSwitch or RDMA and GPUDirect RDMA, and explain the difference in one sentence.

Each label should identify a distinct responsibility in the diagram. The exercise tests whether you can place each technology at the correct layer, not select a universal design.

Review questions

The following questions are original practice items for this chapter. They do not reproduce live NVIDIA exam questions. Use the explanations to review the reasoning behind each answer, not just the letter you selected.

Question 1

A server's operating system is unresponsive, but an administrator can still view hardware sensors and power-cycle the system through a dedicated controller. Which network path is being used?

A. Distributed compute traffic

B. Out-of-band management

C. Client ingress networking

D. Storage traffic over RoCE

Question 2

An enterprise wants RDMA communication while retaining its Ethernet switching and operational model. Which technology most directly addresses the requirement?

A. OpenSM

B. GPUDirect Storage

C. NVSwitch

D. RoCE

Question 3

Which statement best distinguishes converged Ethernet from RoCE?

A. Converged Ethernet shares an Ethernet fabric; RoCE carries RDMA over Ethernet.

B. Converged Ethernet provides GPU switching, while RoCE supplies out-of-band management.

C. Converged Ethernet is a storage protocol, while RoCE is a management-plane service.

D. Converged Ethernet defines an InfiniBand fabric, while RoCE manages PCIe connectivity.

Question 4

A supported multi-GPU system needs switched communication across its NVLink domain. Which component provides that switching role?

A. PCIe link

B. OpenSM

C. NVSwitch

D. Ethernet BMC network

Question 5

What is the main difference between DMA and RDMA?

A. DMA manages InfiniBand subnets; RDMA provides out-of-band host recovery.

B. DMA moves local data without CPU copying; RDMA extends direct-memory access across hosts.

C. DMA carries application packets over Ethernet; RDMA switches traffic across an NVLink domain.

D. DMA coordinates GPU collectives; RDMA monitors cluster hardware health.

Question 6

A distributed training workload spends substantial CPU time copying data between GPU memory and host buffers before network transmission. Which capability most directly addresses that copy path when the complete platform supports it?

A. GPUDirect RDMA

B. Out-of-band management

C. NVSwitch subnet management

D. Object-storage lifecycle policy

Question 7

Which statement best describes GPUDirect Storage?

A. It combines local SSD capacity with GPU memory as one physical address pool.

B. It replaces the filesystem and storage network with an NVLink-only storage path.

C. It initializes InfiniBand adapters and switches before storage traffic can use the subnet.

D. It transfers data between supported storage and GPU memory without staging it in host memory.

Question 8

An InfiniBand cluster needs a service that discovers and configures the subnet. Which component provides this management function?

A. NVSwitch

B. RoCE

C. OpenSM

D. GPUDirect Storage

Before checking the answers, identify the layer involved in each question. In many cases, an option can be eliminated simply because it solves the wrong part of the path.

Answers and explanations

1. **B**: Out-of-band management reaches the BMC independently of the host operating system and supports hardware monitoring, console access, power control, and recovery.
2. **D**: RoCE carries RDMA over Ethernet. The endpoints, fabric, and software stack still need to support the intended RDMA path.

3. **A**: Converged Ethernet describes shared Ethernet infrastructure. RoCE is the RDMA transport used over Ethernet; the terms are related but not interchangeable.
4. **C**: NVSwitch forwards traffic across an NVLink fabric in supported platforms. NVLink supplies the interconnect, while PCIe remains the broader system I/O fabric.
5. **B**: DMA lets a capable local device move payload after software prepares the transfer. RDMA extends direct-memory communication across compatible network endpoints.
6. **A**: GPUDirect RDMA lets a compatible peer device, commonly an RDMA-capable network adapter, exchange data directly with GPU memory where the platform supports the path.
7. **D**: GPUDirect Storage enables a supported DMA path between storage and GPU memory without the traditional host-memory staging step. The filesystem, storage path, and software support still matter.
8. **C**: OpenSM is an InfiniBand Subnet Manager implementation. It discovers and configures the subnet so that the InfiniBand fabric can operate; it is not an accelerator interconnect or a GPU memory-transfer technology.

Summary

In this chapter, you traced data movement across an accelerated environment. You separated compute, storage, in-band management, and out-of-band management traffic, while treating client and external flows as boundary context. You then looked at how each path changes the performance, security, and recovery questions an operator has to answer.

You also compared Ethernet, converged Ethernet, RoCE, and InfiniBand without reducing the decision to a speed contest. Inside the system, you placed PCIe, NVLink, and NVSwitch at the correct scope, then built from DMA and RDMA into GPUDirect RDMA and GPUDirect Storage. Finally, you identified OpenSM as an InfiniBand subnet-management implementation.

The next chapter moves up the stack. With the physical data paths in place, *Chapter 6* will focus on the NVIDIA GPU software foundation, including drivers, CUDA, core libraries, NCCL, and the compatibility relationships that make the stack usable.

6

Understanding the NVIDIA GPU Software Stack

Chapter 5 followed data across the physical paths that connect GPUs, network adapters, storage, and other systems. Those paths matter only when the software above them can recognize the hardware, expose its capabilities, and give applications a usable way to request accelerated work. This chapter moves into that software foundation.

At a high level, the operating system and NVIDIA driver form the host-facing foundation, while CUDA and optimized libraries sit above them. For an infrastructure professional, the useful skill is not memorizing every package name. It is understanding what each layer is responsible for, which information the NVIDIA System Management Interface (`nvidia-smi`) can actually prove, and where a library such as the **NVIDIA Collective Communications Library** (**NCCL**) fits relative to the physical data paths from *Chapter 5*.

We will start with the dependency chain from the GPU upward, then focus on the NVIDIA driver and the software visibility provided by `nvidia-smi`. From there, we will interpret driver-level information accurately, place CUDA and the core libraries in the stack, recognize the official installation path without turning the chapter into an installation lab, and finish with NCCL collective communication. *Chapter 7* will then build on this foundation with **NVIDIA virtual GPU** (**vGPU**) and **Multi-Instance GPU** (**MIG**).

In this chapter, we will cover the following topics:

- Reading the GPU software dependency chain
- Understanding the NVIDIA driver role
- Using `nvidia-smi` for software visibility
- Understanding CUDA and core-library placement

- Recognizing the official CUDA installation path
- Understanding NCCL collectives

By the end of this chapter, you will be able to place the operating system, NVIDIA driver, CUDA, core libraries, frameworks, and applications in the GPU software stack; explain what the NVIDIA driver provides; use `nvidia-smi` output to recognize basic software and device state; distinguish the latest CUDA version supported by the driver from the version of any installed CUDA Toolkit. You will also be able to explain the roles of CUDA, the **NVIDIA CUDA Deep Neural Network library** (**cuDNN**), and NCCL at an associate level; recognize how to begin from the current official CUDA installation documentation; and distinguish NCCL collective communication from the physical interconnects and network transports it can use.

Technical requirements

This chapter is primarily conceptual. You do not need to install CUDA, change a GPU driver, or compile a CUDA program. To work through the examples and review questions, you need the following:

- The data-movement and interconnect concepts established in *Chapter 5*
- Basic familiarity with operating systems, device drivers, and application software
- A browser if you want to inspect the current NVIDIA driver, CUDA, or NCCL documentation
- A notebook, text editor, or spreadsheet for mapping software components to their responsibilities
- A willingness to separate durable software roles from version-specific package and release details

Note

Exam focus: *Chapter 6* concentrates on Essential AI Knowledge objectives 1.1 and 1.7, with supporting context for solution recognition and GPU monitoring. Focus on identifying the responsibility of each software layer, explaining CUDA and the core libraries, and reading driver-level information using `nvidia-smi`. Distinguish NCCL collective communication from the physical interconnects it uses. The goal is accurate recognition rather than memorizing installation commands or release numbers.

Reading the GPU software dependency chain

A GPU application does not reach the hardware through one piece of software. Several layers cooperate, and each layer answers a different question. The operating system provides the host

environment. The NVIDIA driver exposes and manages the GPU for software on that host. CUDA provides the NVIDIA parallel-computing platform and development and runtime foundation. Optimized libraries provide reusable GPU capabilities. Frameworks expose higher-level interfaces, and the application uses those interfaces to run its workload.

Table 6.1 maps each layer to its responsibility and recognition boundary.

Layer	Primary responsibility	What to recognize
Operating system	Provide the host environment, system services, process management, filesystems, and device support.	The host platform underneath the NVIDIA software stack.
NVIDIA driver	Connect the operating system to the NVIDIA GPU and expose device capabilities to CUDA and management software.	A required software path between host software and the GPU.
CUDA platform and Toolkit	Provide NVIDIA parallel-computing interfaces, runtime support, development tools, and related components.	The GPU-computing foundation above the driver, not the driver itself.
Core libraries	Provide optimized functions for common accelerated work and communication.	Examples include cuDNN for deep-learning primitives and NCCL for collective communication.
Framework	Provide higher-level APIs and execution structures for AI and scientific workloads.	Frameworks can use lower-level NVIDIA software without exposing every low-level operation to the application developer.
Application	Perform the training, inference, data-processing, or other workload the organization actually needs.	The workload at the top of the dependency chain.

Table 6.1: Responsibilities across the GPU software dependency chain

With those boundaries clear, start at the operating system.

Placing the operating system at the base

The operating system creates the environment in which the rest of the stack runs. Its kernel, device model, process management, package system, security controls, and system libraries all affect what can be supported above it. At this level, the important recognition point is that GPU software is still host software. It depends on a supported operating environment before an application can use the accelerator.

DGX OS provides a preconfigured operating-system and software baseline for NVIDIA DGX systems. It is derived from Ubuntu Linux and includes NVIDIA drivers, system software, and configurations optimized for DGX hardware.

Reading the software dependency diagram

Figure 6.1 summarizes the dependency chain. Read it from the GPU upward. Each higher layer relies on capabilities exposed by the layers below, but the higher layer has its own responsibility.

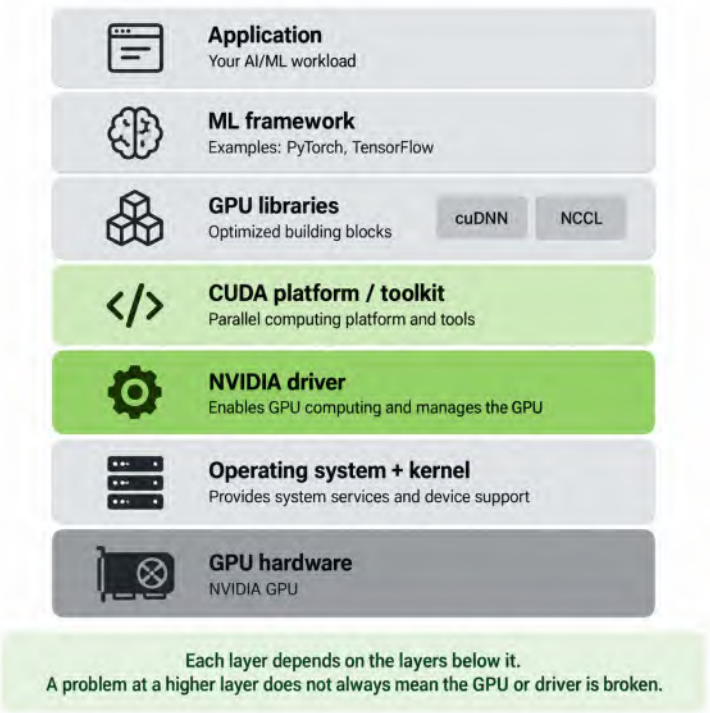


Figure 6.1: GPU software dependency stack

The figure is intentionally broad. Real systems can include container runtimes, language environments, orchestration software, additional libraries, and product-specific management components. Those details do not change the key approach: identify the layer first, then ask what that layer is expected to provide.

Next, examine the NVIDIA driver, which makes the GPU visible and usable to higher-level software.

Understanding the NVIDIA driver role

A device driver lets an operating system work with a hardware device. For an NVIDIA GPU, the driver provides the software path that lets the host operating system and NVIDIA user-space software communicate with the GPU. Without a suitable driver, higher-level CUDA software does not have a usable path to the device.

Connecting the operating system to the GPU

The driver exposes the GPU to management tools and to software that uses NVIDIA compute capabilities. It is responsible for device communication and resource management at the driver layer, while higher software layers decide how a workload uses those capabilities.

Consulting current driver documentation

NVIDIA publishes driver installation documentation for supported operating systems and installation methods. Use the current guide rather than relying on a memorized command sequence. Supported distributions, driver branches, repository details, and installation procedures change over time.

When a deployment decision depends on a driver version or installation method, use the current NVIDIA documentation for the actual operating system and GPU platform. A command copied from an older release can be technically correct for its original environment and still be the wrong choice for a current system.

Keeping driver success separate from application success

A working driver proves an important lower-layer fact: the operating system can communicate with the NVIDIA GPU through the installed driver. It does not, by itself, prove that a local CUDA Toolkit is installed, that a framework has the dependencies it expects, or that a particular training or inference workload will run correctly.

A lower layer can be healthy while a higher layer still needs attention. The next section uses `nvidia-smi` to inspect the driver and GPU without assuming that one command validates the complete software environment.

Using nvidia-smi for software visibility

The `nvidia-smi` command-line utility provides a quick view of the NVIDIA driver and the GPUs that the driver can see. That makes it useful for an initial question: Does this host have an NVIDIA driver that can communicate with the expected GPU? `nvidia-smi` is installed as part of the NVIDIA GPU driver package; you do not normally install it separately.

Reading key nvidia-smi fields

`nvidia-smi` can display the installed driver version, GPU identity, the **CUDA Version** field, memory and utilization information, operating state, and active processes, with the exact fields depending on the driver and device. *Chapter 9* returns to this tool from a monitoring perspective. Here, the focus is narrower: software visibility and correct interpretation of the driver-level information.

Figure 6.2 shows an illustrative output with the main groups of information called out. A different GPU, driver release, or workload can produce different fields and numbers.

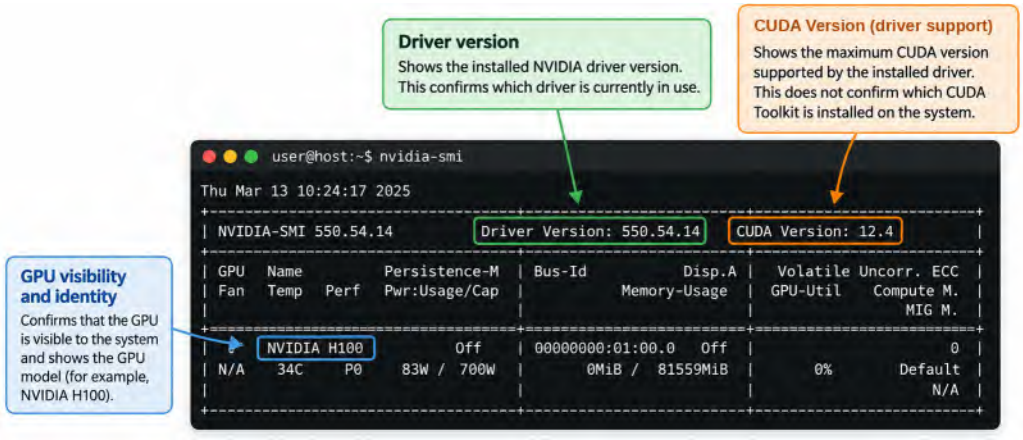


Figure 6.2: Reading nvidia-smi for software and device visibility

Table 6.2 separates software-visibility fields from the monitoring interpretation deferred to *Chapter 9*.

Field or area	Useful question	Boundary to remember
Driver version	Which NVIDIA driver is this host using?	The version is evidence about the driver, not a complete support decision by itself.

Field or area	Useful question	Boundary to remember
GPU identity	Which GPU devices can the driver currently see?	Visibility does not prove that a higher-level application is configured correctly.
CUDA Version field	What is the latest CUDA version supported by the installed driver?	It is not proof of the installed CUDA Toolkit version.
MIG or compute mode	What device mode is reported where the platform supports it?	The field is current state, not a design guide for vGPU or MIG. *Chapter 7* covers those choices.
Memory, utilization, and processes	What is the device doing at this moment?	These are operational observations. *Chapter 9* covers how to interpret them over time.

Table 6.2: Interpreting selected nvidia-smi fields in this chapter

`nvidia-smi` is a driver-level utility. In its summary output, the **CUDA Version** field reports the maximum CUDA version supported by the installed NVIDIA driver. It does not tell you, by itself, which CUDA Toolkit version, if any, is installed locally.

Understanding CUDA and core-library placement

CUDA is NVIDIA's parallel-computing platform and programming model for using NVIDIA GPUs for general-purpose accelerated work. The CUDA Toolkit packages development tools, libraries, headers, and utilities used to build and support CUDA applications.

Placing CUDA between the driver and higher-level software

The GPU hardware sits at the bottom. The NVIDIA driver gives the operating system and NVIDIA software access to that hardware. CUDA provides the runtime foundation above the driver. Optimized libraries and higher-level frameworks then use that foundation to expose useful capabilities to applications.

This arrangement lets application developers work at different levels of abstraction. Some software can call CUDA APIs directly. Other applications use a framework such as PyTorch or TensorFlow, while the framework and its dependencies use lower-level NVIDIA software.

CUDA does not make every workload parallel or automatically faster. Performance still depends on the workload, hardware, memory behavior, and software implementation.

Recognizing optimized libraries

Core libraries sit above or alongside the CUDA foundation and provide optimized building blocks for common work. Two examples are the **NVIDIA CUDA Deep Neural Network library** (**cuDNN**) and the **NVIDIA Collective Communications Library** (**NCCL**).

NVIDIA cuDNN provides GPU-accelerated primitives used by deep-learning software. A framework can use these optimized operations instead of implementing every low-level routine itself.

NCCL solves a different problem. It provides collective communication operations for workloads that use multiple GPUs, including communication that can span multiple systems. It coordinates communication among participating GPUs using the paths available on the platform. We will examine it in detail later in the chapter.

Placing frameworks and applications above the core stack

A framework gives developers a higher-level environment for expressing training, inference, or data-processing work. A simplified software path runs from application code through a framework, optimized NVIDIA libraries, and CUDA to the GPU. Real packaging can vary, but the responsibility boundary remains useful.

This simplified path does not mean that every framework uses or packages the same libraries. Supported combinations depend on the framework, release, platform, and workload. The useful skill is to place the components correctly and understand why the dependency exists.

The next section explains how to find current installation guidance for the target platform.

Recognizing the official CUDA installation path

NVIDIA's official CUDA installation documentation provides the supported installation path for each target platform.

Some cloud images and integrated NVIDIA platforms already include part of the required software stack. Installing another copy can create an unnecessary second software path. Check the existing environment before adding components for the workload.

Following the current installation guide

Use this high-level checklist:

1. Identify the NVIDIA GPU, operating system, and system architecture.
2. Open the current official CUDA installation guide for that platform.
3. Check the supported operating-system release and any documented prerequisites.

4. Choose a documented installation method that fits the environment and support model.
5. Follow the current procedure and keep driver requirements separate from Toolkit requirements.
6. After installation, use the current guide to verify the components you installed.

Next, consider communication between GPUs over the data paths introduced in *Chapter 5*. NCCL provides the collective-communication layer for that work.

Understanding NCCL collectives

A single GPU can execute accelerated work without communicating with another GPU. Distributed training and other multi-GPU workloads are different. Participating GPUs may need to exchange model state, partial results, or other data repeatedly. The hardware and network paths make the transfers possible, but the workload also needs software that organizes the communication pattern.

NCCL provides optimized collective communication for multi-GPU and multi-node workloads. In a collective operation, participating ranks follow a coordinated communication pattern rather than having each application instance construct its own point-to-point transfers.

Separating NCCL from the data path

Chapter 5 introduced PCIe, NVLink, NVSwitch, InfiniBand, RoCE, and RDMA as data paths, interconnects, fabrics, or transfer mechanisms. NCCL sits above those mechanisms as software. It can use supported paths that are available to the platform, but it does not replace the links, switches, adapters, or network transports underneath them.

With that distinction clear, the next question is which collective operation the workload needs and what result each participating rank should receive.

Recognizing common collective operations

A collective operation is defined by how data is distributed among the participating ranks. You do not need the NCCL API syntax for the certification. You do need to recognize the purpose of the main operations and distinguish them when a scenario describes a communication pattern. *Figure 6.3* illustrates four representative patterns. *Table 6.3* adds Reduce and provides examples of all five operations.

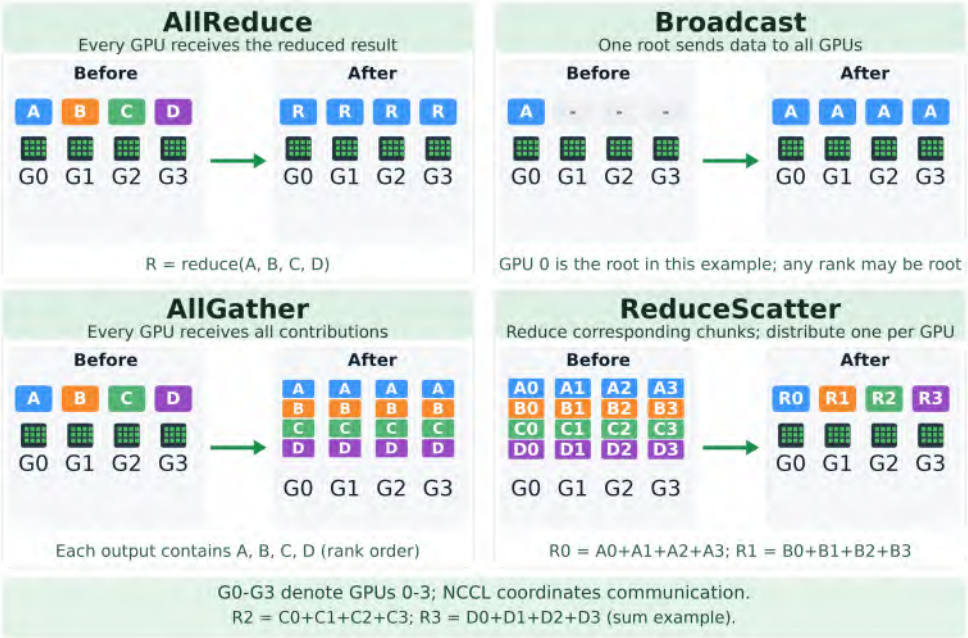


Figure 6.3: Common NCCL collective communication patterns

Collective	**What it does**	**Example use**
AllReduce	Combines data (e.g., sums it) from every GPU and gives the same result to every GPU.	Averaging gradients across GPUs in data-parallel training.
Broadcast	Copies data from one GPU (the root) to every GPU.	Sending the initial model weights from GPU 0 to all GPUs before training starts.
AllGather	Collects a piece of data from each GPU and gives the full set to every GPU.	Reassembling a layer's full weights in sharded training where each GPU stores only part of the model.
Reduce	Combines data from every GPU, but only the root GPU receives the result.	Summing per-GPU sample counts at one root for reporting.

Collective	What it does	Example use
ReduceScatter	Combines data from every GPU, then splits the result so each GPU receives a different part.	In sharded training, reducing gradients so each GPU retains only its assigned portion.

Table 6.3: Reading common NCCL collective operations

These result patterns provide the basis for considering how NCCL uses the paths available in a real system.

Using topology awareness without overclaiming automation

NCCL is topology-aware in supported environments. It can use information about the available communication paths when organizing transfers, which is one reason a framework or distributed application can use NCCL without manually programming every hardware connection. The exact transport choice and performance still depend on the platform, software configuration, and workload.

Topology awareness does not eliminate hardware and network constraints. NCCL can organize communication around supported paths, but it cannot create bandwidth, repair an unhealthy link, or make an unsupported transport usable. The available hardware, network, and software configuration still bound the result.

Keeping framework integration version-aware

Machine-learning frameworks can integrate with NCCL for supported distributed GPU workloads, but packaging and configuration vary by framework and release. The presence of a framework alone does not confirm that NCCL is configured for the workload. For implementation work, verify the current framework and NCCL documentation for the actual environment.

Use the review questions to check whether you can place each component in the correct layer and avoid drawing conclusions that a tool or product name cannot support.

Review questions

These practice questions test software-layer recognition and the distinctions developed in this chapter. They are not official certification questions.

Question 1

Which component most directly provides the software path between the host operating system and an NVIDIA GPU?

A. CUDA Toolkit, which provides development and runtime tools

B. NCCL, which organizes collective GPU communication

C. NVIDIA driver, which connects host software to the GPU

D. cuDNN, which provides optimized deep-learning primitives

Question 2

A host runs `nvidia-smi` successfully, and the header includes a **CUDA Version** value. What is the strongest conclusion from that field?

A. The local CUDA Toolkit is installed at the version shown in the header.

B. Every installed framework supports the CUDA version shown in the header.

C. The installed cuDNN package has the same version shown in the header.

D. The installed driver supports the CUDA version shown in the header.

Question 3

A driver-level check shows the expected GPU, but a framework still reports that no GPU is available. What is the best conclusion?

A. The GPU hardware has failed even though the driver still detects it.

B. The driver path works, but the framework layer still needs to be checked.

C. The CUDA Toolkit must be missing because the framework cannot see the GPU.

D. NCCL must be installed before the framework can access any NVIDIA GPU.

Question 4

Which statement best places CUDA in the software stack?

A. CUDA is the parallel-computing layer between the NVIDIA driver and higher-level GPU software.

B. CUDA is the physical interconnect layer between GPUs and switches inside a server.

C. CUDA is the management layer used for out-of-band administration of GPU servers.

D. CUDA is the device-driver layer that connects the host operating system to the GPU.

Question 5

What is the best starting point when an administrator needs the current supported CUDA installation procedure for a Linux host?

A. Use a package command copied from a previous host with a similar GPU and operating system.

B. Install the machine-learning framework first and let it select the host software stack.

C. Use the current CUDA installation guide for the target operating system and architecture.

D. Choose the installation method from the framework version without checking host support.

Question 6

Which component is best associated with optimized deep-learning primitives used by higher-level software?

A. NCCL, for multi-GPU collective communication

B. cuDNN, for optimized deep-learning primitives

C. OpenSM, for InfiniBand subnet management

D. NVSwitch, for switching traffic within an NVLink domain

Question 7

In an NCCL AllReduce operation, what does every participating rank receive?

A. The reduced result at one designated root rank

B. The complete set of contributions gathered from all ranks

C. One partition of the reduced result assigned to each rank

D. The same reduced result returned to every participating rank

Question 8

Which NCCL collective combines values across the group and then gives a different portion of the reduced result to each participating rank?

A. ReduceScatter

B. AllGather

C. Broadcast

D. Reduce

Answers and explanations

1. **C:** The NVIDIA driver provides the software path between the host operating system and the GPU. CUDA and higher software depend on that path.
2. **D:** The **CUDA Version** field in `nvidia-smi` belongs to the driver view. It reports the maximum CUDA version supported by the driver, not proof of a same-numbered locally installed CUDA Toolkit.
3. **B:** The driver-level check establishes GPU visibility through the driver. A framework can still fail because of its own dependencies, packaging, configuration, or compatibility.
4. **A:** CUDA is NVIDIA's parallel-computing platform and programming model. The driver sits below it, while libraries and frameworks can use CUDA above it.
5. **C:** Installation details are platform- and release-sensitive. Start from the current official CUDA installation guide for the target environment.
6. **B:** cuDNN provides optimized GPU-accelerated primitives for deep-learning software. NCCL, OpenSM, and NVSwitch serve different functions.
7. **D:** AllReduce combines values across participating ranks and returns the reduced result to every participant. Broadcast and Reduce have different distribution patterns.
8. **A:** ReduceScatter first reduces values across the group and then distributes a different portion of the reduced result to each participating rank.

Summary

In this chapter, you moved from the physical data paths in *Chapter 5* into the software layers that make NVIDIA GPUs usable by applications. You placed the operating system, driver, CUDA, core libraries, frameworks, and applications in a dependency chain and separated the responsibility of each layer.

You used `nvidia-smi` as a driver-level visibility tool and distinguished the latest CUDA version supported by the driver from a locally installed CUDA Toolkit. You also positioned CUDA and cuDNN in the core software stack, learned how to find current CUDA installation guidance, and distinguished lasting concepts from version-specific implementation details.

Finally, you placed NCCL above the physical interconnects and network transports from *Chapter 5* and recognized its main collective communication patterns. *Chapter 7* builds on this foundation by comparing NVIDIA vGPU with **Multi-Instance GPU** (**MIG**) and separating hardware partitioning, virtualization, and workload access.

7

Virtualizing Accelerated Infrastructure with vGPU and MIG

Chapter 6 explained the software stack that makes an NVIDIA GPU usable, from the driver and CUDA layer through higher-level libraries. Once that software path is understood, the next operational question is how one physical GPU can be allocated to more than one workload without losing sight of support, resource boundaries, and application requirements.

GPU sharing is attractive when a workload does not need an entire device all the time. A development team may need several GPU-enabled virtual machines, while a group of smaller services may fit into supported partitions of one GPU. The important point is that sharing is not one mechanism. Different approaches divide access, execution time, or hardware resources in different ways.

This chapter focuses on the two NVIDIA mechanisms most relevant to that decision: virtual GPU (vGPU) and **Multi-Instance GPU** (**MIG**). We will use whole-GPU assignment, passthrough, and time slicing only as brief context, then concentrate on how vGPU and MIG differ, what their support boundaries look like, and what evidence helps you recognize a configured MIG environment.

In this chapter, we will cover the following topics:

- Why virtualize or share GPUs
- GPU virtualization foundations
- Understanding NVIDIA vGPU
- Understanding MIG

- Comparing vGPU and MIG
- Inspecting MIG support and profiles
- Review questions

By the end of this chapter, you will be able to explain why accelerated infrastructure is virtualized, distinguish vGPU from MIG at a high level, recognize the purpose of MIG profiles and resource boundaries, identify important support considerations, and interpret basic MIG-related evidence without assuming that one GPU generation or profile layout applies universally.

Technical requirements

Most of this chapter is conceptual and can be followed without GPU hardware. The optional inspection material is read-only and should be used only in an approved environment.

To follow the chapter, you need the following:

- The driver and `nvidia-smi` concepts established in *Chapter 6*
- Basic familiarity with Linux, virtual machines, and resource allocation
- For optional inspection, access to a supported NVIDIA GPU environment where you are permitted to view device information and configuration

MIG support, available profiles, vGPU capabilities, and supported software combinations vary by GPU and platform. Treat profile names, instance counts, hypervisor combinations, and licensing details as version-sensitive rather than universal rules.

Note

Exam focus: Concentrate on the purpose of GPU virtualization, the high-level difference between vGPU and MIG, how MIG profiles describe supported resource shapes, and the evidence used to recognize a configured environment. The objective is to identify key considerations, not to design every possible GPU-sharing architecture.

Why virtualize or share GPUs

A useful sharing decision starts with a simple question: does the workload need the whole GPU all the time? If the answer is no, virtualization or partitioning may improve utilization by allowing more than one workload to use the same physical accelerator.

Consider several development environments that use a GPU only in short bursts, or a set of smaller services whose combined demand fits comfortably within one supported device.

Giving each workload a dedicated GPU may be operationally simple, but it can also leave expensive capacity idle for long periods.

Sharing can therefore improve resource utilization and deployment flexibility, but higher density is not the only goal. The selected mechanism still has to satisfy the workload's memory, compute, access, and separation requirements.

Starting from the workload requirement

Before choosing a technology, identify the workload's practical constraints. Does it need a virtual machine? How much GPU memory does it require? Does it run continuously or in bursts? Do several workloads need to run at the same time? Does it require a defined resource boundary, or can it tolerate shared execution capacity?

These questions matter because two workloads with similar average utilization may need very different allocation models. A latency-sensitive service and an intermittent development VM should not be treated as equivalent simply because both appear lightly loaded.

Use measured workload behavior where it is available, but keep the decision at the level required by this chapter: establish whether sharing is reasonable and which kind of boundary the workload needs before comparing vGPU and MIG.

Recognizing the benefits and limits

GPU sharing also introduces support and operational considerations. The platform has to support the selected sharing mechanism, the software stack must match that deployment, and the application still needs access to the resources it expects.

Sharing also changes the failure and performance context. Several virtual or partitioned resources can still depend on the same physical GPU and host, and some sharing modes allow workloads to compete for execution capacity. Resource separation should therefore be described in terms of the mechanism actually used rather than with broad claims such as fully isolated.

The next section outlines the main ways GPU resources can be allocated or presented to workloads.

GPU virtualization foundations

CPU virtualization provides a familiar mental model. A hypervisor can present virtualized CPU, memory, storage, and device resources to guest virtual machines. GPU virtualization extends that idea to accelerators, but the resource boundaries and software paths are GPU-specific.

It helps to separate three questions: what physical or partitioned GPU resource is allocated, how the workload receives access to it, and whether execution capacity is dedicated or shared over time.

A whole GPU can be reserved for one workload or assigned directly to a VM through passthrough. These approaches use the complete device rather than subdividing it through vGPU or MIG.

Time slicing is different. It allows multiple workloads to share GPU execution capacity over time, but it does not by itself create MIG-style hardware partitions with separate resource allocations.

NVIDIA vGPU adds a supported virtualization stack that presents virtual GPU devices to guest VMs. MIG, by contrast, partitions a supported physical GPU into defined resource instances. On supported platforms, these mechanisms can also be combined, so vGPU and MIG should not be treated as mutually exclusive labels.

The key is to identify which layer provides the boundary. vGPU primarily describes how GPU resources are virtualized and delivered to VMs, while MIG describes how supported GPU hardware resources are partitioned.

Table 7.1 summarizes the main concepts that provide context for the rest of the chapter.

Mechanism	**Main role**	**Key consideration**
Dedicated allocation	Reserves the whole GPU for one workload or environment	Simple boundary, but no sharing benefit
GPU passthrough	Assigns a physical GPU directly to a VM	Whole-device assignment rather than vGPU sharing
Time slicing	Shares GPU execution capacity over time	Does not by itself create MIG-style hardware partitions
NVIDIA vGPU	Presents supported GPU resources to guest VMs	Requires a supported host/guest vGPU software path
MIG	Partitions a supported GPU into defined resource instances	Profiles, placement, and support vary by GPU

Table 7.1: GPU allocation and sharing concepts in brief

Understanding NVIDIA vGPU

NVIDIA Virtual GPU Manager runs on the hypervisor, while a compatible NVIDIA vGPU driver in each guest VM enables applications to use the assigned virtual GPU.

This creates a host-to-guest software path rather than a single validation point. A healthy physical GPU does not prove that a VM has the correct vGPU assignment, guest driver, licensing state, or application environment.

Following the host-to-guest path

Table 7.2 maps the vGPU host-to-guest path to checks at each layer.

Layer	Responsibility	Validation question
Physical GPU and server	Provide the underlying device and platform	Is the proposed configuration supported on this hardware?
Hypervisor and vGPU Manager	Create and manage virtual GPU assignments	Is the required vGPU type available and assigned correctly?
Guest driver	Enable GPU access inside the VM	Does the guest recognize and initialize its assigned device?
Guest application environment	Run the framework, libraries, and workload	Does the application work inside the environment that will actually serve users?

Table 7.2: Validating the vGPU path from host to application

For example, if the host sees the expected physical GPU but a VM cannot initialize its assigned device, the next check is the vGPU assignment and guest driver path rather than reinstalling unrelated application software.

With the vGPU delivery path established, we can now look at MIG, which defines hardware resource partitions on supported GPUs.

Understanding MIG

Multi-Instance GPU (**MIG**) partitions a supported physical GPU into separate GPU Instances with defined portions of compute and memory resources. This allows multiple workloads to use one physical GPU while receiving supported resource allocations rather than simply taking turns on an undivided device.

The partition boundary is useful only when the workload fits the resources assigned to it and the required features are supported in that configuration. MIG should therefore be understood as a resource-partitioning mechanism, not as a guarantee that every application will behave identically to a whole-GPU deployment.

A **GPU Instance (GI)** is a hardware partition with a defined allocation of compute and GPU memory resources.

Reading a MIG profile

A MIG profile describes a combination of compute and memory resources that a particular MIG-capable GPU can provide; it is not an arbitrary memory amount entered by an administrator.

The available profiles and maximum instance counts vary by GPU model and generation.

Figure 7.1 illustrates how MIG divides a supported physical GPU into GPU Instances with defined compute and memory resources. Each instance can serve a different workload; supported partition sizes and placements vary by GPU.

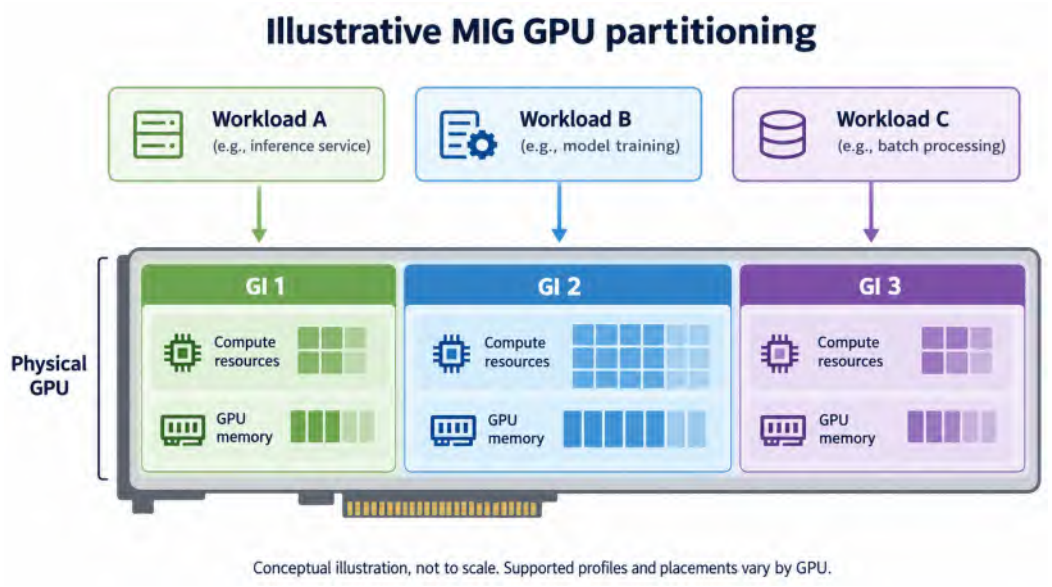


Figure 7.1: MIG partitions a physical GPU into GPU Instances with defined resources

Two profiles can expose similar memory capacities while assigning different amounts of compute. Memory size alone is therefore not enough to decide whether a profile is suitable for a workload.

On data center GPUs such as the A100, H100, and B200, MIG can create up to seven GPU Instances, each with its own dedicated compute and memory.

Comparing vGPU and MIG

The difference becomes important when a supported deployment combines vGPU delivery with MIG hardware partitions. Because the mechanisms operate at different layers, supported environments can combine them. A GPU Instance created through MIG can provide the backing resource for a MIG-backed vGPU presented to a VM.

Figure 7.2 shows the relationship between a vGPU delivery path, a MIG partition, and a supported combined arrangement.

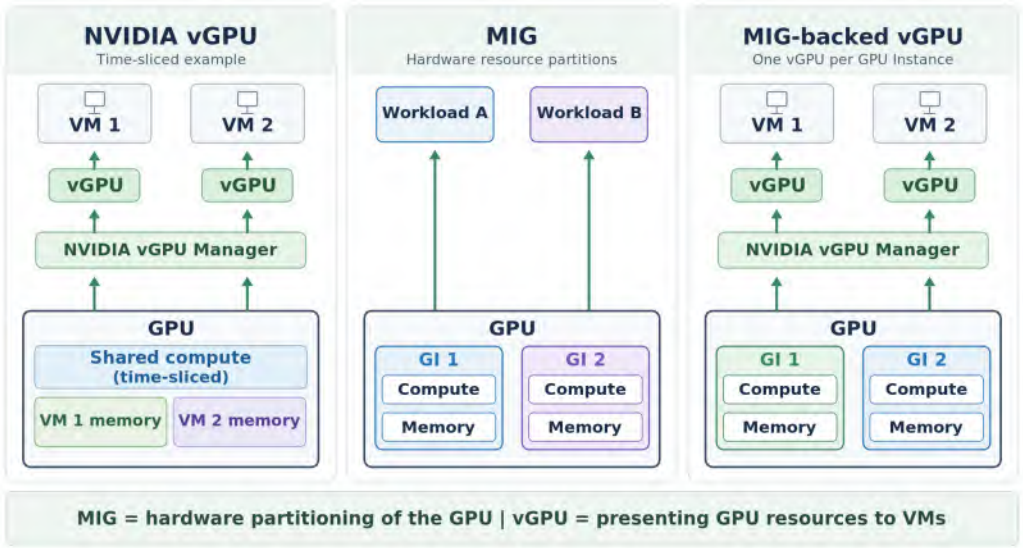


Figure 7.2: vGPU and MIG architecture comparison

When documenting a configuration, name the actual boundary rather than simply stating that the host uses GPU sharing. Record whether the workload receives a whole GPU, a vGPU, a MIG instance, or a vGPU backed by a MIG instance.

This makes troubleshooting easier because another engineer can identify which software and hardware layers need to be inspected.

Choosing between the mechanisms

Use vGPU for supported GPU delivery to VMs, MIG for supported hardware partitions, or MIG-backed vGPU when both are needed.

Neither description is a universal recommendation. The actual choice depends on the workload, whether a VM is required, the GPU model, the supported software stack, and the resource boundary the workload needs.

When stronger separation is required, describe exactly what must be separated: memory allocation, compute resources, guest environments, or operational ownership. Avoid using the word isolation without identifying the boundary being discussed.

A workload can fit within its allocation and still experience performance changes when other workloads share the surrounding system.

Checking support and licensing

A conceptually suitable design can still be unsupported on the proposed platform. Assess support based on the GPU model, host environment, driver or virtualization software, guest environment where applicable, and the selected vGPU type or MIG profile.

For MIG, verify that the GPU supports MIG and that the required profiles are available in the relevant software release. For vGPU, verify the hypervisor, Virtual GPU Manager, guest driver, guest operating system, selected type, and applicable licensing requirements.

Check the current NVIDIA compatibility and licensing documentation for the proposed software release.

A MIG allocation does not itself grant software entitlements. Verify licensing for the NVIDIA product and deployment model where applicable.

Inspecting MIG support and profiles

This section uses read-only checks to recognize the state of an existing environment. It does not enable MIG, create or delete instances, stop workloads, or change the allocation.

Run these checks only where you have permission to inspect the relevant GPU. The goal is to distinguish what the output establishes from what it does not.

Identifying the target GPU

Begin by listing the devices visible to the driver:

```
nvidia-smi -L
```

Figure 7.3 shows an illustrative device listing from an A100 example. The shortened UUIDs identify the physical GPU and its MIG devices; this is not output captured from the inspected system.



Figure 7.3: Illustrative NVIDIA GPU and MIG device listing

Record the physical GPU index, name, and UUID before selecting it for further inspection. If MIG devices are configured and visible to the driver, they can appear beneath the physical GPU in this listing.

For the following examples, GPU index 0 represents the device selected from that listing. Substitute the correct device identifier for your environment.

Inspect its detailed state with:

```
nvidia-smi -i 0 -q
```

Look for the GPU identity and MIG-related state where supported. An `N/A` field means the information is unavailable in that context; it should not automatically be interpreted as a hardware failure.

Comparing available profiles with existing instances

In a supported environment where MIG information is available, list the GPU Instance profiles the device can provide:

```
nvidia-smi mig -i 0 -lgip
```

Then list the GPU Instances that currently exist:

```
nvidia-smi mig -i 0 -lgi
```

Figure 7.4 shows the difference between a profile that the GPU supports and GPU Instances that have actually been created. It uses a simplified A100 example rather than live output from this system.

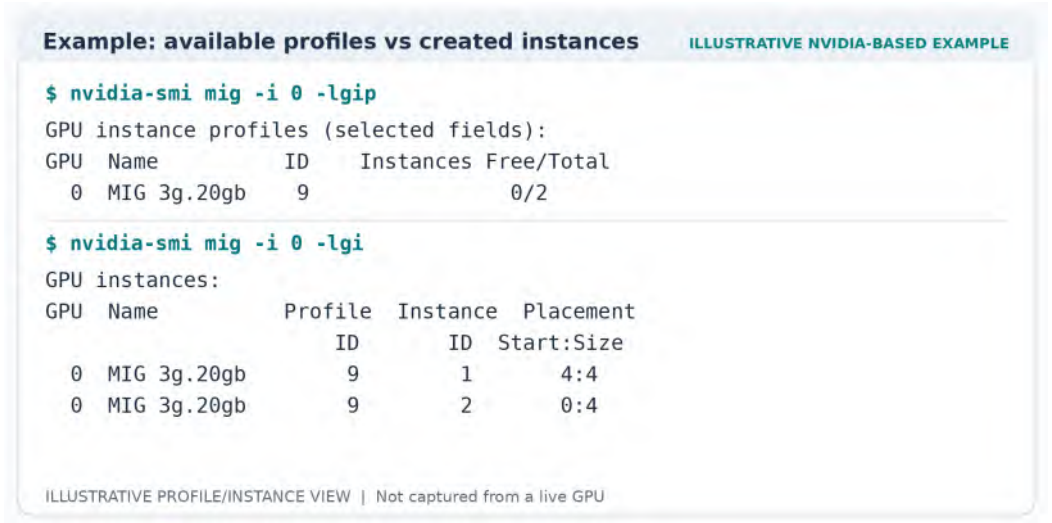


Figure 7.4: Illustrative MIG profile availability and configured GPU Instances

The first command lists supported GPU Instance profiles and reports currently free and total possible instances of each profile as `Free/Total`. The second lists configured GPU Instances. A supported profile does not mean a corresponding instance has been created.

Use the command help for the installed release when you need to confirm the available options:

```
nvidia-smi mig --help
```

Do not enable MIG merely to make an inspection command succeed. If MIG information is unavailable, record the result and check whether the GPU and software context support the requested inspection.

Interpreting what the commands prove

If `nvidia-smi` reports MIG as unavailable on the inspected GPUs, you cannot claim that MIG was configured on those devices. Successful enablement or instance-creation output taken from documentation should be treated as documentation evidence, not as a result produced on the inspected system.

Even a configured GPU Instance does not prove that an application has been assigned to it or validated.

Table 7.3 summarizes several observations and the next useful question. These are investigation prompts rather than guaranteed diagnoses.

Observation	Next question
MIG-related information is unavailable	Does this GPU and software context support the requested inspection?
Profiles are listed but no instances appear	Has a GPU Instance actually been created?
An instance is listed but the application cannot see a usable device	Is the instance correctly assigned or exposed to the application environment?
A vGPU VM does not initialize its assigned device	Is the host/guest vGPU software combination and assignment supported?
A profile from another GPU is not available here	Does the current GPU offer a compatible profile with suitable compute and memory resources?
A field is shown as `N/A`	Is that information supported in this deployment and at this layer?

Table 7.3: Interpreting MIG and vGPU inspection evidence

Review questions

The following practice questions test the distinctions developed in this chapter through comparisons and practical scenarios.

Question 1

A platform team needs several GPU-enabled VMs to share supported NVIDIA GPU resources through the hypervisor. Which mechanism is the most directly relevant starting point?

A. Assign a complete physical GPU to each VM through passthrough
B. Present supported GPU resources to the VMs through NVIDIA vGPU
C. Create MIG profiles without providing a VM virtualization path
D. Expose only virtual CPUs and leave GPU access outside the guests

Question 2

Which statement best distinguishes MIG from NVIDIA vGPU?

A. MIG is a guest-side virtualization layer, while vGPU partitions GPU hardware resources.

B. MIG and vGPU are both time-slicing methods that share the same underlying resource boundary.

C. MIG partitions supported GPU hardware resources, while vGPU virtualizes GPU access for guest VMs.

D. MIG must be enabled on the physical GPU before any vGPU can be assigned to a virtual machine.

Question 3

What does a MIG profile represent?

A. A supported GPU Instance resource shape for a particular MIG-capable GPU

B. A memory amount chosen freely by the administrator for any GPU

C. A universal partition size that is identical across MIG-capable GPU models

D. A software license class that controls whether a VM can access the GPU

Question 4

A read-only `nvidia-smi` check reports MIG information as `N/A`. What is the safest interpretation?

A. MIG is enabled, but every configured GPU Instance is currently unusable on the device.

B. The GPU requires a reset before any additional information can be collected from the device.

C. The system is using vGPU instead of the NVIDIA driver, so MIG fields are intentionally hidden.

D. MIG information is unavailable in this context, so support and configuration need further checking.

Question 5

Why should the maximum MIG instance count listed for one GPU not be treated as a universal rule?

A. Administrators can set any maximum instance count at deployment time.

B. Supported profiles and instance counts depend on the GPU model and configuration.

C. MIG profiles define memory only and therefore have no compute limits.

D. The guest operating system decides how many hardware partitions the GPU can expose.

Question 6

A vGPU VM cannot initialize its assigned virtual GPU even though the host sees the physical device. Which path should be checked next?

A. The application framework only, because host visibility proves virtualization is healthy

B. The physical GPU temperature only, because thermal state controls guest assignment

C. The vGPU assignment, Virtual GPU Manager, and compatible guest-driver path

D. The profile count on another MIG-capable GPU in the same data center

Question 7

The command that lists MIG profiles returns several supported shapes, but the command that lists GPU Instances returns none. What does this show?

A. The current context exposes supported profile shapes, but no GPU Instances are currently listed.

B. Every profile shown by the driver has already been assigned to an application on the system.

C. The driver automatically created one GPU Instance for each supported profile that it displayed.

D. The host is using passthrough instead of MIG, so GPU Instance information cannot be reported.

Question 8

On a supported platform, how can MIG and vGPU relate to each other?

A. MIG replaces the vGPU guest driver whenever both are present.

B. MIG and vGPU cannot be combined because they operate at the same layer.

C. A vGPU can be created only when it uses the entire physical GPU.

D. A MIG GPU Instance can back a vGPU that is presented to a VM.

Before checking the answers, identify which concept each question is testing: virtualization delivery, MIG partitioning, profile interpretation, support, or inspection evidence.

Answers and explanations

1. **B:** NVIDIA vGPU is the mechanism designed to present supported GPU resources as virtual devices to guest VMs through the vGPU software stack.
2. **C:** MIG partitions supported GPU hardware resources, while vGPU provides the virtualization path that presents supported GPU resources to VMs.

3. **A:** A MIG profile is a supported GPU Instance resource shape for a specific MIG-capable GPU. Its exact characteristics depend on the GPU and software release.
4. **D:** `N/A` means the requested information is unavailable in that context. It does not, by itself, prove a hardware failure or a configured MIG environment.
5. B: MIG capabilities are GPU- and configuration-specific. Profile names, capacities, placement rules, and maximum instance counts must be verified for the actual platform.
6. **C:** vGPU troubleshooting follows the path from the host-side Virtual GPU Manager and assignment into the compatible guest driver and application environment.
7. **A:** Listing profiles shows resource shapes exposed by the current context. It does not prove that any GPU Instance has been created.
8. **D:** On supported platforms, a MIG GPU Instance can back a vGPU. This is why vGPU and MIG should be understood as mechanisms at different layers rather than mutually exclusive choices.

Summary

In this chapter, you examined the key considerations for virtualizing accelerated infrastructure rather than treating GPU sharing as a single technology.

You first considered why organizations share GPUs and separated whole-device assignment, time-shared execution, virtual GPU delivery, and hardware partitioning. You then followed the vGPU path from the hypervisor and Virtual GPU Manager into the guest environment.

For MIG, you identified GPU Instances as resource partitions and interpreted profiles as GPU-specific compute and memory allocations.

The comparison showed that vGPU and MIG operate at different layers and can be combined on supported platforms. The inspection section then distinguished what read-only `nvidia-smi` evidence can establish from what still requires support or application-level verification.

Chapter 8 moves from allocating accelerator resources to mapping NVIDIA software across the AI lifecycle, including data preparation, training, optimization, and inference or deployment responsibilities.

8

Mapping NVIDIA Software to the AI Lifecycle

Chapter 7 examined how accelerated GPU resources can be virtualized or partitioned for different workloads. Once the hardware and software foundation is available, the next question is what software the workload needs at each point in the AI lifecycle. Data preparation, model development, optimization, and inference or deployment are related activities, but they are not the same job.

Consider a team taking a trained model from a development environment into a customer-facing service. The model runs correctly in a notebook, but the team still needs to decide how to package it, how to optimize its execution, how applications will send requests, and which software versions it can maintain. A model-serving product might address the request-handling problem but not fix a slow data-preparation job or an unsupported GPU configuration.

This chapter uses the AI lifecycle as the organizing model. We will follow the workflow from data processing through inference and deployment and identify representative NVIDIA technologies for each stage. The goal is role recognition rather than memorizing a product catalog.

In this chapter, we will cover the following topics:

- The four-stage AI workflow
- Frameworks and accelerated libraries
- Data preparation and development tools
- Optimization and inference tools
- NVIDIA AI Enterprise

- Selected vertical solutions and NVIDIA GPU Cloud (NGC)
- Understanding the AI factory concept
- Review questions

By the end of this chapter, you will be able to map software responsibilities to the AI lifecycle, distinguish frameworks and accelerated libraries from optimization and serving tools, recognize the high-level roles of representative NVIDIA software, and choose a relevant software category without assuming that every product belongs in every deployment.

Technical requirements

Most of this chapter is conceptual. No GPU, software installation, or live deployment is required. The examples assume the following background:

- The AI workload, training, and inference vocabulary established in *Chapter 2*
- The NVIDIA GPU software-stack concepts established in *Chapter 6*
- The resource-allocation and virtualization concepts established in *Chapter 7*
- Basic familiarity with applications, containers, and operating environments

Product packaging, component names, support matrices, and deployment combinations can change over time. Treat them as version-sensitive details and use current NVIDIA documentation when making a real deployment decision.

Note

Exam focus: Start with the lifecycle responsibility, then identify the software category or product that matches it. The objective is to recognize what the major components do and where they fit.

The four-stage AI workflow

An AI workflow typically progresses through four stages, from preparing data to delivering a working AI application.

Teams may participate in different stages of the AI lifecycle depending on their responsibilities and requirements. A team deploying a pretrained model may begin with selection and evaluation rather than training. Another team may own data preparation while a separate group owns the model. The lifecycle still helps because it identifies what must pass between those groups.

The workflow is organized into four stages: data processing, model training, model optimization, and inference or deployment. Data processing prepares usable inputs. Training

creates or updates model parameters. Optimization prepares a trained model for efficient execution. Inference uses the trained model for new inputs, while deployment makes that capability available in an operating environment.

Inference and deployment are related but not identical. A runtime can execute a model without providing a complete application-facing service. Keeping that distinction visible will help later when we separate TensorRT, NVIDIA Dynamo-Triton (formerly NVIDIA Triton Inference Server), and NVIDIA NIM.

Figure 8.1 provides the chapter's primary visual anchor. It shows the four lifecycle stages from left to right and makes the output of each stage explicit. Product names are intentionally kept out of this first figure so that the workflow remains the organizing model.

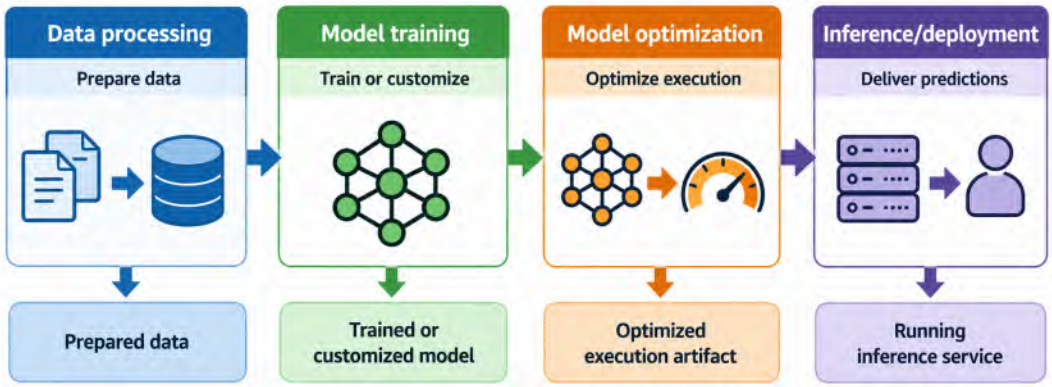


Figure 8.1: Four-stage AI workflow and stage outputs

For each stage, identify what is input, what work is required, and what is the output. The outputs may be artifacts (stored, versioned items such as model files or optimized engines) or a running service, depending on the stage. We will begin with frameworks and accelerated libraries, then move to data preparation, because faster model execution has limited value when the input data is incomplete, inconsistent, or unavailable when needed.

Frameworks and accelerated libraries

A model-development framework gives developers higher-level ways to define a model, train it, evaluate it, and save its state. PyTorch and TensorFlow are familiar examples. NVIDIA-enabled environments can combine such frameworks with CUDA and optimized libraries, but the exact integration depends on the framework build and the selected packages.

The infrastructure question is therefore not whether a familiar framework name appears on the host. It is whether the application uses the intended build and whether that build has a

supported path to the GPU resources assigned to it. *Chapter 6* established the underlying driver, CUDA, and library relationships.

Placing accelerated libraries underneath the workflow

CUDA-X is NVIDIA's collection of accelerated libraries built on CUDA for AI and other compute-intensive work. It is an umbrella term for capabilities, not another driver or a single training application. The CUDA-X libraries overview (`https://developer.nvidia.com/cuda/cuda-x-libraries`) organizes those capabilities by function.

Two examples connect directly to the earlier chapters. The **NVIDIA CUDA Deep Neural Network** (**cuDNN**) library provides optimized operations used in neural networks. **NVIDIA Collective Communications Library** (**NCCL**) provides communication operations for supported multi-GPU workloads. cuDNN's function is described in the cuDNN documentation (`https://docs.nvidia.com/deeplearning/cudnn/latest/`); NCCL's communication role is described in its user guide (`https://docs.nvidia.com/deeplearning/nccl/user-guide/docs/overview.html`).

A developer may use these capabilities through a framework rather than call them directly. That does not mean every framework operation uses every installed library. When investigating performance, identify the application operation and the execution path it follows before changing unrelated components.

Table 8.1 summarizes the main development components and their roles. It is not an installation order, and a successful check at one layer does not validate the entire workflow.

Component	Role in model development	Evidence that matches the role
CUDA foundation	Enables supported GPU programming and execution	A compatible environment and a working GPU workload
cuDNN	Supplies optimized neural-network operations	A supported library combination and a workload that uses those operations
NCCL	Coordinates supported multi-GPU communication	A successful communication test in the intended topology
ML framework	Organizes model definition, training, evaluation, and execution	The intended framework build and a representative model run

Component	Role in model development	Evidence that matches the role
NeMo Framework	Provides tools for building and customizing generative models	Supported model workflow, configuration, and framework dependencies

Table 8.1: Connecting frameworks and accelerated libraries to their roles

Some packages provide dependencies together, and others require a separately managed environment. Use the selected framework's support information to choose a compatible combination.

For this chapter, the important boundary is simple: frameworks provide higher-level development and execution structures, while accelerated libraries provide optimized capabilities underneath supported software paths. The next section applies the same role-first approach to data preparation and model development.

Data preparation and development tools

Data preparation includes turning source data into inputs the model workflow can use. For a tabular workload, that might involve filtering records, joining datasets, and calculating features. For a language-model workflow, it might involve removing duplicate documents and applying quality filters before customization.

The data type and operation should guide the choice. A library that processes tabular data and a tool designed for large-scale content curation address related but different tasks. Neither can decide the business meaning of a field or the organization's rules for acceptable training data.

Using RAPIDS for data processing and analytics

RAPIDS provides GPU-accelerated libraries for data science workflows. Its CUDA-based DataFrame library, cuDF, accelerates operations on tabular data, while its CUDA-based machine learning library, cuML, offers GPU-accelerated implementations of selected machine learning algorithms. RAPIDS therefore extends beyond preparation, even though data processing is a useful entry point for this chapter. The RAPIDS documentation (`https://docs.rapids.ai/`) and cuDF documentation (`https://docs.rapids.ai/api/cudf/stable/`) describe the supported interfaces and execution options.

Suppose a team joins transaction records with customer attributes before each training run. The key question is whether the supported operations in that preparation step can benefit from acceleration. Record the time spent reading data, transforming it, and writing the result before attributing the whole delay to computation.

Acceleration only helps the part of the workflow that can use it. A faster transformation does not automatically remove storage, data-movement, or source-data bottlenecks, so the complete workflow still needs to be evaluated.

Using NeMo Curator for training-data curation

NeMo Curator supports scalable data curation, including filtering and deduplication workflows for supported text, image, audio, and video data. It focuses on preparation and quality control, rather than serving inference requests from applications. The NeMo Curator overview (`https://docs.nvidia.com/nemo/curator/about`) provides the current scope and component guidance.

For example, a team customizing a language model may need to remove repeated documents from a large collection. The team needs more than faster model execution: a repeatable preparation process, a record of filtering decisions, and a dataset that can be used in a later model run.

Using NeMo Framework for model development

NVIDIA NeMo covers a broader set of tools than model training alone. Within that family, **NeMo Framework** supports building and customizing generative models, while **NeMo Curator** addresses data preparation. Other NeMo components have their own responsibilities. Use the NeMo documentation hub (`https://docs.nvidia.com/nemo/`) to identify the particular component, then follow its release-specific instructions.

For an infrastructure team, this distinction turns a broad request into a deployable requirement. A request to install NeMo is incomplete until the team names the component and workflow. Preparing a document collection, fine-tuning a model, and deploying a supported inference service can require different artifacts and operating environments even when the project uses the same product-family name.

Once a model has been selected or trained, the focus shifts to running it efficiently for its intended purpose. This involves optimizing how the model executes rather than changing what it has learned.

Optimization and inference tools

The following sections distinguish optimizing a model's execution from serving inference requests to applications.

Consider a team that already has an acceptable image-classification model. It wants to reduce response time on its chosen GPU. Before replacing the training framework, the team should examine the inference path, the supported model operations, and the optimization options available for that path.

Understanding TensorRT's build and runtime roles

TensorRT provides inference optimization and runtime capabilities. Its build process prepares an execution engine for a supported model and target environment; its runtime executes that engine. The engine is sometimes called a **plan**. It represents an execution artifact. NVIDIA explains this separation in How TensorRT works (`https://docs.nvidia.com/deeplearning/tensorrt/latest/architecture/how-trt-works.html`).

The team must retain both the source model and the information needed to reproduce the optimized artifact. Record the model version, build configuration, TensorRT version, and intended platform. Keeping only the generated engine makes later changes harder to investigate.

TensorRT is one option for inference optimization, not a required step for every model. The appropriate runtime depends on model support, the target platform, and the software combination in use.

Optimization establishes how a model can execute. Serving determines how applications reach that execution path and how incoming requests are handled.

A model that runs inside a script is not yet a complete network service. Clients need a request interface, the service needs to load the intended model, and the operator needs evidence that it is ready to handle work. These requirements explain why serving software is a separate category.

Using NVIDIA Dynamo-Triton for model serving

NVIDIA Dynamo-Triton provides model-serving capabilities through supported execution backends. A **backend** is the component that executes a particular model format or runtime. Dynamo-Triton exposes **application programming interfaces** (**APIs**) for inference and manages model execution without requiring every model to be converted to the same engine. Its supported features and backends are documented in the Triton user guide (`https://docs.nvidia.com/deeplearning/triton-inference-server/user-guide/docs/index.html`).

For example, a team may need to serve several supported model formats through a consistent service interface. Dynamo-Triton is relevant to that request. The team must still provide the model artifacts and configuration, select a container with the required backends, and validate the behavior of each model.

Using NIM for a packaged inference capability

NVIDIA NIM provides packaged inference microservices. A **microservice** exposes a focused capability through an API; in this case, that capability is inference. Packaging can reduce the work of assembling the model-serving environment, but it does not remove requirements for supported models, GPUs, software, or deployment settings.

NIM should not be described as a universal wrapper for any model, nor as a mandatory stage after Dynamo-Triton. Supported models, runtimes, hardware requirements, and maintenance lifecycles vary by NIM offering and release. Treat NIM as a packaged inference option and verify the selected offering in the current NVIDIA NIM documentation (https://docs.nvidia.com/nim/).

The choice depends on which deployment tasks the team wants to manage. One team may need direct control over a serving configuration and its model backends. Another may prefer a supported packaged inference path for its selected model. Both still need an application test, a resource plan, and a way to record the deployed version.

Choosing between responsibilities rather than names

Table 8.2 compares TensorRT, Dynamo-Triton, and NIM. Use the final column to identify what a deployment still needs to verify.

Technology	Primary question it addresses	What the team still needs to verify
TensorRT	How can a supported model execute efficiently on the target platform?	Model support, engine compatibility, quality, and measured performance
NVIDIA Dynamo-Triton	How can clients request inference from models using supported backends?	Included backends, model configuration, request behavior, and deployment controls
NVIDIA NIM	Is there a suitable packaged inference microservice for this model and environment?	Model and engine support, resource requirements, selected offering, and maintenance scope

Table 8.2: Separating inference optimization, model serving, and packaged microservices

Figure 8.2 maps representative software to the lifecycle responsibility it supports.

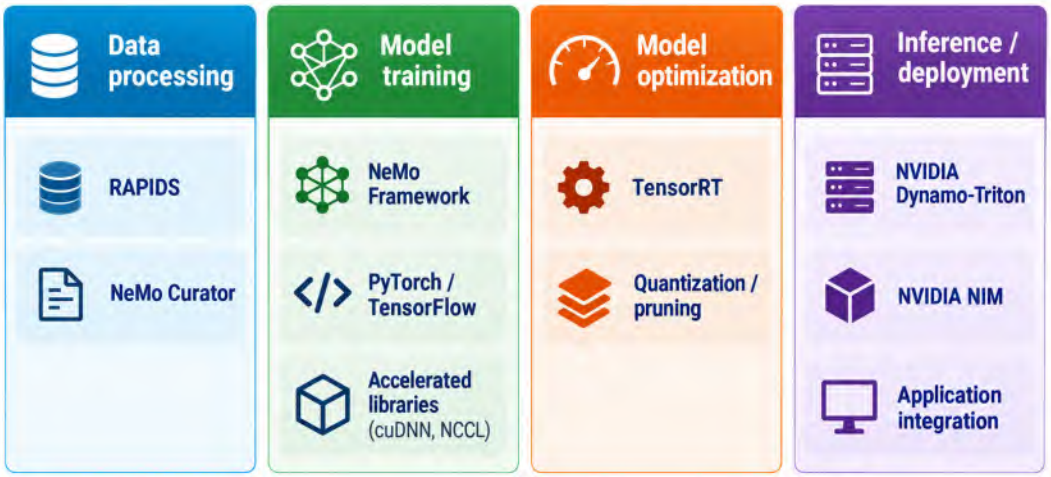


Figure 8.2: Representative NVIDIA software mapped to lifecycle responsibilities

Choosing a serving component also means checking the software environment in which it will run. NVIDIA AI Enterprise provides an enterprise software and support context for those choices.

NVIDIA AI Enterprise

NVIDIA AI Enterprise is a software platform for developing, deploying, and managing AI applications, not a replacement operating system. NVIDIA organizes the platform into the Application Layer and Infrastructure Layer, with separate release branches and lifecycles. This separation lets teams plan application and infrastructure updates independently while still checking cross-layer compatibility. Use the current NVIDIA AI Enterprise documentation (https://docs.nvidia.com/ai-enterprise/) as the source of truth for the components and branches available in a real deployment.

For this chapter, place application frameworks, libraries, **software development kits (SDKs)**, and inference services within the application-development context. The infrastructure-management context connects to the GPU foundation, resource management, and operational tooling addressed elsewhere in the book. A deployment selects only the components it needs and verifies their versions, compatibility, and applicable support entitlements.

Selected vertical solutions and NVIDIA GPU Cloud (NGC)

Beyond tools associated with individual AI lifecycle stages, NVIDIA provides software distribution services and solutions for specific application areas. This section examines NGC first, followed by Clara and Merlin as examples of domain-specific software.

Using NGC as a catalog and distribution source

NGC provides access to NVIDIA software and related artifacts. Within that wider platform, the **NGC Catalog** helps users discover curated containers, models, charts, and other resources. The **NGC Private Registry** provides controlled storage and distribution for an organization's own containers, models, and resources. The NGC documentation (`https://docs.nvidia.com/ngc/latest/ngc-user-guide.html`) distinguishes these services.

Keeping distribution separate from support

A public catalog entry does not establish that an organization has every entitlement associated with the software. Likewise, putting an artifact in a private registry does not make its full application stack supported. Record distribution, licensing, and support as separate parts of the decision.

An organization can use the NGC Private Registry (`https://docs.nvidia.com/ngc/latest/ngc-private-registry-user-guide.html`) for controlled artifact access, but it still needs to define who approves versions and how applications reference them. The registry supplies a distribution function; the team establishes the release and change process.

NGC helps teams discover and distribute software artifacts. Domain-specific software answers a different question: which capabilities are organized around a particular industry or application pattern.

Recognizing selected domain-specific software

Some software is organized around an industry or application pattern rather than a single lifecycle stage. Two examples from the wider NVIDIA portfolio are useful here: Clara and Merlin. These families bring together components for an application area, so the specific task still determines what the team should use.

Clara groups healthcare and life-sciences models, tools, and frameworks. A medical-imaging workflow and a biological-research workflow may require different components within that wider family. Start with the task and consult the Clara documentation (`https://docs.nvidia.com/clara/`) for the appropriate component. A healthcare label alone does not establish clinical suitability, authorization, or regulatory compliance.

Merlin addresses recommendation-system workflows, spanning preparation, training, and inference-related components. Its developer overview (`https://developer.nvidia.com/merlin`) describes that scope. For a recommendation service, identify whether the immediate requirement concerns feature preparation, model development, or serving before choosing a component.

To see how these tools fit alongside the infrastructure they require, consider the AI factory concept.

Understanding the AI factory concept

An AI factory is a full-stack environment that brings accelerated computing, networking, storage, software, and data pipelines together to produce and operate AI capabilities at scale. NVIDIA publishes reference architectures and related design guidance for enterprise AI factories, but an AI factory is not a single product that an administrator installs. The current NVIDIA Enterprise Reference Architecture documentation (https://docs.nvidia.com/enterprise-reference-architectures/white-paper/latest/) provides that broader infrastructure context.

Think of a conventional factory: raw materials enter, machines and defined processes transform them, and finished products emerge. In an AI factory, curated data is an input, accelerated computing and software run AI workflows, and trained models or deployed inference services are the outputs.

The earlier chapters supply its foundation: facility capacity, compute, storage, networks, GPU software, and resource allocation. This chapter adds the application-software responsibilities. Together, they explain how data and model artifacts become usable services rather than a collection of individually installed products.

Figure 8.3 shows how AI applications, NVIDIA AI Enterprise, and accelerated infrastructure fit together within an AI factory.

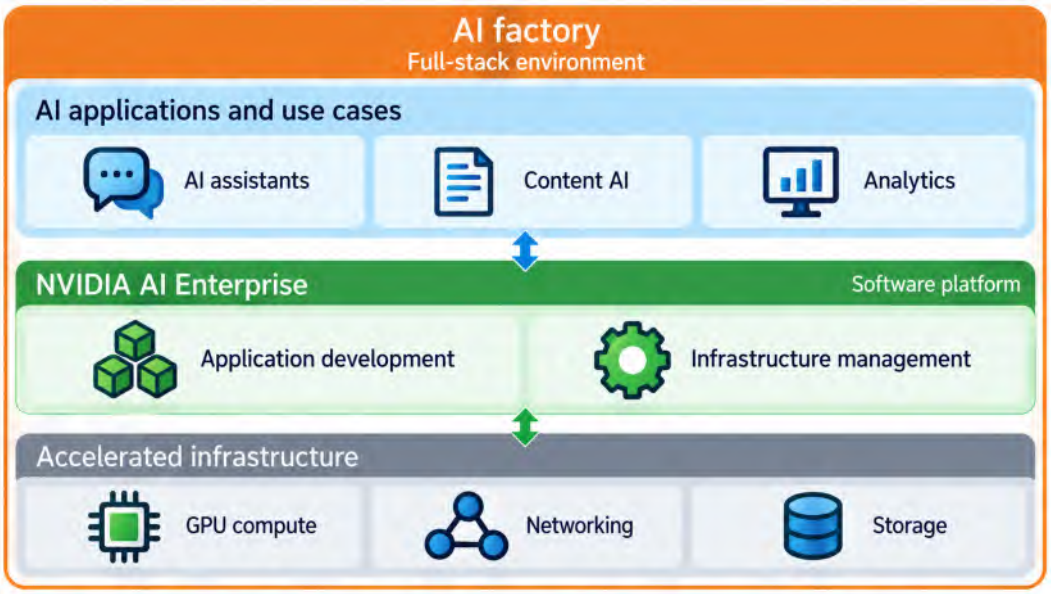


Figure 8.3: NVIDIA AI Enterprise and AI factory context from applications to infrastructure

Connecting the architecture to an outcome

Consider an organization that maintains an existing model and provides inference to several internal applications. The organization may serve requests far more often than it retrains the model. The factory concept does not require it to train a new model for each request or build every model from scratch.

Instead, describe the outcome the platform must support: what data enters, which model and software versions are approved, which services consume the results, and what availability and performance those services require. These requirements connect the architecture to the workload it must support.

Review questions

The following practice questions test lifecycle and software-role recognition. They do not reproduce live certification questions. Choose the answer that best matches the responsibility described in each scenario.

Question 1

Which sequence matches the four-stage AI workflow used in this chapter?

A. Model training, data processing, inference/deployment, model optimization

B. Data processing, model training, model optimization, inference/deployment

C. Data processing, inference/deployment, model training, model optimization

D. Model optimization, data processing, model training, inference/deployment

Question 2

A developer uses a machine-learning framework on an NVIDIA GPU. Which statement best describes the relationship between the framework and accelerated libraries?

A. The framework replaces CUDA and every accelerated library in the environment.

B. Every framework operation must call every installed CUDA-X library.

C. The framework can use supported accelerated libraries beneath higher-level interfaces.

D. Accelerated libraries are used only after the model has been deployed as a service.

Question 3

A team needs to remove duplicate and low-quality documents before customizing a generative model. Which component is the most directly relevant starting point?

A. NeMo Curator

B. NVIDIA Dynamo-Triton

C. TensorRT

D. NGC Private Registry

Question 4

A trained model already meets its quality target. The team wants to prepare it for more efficient inference execution on a supported NVIDIA platform. Which technology is the most directly relevant candidate?

A. RAPIDS

B. NeMo Curator

C. NGC Catalog

D. TensorRT

Question 5

Which statement best distinguishes NVIDIA Dynamo-Triton from NVIDIA NIM?

A. Dynamo-Triton trains models for deployment, while NIM manages serving backends.

B. Dynamo-Triton is an engine-optimization toolkit, while NIM distributes software artifacts.

C. Dynamo-Triton provides model serving through supported backends, while NIM packages supported inference microservices.

D. Dynamo-Triton and NIM are interchangeable names for one required inference service.

Question 6

What is the most useful high-level description of NVIDIA AI Enterprise in this chapter?

A. An enterprise-supported platform for AI application and infrastructure software

B. A replacement operating system that every NVIDIA GPU server must install

C. A single inference runtime that replaces frameworks, TensorRT, and Dynamo-Triton

D. A hardware partitioning mechanism for dividing one GPU among virtual machines

Question 7

Which statement correctly describes the role of NGC?

A. It replaces the model runtime after a container is downloaded.

B. It guarantees that every catalog artifact is supported on every NVIDIA GPU.

C. It automatically trains and deploys any model stored in the catalog.

D. It distributes software artifacts; execution requires an appropriate runtime or service.

Question 8

A scenario lists several NVIDIA products and asks which one to investigate first. What is the most reliable starting point?

A. Choose the product with the newest release number.

B. Identify the lifecycle responsibility the workload needs to satisfy.

C. Choose the product family with the largest number of components.

D. Start with monitoring software regardless of the lifecycle stage.

Before checking the answers, identify the lifecycle stage and responsibility being tested. That habit remains useful even when product packaging changes.

Answers and explanations

1. **B:** The four-stage workflow is data processing, model training, model optimization, and inference/deployment. It is a responsibility map rather than a requirement that every team perform every stage.
2. **C:** A framework provides higher-level model-development interfaces and can use supported accelerated libraries beneath them. The exact backend and library paths depend on the software combination.

3. **A:** NeMo Curator supports scalable data curation. The other choices address serving, optimization, or artifact distribution rather than preparation of the source collection.
4. **D:** TensorRT is an inference-optimization and runtime technology for supported models and target environments. The requirement concerns execution of an already trained model, not data preparation or artifact discovery.
5. **C:** Dynamo-Triton provides model-serving capabilities through supported backends, while NIM provides packaged inference microservices for supported models and environments. They address related but distinct deployment responsibilities.
6. **A:** NVIDIA AI Enterprise is an enterprise AI software platform that provides a broader supported and maintained software context. It is not an operating system, a single inference runtime, or a GPU-partitioning mechanism.
7. **D:** NGC helps users discover and distribute artifacts. A downloaded or stored artifact still needs the appropriate framework, runtime, or serving process to execute its workload.
8. **B:** Start with the responsibility. Once the lifecycle stage and required work are clear, the relevant software category or representative product becomes easier to identify and verify.

Summary

In this chapter, you used the four-stage AI workflow to organize NVIDIA software by responsibility. Data processing prepares usable inputs, model training creates or updates model parameters, model optimization prepares trained models for efficient execution, and inference or deployment makes model outputs available to applications.

You identified frameworks and accelerated libraries used in model development, then distinguished RAPIDS, NeMo Curator, and NeMo Framework by their roles. You also separated TensorRT's optimization/runtime role from Dynamo-Triton's serving role and NIM's packaged inference approach.

You considered NGC for artifact distribution, NVIDIA AI Enterprise for enterprise software and maintenance, and Clara and Merlin as examples of domain-specific software. The AI factory concept connected these choices to the wider infrastructure environment without turning it into a single product.

When selecting NVIDIA software, start with the lifecycle responsibility and then choose a software category or representative product. *Chapter 9* moves from software-role recognition to monitoring and managing GPU infrastructure, where the focus shifts to observing the health, utilization, and behavior of the accelerated environment.

9

Monitoring and Managing GPU Infrastructure

Chapter 8 mapped NVIDIA software to the AI lifecycle and separated the roles of development, optimization, inference, and platform software. Once those components are running, the operational question changes: are the GPUs healthy, is the workload using them as expected, and is a developing condition visible before it becomes an incident?

That sounds like a monitoring problem, but monitoring is more than collecting every number a GPU can expose. A temperature reading is useful only when the operator knows what device produced it, what workload was active, how the value compares with a meaningful threshold or baseline, and what action should follow. The same principle applies to utilization, memory utilization, power, error counters, and process information.

This chapter moves from monitoring goals to immediate node inspection, structured GPU health and diagnostics, continuous telemetry, and broader infrastructure management. The emphasis is on choosing the right evidence and the right tool for the responsibility, not on building a general-purpose observability platform.

In this chapter, we will cover the following topics:

- Defining useful GPU health, utilization, memory, thermal, power, and error measures
- Using the **NVIDIA System Management Interface** (**nvidia-smi**) for immediate node-level inspection
- Understanding **NVIDIA Data Center GPU Manager** (**DCGM**) telemetry, health monitoring, and diagnostics
- Using DCGM Exporter, Prometheus, and Grafana for continuous monitoring
- Positioning **Base Command Manager** (**BCM**) at the infrastructure-management layer

- Choosing the right monitoring or management tool for the task
- Applying simple first-response reasoning to GPU symptoms

By the end of this chapter, you will be able to interpret core GPU monitoring measures, use `nvidia-smi` for immediate inspection, explain the difference between DCGM telemetry, passive health monitoring, and active diagnostics, place DCGM Exporter, Prometheus, Grafana, and BCM at the correct operational layer, and choose a sensible first response from the evidence available.

Technical requirements

This chapter can be completed as a documentation and analysis exercise. If you have access to an authorized NVIDIA GPU test host, you can also run the small inspection commands shown in the chapter. You need the following:

- The software-stack and GPU-allocation concepts from *Chapters 6 through 8*
- A Linux shell on an NVIDIA GPU host if you want to run the local inspection commands
- DCGM and `dcgmi` on a supported test host if you want to inspect DCGM discovery, health, or diagnostics
- A browser for current NVIDIA DCGM, DCGM Exporter, Prometheus, Grafana, and BCM documentation

Commands in this chapter are intentionally small and evidence-focused. Run hardware-dependent commands only in an environment where you are authorized to inspect or test the GPUs. When a command or feature varies by release, verify the syntax against the documentation for the installed version.

> **Note**
>
> **Exam focus**: This chapter concentrates on AI Operations. In a scenario, identify the operational responsibility before choosing a tool. Immediate inspection, structured GPU telemetry, retained time-series data, visualization, diagnostics, and infrastructure management are related responsibilities, but they are not interchangeable.

Monitoring goals and GPU measures

An operator usually starts with a symptom, not a metric. A training run finishes late. An inference service becomes inconsistent. One node behaves differently from its peers. The first job is to turn that signal into a time window and a set of observations that can be tested.

A local command can answer an immediate question quickly: Which GPU is present? Is the driver responding? How much memory is allocated? Is a process using the device? What are the current temperature and power readings? Those answers are valuable, but they describe a moment. They do not automatically show what happened ten minutes earlier or whether the same condition has been developing across several nodes.

Continuous monitoring adds responsibilities that a local snapshot cannot provide. Metrics need to be exposed, collected over time, retained, queried, and visualized. Infrastructure management is broader again: a management platform can provision and operate cluster resources while still consuming telemetry from specialist monitoring components.

Table 9.1 maps these responsibilities to operational questions and representative components.

Operational responsibility	**Question it answers**	**Representative component**
Immediate inspection	What is the GPU or node reporting right now?	`nvidia-smi`
GPU telemetry and health	What structured device evidence is available, and do watched conditions indicate a problem?	DCGM
Metric exposure	How can selected GPU metrics be made available for collection by a monitoring system?	DCGM Exporter
Collection, retention, and rule evaluation	What did the metric do over time, and does it meet a rule condition?	Prometheus
Visualization	How can operators query and compare retained telemetry in dashboards?	Grafana
Infrastructure management	How are cluster nodes, software, workloads, and monitoring managed as an operating platform?	Base Command Manager

Table 9.1: Separating monitoring and management responsibilities

These components can coexist. An operator may use `nvidia-smi` while investigating a node that is already covered by DCGM and Prometheus. The local command is not made obsolete by the cluster monitoring system; it simply answers a different question.

Figure 9.1 separates four operational responsibilities: immediate node inspection, structured GPU telemetry and health, retained monitoring and visualization, and broader infrastructure management. The sequence does not mean that one responsibility replaces another as the environment grows.

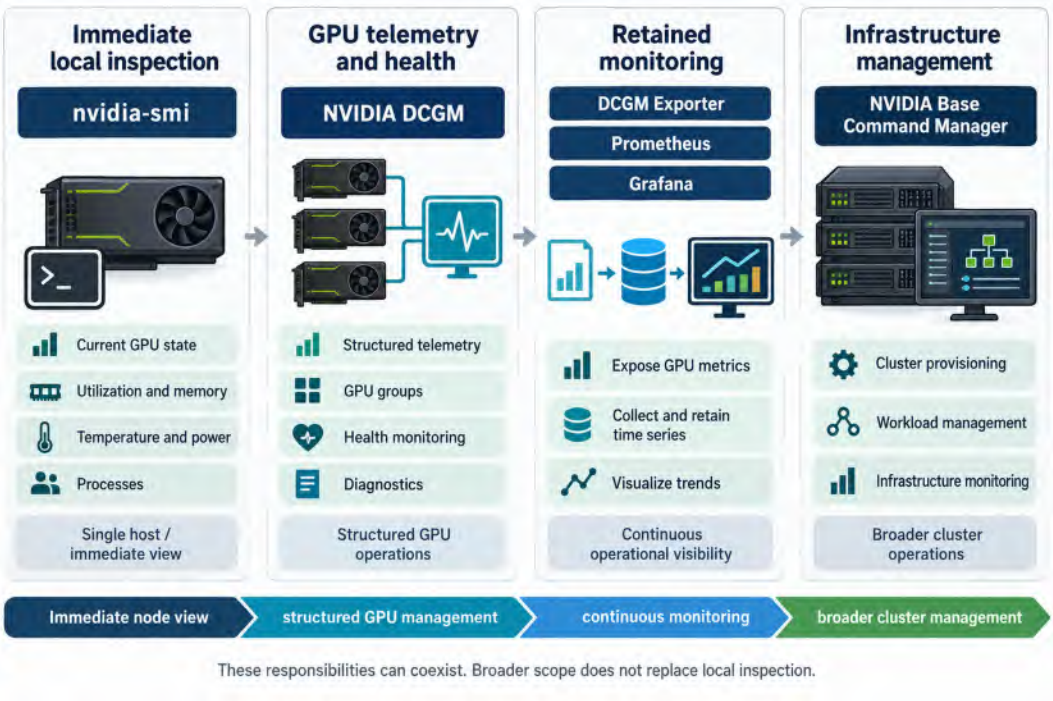


Figure 9.1: Node inspection and cluster monitoring tool scopes

With the tool scopes separated, the next step is to choose GPU measures that answer specific operational questions.

Choosing metrics that answer operational questions

A monitoring system becomes noisy when every available field is treated as equally important. Start with the workload and the failure modes that matter. A distributed training cluster may prioritize sustained execution, device memory pressure, thermal behavior, link errors, and whether one GPU falls behind its peers. An interactive inference service may emphasize response latency, memory headroom, power and thermal stability, and process or container identity.

The most useful view combines device telemetry with workload context. A utilization number without a job, process, or expected operating state is often ambiguous. Low activity can represent idle capacity, a blocked workload, slow data delivery, synchronization delays, an application issue, or a device problem.

Reading GPU execution utilization

GPU utilization is often the first metric people look at, but it is easy to overinterpret. The value reported through NVIDIA management interfaces represents activity during a sampling period. It does not directly indicate that a GPU is delivering a particular percentage of its theoretical performance. A value near zero can be correct when the device has no work to execute, and a high value does not by itself prove that the complete application is meeting its objective.

Use utilization together with the expected workload state. If a scheduled training job is supposed to be actively computing and utilization remains materially below its established baseline, investigate whether the process is running, whether data is arriving, whether distributed communication is waiting, and whether the allocation is larger than the job can use.

Separating memory capacity from memory activity

GPU memory usage answers a capacity question: how much device memory is allocated or occupied. Memory activity answers a different question: whether the device memory subsystem is being read or written during the sample period. A model can remain resident in GPU memory while execution is temporarily quiet, so substantial memory use and low execution utilization are not contradictory.

This distinction matters when diagnosing out-of-memory risk as well. A rising allocation trend can indicate shrinking headroom even if current compute activity is moderate. Conversely, an empty device during a scheduled workload may point toward application startup, visibility, or placement problems rather than insufficient capacity.

Interpreting temperature, power, and clocks together

Temperature and power are most useful when interpreted with device-specific limits and workload state. Avoid a universal temperature alarm copied from another GPU generation. Devices report thermal and power information differently, and an operator should use the values and event conditions documented for the actual platform.

A temperature increase under sustained compute may be expected. The stronger operational signal is the relationship among temperature, power draw, clock behavior, slowdown or throttling indications, and the workload. If clocks fall while a device reports thermal or power constraints, the evidence is more actionable than a temperature number viewed by itself.

Using error and health evidence proportionally

Error counters need the same context. Correctable and uncorrectable memory events do not have the same meaning, and a single historical count is not equivalent to a recurring active fault. Record the type of event, whether the counter is volatile or persistent for the device, whether the value is increasing, and whether DCGM or the driver reports a health incident that changes the operational response.

Identifying the process and workload completes the picture. Knowing that GPU 2 is warm and busy is much more useful when the monitoring record also identifies the workload responsible for that period. In orchestrated environments, metric labels can help connect telemetry to a pod or other workload identity, but *Chapter 10* will cover how those workloads are actually scheduled and exposed to GPUs.

Table 9.2 summarizes the metric families used throughout the chapter.

Metric family	Operational question	Interpret with
Execution utilization	Was GPU compute active during the interval?	Expected workload state, peer GPUs, data delivery and communication paths
GPU memory	How much GPU memory capacity is allocated?	Model size, batch size, concurrency, headroom, allocation
Memory activity	Was device memory actively being accessed?	Execution phase and workload behavior
Temperature and power	Is the device operating within meaningful platform conditions?	Device-specific thresholds, clocks, slowdown events, workload
Clocks and constraint state	Is performance being limited by a reported thermal or power condition?	Temperature, power, platform policy, workload
Errors and health events	Is there evidence of a hardware, driver, fabric, or memory problem?	Severity, recurrence, trends, health and diagnostic results

Metric family	Operational question	Interpret with
Processes/ workload labels	Which workload is associated with the observed resource use?	Scheduler/orchestrator context and time window

Table 9.2: GPU metric families and the questions they help answer

With the metric questions clear, we can collect a local snapshot without building a monitoring stack first.

Using nvidia-smi for immediate inspection

Chapter 6 used `nvidia-smi` to verify GPU visibility and driver-level state. Here we use the same tool operationally: capture a quick baseline, identify the affected device, and decide whether the investigation needs retained telemetry or a different subsystem.

The default display is convenient for a human at a terminal because it combines identity, driver information, temperature, power, memory, utilization, and processes in one view. Its layout is not the best interface for maintainable automation, however. For scripts, use selective query output or an API intended for programmatic use.

The current command options and field definitions are documented in the NVIDIA System Management Interface reference (`https://docs.nvidia.com/deploy/nvidia-smi/index.html`).

Capturing a selective, script-friendly snapshot

For scripts, select the required fields instead of parsing the default table by screen position. The following concise example requests identity, utilization, memory, temperature, and power as CSV output:

```
nvidia-smi --query-gpu=index,uuid,name,utilization.gpu,memory.used,\
memory.total,temperature.gpu,power.draw --format=csv
```

Not every field is available on every GPU or platform, so automation should handle unsupported values rather than treating them as zero. Stable device identity also matters. Index numbers are convenient during a local session, but UUIDs or PCI bus identifiers are better when observations must be correlated over time.

Repeating a local observation without mistaking it for history

A short repeated view can help while reproducing an immediate symptom:

```
nvidia-smi -l 5
```

Here, `-l` enables repeated execution (loop mode), and `5` specifies the interval in seconds. The command displays GPU status every five seconds until stopped with *Ctrl+C*.

A repeated terminal view is still a local observation, not durable time-series retention. When the question becomes "what happened before I opened the terminal?" or "did this pattern occur across several nodes?", move to retained telemetry rather than stretching a local CLI beyond its role.

Turning the snapshot into a next action

A useful first-response note records more than the command output. Capture the hostname, timestamp, affected workload, GPU UUID or PCI identifier, current process list, and any recent change that could explain the symptom. Treat the values as evidence, not verdicts.

Table 9.3 shows how a GPU observation can guide further investigation, sometimes requiring additional tools or logs.

Observation	What it establishes	Useful next check
GPU absent from expected inventory	The device is not visible through the current driver view	Driver/device state, allocation, platform inventory, recent change
High memory usage, low GPU utilization	GPU memory is occupied, but compute activity is low during the sampling interval	Process identity, workload phase, input/communication waits
Sustained low utilization during expected active work	Compute is less active than expected	CPU and data paths, scheduler context, communication, application behavior
Rising temperature with reduced GPU clock speeds	Thermal or power constraints may be affecting execution	Cooling, airflow, thermal/power limits, clock limitation indicators

Observation	What it establishes	Useful next check
GPU error indicators or Xid events reported in driver logs	Driver or device reported an event	DCGM health evidence, logs, support guidance, controlled diagnostics
One GPU differs consistently from comparable peers	The anomaly may be device- or path-specific	Historical trend, peer comparison, fabric, path, and health checks

Table 9.3: Converting nvidia-smi observations into next checks

When the local evidence suggests that history, group-level health, or a structured diagnostic is needed, DCGM provides the next operational layer.

Understanding DCGM

DCGM provides libraries and tools for monitoring, managing, and validating supported NVIDIA GPUs and related hardware. Its value is not simply that it can see more than one GPU. DCGM gives operators a structured model for selecting entities, watching fields, evaluating health conditions, and running active diagnostics when the situation warrants them.

For current installation, supported entities, command syntax, and release-specific behavior, use the NVIDIA DCGM documentation (`https://docs.nvidia.com/datacenter/dcgm/latest/`)

Understanding entities, groups, fields, and watches

Start with four terms. An **entity** is something DCGM can identify, such as a supported GPU or another managed component. A **field** is a property, metric, counter, or event associated with an entity. A **group** is a reusable selection of entities. A **watch** instructs the DCGM host engine to periodically collect selected fields and retain their samples for a configured period.

The distinction between a DCGM group and a GPU allocation is important. Creating a group does not reserve a GPU for a job, create a MIG instance, or change what a container can see. It gives DCGM a reusable target for monitoring or management operations. *Chapter 7* remains the place where GPU allocation and sharing are defined.

Use discovery before assuming entity identifiers. On a supported host, the current DCGM documentation uses the following command to list detected GPUs and supported entities:

```
dcgmi discovery --list
```

A group can then be created as a reusable target for ongoing monitoring or a temporary investigation. Groups are runtime state owned by the host engine rather than durable on-disk configuration, so recreate them after a host-engine restart when needed. Exact entity-selection syntax depends on the DCGM release and the entities being targeted, so confirm the command against the installed version before using it.

Together, these concepts define what DCGM monitors, which measurements it collects, which devices are grouped, and how samples are retained.

Table 9.4 highlights their operational distinctions.

DCGM concept	What it represents	Important distinction
Entity	A device or managed object known to DCGM	Identifies a managed object; does not allocate it to a workload
Field	A metric, property, counter, or event	Represents one measurement or property, not a complete health assessment
Group	A reusable selection of entities	Selects entities without allocating or isolating them
Watch	Sampling state maintained by the running host engine	Retains samples temporarily, not as durable historical data

Table 9.4: Reading DCGM entities, groups, fields, and watches

Watches are especially easy to misunderstand. They preserve sampled state in the running host engine according to their configuration, but they are not a substitute for the organization's durable historical database. A host-engine restart can remove that runtime state. When long-term trend analysis is required, export the telemetry to a monitoring system designed to retain it.

Separating passive health monitoring from active diagnostics

DCGM health monitoring and DCGM diagnostics both help answer whether the platform is healthy, but they gather evidence in different ways. Passive health monitoring evaluates retained telemetry and events from configured watches. Active diagnostics run tests against the hardware and software environment.

This distinction changes how safely each can be used. Passive monitoring is designed to coexist with ordinary operation. Active diagnostics can consume GPU, CPU, memory, power, and

fabric resources, and some tests require exclusive access. Coordinate them with workload owners and the maintenance process rather than launching a stress test on a busy production node.

The `dcgmi health` interface can configure watches, display the configuration, and evaluate retained samples. The following examples show the relationship among those operations:

```
dcgmi health --set a --group g
dcgmi health --fetch --group g
dcgmi health --check --group g
```

Here, `a` enables the available health-watch categories, and `g` selects all GPUs. The `--set` option configures health watches, `--fetch` displays their configuration, and `--check` evaluates the collected samples.

Some watches need time to collect useful samples, so an immediate check after enabling them may not tell you much. A `Healthy` result also has a precise boundary: none of the enabled health rules found an incident in the retained data. It does not prove that every subsystem was watched or that an active stress test passed.

Current selector meanings, sampling intervals, and result semantics are documented in the dcgmi health command reference (`https://docs.nvidia.com/datacenter/dcgm/latest/reference/command-line-reference/dcgmi/dcgmi-health.html`).

When active testing is justified and the target can be taken out of ordinary service, `dcgmi diag` provides diagnostic suites and named tests. After verifying that GPU `0` is the intended idle target, a quick deployment-check example is:

```
dcgmi diag -r 1 --entity-id gpu:0
```

Without an explicit entity selector, DCGM diagnostics can target all GPUs and NVIDIA CPUs by default. Run active diagnostics only on an appropriate target, after coordinating workload availability. Select the test according to the suspected problem and record its results.

The active-test scope, parameters, and operational considerations are documented in DCGM Diagnostics (`https://docs.nvidia.com/datacenter/dcgm/latest/user-guide/dcgm-diagnostics.html`).

DCGM provides GPU telemetry and health evidence, but monitoring trends over time requires collecting and retaining those metrics. DCGM Exporter and Prometheus provide that capability.

Using DCGM Exporter, Prometheus, and Grafana

DCGM Exporter makes selected GPU metrics available in Prometheus format for collection by monitoring systems. It does not provide long-term storage or visualization.

NVIDIA documents current deployment modes, metric configuration, and compatibility considerations in the DCGM Exporter documentation (`https://docs.nvidia.com/datacenter/dcgm/latest/installation/install-dcgm-exporter.html`).

DCGM Exporter is typically deployed on each monitored GPU node as a service, container, or Kubernetes DaemonSet. NVIDIA GPU Operator can also manage its deployment in Kubernetes environments.

Using Prometheus for collection and retained telemetry

Prometheus periodically collects metrics from DCGM Exporter and stores them as time-series data. This allows operators to analyze historical trends, compare GPU performance across nodes, and identify unusual patterns over time.

Prometheus can also evaluate alerting rules to identify conditions that require attention.

Using Grafana to visualize retained telemetry

Grafana retrieves metrics from Prometheus and visualizes them through dashboards. Operators can monitor GPU utilization, memory usage, temperature, power, and other available metrics across GPUs and nodes over time. Filter the view by node, GPU, workload, and time window before comparing trends.

Dashboards help identify potential problems, but metrics must be interpreted together and in the context of the workload before drawing conclusions.

Grafana documents Prometheus as a supported data source and explains how queries feed panels in its Prometheus data-source documentation (`https://grafana.com/docs/grafana/latest/datasources/prometheus/`).

Figure 9.2 illustrates how DCGM Exporter exposes GPU metrics for collection and storage by Prometheus, while Grafana queries Prometheus to visualize the collected data.

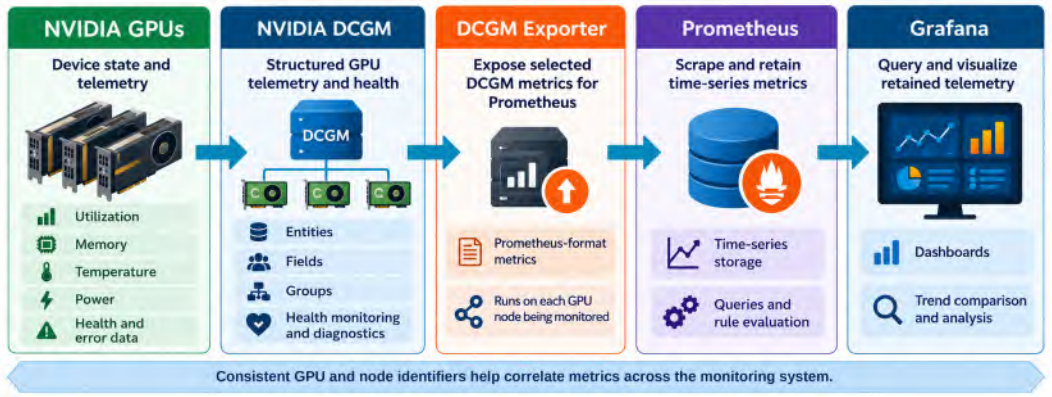


Figure 9.2: DCGM Exporter, Prometheus, and Grafana monitoring flow

Positioning Base Command Manager

NVIDIA currently describes BCM as streamlining cluster provisioning, workload management, and infrastructure monitoring. That places BCM at a broader cluster-management layer than the GPU telemetry and visualization tools in this chapter.

The current product scope and manuals are available from the NVIDIA Base Command Manager documentation hub (`https://docs.nvidia.com/base-command-manager/`).

That does not make BCM interchangeable with DCGM, Prometheus, or Grafana. A management platform can integrate monitoring information while also owning capabilities that a telemetry component does not. The important exam distinction is the responsibility boundary, not a fixed cluster-size threshold.

Chapter 10 goes deeper into workload scheduling and orchestration.

Choosing the right monitoring or management tool

Choose the tool from the operational question. Use `nvidia-smi` when you need an immediate local view of GPU identity, utilization, memory, temperature, and power. Check driver or system logs for Xid errors when investigating suspected GPU faults. Use DCGM when you need structured GPU telemetry, health monitoring, groups, or diagnostics across supported data-center accelerators.

Do not turn scale into a magic threshold. A small environment can still use DCGM, and a large environment can still use `nvidia-smi` for a node-level check. Choose based on the operational responsibility and the evidence required.

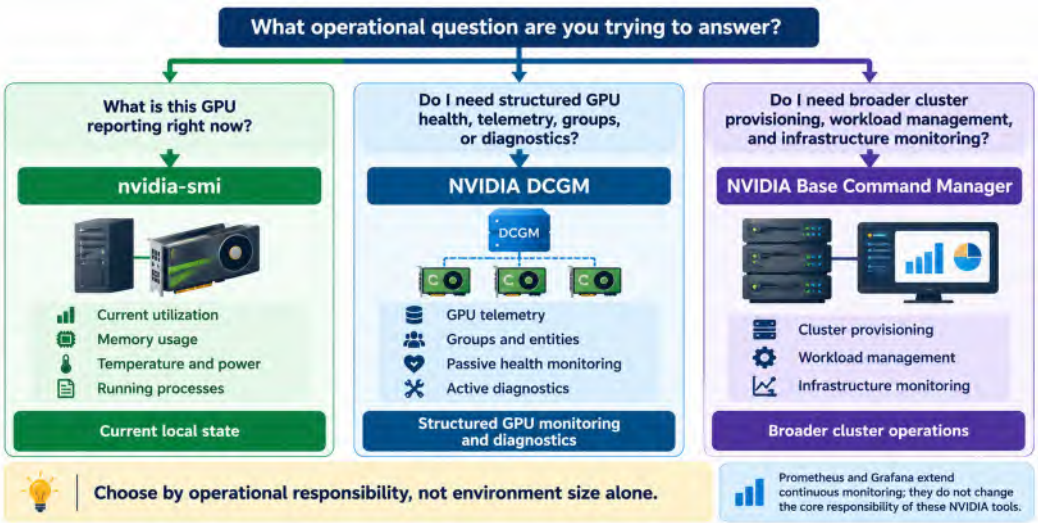


Figure 9.3: Choosing among nvidia-smi, DCGM, and Base Command Manager

With the tool boundaries clear, the final step is to use the lightest evidence path that can narrow a reported symptom.

Simple first-response reasoning

A first response should narrow the problem before making disruptive changes. Start with the workload symptom and expected state, then collect evidence at the lightest layer that can answer the next question.

Use this sequence to narrow a reported symptom:

1. Confirm the symptom, workload, affected node or GPU, and time window.
2. Check immediate GPU state with `nvidia-smi`: visibility, processes, utilization, memory, temperature, power, and obvious error evidence.
3. Compare retained telemetry and peers to determine whether the condition is new, recurring, or isolated.
4. Inspect DCGM health evidence when a GPU problem is suspected. Also investigate CPU, storage, networking, scheduling, or application issues when the available GPU metrics do not explain the symptom.
5. Use active diagnostics only when justified and coordinated. When escalating, provide the time window, identifiers, trends, health or diagnostic results, and recent changes without claiming an unconfirmed root cause.

Review questions

For each scenario, choose the answer that best matches the monitoring responsibility and available evidence.

Question 1

An operator needs an immediate local view of GPU utilization, memory use, temperature, and active processes on one host. Which tool best matches that responsibility?

A. DCGM Exporter

B. `nvidia-smi`

C. Grafana

D. Base Command Manager

Question 2

A GPU shows high framebuffer memory usage but near-zero execution utilization at a single point in time. What is the strongest conclusion?

A. The GPU memory is physically faulty

B. The scheduler has definitely assigned the wrong GPU

C. Memory is occupied, but compute is quiet in this sample

D. The device must be thermally throttled

Question 3

What does a `Healthy` result from a configured DCGM passive health check establish?

A. No enabled health rule found an incident in the retained data

B. Every possible GPU subsystem has passed an active stress test

C. The application has met its latency and throughput targets

D. Prometheus has retained all telemetry since the node was installed

Question 4

Which component primarily exposes selected DCGM GPU metrics in Prometheus format for scraping?

A. Grafana

B. Alertmanager

C. Base Command Manager

D. DCGM Exporter

Question 5

An operator needs structured GPU health and diagnostic evidence across a set of supported data-center GPUs. Which tool best matches that responsibility?

A. Grafana

B. Base Command Manager

C. DCGM

D. `nvidia-smi`

Question 6

A team needs broader cluster provisioning, workload management, and infrastructure monitoring in addition to GPU telemetry. Which tool best matches that requirement?

A. Base Command Manager

B. DCGM Exporter

C. `nvidia-smi`

D. Grafana

Question 7

An operator is considering an active DCGM diagnostic on a production node running a training workload. What should happen first?

A. Run the quick diagnostic immediately because it is short

B. Increase the diagnostic level while training is still running

C. Start diagnostics after a `Healthy` passive check, without coordinating downtime

D. Coordinate workload availability and an appropriate maintenance window

Question 8

Which component primarily stores scraped GPU metrics as time series so operators can query behavior over time?

A. Grafana

B. Prometheus

C. `nvidia-smi`

D. Base Command Manager

Before reading the explanations, identify the responsibility in each question: local inspection, metric interpretation, passive health, metric exposure, structured GPU health/diagnostics,

infrastructure management, active diagnostics, or retained time-series data. This helps rule out options that address a different operational problem.

Answers and explanations

1. **B**: `nvidia-smi` is the local command-line tool for immediate GPU state and process visibility. DCGM Exporter exposes metrics, Grafana visualizes retained data, and BCM operates at a broader cluster-management layer.
2. **C**: Framebuffer memory occupancy and GPU execution activity measure different things. A model can remain resident while compute is quiet, so the operator still needs workload phase, process state, and trend context.
3. **A**: Passive DCGM health checks evaluate the enabled watches over retained samples. A `Healthy` result does not mean every subsystem was watched, an active diagnostic ran, or the application met its service objective.
4. **D**: DCGM Exporter exposes selected GPU telemetry in Prometheus format. Prometheus collects and retains samples, while Grafana visualizes them.
5. **C**: DCGM is the structured GPU telemetry, health, grouping, and diagnostics layer. Grafana visualizes data, BCM operates at a broader cluster-management layer, and `nvidia-smi` is primarily an immediate local inspection tool.
6. **A**: BCM operates at the broader cluster-management layer for provisioning, workload management, and infrastructure monitoring.
7. **D**: Active diagnostics can consume resources or require exclusive access. Preserve evidence and coordinate workload availability and the maintenance window before running them on an active production node.
8. **B**: Prometheus scrapes configured metric targets and stores samples as time series. Grafana queries and visualizes a data source such as Prometheus; `nvidia-smi` is an immediate local tool, and BCM is a broader cluster-management platform.

Summary

In this chapter, you moved from GPU monitoring goals to the tools that answer them. You interpreted utilization, memory, temperature, power, clocks, errors, and workload identity as evidence to consider in context, not as automatic diagnoses.

You explored `nvidia-smi` for immediate GPU inspection and DCGM for structured telemetry, groups, passive health monitoring, and active diagnostics. You also separated DCGM Exporter, Prometheus, and Grafana into metric exposure, retained time-series data, and visualization responsibilities.

Finally, you placed BCM at the broader cluster-management layer. You also learned a practical approach to investigating GPU problems using local observations, historical metrics, DCGM health checks, and controlled diagnostics.

10

Scheduling and Orchestrating Accelerated Workloads

In *Chapter 9, Monitoring and Managing GPU Infrastructure*, you separated immediate GPU inspection from continuous telemetry and learned to ask what an observation actually proves. Monitoring tells you what the infrastructure is doing. It does not, by itself, decide which workload should run next, where that workload should be placed, or how a service should be kept at its intended state when a component fails.

Those decisions belong to workload-management systems. In an accelerated environment, two related models appear repeatedly: job scheduling and container orchestration. A job scheduler is primarily concerned with admitting queued work, matching requested resources to available capacity, and managing jobs through completion. A container orchestrator manages a broader desired state for containerized workloads, including placement, lifecycle, scaling, and reconciliation. Both can use GPUs, and both can appear in the same AI platform, but they solve different control problems.

This chapter builds that distinction first, then uses Slurm and Kubernetes as the representative systems. The goal is not to turn you into a Slurm or Kubernetes administrator. Instead, you will learn enough to recognize their responsibilities, understand how GPUs become schedulable resources, separate native placement behavior from monitoring integrations, and reason about environments where batch training, experimentation, and always-on services coexist.

In this chapter, we will cover the following topics:

- Distinguishing scheduling from orchestration
- Understanding Slurm essentials
- Understanding Kubernetes essentials

- Comparing Slurm and Kubernetes
- Recognizing NVIDIA integration points
- Operating mixed scheduling and orchestration environments

By the end of this chapter, you will be able to identify whether a scenario is primarily a scheduling or orchestration problem, recognize the core Slurm and Kubernetes responsibilities, explain how GPU resources are exposed to workloads, place NVIDIA components at the correct layer, and avoid common shortcuts such as treating Slurm as a training-only system or Kubernetes as an inference-only system.

Technical requirements

This chapter is intentionally conceptual. You do not need to deploy a Slurm cluster, write a Kubernetes manifest, or install NVIDIA GPU Operator to complete it. The following background is enough:

- The networking and accelerated data-path concepts from *Chapter 5, Moving Data Across Accelerated AI Infrastructure*
- The GPU software-stack concepts from *Chapter 6, Understanding the NVIDIA GPU Software Stack*
- The **Multi-Instance GPU** (**MIG**) concepts from *Chapter 7, Virtualizing Accelerated Infrastructure with vGPU and MIG*
- The monitoring and **Data Center GPU Manager** (**DCGM**) responsibilities from *Chapter 9, Monitoring and Managing GPU Infrastructure*
- A browser for current Slurm, Kubernetes, and NVIDIA documentation when you need to verify version-sensitive implementation details

If you have read-only access to a non-production Slurm or Kubernetes environment, you may use it to connect the concepts to familiar objects. That access is optional. Exact command syntax, deployment steps, component versions, and support matrices change over time and are outside the scope of this chapter.

Note

Exam focus: Start by identifying the control responsibility. Is the problem about queued work and resource allocation, or about maintaining the desired state of a containerized workload? Then decide which platform responsibility fits. Monitoring, communication libraries, GPU partitioning, and network data movement may support the workload, but they are not the same thing as scheduling or orchestration.

Scheduling versus orchestration

A workload-management system has to answer more than one question. First, there is a placement question: which available resources can satisfy this workload? Second, there is a lifecycle question: what should happen before the workload starts, while it runs, when it completes, and when something fails? Job schedulers and container orchestrators both answer parts of those questions, but they begin from different operating models.

A job scheduler starts with units of work that are submitted, queued, admitted according to policy, assigned resources, run, and eventually reach an end state. This model is a natural fit for **high-performance computing** (**HPC**), batch analytics, simulation, and many distributed training workloads. The scheduler can consider requested CPUs, memory, GPUs, runtime limits, partition policy, priority, reservations, and other site-defined constraints before allocating capacity.

A container orchestrator starts with a declared desired state. Instead of saying only, "run this once when resources become available," the user can specify the number of workload replicas, a service endpoint, resource requirements, and placement constraints. Controllers then keep comparing the observed state with the desired state and act when they differ. A failed container may be replaced, a replica count may change, or a workload may be moved to another suitable node.

Kubernetes illustrates an important detail: orchestration still contains scheduling. The Kubernetes scheduler selects a node for a Pod that has not yet been placed, but Kubernetes as a platform does more than schedule. Controllers, the node agent, networking, service discovery, and other control loops keep the application state aligned with the declared configuration. That broader control loop is what makes "orchestration" more than a synonym for scheduling.

Two useful analogies make the distinction concrete. A job scheduler resembles an airport allocating runways and gates to flights that must wait, use a finite resource, and then leave. A container orchestrator resembles a city traffic-control system that continuously manages movement and responds when conditions change. The analogy is helpful as long as you do not turn it into a hard technology boundary. Real platforms overlap, and both can support workload types outside their most common examples.

The difference is easier to see as a pair of control flows. *Figure 10.1* shows the job path on the left and the desired-state loop on the right.

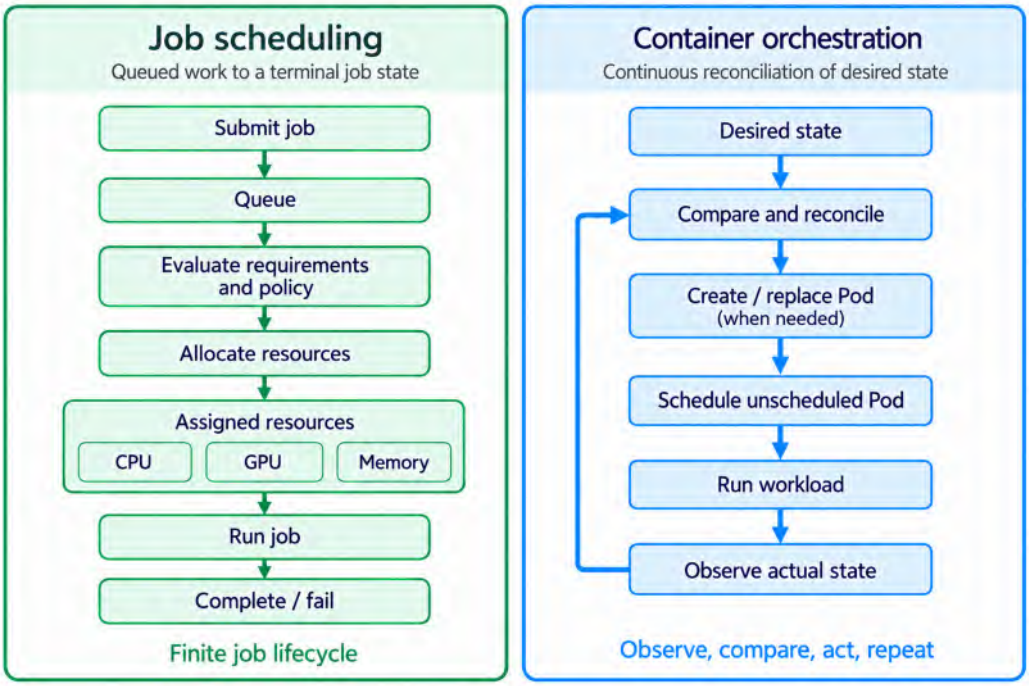


Figure 10.1: Job scheduling and container orchestration control flows

The left-hand flow reaches a terminal job state after allocated work runs. The right-hand flow continues because the orchestrator keeps checking whether the running system matches the declared intent. This difference also shapes the wording of exam scenarios. Queues, priorities, reservations, pending jobs, and completion point toward scheduling. Replicas, reconciliation, services, health, and replacement point toward orchestration.

Workload type is evidence, not a verdict

It is tempting to compress the distinction into "training uses Slurm and inference uses Kubernetes." That shortcut is easy to remember and often wrong. Many organizations do use Slurm for large queued training jobs and Kubernetes for containerized inference services, so the association is useful as a starting clue. It is not a rule.

Kubernetes can run finite batch jobs and distributed training workloads. Slurm environments can run long-lived processes or support serving patterns when the surrounding platform is designed for them. The stronger distinction is the operating model: how work is described, admitted, placed, supervised, and integrated with the rest of the platform.

For certification exam scenarios, prioritize the control responsibility rather than the workload label. If both problems exist, a mixed environment may be appropriate. We will return to that case after examining the two representative platforms.

Understanding Slurm essentials

Slurm Workload Manager is widely used in HPC and accelerated-computing environments because it organizes queued work around explicit resource requests and site policy. Its policies determine when eligible jobs receive their requested resources.

Thinking in jobs, allocations, and partitions

A job represents requested work plus the resources and constraints needed to execute it. Before the job can run, Slurm needs an allocation. The allocation is the set of resources reserved for that job, such as compute nodes, CPUs, memory, and GPUs.

Clusters are commonly divided into partitions. A partition groups nodes and associates them with scheduling policy. One partition might contain GPU-dense systems, another might contain general-purpose CPU nodes, and another might be reserved for a particular class of work. The partition is therefore both a resource grouping and a policy boundary. A job can remain pending because the requested partition lacks a currently available resource even when capacity exists somewhere else in the cluster.

This is one reason "the cluster has idle GPUs" is not enough to conclude that a waiting job should be running. The idle devices may be outside the requested partition, reserved for another use, incompatible with the request, unavailable under policy, or fragmented in a way that prevents the requested allocation. Scheduling always has to be interpreted in the context of the resource request and site policy.

Priorities, reservations, and policy controls

When more jobs are eligible than the cluster can run at once, Slurm uses configured policy to decide the order in which work is considered. A site can incorporate factors such as job age, fair-share history, partition priority, job size, and **Quality of Service** (**QOS**). Exact weighting is an administrative choice, so you should not assume that "oldest job runs first" or that one factor always dominates.

Reservations solve a different problem. They can set aside resources for a defined period or purpose so that a planned workload, maintenance activity, class, or event has capacity when required. QOS settings can affect priority, preemption, and resource limits depending on site configuration. Together, these controls let administrators express operational policy around shared infrastructure rather than leaving resource access to a simple first-come, first-served queue.

Quotas and accounting add another dimension. A site may constrain how much of a resource a user, group, or project can consume, or use historical consumption as an input to fair-share policy.

Exposing GPUs as schedulable resources

Slurm can represent GPUs through its **Generic Resources** (**GRES**) mechanism. From the scheduler's perspective, this makes an accelerator something a job can request and something the allocation system can reserve. A site can distinguish GPU types and other characteristics when configured to do so. Slurm also supports GPU-specific accounting and resource controls, but the durable concept is simpler: the scheduler needs an inventory of allocatable accelerator resources and a job needs to express what it requires.

When a GPU allocation is granted, the workload should see the devices assigned to that allocation rather than treating every GPU in the node as freely available. Slurm can coordinate that exposure for launched tasks, including the device visibility that CUDA applications use.

This distinction becomes especially important with MIG. As discussed in *Chapter 7, Virtualizing Accelerated Infrastructure with vGPU and MIG*, MIG partitions supported GPUs into isolated GPU instances. Slurm can schedule MIG instances when the site has configured those instances as resources. Slurm does not dynamically decide how a GPU should be partitioned merely because a job is waiting. Resource creation and resource allocation are separate responsibilities.

Following the job lifecycle

A submitted job typically moves through a small set of conceptual states: waiting for eligibility and resources, running on an allocation, and then completing, failing, or being canceled. A running state tells you that an allocation was made; it does not tell you that the workload is using the GPUs efficiently.

That last point connects directly to *Chapter 9, Monitoring and Managing GPU Infrastructure*. Slurm tracks allocations and, when configured, selected GPU usage metrics. DCGM and related tools provide detailed GPU telemetry. A monitoring system can inform capacity planning, troubleshooting, or a custom policy extension, but native Slurm placement should not be described as though DCGM utilization automatically determines where every job runs.

Keeping communication separate from scheduling

Multi-GPU and multi-node jobs also use communication technologies introduced earlier in the book. **NVIDIA Collective Communications Library** (**NCCL**) coordinates collective communication inside supported applications. GPUDirect technologies and the underlying PCIe, NVLink, Ethernet, InfiniBand, or **remote direct memory access** (**RDMA**) paths move

data. Slurm can allocate the nodes and GPUs on which the job runs, but it does not perform the NCCL collective or the GPUDirect transfer itself.

This responsibility map prevents a common reasoning error. If a distributed training job has already been scheduled and then scales poorly, the next question may concern communication, topology, software, or data movement rather than scheduling. The scheduler answered "where and when can this job run?" It did not guarantee that every layer inside the allocation will deliver the expected application performance.

Understanding Kubernetes essentials

Kubernetes uses API objects and controllers to manage containerized workloads. Scheduling is one part of the process, but the platform also handles workload controllers, service discovery, health-driven replacement, configuration, and other lifecycle responsibilities.

Starting with Pods and desired state

A Pod is the basic schedulable unit in Kubernetes. It contains one or more containers that share a lifecycle and selected resources. In most production environments, users do not manage isolated Pods directly. Higher-level controllers describe the desired workload and create or replace Pods as needed. A Deployment, for example, can express a desired replica count for a service, while a Job represents finite work that should run to completion.

Reconciliation makes the difference concrete. Suppose a service should have four replicas and one Pod disappears because a node fails. The controller observes that the actual state has fallen below the desired state and creates a replacement Pod. The scheduler then finds a suitable node for the new Pod. The controller owns the desired replica count; the scheduler owns node placement for an unscheduled Pod. Keeping those responsibilities separate makes Kubernetes behavior easier to reason about.

Services, health, and scaling

Containerized applications also need a stable way to reach changing Pods. A Kubernetes Service provides a durable network identity in front of a changing set of matching Pods. Readiness and liveness mechanisms can help determine whether containers should receive traffic or be restarted, while controllers can scale replica counts manually or through additional automation. These are orchestration responsibilities around the workload lifecycle, not GPU-scheduling algorithms.

Autoscaling deserves the same careful boundary. Kubernetes supports mechanisms that can change workload replicas or cluster capacity based on configured signals and components. Scheduling normally uses declared resource requirements plus scheduling constraints and

available resource information. More specialized placement policies can be added, but they should be described as explicit extensions rather than assumed native behavior.

Requesting GPU resources conceptually

Kubernetes exposes specialized hardware through its device-plugin framework. When the NVIDIA device plugin is present, GPU capacity can be advertised to the cluster as an extended resource such as `nvidia.com/gpu`. A Pod can then request that resource, and the scheduler considers nodes with sufficient advertised capacity alongside the Pod's other constraints.

The scheduler does not need to understand CUDA kernels or model architecture. It needs a resource model that states how many eligible devices are available and how many the workload requires. Once placed, the container runtime and NVIDIA software stack make the allocated devices available to the application.

The Kubernetes resource model also reinforces the difference between capacity and activity. A GPU can be allocatable to Kubernetes even though it was highly utilized a minute earlier, and a GPU can be allocated to a Pod that is currently idle. Resource assignment and telemetry answer different questions.

With the two models established independently, we can compare them without forcing either platform into a narrow workload stereotype.

Comparing Slurm and Kubernetes

A useful comparison focuses on control responsibilities and operating assumptions rather than product features. *Table 10.1* summarizes the differences that matter for certification-level reasoning while keeping the overlap visible.

Dimension	**Slurm**	**Kubernetes**
Primary control problem	Queue and prioritize jobs, allocate resources, and track execution through completion.	Maintain declared state for containerized workloads, including placement and lifecycle control.
Common workload shape	Often HPC, queued batch, simulation, analytics, and distributed training; not limited to these.	Often services, microservices, Jobs, batch pipelines, and containerized AI workloads; not limited to inference.

Dimension	Slurm	Kubernetes
Execution model	Work is submitted as jobs that wait for an eligible allocation and then run on assigned resources.	Work is declared through API objects; controllers create Pods and the scheduler places unscheduled Pods.
Resource model	Nodes, CPUs, memory, GPUs, and other configured resources are allocated under site policy.	Node capacity plus standard and extended resources are matched to Pod requirements and constraints.
GPU exposure	GPUs can be represented as configured GRES resources; the job receives the GPUs in its allocation.	A vendor device plugin advertises GPUs as extended resources such as `nvidia.com/gpu` for Pod allocation.
Policy emphasis	Partitions, priority, fair share, reservations, QOS, limits, and accounting commonly shape admission and order.	Requests, limits, selectors, affinity, taints/tolerations, priorities, quotas, and policies can shape placement and admission.
Lifecycle emphasis	Jobs move from queued to running to terminal states; retries and workflow behavior depend on surrounding tools and configuration.	Controllers continuously reconcile desired and observed state and can recreate or scale workload replicas.
Monitoring relationship	Scheduler state explains allocation decisions; GPU telemetry remains a separate monitoring responsibility.	Scheduler and controller state explain orchestration decisions; DCGM telemetry remains separate unless explicitly integrated into additional policy.

Table 10.1: Slurm and Kubernetes responsibility comparison

If the organization already operates a large HPC environment with established fair-share policy and queued accelerator jobs, Slurm may fit naturally. If the organization's workloads are already packaged and run as containerized services with declarative lifecycle controls, Kubernetes may fit naturally. Some organizations use each system for the workloads and teams that match the platform's operating model.

Recognizing NVIDIA integration points

Accelerated workload managers need a bridge between generic cluster control and GPU-specific capabilities. That bridge can involve resource discovery, device exposure, partitioning, node software, communication libraries, and telemetry. The key is to keep each capability attached to the layer that actually owns it.

Slurm and NVIDIA GPU resources

GRES provides the scheduling abstraction, while the NVIDIA driver and runtime stack on the node make the GPUs usable by the application.

Kubernetes and NVIDIA GPU enablement

The NVIDIA Container Toolkit and driver stack support container access to GPUs. Node labels and feature-discovery components can expose useful hardware characteristics for scheduling constraints.

NVIDIA GPU Operator automates deployment and lifecycle management of several node-side NVIDIA components used in Kubernetes GPU environments. Depending on the supported configuration, these components can include the NVIDIA driver, Container Toolkit, device plugin, GPU feature discovery, DCGM or DCGM Exporter, and MIG Manager. This is a node-software management role.

MIG Manager can apply supported MIG configurations to Kubernetes GPU nodes, while the device plugin can expose the resulting GPU-instance resources according to the chosen strategy. The scheduler can then allocate exposed resources to Pods.

DCGM Exporter can expose GPU metrics for collection by Prometheus-based monitoring systems, as discussed in *Chapter 9*. Those metrics are valuable for dashboards, alerts, troubleshooting, and capacity decisions. They do not mean that the default Kubernetes scheduler automatically rebalances GPU workloads based on utilization. Any utilization-aware placement or rebalancing mechanism should be treated as an additional policy or extension with its own assumptions.

The integration map in *Figure 10.2* keeps these responsibilities visible. Read it vertically: the workload manager allocates an exposed resource, while software, communication, partitioning, and monitoring capabilities support the workload at adjacent layers.

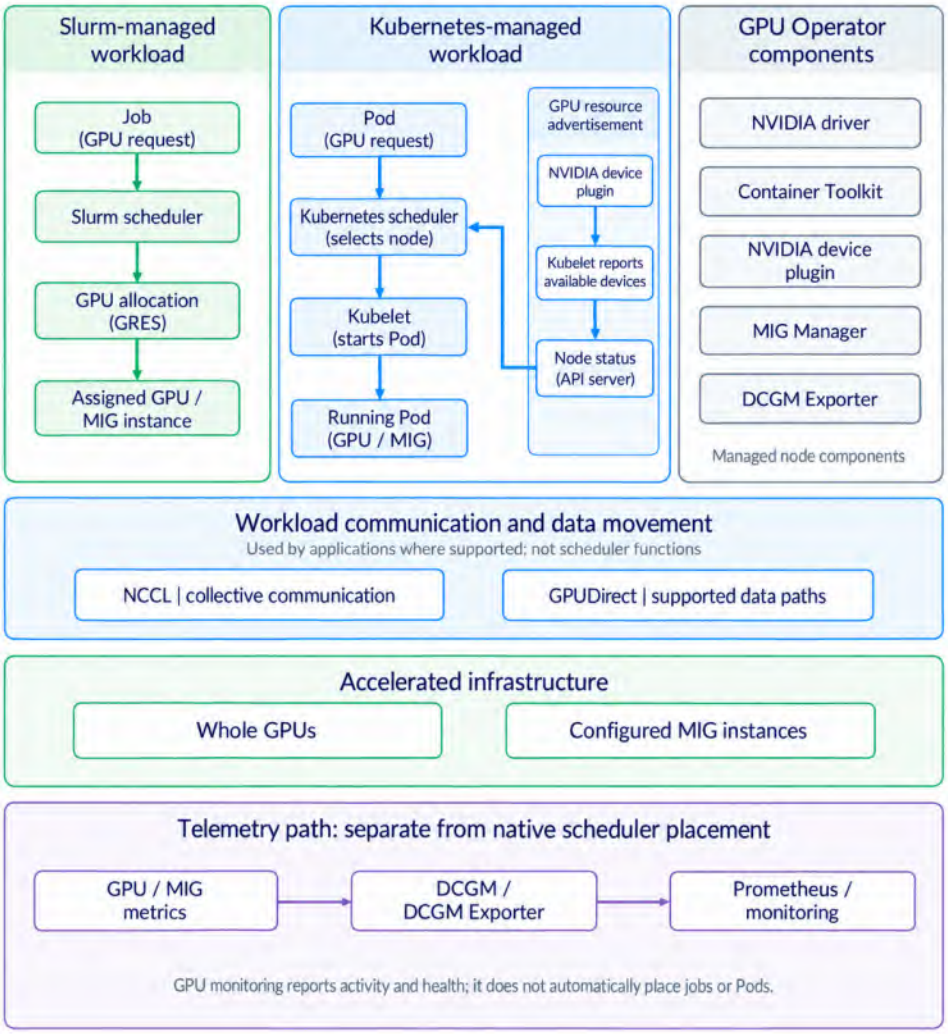


Figure 10.2: NVIDIA integration context for Slurm and Kubernetes

Operating mixed environments

Real AI platforms can contain more than one workload-management model. A research group may use Slurm for large queued experiments while an application team runs containerized services in Kubernetes. The design is workable when ownership is explicit: teams need to know which system owns each resource pool, where workloads are submitted, how capacity is reserved, and where failures are handled. A GPU should not be treated as simultaneously free

to two independent allocators unless an explicit sharing or partitioning design sits underneath them.

Separating resource ownership from workload handoff

One straightforward pattern uses separate GPU pools, with Slurm owning one set of nodes and Kubernetes another. Another pattern uses a lifecycle handoff: a training job runs under one control plane and publishes a versioned model artifact; a separate deployment process then makes that artifact available to an inference environment under the other control plane. In both cases, the boundary is easier to operate because the two systems do not independently schedule the same running workload.

Organizations can also standardize more of the lifecycle on one platform when that better matches their operating model. Kubernetes can run training and batch jobs as well as services, and Slurm-centric environments can extend existing practices to additional workload patterns. At this level, the goal is to identify what each control plane owns rather than choose one universal architecture.

Using monitoring without confusing it with control

Shared observability can span both environments. DCGM telemetry and infrastructure dashboards can support troubleshooting, capacity planning, and human decisions about reservations or resource-pool size. If telemetry feeds a custom placement controller or policy extension, that additional control path should be named explicitly rather than treated as default scheduler behavior.

A responsibility-first scenario method

For a mixed-environment scenario, work through the problem in the following order:

1. Identify the unit of work: a queued job, a finite containerized job, a long-running service, or a combination.
2. Identify the control objective: fair allocation and completion, desired-state reconciliation, or both.
3. Identify the accelerator resource model: whole GPUs, configured MIG instances, or another explicitly supported sharing model.
4. Identify the policies and ownership boundaries that determine which resources the workload can use.
5. Separate supporting layers such as NCCL, networking, storage, software dependencies, and monitoring from the workload-management decision.
6. Choose the scheduler or orchestrator responsibility only after those boundaries are clear.

Review questions

The following questions are original practice items for this chapter. They do not reproduce live NVIDIA exam questions. For each scenario, identify the control responsibility before choosing a platform or component.

Question 1

A team submits finite GPU jobs that may wait for capacity, and the main requirement is to prioritize queued work under the cluster's scheduling policy. Which responsibility is primary?

A. Container health reconciliation across service replicas

B. Job scheduling and resource allocation

C. GPU telemetry visualization

D. Model artifact distribution

Question 2

A Slurm job is pending even though an operator can see some idle GPUs elsewhere in the cluster. Which explanation should the operator consider first?

A. Idle GPU capacity always satisfies a pending job if the device is healthy

B. DCGM utilization is used by default to bypass partition and reservation policy

C. NCCL must claim an idle GPU before Slurm can create the job allocation

D. The request or site policy may make that idle capacity ineligible for the job

Question 3

What is the main reason Slurm represents GPUs through a resource mechanism such as GRES?

A. So GPU requirements can participate in the job's resource allocation

B. So Slurm can execute NCCL collectives for the application

C. So DCGM metrics become the default signal for job placement

D. So Slurm can create MIG geometry dynamically when demand changes

Question 4

A Kubernetes cluster has the NVIDIA device plugin installed. What does that primarily enable?

A. Prometheus to act as the placement scheduler for GPU Pods

B. Kubernetes to repartition GPUs from live utilization signals

C. GPU capacity to be exposed as a schedulable resource for Pods

D. NCCL to replace the container runtime on GPU worker nodes

Question 5

Which statement best describes the NVIDIA GPU Operator in the context of this chapter?

A. It replaces Kubernetes controllers and schedules GPU Pods directly

B. It manages supported NVIDIA software components on Kubernetes GPU nodes

C. It packages each application's frameworks and model dependencies

D. It makes DCGM utilization the default input for Kubernetes placement

Question 6

A Kubernetes environment uses MIG on supported GPUs. Which responsibility boundary is most accurate?

A. Configured MIG resources are exposed for the scheduler to allocate

B. The scheduler creates MIG profiles from each Pod's live utilization

C. DCGM Exporter repartitions the GPU before each scheduling cycle

D. NCCL selects MIG geometry because it manages GPU communication

Question 7

A multi-node training job has already been allocated GPUs but performs poorly during collective communication. Which statement is most accurate?

A. GPU allocation proves that the network path is already optimal

B. Kubernetes reconciliation is required even for a Slurm job

C. The scheduler performs GPUDirect transfers after allocating GPUs

D. The next investigation may involve NCCL, topology, or the data path

Question 8

An organization trains models in a Slurm-managed research cluster and serves selected models in Kubernetes. What is the best interpretation of this design?

A. One platform must be removed because the control models cannot coexist

B. Both platforms should schedule the same GPU nodes without an ownership boundary

C. The design can be valid when ownership and the model-artifact handoff are explicit

D. DCGM should select the platform that owns each model from utilization

Answers and explanations

1. **B:** The defining requirement is queued work competing for finite resources under policy, which is the core job-scheduling problem. Reconciliation and telemetry may support the environment, but they are not the primary control responsibility in the scenario.
2. **D:** Visible idle capacity does not prove that the pending job is eligible to use it. Partition membership, resource type, reservations, policy, dependencies, and the job's resource request can all affect whether an allocation is possible. DCGM telemetry does not automatically override those scheduling rules.
3. **A:** A resource abstraction lets a job declare accelerator needs and lets Slurm reserve eligible devices in the allocation. NCCL communication, DCGM monitoring, and MIG configuration are different responsibilities.
4. **C:** The NVIDIA device plugin advertises GPU capacity to Kubernetes as an extended resource, allowing Pods to request accelerators that the scheduler can consider. It does not turn Prometheus into a scheduler or make utilization-driven placement the default behavior.
5. **B:** GPU Operator manages supported NVIDIA components needed for Kubernetes GPU nodes, such as drivers, the Container Toolkit, the device plugin, monitoring components, and MIG-related components according to configuration. Application libraries and model dependencies still belong to the workload packaging and software lifecycle.
6. **A:** MIG creates partitioned GPU resources at the infrastructure layer. Once supported instances are configured and exposed, the workload-management system can allocate them. Scheduler placement and MIG geometry management are related but separate responsibilities.
7. **D:** Allocation only establishes that the workload received resources. Collective communication performance depends on NCCL, topology, interconnects, RDMA or GPUDirect paths, software versions, and workload behavior. Those layers should be investigated without blaming scheduling simply because the job is slow.
8. **C:** A mixed design is reasonable when each platform has clear resource ownership and the lifecycle handoff is explicit. A model can be trained under one control plane and later deployed under another without both systems independently scheduling the same running workload.

Summary

In this chapter, you separated two workload-control models that are easy to confuse. Job scheduling is centered on queued work, resource requests, policy, allocation, and completion. Container orchestration is centered on desired state, placement, workload lifecycle, services, scaling, and reconciliation. Kubernetes still contains a scheduler, but orchestration extends beyond the placement decision.

You used Slurm to understand jobs, partitions, priorities, reservations, QOS, and GPU resource allocation. You then used Kubernetes to understand Pods, controllers, Services, desired-state reconciliation, and device-plugin-based GPU exposure. The comparison showed why training and inference are useful workload clues, not strict platform rules: either system can support workloads outside its most common association when the operating model fits.

You also placed NVIDIA integrations at the correct layer. Slurm and Kubernetes allocate exposed GPU resources. MIG provides partitioned accelerator resources when configured. GPU Operator manages supported NVIDIA software components in Kubernetes environments. NCCL and GPUDirect support communication and data movement. DCGM and DCGM Exporter provide GPU health and performance telemetry. None of those supporting capabilities should be mistaken for native scheduler behavior.

Finally, you applied a responsibility-first method to mixed environments. Clear resource ownership, explicit lifecycle handoffs, and separation between monitoring and control make it possible for scheduling and orchestration systems to coexist. The next chapter moves from workload placement and lifecycle control to MLOps practices that connect software, model artifacts, automation, monitoring, and repeatable AI operations.

11

Recognizing MLOps Practices in the AI Lifecycle

Chapter 10, Scheduling and Orchestrating Accelerated Workloads, focused on how accelerated work is admitted, placed, and kept running. Once a model reaches production, however, a different operational question arises: how can you make each change repeatable, traceable, and controlled so the result can be reproduced later?

That is where **machine learning operations** (**MLOps**) can help. MLOps is not a single product and it is not a replacement for the infrastructure practices discussed earlier in the book. It is a set of practices that connects development and operations across the machine learning lifecycle so that data, code, model artifacts, environments, deployments, and operational evidence can be managed as related parts of one process.

The certification does not treat MLOps as a separate domain. This chapter therefore stays at recognition depth: why versioning matters, what a registry is for, how CI/CD and retraining triggers fit the lifecycle, and how monitoring evidence feeds back into controlled change.

In this chapter, we will cover the following topics:

- Understanding why MLOps matters
- Tracking data, code, models, and environments
- Recognizing CI/CD and event-driven retraining
- Mapping representative NVIDIA tools to lifecycle operations
- Connecting monitoring feedback to the AI lifecycle

By the end of this chapter, you will be able to explain why MLOps practices matter after an experiment succeeds, identify the artifacts that need to be tracked together. You will also be able to distinguish automated pipeline steps from retraining triggers, place representative NVIDIA technologies at the correct lifecycle stage, and interpret monitoring evidence as a basis for evaluation and controlled change rather than a guarantee of model quality.

Technical requirements

This chapter is conceptual and does not require a GPU, a model-training environment, or access to a CI/CD platform. The examples are designed to reinforce lifecycle responsibilities rather than teach a specific MLOps implementation.

When a named product or service is part of a real implementation, verify its current documentation, supported integrations, and lifecycle policy before treating a workflow as fixed. Product offerings may change, but these lifecycle responsibilities remain.

> **Note**
>
> **Exam focus**: Treat MLOps as a lifecycle discipline. Look for repeatability, versioning, lineage, validation, controlled promotion, retraining triggers, and feedback. Identify the responsibility before the product.

Why MLOps matters

A successful experiment answers a narrow question: can this model produce useful results under the conditions in which it was tested? Production creates a wider set of questions. Which dataset produced the result? Which code and dependencies were used? Which model artifact was approved? Which environment was deployed? What changed after the last release? What evidence would justify another training cycle?

MLOps introduces discipline around those handoffs. A useful process makes important artifacts identifiable, records their relationships, applies checks before promotion, and creates a path from operational evidence back to the next lifecycle decision.

From a linear workflow to a managed loop

It is easy to draw the AI lifecycle as a straight line: prepare data, train a model, optimize it, deploy it, and monitor it. Operations adds the return path. Monitoring and new data can create evidence that a model or its environment should be reevaluated, but the response still needs validation before another version is promoted.

Figure 11.1 summarizes that managed feedback loop. Notice that the arrow from monitoring does not jump directly to deployment. Evidence leads to evaluation or a change trigger, and any updated model still passes through validation before promotion.

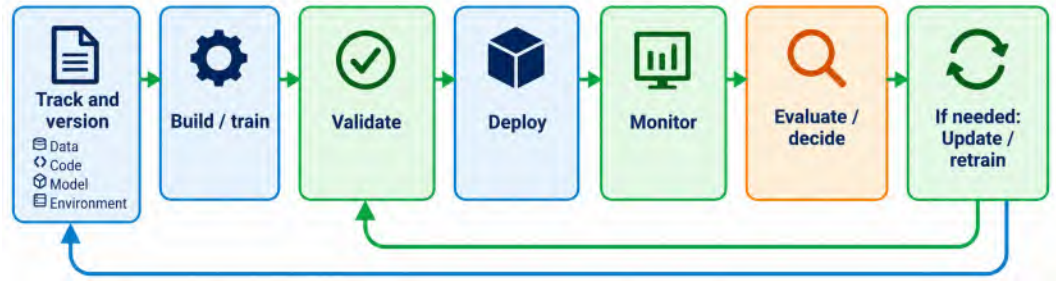


Figure 11.1: MLOps feedback loop from tracked change to monitored operation

Tracking data, code, models, and environments

Reproducibility starts with the ability to identify the inputs and artifacts behind a result. A model artifact is only one part of that record. The same training code can produce a different model when the data changes. The same model file can behave differently when the runtime environment changes. A useful record therefore links several classes of artifact rather than storing a model in isolation.

At the associate level, four categories are enough to organize the problem:

- **Data**: the dataset or data snapshot used for training, tuning, evaluation, or validation.
- **Code and configuration**: the training logic, preprocessing steps, hyperparameters, and configuration that define the run.
- **Model artifacts**: the resulting model version together with metadata that identifies how it was produced and evaluated.
- **Environment**: the runtime context, such as the container image, framework and library versions, driver and runtime dependencies, and other software needed to reproduce or serve the model.

These categories can change independently. A new dataset creates a new lifecycle state even when the code is unchanged, and a new container image changes the deployment environment even when the model artifact stays the same.

Versioning and lineage answer different questions

Versioning tells you which instance of an artifact you are using. Lineage tells you how artifacts are related. A model version without lineage might tell you that model v12 is deployed. A

lineage record adds the information needed to connect v12 to the data snapshot, code revision, configuration, environment, and evaluation that produced it.

When a production issue appears after a change, lineage helps identify what changed between the previous known-good state and the current one, not just which model is running.

Understanding the model registry concept

A model registry provides a managed place to organize model artifacts and metadata. Its responsibilities include distinguishing versions, associating evaluation evidence, making approved artifacts discoverable, and supporting controlled movement through lifecycle states such as candidate, approved, or deployed.

A model registry should not be confused with a general software catalog or container registry. Those systems may participate in the same workflow, but the certification-level distinction is the responsibility: traceable model-version management versus broader software or artifact distribution.

NVIDIA GPU Cloud (**NGC**) is useful in the neighboring distribution role because it provides containers, models, Helm charts, and related GPU-optimized resources. NGC Private Registry also supports model versioning; artifact storage and versioning do not, by themselves, establish model evaluation and promotion policies.

Once data, code, model artifacts, and environments are identifiable, automation can connect their changes to repeatable checks. That is the role of CI/CD in the next section.

Recognizing CI/CD and event-driven retraining

Continuous integration and continuous delivery or deployment, usually shortened to CI/CD, bring repeatable automation to changes. In ML workflows, code is only one source of change. Data, model configuration, framework dependencies, and evaluation thresholds can also affect the result.

Continuous integration in an ML workflow

Continuous integration (**CI**) repeatedly validates changes to code, configuration, or related workflow inputs. Automation can run the checks, but the organization still defines what counts as a passing result.

Continuous delivery and controlled promotion

The CD part of CI/CD concerns moving a validated change toward a usable environment. Some teams keep an approval step while others automate deployment after gates pass, so the durable concept is controlled promotion rather than a particular tool chain.

A model should not move forward merely because training completed. A completed job shows only that execution ended; promotion requires separate evidence that the resulting artifact passed the checks defined for the next lifecycle stage.

Event-driven retraining is a trigger pattern, not a rule

Event-driven retraining is a representative MLOps pattern. A trigger might be the arrival of new data, an approved code change, a change in an upstream data source, or evidence from production that justifies reevaluation. A scheduled refresh is a separate, time-based trigger. An event-driven or scheduled trigger starts a workflow; it does not prove that a new model should be deployed.

A safe conceptual sequence is: detect a trigger, prepare the inputs, run training or an update, evaluate the result, and promote it only if the defined checks pass.

Mapping NVIDIA tools to lifecycle operations

Table 11.1 maps lifecycle operations to technologies. Treat each entry as representative rather than exclusive. A product can span more than one stage, and a lifecycle stage can use products from several vendors.

Lifecycle operation	**Primary responsibility**	**Representative NVIDIA technology**	**Covered in**
Data preparation	Prepare and transform data	RAPIDS	*Chapter 8*
Training and customization	Run training or customization on the accelerated stack	Frameworks and NVIDIA-accelerated libraries	*Chapters 6 and 8*
Inference optimization	Optimize trained models for GPU inference	TensorRT	*Chapter 8*
Deployment and serving	Serve models or deploy inference endpoints	NVIDIA Dynamo-Triton; NVIDIA NIM	*Chapter 8*
Monitoring	Expose GPU health and performance telemetry	DCGM; DCGM Exporter	*Chapter 9*

Lifecycle operation	Primary responsibility	Representative NVIDIA technology	Covered in
Artifact distribution	Distribute containers, models, and related software assets	NGC	*Chapters 8 and 11*

Table 11.1: Lifecycle operations and representative NVIDIA technologies

Three boundaries matter: TensorRT optimizes inference, while NVIDIA Dynamo-Triton serves models; NVIDIA NIM provides inference microservices rather than a complete MLOps workflow; and DCGM telemetry informs infrastructure decisions, not automatic retraining.

NGC Private Registry can also hold organization-specific assets. Distinguish an artifact distribution or storage role from the policies governing evaluation and promotion across a model lifecycle.

With the software roles separated, we can now connect the final operational input to the lifecycle: monitoring evidence.

Connecting monitoring feedback to the lifecycle

Monitoring closes the loop only when the evidence is interpreted correctly. GPU monitoring provides infrastructure observations such as utilization, memory, temperature, power, health, and exporter-based telemetry. Those signals answer infrastructure questions. They can show that a workload is resource constrained or that a node is unhealthy, but they do not directly measure whether the model's predictions are appropriate for the business problem.

A complete lifecycle can use several types of signals. Infrastructure telemetry describes the platform, service observations describe runtime behavior, and model evaluation describes whether a candidate meets defined quality criteria. MLOps connects those signals to the next controlled change.

Monitoring creates evidence, not automatic action

Suppose a deployed inference service becomes slower. High GPU utilization may indicate that the infrastructure is saturated, but retraining the model would not automatically solve that problem. An appropriate response may involve adjusting capacity, batching, optimization settings, or serving configuration. Conversely, a model can continue to use healthy GPUs while its evaluation quality degrades because the data it sees has changed. Infrastructure health alone does not prove model quality.

Classify the signal first: infrastructure, service behavior, data, or model quality. Then decide whether the next action is investigation, reconfiguration, rollback, data review, or a new training cycle.

A simple production scenario

Consider a team that deploys a new model version for an internal inference service. The model is tied to a known data snapshot, code revision, evaluation result, and serving environment. After deployment, GPU health remains normal but service latency increases.

The recorded lineage shows that the model changed while the environment did not, narrowing the investigation without claiming a root cause. The team can compare model versions, optimization settings, or request behavior and still identify the previous approved artifact if the change must be reversed.

Review questions

Use the following questions to check whether you can recognize MLOps responsibilities without relying solely on product-name recall.

Question 1

A team can reproduce the training code but cannot determine which dataset snapshot and environment produced the model currently in production. Which MLOps capability is most directly missing?

A. A scheduler for assigning GPUs to the next training run

B. Versioning and lineage linking the model to its inputs

C. A telemetry exporter for collecting node-level GPU metrics

D. An inference server for exposing the deployed model to clients

Question 2

New labeled data arrives in a production ML workflow. Which response best represents event-driven retraining?

A. Replace the deployed model immediately because the dataset changed

B. Ask the scheduler to choose the model with the lowest GPU utilization

C. Store the data but avoid automation because retraining must be manual

D. Start controlled training and evaluation before any promotion

Question 3

What is the main role of a model registry in the context of this chapter?

A. Organize model versions and metadata so lifecycle states are traceable

B. Collect GPU temperature and power telemetry from compute nodes

C. Allocate MIG resources to training jobs through scheduler policy

D. Optimize a trained model into a GPU inference engine

Question 4

A training pipeline completes successfully, but the resulting model has not yet passed the required evaluation checks. What should the workflow conclude?

A. The completed job proves the model is ready for production

B. Deploy the model first so monitoring can decide whether it is valid

C. Execution completed, but promotion should wait for required validation

D. Retrain the model because every successful job creates a production version

Question 5

Which statement best describes the relationship between DCGM telemetry and the MLOps lifecycle?

A. DCGM telemetry automatically decides whether a model should be retrained

B. DCGM provides infrastructure evidence for lifecycle investigation and decisions

C. DCGM replaces tracking of model, code, data, and environment versions

D. DCGM provides the registry for approved production model artifacts

Question 6

Which NVIDIA technology is primarily associated with optimizing a trained model for efficient GPU inference?

A. RAPIDS

B. DCGM Exporter

C. NGC Catalog

D. TensorRT

Before checking the answers, confirm that each choice reflects the lifecycle responsibility described in the question, not simply a familiar NVIDIA product name.

Answers and explanations

1. **B:** Versioning identifies individual artifacts, while lineage connects the deployed model to the data, code, configuration, and environment that produced it. A scheduler, telemetry exporter, or inference server addresses a different responsibility.
2. **D:** The arrival of new data can be a trigger, but the trigger should start a controlled workflow. Training and evaluation still have to occur before an updated model is promoted. New data does not make replacement automatic.
3. **A:** A model registry helps organize model artifacts, versions, metadata, and lifecycle state so teams can identify and control the models they use. GPU monitoring, scheduling, and inference optimization belong to other layers.
4. **C:** Job completion proves only that execution finished. Promotion is a separate lifecycle decision that depends on the required validation and approval gates.
5. **B:** DCGM and DCGM Exporter provide GPU health and performance evidence. That evidence can support investigation and operational decisions, but it does not automatically determine model quality or trigger retraining.
6. **D:** TensorRT is the NVIDIA inference SDK used to optimize trained models for efficient execution on supported NVIDIA GPUs. RAPIDS focuses on accelerated data science, DCGM Exporter exposes telemetry, and NGC provides software and artifact distribution services.

Summary

In this chapter, you treated MLOps as the discipline that makes model-lifecycle changes repeatable and traceable. Lineage connects a deployed model to its tracked inputs and environment.

You used CI/CD for repeatable checks and controlled promotion, and treated event-driven retraining as a trigger rather than automatic deployment.

Finally, you mapped representative NVIDIA technologies to lifecycle roles and connected monitoring evidence to controlled change. *Chapter 12* shifts to NCA-AIIO preparation.

12

Preparing for NCA-AIIO

Chapter 11, Recognizing MLOps Practices in the AI Lifecycle, completed the technical journey by connecting tracked changes, validation, deployment, monitoring, and feedback. At this point, the task changes. You no longer need another layer of technology. You need a reliable way to retrieve what you already learned, distinguish similar concepts under time pressure, and recognize which part of the stack a scenario is actually testing.

That is the purpose of this chapter. We will return to the current NCA-AIIO blueprint, map every objective to its canonical chapter, review the comparisons that are easiest to blur together, and build a final revision process around evidence rather than confidence alone. The goal is not to memorize this book. It is to draw on the reasoning developed throughout the book when a short exam scenario gives you only a few clues.

Certification details are version-sensitive. The blueprint and logistics in this chapter were verified against NVIDIA's official certification page and January 2026 study guide while the chapter was prepared in October 2026. Recheck the official page before scheduling the exam, especially if you are reading a later edition or using this chapter long after publication.

In this chapter, we will cover the following topics:

- Rechecking the current blueprint and exam logistics
- Reviewing Essential AI Knowledge
- Reviewing AI Infrastructure
- Reviewing AI Operations
- Recognizing high-value comparisons and terminology traps
- Reading scenario questions systematically
- Planning targeted revision and closing weak areas
- Working through a mixed-domain practice set

By the end of this chapter, you will be able to map each current NCA-AIIO objective to its primary chapter, identify the distinction a scenario is testing, eliminate answers that solve the wrong layer or overstate a product capability, and turn practice results into a focused final revision plan.

Technical requirements

This chapter is a review and planning chapter. You do not need a GPU, a cloud account, or access to an AI cluster. You do need the evidence you created while working through the book and access to the current NVIDIA certification material.

- The completed chapters and their review questions, especially any questions you previously answered incorrectly or with low confidence
- The current NCA-AIIO certification page and exam study guide so you can verify version-sensitive logistics and objective wording
- A notebook, document, or spreadsheet for a weak-area tracker and final revision plan
- A timer for the mixed-domain practice in this chapter and the full mock exams in *Appendix A*

Use the official NCA-AIIO page as the source of truth for current logistics and policy details: `https://www.nvidia.com/en-us/learn/certification/ai-infrastructure-operations-associate/`

Note

Exam focus: This chapter spans all three NCA-AIIO domains. Use the published weights to guide attention, but do not treat a percentage as permission to ignore any objective. The exam blueprint is the boundary; the chapter map tells you where to revise each part of it.

Rechecking the current blueprint and exam logistics

Before final preparation, confirm that the blueprint and exam logistics are current. A study plan can be well organized yet miss the target if the objectives have changed.

At the time this chapter was prepared, NVIDIA described NCA-AIIO as an entry-level associate certification that validates foundational AI infrastructure and operations knowledge. The exam was listed as online and remotely proctored, with 50 questions and a 60-minute time limit. The official page listed a price of US$125, English as the exam language, basic data center infrastructure knowledge as a prerequisite, and a two-year certification validity period. Treat these details as a dated snapshot and verify them again before registering.

Table 12.1 summarizes the current logistics and suggests preparation actions.

Item	Verified detail	Preparation action
Certification level	Associate; entry-level/ foundational	Practice recognizing and comparing concepts before studying implementation details.
Delivery	Online, remotely proctored	Read the current exam and proctoring policy before scheduling.
Questions and time	50 questions; 60 minutes	Use timed mixed practice to avoid spending too long on one difficult item.
Price	US$125	Verify current regional registration details before purchase.
Language	English	Practice reading short technical scenarios and distinguishing product roles precisely.
Prerequisite	Basic understanding of data center infrastructure	Refresh facility, server, storage, and networking fundamentals where they affect an objective.
Validity	Two years from issuance	Treat recertification rules as policy details to recheck when needed.

Table 12.1: Current NCA-AIIO exam snapshot and preparation action

The blueprint is more durable than the logistics because it defines what the exam is intended to measure. It currently contains 22 objectives distributed across three domains: Essential AI Knowledge at 38%, AI Infrastructure at 40%, and AI Operations at 22%. Those percentages describe emphasis across the exam; they do not turn the objectives into independent silos. A networking scenario can depend on an accelerated-compute requirement, and an operations question can require you to know which software layer provides the evidence.

Figure 12.1 groups related exam objectives into broad revision clusters and points to the chapters most useful for a first review pass. The numbers shown at the top of each card (for example, **1.1** or **1.2–1.5**) are objective numbers from the exam blueprint, and **Ch./Chs.** indicate the chapter or chapters to review for that cluster. This figure is intentionally a high-level navigation aid, not an exact one-to-one objective map; *Tables 12.2* through *12.4* provide the detailed objective-level chapter mapping.

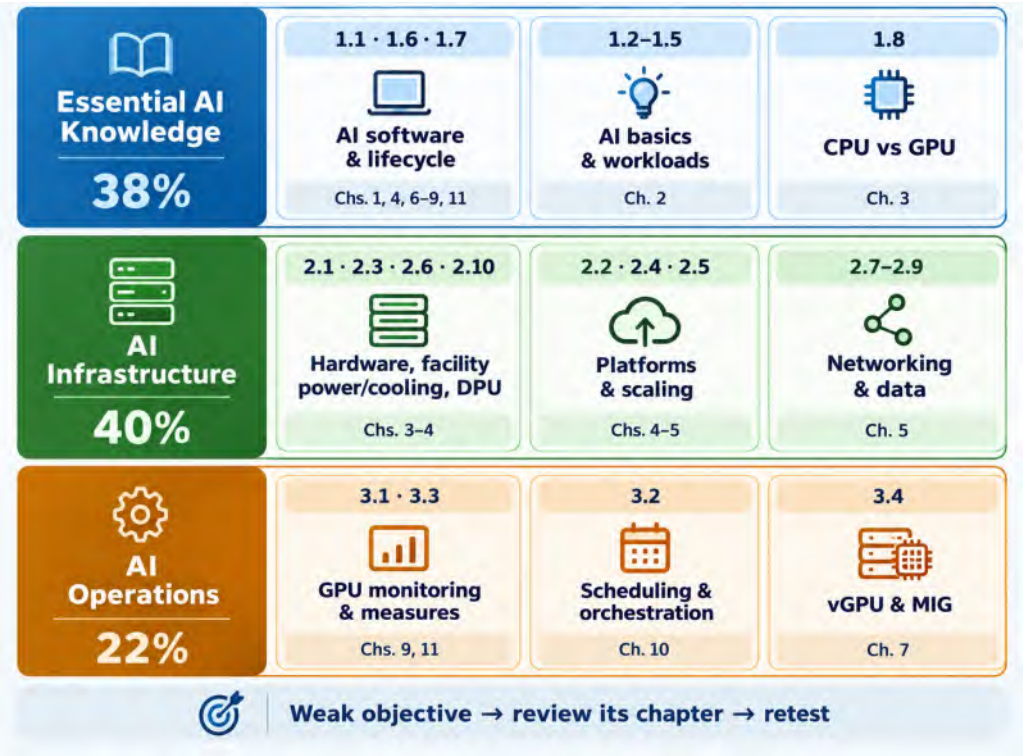


Figure 12.1: NCA-AIIO domain and chapter revision map

Use the map to locate the chapter for a weak objective, then retest after revision. Classify practice mistakes by objective before deciding what to review.

Reviewing Essential AI Knowledge

Essential AI Knowledge accounts for 38% of the published blueprint and contains eight objectives. The word knowledge can make this domain sound like vocabulary recall, but several objectives are comparative. You need to recognize not only what a component is, but also why one architecture, software role, or lifecycle responsibility differs from another.

Table 12.2 lists recall cues and the chapters where each concept is developed.

Objective	Recall cue	Primary chapters
1.1	Place driver, CUDA, libraries, frameworks, and higher-level software in the correct stack relationship.	*1, 6, 8*
1.2	Compare training and inference by workload behavior and infrastructure priorities rather than by rigid product rules.	*2*
1.3	Differentiate AI, machine learning, and deep learning by scope and relationship.	*2*
1.4	Connect modern AI adoption to data growth, compute capability, and algorithmic progress.	*2*
1.5	Recognize AI use cases across industries and the workload characteristics relevant to infrastructure.	*2*
1.6	Choose representative NVIDIA solutions by the responsibility they address, not by product-name familiarity.	*4, 7, 8, 9, 11*
1.7	Map data preparation, training, optimization, deployment, monitoring, and lifecycle practices to software roles.	*6, 8, 11*
1.8	Compare CPU and GPU architectures by execution model and workload fit.	*3*

Table 12.2: Essential AI Knowledge revision map

Build relationships, not isolated definitions

The most useful review focuses on relationships. For objective 1.1, a driver does not replace CUDA, CUDA does not replace a framework, and a serving layer does not become a training framework merely because all of them appear in an AI software environment. Reconstruct the dependency or responsibility chain from memory and explain what would be missing if one layer were removed.

The same approach works for objectives 1.2 and 1.8. Do not reduce training versus inference to "training uses more GPUs" or CPU versus GPU to "GPU is faster." Ask what kind of parallelism exists, where latency or throughput matters, how memory and data movement affect the

workload, and whether the scenario is asking about one request, a large batch, or distributed execution. The correct comparison follows the workload.

Use lifecycle responsibility to control product-name confusion

Objectives 1.6 and 1.7 contain the largest concentration of NVIDIA product names, making it easy to focus on memorizing product names rather than understanding their roles. A better revision method begins with the responsibility: data preparation, model development, inference optimization, serving, artifact distribution, monitoring, or lifecycle control. Only then attach representative software.

For example, TensorRT and NVIDIA Dynamo-Triton (formerly Triton Inference Server) can appear next to each other in an inference workflow, but they answer different questions. TensorRT optimizes model execution for inference, while Dynamo-Triton provides a serving layer for inference workloads. NVIDIA NIM provides containerized, API-accessible inference microservices. RAPIDS belongs to accelerated data science and preparation workflows. DCGM belongs to GPU monitoring and management rather than model optimization. If you can state the responsibility before the product name, distractors become easier to recognize.

Once these eight objectives are stable, move to the largest domain: the infrastructure that supplies power, compute, storage, networking, and scale.

Reviewing AI Infrastructure

AI Infrastructure carries 40% of the current blueprint and contains ten objectives. This domain is broad because an accelerated workload depends on more than the GPU. Facility constraints, platform design, scale, deployment location, storage and data movement, networking, and DPU roles all contribute to whether the workload can run as intended.

Use *Table 12.3* as a fast objective check. Each cue points back to the detailed explanations in *Chapters 3 through 5*; it does not replace them.

Objective	Recall cue	Primary chapters
2.1	Match training-task requirements to compute, memory, storage, networking, and platform capability.	*3, 4*
2.2	Recognize how GPU infrastructure scales and which system resources must scale with compute.	*4*

Objective	Recall cue	Primary chapters
2.3	Connect rack density and sustained load to power delivery, cooling, and facility limits.	*3*
2.4	Compare on-premises and cloud choices by control, access, capacity, economics, and operational constraints.	*4*
2.5	Recognize the components and considerations that make an accelerated cluster operate as a system.	*4, 5*
2.6	Identify facility requirements that support accelerated infrastructure.	*3*
2.7	Derive network requirements from workload communication and data-movement behavior.	*5*
2.8	Recognize data center networking protocols and concepts, including RDMA-related responsibilities.	*5*
2.9	Distinguish high-speed network options and the use cases they support.	*5*
2.10	Explain how a DPU can offload, accelerate, and isolate infrastructure services from the host CPU.	*3*

Table 12.3: AI Infrastructure revision map

Review the system path from facility to workload

A useful way to retrieve this domain is to walk the dependency path upward. The facility supplies power and cooling. The server supplies CPUs, GPUs, memory, local connectivity, and platform management. Storage and networks keep data moving. Scale-up technologies connect accelerators within a system or tightly coupled domain, while scale-out networks connect nodes and services across the data center. Software then uses those resources on behalf of the workload.

This path helps with scenario questions because the cause may lie outside the component showing the symptom. A GPU that cannot sustain expected performance may be waiting for data, constrained by power, limited by thermals, or participating in an inefficient communication path. NCA-AIIO is an associate-level exam, so the goal is not to perform a

professional-level root-cause analysis. It is to identify the layer and the class of requirement that should be checked next.

Keep network terms in the correct category

Networking terminology creates a predictable source of confusion because several names can coexist in one architecture. Ethernet and InfiniBand are network technologies. RoCE is a way to carry RDMA semantics over Ethernet. RDMA describes direct-memory-access behavior across the network, while GPUDirect RDMA extends the data path so supported devices can exchange data with GPU memory more directly. NCCL is not another fabric; it is a collective-communication library that can use available data paths.

Draw the network layers from memory, then check that each term sits in the correct category before attempting more questions.

Distinguish scale from completeness

Objective 2.2 can also tempt you into a GPU-count answer. Adding GPUs increases potential compute, but useful scaling also depends on removing the bottleneck identified in the scenario. Address that constraint rather than increasing device count alone.

With the infrastructure relationships refreshed, the remaining domain asks how the environment is operated: how resources are shared, scheduled, observed, and managed.

Reviewing AI Operations

AI Operations accounts for 22% of the published blueprint and has four objectives. The smaller weight does not make the domain trivial. These objectives contain several tools whose boundaries are easy to blur, especially when monitoring, scheduling, orchestration, and virtualization appear together in one operational scenario.

Objective	Recall cue	Primary chapters
3.1	Separate immediate inspection, telemetry, monitoring, visualization, and broader infrastructure-management responsibilities.	*9, 11*
3.2	Distinguish job scheduling from container orchestration and recognize the roles of Slurm and Kubernetes.	*10*

Objective	Recall cue	Primary chapters
3.3	Recognize key GPU measures such as utilization, memory, temperature, power, processes, health, and related evidence.	*9*
3.4	Compare accelerated-infrastructure virtualization choices, especially vGPU and MIG, by isolation and resource-sharing goals.	*7*

Table 12.4: AI Operations revision map

Choose monitoring tools by responsibility

For objective 3.1, start with scope. `nvidia-smi` is useful for immediate host-level inspection. DCGM provides NVIDIA GPU telemetry, health, diagnostics, and management capabilities for data-center GPUs. DCGM Exporter exposes GPU telemetry in Prometheus format for collection. Prometheus handles time-series collection and can support alerting; Grafana visualizes the resulting data. Base Command Manager belongs at a broader cluster and infrastructure management layer.

Do not choose a monitoring tool simply because it is NVIDIA-branded; match its scope to the requested outcome.

Separate job scheduling from continuous orchestration

For objective 3.2, remember the control model from *Chapter 10*. A job scheduler such as Slurm admits and queues jobs, allocates resources, and tracks their completion. Kubernetes includes scheduling, but its broader orchestration model continuously reconciles declared workload state. Either platform can support more than one workload pattern, so reject answers based on a rigid rule such as "training always means Slurm" or "inference always means Kubernetes."

Keep virtualization terms tied to the sharing goal

For objective 3.4, MIG and vGPU both help share accelerated infrastructure, but they solve different problems. MIG partitions supported physical GPUs into hardware-isolated GPU instances with defined compute and memory resources. NVIDIA vGPU is a virtualization solution that exposes GPU capability to virtual machines through a supported virtualization stack. The right answer depends on the required environment, sharing model, isolation, management layer, and supported configuration. Remember, a supported NVIDIA GPU can be partitioned into up to seven MIG instances, depending on the GPU model.

The three domain reviews give you the coverage map. The next step is to compress the most error-prone distinctions into one final comparison pass.

Recognizing high-value comparisons and terminology traps

Comparisons are especially useful when two answers sound plausible until you identify the relevant responsibility or layer. *Table 12.5* collects the distinctions that recur across this book. Read the middle column first, then use the last column as a self-test: can you explain why the trap is wrong without looking up the answer?

Comparison	Key distinction	Common trap to avoid
AI / ML / deep learning	Nested concepts with different scope; deep learning is a subset of machine learning, which is part of the broader AI field.	Using the terms as interchangeable labels.
Training / inference	Training updates model parameters; inference applies a trained model to inputs. Infrastructure priorities depend on workload behavior.	Assuming one fixed hardware pattern for every training or inference workload.
CPU / GPU / DPU	CPU provides general-purpose control and serial performance; GPU emphasizes parallel throughput; DPU offloads and isolates infrastructure services.	Treating the DPU as a replacement for the application CPU or GPU.
On-premises / cloud	Different control, access, capacity, cost, security, and operational trade-offs.	Assuming one deployment model is always cheaper or always faster.
Ethernet / InfiniBand / RoCE	Ethernet and InfiniBand are network technologies; RoCE carries RDMA over Ethernet.	Treating RoCE as an unrelated third physical-fabric category.
PCIe / NVLink / RDMA / NCCL	Data paths and interconnects move data; NCCL is collective-communication software that uses available paths.	Treating NCCL as a network fabric or switch technology.

Comparison	Key distinction	Common trap to avoid
Driver-supported CUDA / CUDA Toolkit	The **CUDA Version** field in `nvidia-smi` reflects the latest CUDA version supported by the installed driver; it does not prove that a CUDA Toolkit is installed.	Using the `nvidia-smi` CUDA field as evidence that the CUDA Toolkit is installed.
vGPU / MIG	vGPU supports GPU access through virtualization; MIG partitions supported GPUs into hardware-isolated instances.	Treating them as interchangeable names for GPU sharing.
`nvidia-smi` / DCGM / broader cluster management	Immediate inspection, data-center GPU telemetry/health, and cluster management have different scopes.	Selecting the broadest tool simply because the environment has many GPUs.
DCGM / DCGM Exporter / Prometheus / Grafana	Telemetry and diagnostics, metric export, collection/retention/alerting, and visualization are separate responsibilities.	Assigning history, alerting, or dashboards directly to DCGM without the surrounding stack.
Slurm / Kubernetes	Job scheduling and desired-state orchestration use different control models, with overlap in supported workloads.	Applying a rigid training-versus-inference split.
TensorRT / Dynamo-Triton / NIM	Inference optimization, inference serving, and packaged deployment-focused inference microservices are distinct roles.	Choosing by the word inference without identifying the requested responsibility.
Monitoring / MLOps retraining	Monitoring creates evidence; evaluation and policy determine whether retraining or promotion should occur.	Assuming a metric change automatically retrains and deploys a model.

Table 12.5: High-value comparisons and common terminology traps

Use the traps as self-tests, not answers to memorize. The six-step method that follows shows how to apply these distinctions to scenarios.

Reading scenario questions systematically

A disciplined reading method helps you evaluate unfamiliar technical scenarios without relying on familiar product names.

Figure 12.2 summarizes the flow. It begins with the requested outcome, not with the longest technical noun in the scenario.

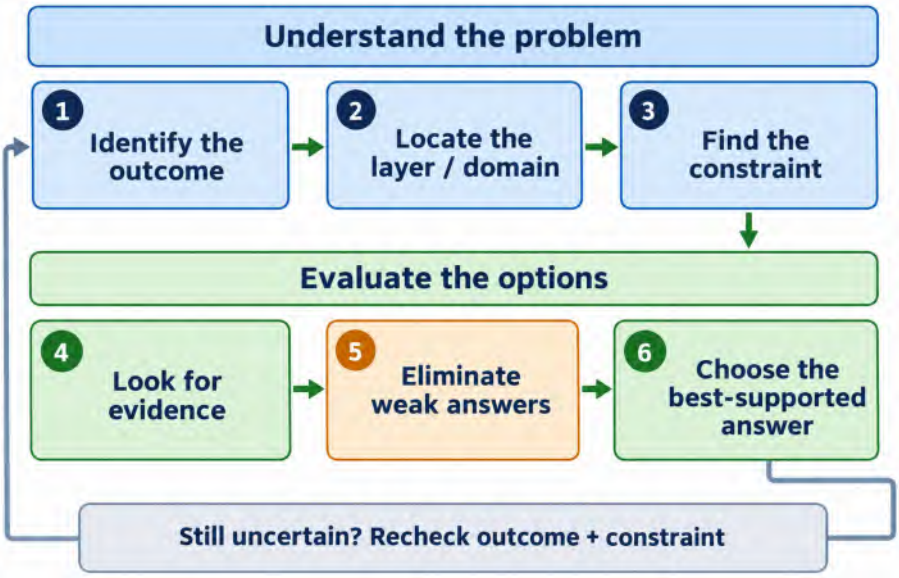


Figure 12.2: Scenario-question reasoning flow

Step 1: Identify the requested outcome

Start with the final ask. Is the scenario asking you to monitor GPUs, select a data path, share a GPU, choose a serving component, identify a facility constraint, or distinguish training from inference? This establishes the responsibility that the answer must satisfy.

Step 2: Locate the layer and domain

Place the outcome in the model you used throughout the book. A temperature or power problem is not automatically a model problem. A model-serving requirement is not automatically a scheduler problem. A slow distributed workload can involve compute, storage, networking, communication libraries, or several layers, so use the wording to identify which layer the question actually gives you evidence about.

Step 3: Extract the constraint

Look for the detail that removes otherwise reasonable choices: virtual machines, hardware-isolated GPU partitions, remotely shared infrastructure, low-latency communication, immediate host inspection, long-term metrics, desired-state reconciliation, or a need to optimize model execution. The constraint is often more useful than the product names.

Step 4: Match the evidence to the responsibility

Ask what the proposed answer would actually do. If the requirement is to visualize retained metrics, a diagnostic tool alone is incomplete. If the requirement is to optimize inference execution, a serving layer alone solves a neighboring problem. If a scenario asks whether a change caused a model-quality regression, GPU utilization is evidence about infrastructure behavior, not proof about model quality.

Step 5: Eliminate wrong-layer and overclaim answers

Two distractor patterns recur in technical certification material even when the exact wording changes. A wrong-layer answer names a real technology that solves a different problem. An overclaim starts with a real capability but exaggerates its scope: monitoring that "guarantees" model accuracy, a scheduler that "automatically" uses arbitrary utilization telemetry, or one tool that supposedly replaces every surrounding layer.

Step 6: Choose the answer that satisfies the scenario as written

Do not improve the scenario in your head. If the question asks for immediate inspection, do not choose a larger management platform because it would be useful in production. If it asks for a foundational comparison, do not reject the intended answer because a specialized implementation could create an exception. Use the stated scope, then move on.

This method also makes your practice notes more specific. Instead of recording only "Question 14 wrong," you can record "misclassified monitoring scope" or "confused network fabric with communication library." That is the bridge from practice to revision.

Readiness planning and targeted revision

A final study plan should be driven by evidence of gaps, not by equal time per chapter. You already created the raw material while reading: review-question results, diagrams, command outputs where applicable, comparison notes, and areas where you had to reread an explanation. Convert that evidence into the next action.

Maintain a weak-area tracker

A useful tracker identifies gaps that can be addressed. "Networking weak" is not actionable. "I confuse RoCE with a separate fabric and cannot explain where RDMA sits" points to one section, one diagram, and one retest. *Table 12.6* shows the structure.

Topic or objective	Evidence of gap	Next action	Checkpoint
1.2 Training vs inference	Can define both but choose hardware by a rigid rule.	Redo the *Chapter 2* comparison from workload requirements and explain two scenarios aloud.	Retest in 48 hours
2.8-2.9 Networking	Confuse RoCE, RDMA, InfiniBand, and NCCL categories.	Redraw the *Chapter 5* data-path stack from memory and answer the network review questions.	End of infrastructure review
3.1 Monitoring	Select DCGM for every monitoring task.	Rebuild the `nvidia-smi` / DCGM / Exporter / Prometheus / Grafana responsibility chain.	Before mixed practice

Table 12.6: Weak-area tracker example

Use measurable evidence of readiness

Confidence is useful but difficult to audit. Replace it with observable outputs. You should be able to draw the major stack relationships without notes, explain a comparison in your own words, identify the canonical chapter for an objective, and justify why each distractor in a practice question is wrong. A topic is not ready because it looks familiar; it is ready when you can retrieve and apply it.

A practical checkpoint for each weak area is three passes: explain the concept without notes, apply it to a short scenario, and repeat the check after a delay. If you cannot explain the concept two days later, familiarity has not become reliable recall. Return to the canonical chapter and rebuild the relationship instead of rereading the whole book.

Build the final two-week pass around gaps

If you have roughly two weeks before the exam, use the first part for domain strengthening and the second for integration. One workable pattern is to spend the first four days on Essential AI Knowledge and its product-role boundaries, the next four on AI Infrastructure, and the following two on AI Operations. Use the remaining days for mixed questions, weak-area retests, and the *Appendix A* mock exams. Adjust the allocation when the tracker shows a different need; the domain weights guide attention, but your errors should control the final schedule.

Reserve the day before the exam for consolidation rather than first exposure to a major topic. Recheck current exam logistics, review the comparison table, revisit only unresolved tracker items, and stop adding new sources simply because they exist. The goal is reliable recall, not maximum reading volume.

Use mock exams as measurement, not prediction

Appendix A contains the full original mock exams. Their purpose is to expose gaps across the finalized book coverage; they are not copies of live NVIDIA questions and they cannot predict the exact composition of an exam attempt. Take them under consistent conditions, record why you missed each item, revise the relevant objective, and retest the concept with a different question or explanation.

Before moving to the full mocks, use the shorter set below as a mixed-domain check. Treat each item as a reasoning exercise: state the outcome, layer, constraint, and evidence before selecting an option.

Mixed-domain practice set

Answer the following eight questions without referring to the chapter tables. Each tests a relationship covered in this guide.

Question 1

A team is choosing infrastructure for two workloads. Workload A repeatedly updates billions of model parameters over a large dataset. Workload B applies a fixed trained model to interactive requests where response time matters. Which distinction should guide the comparison?

A. Both workloads should be treated as identical because they use the same model family

B. Assume inference always needs more GPUs because production requests are latency-sensitive

C. Compare training and inference by throughput, latency, memory, and communication needs

D. Use CPUs for training because parameter updates are control-plane operations

Question 2

A new rack of accelerated servers runs briefly at full load, but performance repeatedly drops when all systems remain busy. GPU health checks show no device faults. Which area should be investigated first?

A. Power delivery and cooling capacity for the sustained rack load

B. Model-registry metadata for the workloads running on the rack

C. Whether every workload is packaged as a Kubernetes service

D. Whether the CUDA Toolkit version matches the `nvidia-smi` header

Question 3

An Ethernet-based AI cluster needs RDMA semantics for low-overhead data movement between nodes. Which option most directly describes that requirement?

A. NCCL, because it is the physical fabric used for GPU communication

B. NVLink, because it replaces the data-center network between all nodes

C. InfiniBand only, because RDMA cannot operate over Ethernet

D. RoCE, which carries RDMA over an Ethernet network

Question 4

An administrator runs `nvidia-smi` and sees a **CUDA Version** value in the header. What can be concluded from that field alone?

A. The matching CUDA Toolkit, compiler, and development libraries are installed and ready for use

B. The field reports the driver's CUDA support level; CUDA Toolkit installation must be checked separately

C. All containers on the host use the same CUDA Toolkit version reported by `nvidia-smi`

D. A CUDA application has already run successfully with the Toolkit version shown

Question 5

A team needs NVIDIA GPU telemetry across a cluster for a Prometheus-based monitoring pipeline. Which component most directly exposes those metrics for collection?

A. Grafana, because it collects GPU metrics directly from device drivers

B. Slurm, because job scheduling automatically exports GPU health metrics

C. DCGM Exporter, because it exposes DCGM-backed GPU telemetry for collection

D. TensorRT, because inference optimization includes GPU monitoring

Question 6

Several teams need isolated portions of a supported physical GPU for separate workloads on the same server without placing each workload in a different virtual machine. Which technology most directly fits the stated sharing goal?

A. GPUDirect RDMA, because it partitions GPU compute and memory into isolated instances

B. NVIDIA vGPU, because every shared GPU workload requires a virtual machine

C. DCGM, because monitoring groups create isolated execution partitions

D. MIG, because it partitions supported GPUs into hardware-isolated GPU instances

Question 7

A platform must continuously maintain a declared set of containerized services, replace failed instances, and reconcile actual state with desired state. Which control model best matches the requirement?

A. Kubernetes-style container orchestration with continuous reconciliation

B. Slurm-style finite job scheduling only, because every container is a batch job

C. DCGM-based monitoring, because telemetry determines the desired application state

D. NCCL collective communication, because service replicas require synchronization

Question 8

Monitoring shows that a deployed model service is experiencing a change in behavior after a new model version was promoted. What is the best MLOps response?

A. Retrain and redeploy automatically because any monitoring change proves model drift

B. Evaluate the evidence, then decide whether rollback, correction, or retraining is warranted

C. Increase GPU allocation first because every model-quality issue is a capacity problem

D. Remove version tracking so the new model can be treated as the current baseline

Before checking the answers, classify each question by domain and objective. If you chose the correct option for the wrong reason, record the concept as a weak area rather than counting it as fully ready.

Answers and explanations

1. **C:** Training updates model parameters while inference applies a trained model. Compare the workloads by their execution goals and by requirements such as throughput, latency, memory, data movement, and communication rather than by a fixed product rule.
2. **A:** The symptom appears only under sustained rack load and the GPUs report no device fault. Power delivery and cooling are therefore the first infrastructure layer to examine. Registry metadata, orchestration packaging, and the `nvidia-smi` CUDA header do not explain a rack-wide sustained-load limit.
3. **D:** RoCE provides RDMA over an Ethernet network. NCCL is collective-communication software, NVLink is a high-speed GPU interconnect used within supported systems and topologies, and InfiniBand is another high-speed network technology rather than the only way to use RDMA semantics.

4. **B:** The **CUDA Version** field shown by `nvidia-smi` reflects the latest CUDA version supported by the installed driver. It does not confirm that a CUDA Toolkit, compiler, or development libraries are installed, or that a particular application will run; compatibility requires separate checks.
5. **C:** DCGM Exporter exposes DCGM telemetry in Prometheus format for collection. Grafana visualizes data, Slurm schedules work, and TensorRT optimizes inference.
6. **D:** MIG partitions supported physical GPUs into hardware-isolated instances with defined compute and memory resources. vGPU addresses virtualization; the scenario asks for hardware partitioning without requiring virtual machines.
7. **A:** Kubernetes continuously reconciles actual state with declared state, so this is desired-state orchestration rather than one-time job placement.
8. **B:** Monitoring provides evidence, not an automatic retraining decision. Evaluate the evidence and lineage, then decide whether rollback, correction, retraining, or another action is warranted.

Summary

In this chapter, you returned to the current NCA-AIIO blueprint and mapped all 22 objectives to the chapters that cover them. Essential AI Knowledge asks you to recognize foundational concepts and software roles, AI Infrastructure asks you to reason across facility, compute, platform, storage, networking, and DPU responsibilities, and AI Operations asks you to distinguish how accelerated environments are virtualized, monitored, scheduled, and orchestrated.

You also reviewed the most easily confused technologies in a comparison table and used a scenario-reading method built around outcome, layer, constraint, evidence, and distractor elimination. That method is valuable because it prevents a familiar product name from pulling you toward the wrong responsibility.

Finally, you used practice results to build a weak-area tracker and targeted revision plan. *Appendix A* contains the objective map and mock exams; *Appendix B* lists resources for checking version-sensitive facts.

13

Unlock Your Exclusive Benefits

Your copy of this book includes the following exclusive benefits:







Follow the guide below to unlock them. The process takes only a few minutes and needs to be completed once.

Unlock this Book's Free Benefits in 3 Easy Steps

Step 1

Keep your purchase invoice ready for *Step 3*. If you have a physical copy, scan it using your phone and save it as a PDF, JPG, or PNG.

For more help on finding your invoice, visit `https://www.packtpub.com/en-us/unlock?step=1`.

> **Note**
>
> If you bought this book directly from Packt, no invoice is required. After *Step 2*, you can access your exclusive content right away.

Step 2

Scan the QR code or go to `packtpub.com/unlock`.

On the page that opens (similar to *Figure 13.1* on desktop), search for this book by name and select the correct edition.

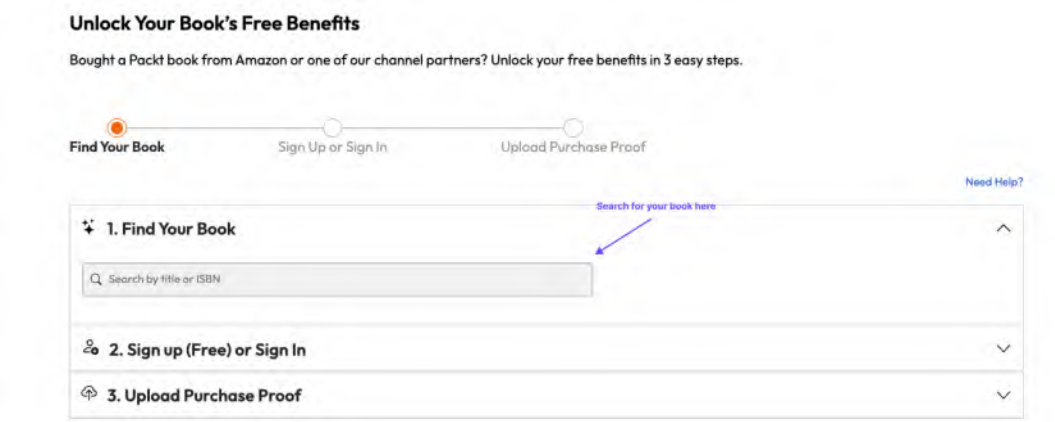


Figure 13.1: Packt unlock landing page on desktop

Step 3

After selecting your book, sign in to your Packt account or create one for free. Then upload your invoice (PDF, PNG, or JPG, up to 10 MB). Follow the on-screen instructions to finish the process.

Need Help

If you get stuck and need help, visit `https://www.packtpub.com/unlock-benefits/help` for a detailed FAQ on how to find your invoices and more. This QR code will take you to the help page.

Note

If you are still facing issues, reach out to `customercare@packt.com`.

Appendix A

NCA-AIIO Objective Map and Mock Exams

This appendix converts the current NVIDIA-Certified Associate: AI Infrastructure and Operations (NCA-AIIO) blueprint into a final revision map and two original practice exams. Use it after completing Chapters 1–12. The questions are designed to test the concepts and relationships taught in this guide; they are not copied from, reconstructed from, or claimed to represent live NVIDIA exam questions.

At the time of final verification in October 2026, NVIDIA listed 50 questions in 60 minutes, with domain weights of 38% Essential AI Knowledge, 40% AI Infrastructure, and 22% AI Operations. Certification details can change, so verify the official certification page before scheduling or sitting the exam.

> **Note**
>
> Practice design: Each mock contains 50 four-option questions with one best answer. The domain totals mirror the published 38/40/22 weighting, but the objective-level question counts are an editorial study design, not a statement about NVIDIA's internal item distribution.

Objective-to-chapter map

Use this table to move from an incorrect answer to the chapter that owns the concept. Chapter 12 provides the final domain-level review, while the primary chapters listed here contain the canonical teaching.

Objective	Official blueprint wording	Primary chapter(s)	Secondary review
1.1	Describe the NVIDIA software stack used in an AI environment.	Chapters 1, 6, 8	Table 12.2; Figure 12.1; chapter review questions
1.2	Compare and contrast training and inference architecture requirements and considerations.	Chapter 2	Table 12.2; Figure 12.1; Chapter 2 review
1.3	Differentiate the concepts of AI, machine learning, and deep learning.	Chapter 2	Table 12.2; Chapter 2 review
1.4	Explain the factors contributing to recent rapid improvements and adoption of AI.	Chapter 2	Table 12.2; Chapter 2 review
1.5	Explain the key AI use cases and industries.	Chapter 2	Table 12.2; Chapter 2 review
1.6	Explain the purpose and use case of various NVIDIA solutions.	Chapters 4, 7, 8, 9, 11	Table 12.2; Figure 12.1; relevant chapter reviews
1.7	Describe the software components related to the life cycle of AI development and deployment.	Chapters 6, 8, 11	Table 12.2; Chapters 8 and 11 reviews
1.8	Compare and contrast GPU and CPU architectures.	Chapter 3	Table 12.2; Chapter 3 review
2.1	Identify hardware requirements for specific AI training task use cases.	Chapters 3, 4	Table 12.3; Chapters 3-4 reviews
2.2	Scale a GPU infrastructure for different use cases.	Chapter 4	Table 12.3; Chapter 4 review

Objective	Official blueprint wording	Primary chapter(s)	Secondary review
2.3	Identify key concepts and high-level specifications related to power and cooling requirements within a data center.	Chapter 3	Table 12.3; Chapter 3 review
2.4	Articulate the key advantages, challenges, and considerations related to on-prem vs cloud infrastructures.	Chapter 4	Table 12.3; Chapter 4 review
2.5	Identify key components and considerations of a cluster of an accelerated infrastructure.	Chapters 4, 5	Table 12.3; Chapters 4-5 reviews
2.6	Identify facility requirements.	Chapter 3	Table 12.3; Chapter 3 review
2.7	Determine networking requirements for AI workloads.	Chapter 5	Table 12.3; Chapter 5 review
2.8	Identify and describe data center networking protocols and key concepts.	Chapter 5	Table 12.3; Chapter 5 review
2.9	Identify high-speed data center network options and their use cases.	Chapter 5	Table 12.3; Chapter 5 review
2.10	Explain the purpose and benefits of a DPU in a data center.	Chapter 3	Table 12.3; Chapter 3 review
3.1	Describe AI data center management and monitoring essentials.	Chapters 9, 11	Table 12.4; Chapters 9 and 11 reviews
3.2	Describe AI cluster orchestration and job scheduling essentials.	Chapter 10	Table 12.4; Chapter 10 review

Objective	Official blueprint wording	Primary chapter(s)	Secondary review
3.3	Articulate the key measures and criteria related to monitoring GPUs.	Chapter 9	Table 12.4; Chapter 9 review
3.4	Identify the key considerations for virtualizing accelerated infrastructure.	Chapter 7	Table 12.4; Chapter 7 review

Table A.1: NCA-AIIO objective-to-chapter revision map

Mock-exam weighting and final review cues

Domain	Published weight	Questions per mock	Coverage
Essential AI Knowledge	38%	19	Objectives 1.1–1.8
AI Infrastructure	40%	20	Objectives 2.1–2.10
AI Operations	22%	11	Objectives 3.1–3.4

Table A.2: Domain distribution used in each mock exam

Comparison	Revision cue
Training vs inference	Training updates parameters; inference applies trained parameters. Compare throughput, latency, memory, and communication without rigid platform rules.
CPU vs GPU vs DPU	CPU: versatile general-purpose control; GPU: parallel accelerated compute; DPU: infrastructure-service offload.
Driver vs CUDA Toolkit	The driver provides hardware access and CUDA compatibility support; a displayed supported CUDA version does not prove the Toolkit is installed.

Comparison	Revision cue
NCCL vs network fabric	NCCL is collective-communication software; Ethernet/InfiniBand/RoCE provide underlying network/transport capabilities.
vGPU vs MIG	vGPU supports virtualized GPU access in VM environments; MIG partitions supported GPUs into isolated instances.
nvidia-smi vs DCGM	nvidia-smi is immediate host inspection; DCGM provides structured GPU telemetry, health monitoring, policy, and diagnostics.
DCGM Exporter vs Prometheus vs Grafana	Exporter exposes metrics, Prometheus collects/stores them, Grafana visualizes them.
Slurm vs Kubernetes	Slurm emphasizes queued jobs and resource scheduling; Kubernetes emphasizes container orchestration and desired-state reconciliation.
TensorRT vs Dynamo-Triton vs NIM	TensorRT optimizes inference, Dynamo-Triton serves models, NIM provides packaged deployment-focused inference microservices.
Monitoring vs retraining	Monitoring supplies evidence. It does not automatically prove drift or justify retraining/deployment without evaluation.

Table A.3: High-value comparison cues before attempting the mocks

How to attempt the mock exams

- Take each mock in one uninterrupted 60-minute session if possible. This mirrors the current published time budget without implying that the practice questions reproduce the live exam format.
- Answer every question on the first pass. Mark uncertain items rather than spending several minutes on a single question.

- When two options look plausible, identify the requested outcome, the layer or domain, the limiting constraint, and the evidence given. Eliminate answers that solve the wrong layer or overstate a product capability.
- After the attempt, score by objective as well as by total. A correct answer reached by guessing should remain in the weak-area tracker.

Mock Exam 1

Questions: 50 | Suggested time: 60 minutes | Choose the one best answer for each question. Do not look at the answer section until the attempt is complete.

1. A containerized CUDA application starts on a newly provisioned server but cannot access any GPU device. Which host-side prerequisite should be verified first?

A. A compatible NVIDIA GPU driver is installed and loaded on the host

B. NVIDIA Dynamo-Triton is running as a system service

C. Grafana has a dashboard for the server

D. A CUDA compiler is installed inside every application container

2. A large training job cannot fit its model state into the memory of one accelerator and must distribute work across several GPUs. Which hardware consideration is most directly relevant?

A. Higher CPU clock speed while accelerator memory remains unchanged

B. More local disk capacity without changing accelerator memory or communication

C. A faster management network with the same accelerator resources

D. Sufficient aggregate GPU memory and a fast inter-GPU communication path, with fast inter-node links if the job spans nodes

3. A user reports that one GPU server is slow right now. You need an immediate view of GPU utilization, memory use, temperature, and running processes on that host. Which tool is the most direct first check?

A. TensorRT

B. nvidia-smi

C. NGC Catalog

D. Slurm

4. A team compares a model-training service with an interactive inference service. Which architectural distinction is the most useful starting point?

A. Training updates model parameters, while inference applies trained parameters to new inputs

B. Inference requires parameter synchronization after every request

C. Inference always needs more GPUs than training

D. Training is primarily a latency-optimization problem for individual requests

5. Multi-node training scales poorly because gradient exchanges dominate iteration time even though individual GPUs are not saturated. Which network characteristic should receive the most attention?

A. North-south bandwidth to end-user web clients only

B. Lower storage capacity to reduce network traffic

C. High east-west bandwidth with low latency for inter-node communication

D. A larger management VLAN with no data-path change

6. A system improves its predictions by learning patterns from labeled examples rather than following only hand-written rules. Which description is most accurate?

A. It is only data analytics and cannot be machine learning

B. It is a machine-learning system within the broader field of AI

C. It is deep learning simply because data is used

D. It is not AI because no explicit rules are written

7. A rack design calls for 30 kW of sustained IT load, but the available branch power can deliver only 20 kW continuously. Cooling capacity is adequate. What is the primary issue?

A. The rack is acceptable because cooling capacity is sufficient

B. The issue is only the configured GPU power limit, not facility capacity

C. The scheduler can compensate by allocating more GPUs to each job

D. The planned rack exceeds the available electrical delivery capacity

8. An HPC team submits finite jobs that wait in queues, receive priorities, request resources, and run to completion. Which control system best matches that operating model?

A. A dashboard such as Grafana

B. An inference optimizer such as TensorRT

C. A job scheduler such as Slurm

D. A model registry

9. A trained neural network must be optimized for efficient inference on NVIDIA GPUs before deployment. Which NVIDIA technology is the most direct fit?

A. NCCL

B. TensorRT

C. Base Command Manager

D. DCGM Exporter

10. A project has highly variable short-term demand and wants to add GPU capacity quickly without purchasing a permanent data-center footprint. Which characteristic most strongly favors cloud infrastructure?

A. Complete physical control of all hardware and facility systems

B. Guaranteed lower lifetime cost for every workload

C. Elastic access to capacity with less up-front physical infrastructure ownership

D. Removal of all data-governance responsibilities

11. Which combination best explains the recent acceleration in AI capability and adoption?

A. Larger monitors, faster office networks, and more storage formats

B. More available data, increased computational power, and algorithmic advances

C. Only the growth of social media datasets

D. Only the creation of cloud billing models

12. A data-center design wants to move infrastructure services such as networking, storage processing, and security tasks away from host CPUs. Which component is intended for this role?

A. A DPU

B. A Kubernetes namespace

C. A GPU display engine

D. A model registry

13. GPU clocks drop during heavy operation and the team suspects thermal or power limiting. Which monitoring evidence is most useful to correlate with the clock reduction?

A. GPU temperature and power draw or limit data

B. Only the model name

C. Only the number of files in the training dataset

D. Only the container image tag

14. A team wants every model release to be reproducible and traceable from data and code through validation and deployment. Which practice most directly supports that goal?

A. Treat the current production model as the only version that matters

B. Delete older model artifacts after each deployment

C. Version and track the relevant data, code, model artifacts, and environments

D. Use monitoring dashboards as a replacement for artifact tracking

15. Which set best represents the core infrastructure categories that must work together in an accelerated cluster?

A. Compute, high-speed networking, storage/data access, and management

B. Compute and storage only; the inter-node network is not a cluster dependency

C. GPU count and a scheduler only; data access can be treated independently

D. CPU capacity and user accounts only; management and networking are optional

16. Which workload characteristic most strongly favors GPU execution over CPU execution?

A. A small branch-heavy control task with little parallel work

B. A serial configuration parser

C. Operating-system process scheduling

D. A large amount of data-parallel numerical work that can run concurrently

17. A network design calls for host-to-host memory transfers with minimal CPU involvement. Which data-center networking concept directly describes this capability?

A. NCCL

B. MIG

C. RDMA

D. NVLink

18. A manufacturer uses camera and sensor data to detect defective products on a production line. Which AI use-case category best describes this example?

A. Cluster job scheduling

B. Industrial quality inspection and anomaly detection

C. Facility power management

D. GPU virtualization

19. A team plans to use MIG for several workloads on one supported GPU. Which consideration should be confirmed before assigning instances?

A. The GPU supports MIG and the required instance profiles or geometry can be configured for the workloads

B. Every workload runs in a separate virtual machine with NVIDIA vGPU

C. DCGM automatically creates the required MIG layout from utilization telemetry

D. GPUDirect RDMA partitions the GPU before MIG is enabled

20. An organization adds many GPUs to a cluster but leaves the network, storage path, power, and cooling unchanged. Which statement best describes the scaling problem?

A. The only remaining requirement is a larger model registry

B. Scaling is complete once all GPUs have identical firmware

C. GPU count increased, but the infrastructure was not scaled as a balanced system

D. The cluster is fully scaled because accelerators are the only capacity that matters

21. An application has a compatible NVIDIA driver and now needs the programming/runtime layer used to launch GPU kernels and access GPU-accelerated libraries. Which component is the best fit?

A. A model registry

B. CUDA

C. Grafana

D. A Slurm partition

22. A team needs structured NVIDIA GPU health telemetry and diagnostic capabilities across its cluster nodes rather than only a one-time local snapshot. Which NVIDIA framework is the best fit?

A. NVLink

B. NIM

C. TensorRT

D. DCGM

23. Before installing several high-density GPU racks in an existing room, which facility question should be resolved first?

A. Whether every user prefers the same deep-learning framework

B. Whether Grafana is installed on each compute node

C. Whether sufficient rack power, cooling, space, and environmental capacity are available

D. Whether all models can use one registry tag

24. A service scores millions of records overnight with a fixed trained model. It is optimized for throughput and does not update any model weights. How should the workload be classified?

A. Inference, because trained parameters are applied without being updated

B. Training, because the workload runs as a scheduled batch

C. Training, because throughput is more important than response latency

D. Orchestration, because the records are processed overnight

25. A new tightly coupled training cluster will use a dedicated high-performance fabric with native RDMA and an ecosystem designed for large-scale HPC/AI communication. Which option is the most direct fit?

A. Wi-Fi

B. A management-only 1 GbE network

C. InfiniBand

D. SATA

26. A team needs a production inference server that can expose models through standard service interfaces and support models from multiple frameworks. Which NVIDIA technology is the most direct match?

A. NCCL

B. DCGM

C. RAPIDS

D. NVIDIA Dynamo-Triton

27. A training cluster shows low GPU utilization because accelerators repeatedly wait for the next batch of data. Which hardware/data-path requirement should be investigated?

A. Increase GPU clock limits even though the accelerators are waiting for input

B. Storage and input-pipeline throughput sufficient to keep the GPUs fed

C. Increase scheduler queue depth without changing the data path

D. Add more GPU memory while leaving input throughput unchanged

28. An organization uses Slurm for queued research jobs and Kubernetes for long-running container services on the same accelerated environment. Which conclusion is most accurate?

A. The two can coexist because scheduling and orchestration responsibilities overlap but are not identical

B. Kubernetes cannot run batch workloads, so all containers must be long-running services

C. Slurm cannot manage GPU resources when Kubernetes is present

D. One platform must replace the other before the environment can support GPU workloads

29. Which hierarchy correctly relates AI, machine learning, and deep learning?

A. Deep learning is the broadest field and includes all forms of AI

B. AI and machine learning are separate fields with no conceptual overlap

C. Deep learning is a subset of machine learning, which is a subset of AI

D. Machine learning is a subset of deep learning, which is a subset of AI

30. Training nodes read batches from shared remote storage. GPU utilization stays low even though inter-GPU communication is healthy. Which network path should be examined first?

A. Bandwidth and congestion on the storage-to-compute data path

B. Collective bandwidth between GPUs, even though that path is already healthy

C. North-south user-access latency, which is outside the training data path

D. Management-plane bandwidth used mainly for provisioning and administration

31. Why did AI progress accelerate when modern GPU computing became widely available alongside larger datasets?

A. Large datasets eliminated the need for compute

B. GPUs removed the need for algorithms

C. Compute and data made increasingly capable algorithms practical at larger scale

D. The change was caused only by new data-center cooling systems

32. A model loads successfully but repeatedly approaches the physical GPU memory limit and fails when batch size increases. Which metric should be watched most directly?

A. GPU temperature relative to its operating limit

B. GPU memory used compared with total available GPU memory

C. CPU utilization on the host

D. Management-network throughput

33. A data center has a PUE closer to 1.0 after an efficiency project. What does the lower PUE generally indicate?

A. A greater share of facility energy is being lost to non-IT overhead

B. GPU memory bandwidth has increased relative to CPU bandwidth

C. Less facility overhead energy is used relative to the energy consumed by IT equipment

D. More GPUs can be scheduled per rack without checking electrical capacity

34. What is the main reason to keep a versioned record of model artifacts and their associated metadata?

A. To eliminate the need for model validation after training

B. To support lineage, reproducibility, controlled promotion, and recovery decisions

C. To force retraining whenever any metadata field changes

D. To guarantee that every stored model version is production-ready

35. A regulated organization has predictable long-term demand, existing data-center capacity, and strong requirements for direct hardware and data-location control. Which deployment model is most naturally aligned with those priorities?

A. A short-lived public cloud trial only

B. Consumer edge devices only

C. A software registry with no compute environment

D. On-premises infrastructure

36. Which statement best contrasts CPU and GPU architecture at a high level?

A. GPUs replace CPUs for all operating-system and control tasks

B. CPUs emphasize versatile general-purpose execution, while GPUs emphasize parallel throughput

C. GPUs are optimized mainly for serial branch-heavy control logic

D. CPUs cannot execute vector or floating-point workloads

37. What is the main operational benefit of RDMA for suitable data-center communication?

A. It partitions one physical GPU into isolated execution instances

B. It enables memory-oriented network transfers with reduced CPU involvement

C. It replaces the need for a network fabric by moving data only inside one host

D. It automatically chooses the best collective algorithm for each model

38. A retailer predicts which products a customer is likely to prefer based on prior behavior. Which AI use case does this represent?

A. Job scheduling

B. Data-center cooling optimization only

C. Recommendation and personalization

D. GPU health monitoring

39. A company runs GPU-accelerated virtual desktops and wants the hypervisor to assign GPU capability to separate virtual machines. Which technology is the most direct match?

A. NVIDIA vGPU

B. NCCL

C. MIG without virtualization

D. RoCE

40. A cluster adds more compute nodes, but training jobs still stall because the shared data source cannot feed them fast enough. Which cluster component has become the limiting factor?

A. Only the rack labels

B. Only the user directory

C. Only the GPU firmware name

D. The storage/data-access layer

41. What is the primary role of NCCL in the NVIDIA software stack?

A. Collect and visualize GPU telemetry from a Prometheus data source

B. Provide topology-aware collective and point-to-point communication primitives for GPUs

C. Install and manage the NVIDIA GPU driver on cluster nodes

D. Partition a supported GPU into isolated compute and memory instances

42. What is a key benefit of using a DPU in an accelerated data center?

A. Offload networking, storage, and security-related infrastructure processing from host CPUs

B. Replace the physical network fabric between accelerated nodes

C. Manage model versions and promotion through the AI lifecycle

D. Convert serial CPU control code into GPU kernels automatically

43. Which pairing correctly describes a common monitoring pipeline role?

A. Grafana collects GPU telemetry directly from hardware with no data source

B. Prometheus schedules GPU jobs, while Grafana allocates resources

C. Prometheus collects time-series metrics, while Grafana visualizes them

D. TensorRT stores metrics, while Dynamo-Triton renders dashboards

44. Which comparison is generally most appropriate when contrasting training and inference?

A. Training usually optimizes single-request latency, while inference focuses on gradient synchronization across nodes

B. Inference always requires more accelerator memory than training because it serves production traffic

C. Training is defined by Slurm batch jobs, while inference is defined by Kubernetes services

D. Training often emphasizes parameter-update throughput and communication; inference often emphasizes request latency or serving throughput

45. A team wants to double training throughput by doubling node count. Which planning approach is most appropriate?

A. Replace the scheduler with a monitoring dashboard so jobs start sooner

B. Add nodes first and assume network, storage, and facility capacity scale automatically

C. Reduce inter-node bandwidth so collective traffic cannot saturate the fabric

D. Scale compute together with networking, data access, facility capacity, and workload efficiency

46. A team wants a standardized NVIDIA-packaged inference microservice for deploying a supported AI model through an API. Which solution best matches that role?

A. NVIDIA NIM

B. DCGM

C. NVLink

D. MIG

47. A facility team is evaluating whether an existing row can accept another dense accelerator rack. Which measurement is most directly relevant?

A. The count of Grafana dashboards

B. The number of Python packages in the image

C. The number of model versions in NGC

D. Available electrical and cooling capacity at the rack/row

48. Which scenario is the clearest fit for Slurm rather than a continuously reconciled service orchestrator?

A. A queue of finite GPU jobs with priorities, resource requests, and completion states

B. A service that must continuously maintain a target replica count after failures

C. A container platform reconciling declared state with actual state

D. An inference endpoint that must remain continuously available

49. A team is comparing InfiniBand with high-speed Ethernet for an AI cluster. Which statement is accurate?

A. Only InfiniBand can support RDMA semantics in a data-center network

B. Ethernet cannot be used as a high-speed AI fabric

C. NVLink replaces both InfiniBand and Ethernet as the rack-scale data-center network

D. InfiniBand is an RDMA-capable fabric, while Ethernet can provide RDMA semantics through RoCE

50. Which metric most directly indicates how busy the GPU execution engines are over a sampling interval?

A. Model version number

B. Container image size

C. Rack position

D. GPU utilization

Mock Exam 1 answers and explanations

Review the explanation even when you selected the correct option. Record the objective when your reasoning was uncertain, incomplete, or based on elimination alone.

1. A: The NVIDIA driver is the host-side layer that enables software to communicate with the GPU. A compiler, inference server, or dashboard cannot substitute for a missing driver. *Objective 1.1 | Review: Chapters 1, 6, 8*

2. D: A model that must span accelerators needs enough aggregate GPU memory and an efficient inter-GPU path; if placement spans nodes, the inter-node fabric also matters. CPU clocks, local disk, and management networking do not address those constraints. *Objective 2.1 | Review: Chapters 3, 4*

3. B: nvidia-smi is designed for immediate local GPU inspection. TensorRT optimizes inference, Slurm schedules jobs, and NGC distributes software and model assets. *Objective 3.1 | Review: Chapters 9, 11*

4. A: The defining distinction is that training changes model parameters, while inference uses a trained model. Their throughput, latency, memory, and communication priorities can then be compared from that starting point. *Objective 1.2 | Review: Chapter 2*

5. C: Distributed training can create heavy east-west communication between nodes. High bandwidth and low latency on that path are therefore central network requirements. *Objective 2.7 | Review: Chapter 5*

6. B: Machine learning is a subset of AI in which systems learn patterns from data. Deep learning is a further subset that specifically uses deep neural-network approaches. *Objective 1.3 | Review: Chapter 2*

7. D: The planned sustained IT demand exceeds the available electrical capacity. Facility power delivery must be sized for the load; scheduler or GPU settings cannot create missing electrical capacity. *Objective 2.3 | Review: Chapter 3*

8. C: Slurm is built around queued jobs, resource allocation, priorities, partitions and related scheduling policies. The other choices serve different responsibilities. *Objective 3.2 | Review: Chapter 10*

9. B: TensorRT is an inference-optimization SDK. DCGM Exporter exposes telemetry, NCCL provides collective communication, and Base Command Manager manages cluster infrastructure. *Objective 1.6 | Review: Chapters 4, 7, 8, 9, 11*

10. C: Cloud infrastructure can provide elastic capacity without owning the underlying facility. It does not guarantee lower lifetime cost or remove governance obligations. *Objective 2.4 | Review: Chapter 4*

11. B: The course frames modern AI growth around data availability, compute improvements, and algorithmic progress working together. *Objective 1.4 | Review: Chapter 2*

12. A: A DPU is designed to accelerate and offload data-center infrastructure services. It complements CPUs and GPUs rather than replacing their compute roles. *Objective 2.10 | Review: Chapter 3*

13. A: Temperature and power measurements can reveal thermal or power constraints that correlate with reduced clocks. Metadata about models or files does not diagnose GPU operating limits. *Objective 3.3 | Review: Chapter 9*

14. C: Lifecycle reproducibility depends on tracking the inputs, code, models, and environments associated with a release. Monitoring complements that record but does not replace it. *Objective 1.7 | Review: Chapters 6, 8, 11*

15. A: An accelerated cluster is a system of compute, network, storage/data paths, and management components. Focusing on one category alone misses the dependencies that determine cluster behavior. *Objective 2.5 | Review: Chapters 4, 5*

16. D: GPUs are designed for high-throughput parallel computation. CPUs remain strong for general-purpose, branch-heavy and serial control work. *Objective 1.8 | Review: Chapter 3*

17. C: RDMA enables memory-oriented transfers between hosts with reduced CPU involvement. RoCE and InfiniBand can provide RDMA-capable networking; NCCL is communication software, NVLink is a GPU interconnect, and MIG partitions GPUs. *Objective 2.8 | Review: Chapter 5*

18. B: Computer vision and anomaly detection for manufacturing quality are common AI use cases. The other choices are infrastructure or operations functions, not business AI applications. *Objective 1.5 | Review: Chapter 2*

19. A: MIG requires a supported GPU and an appropriate configured GPU/compute-instance layout. It does not require each workload to run in a VM, and monitoring or communication technologies do not automatically create the partitioning. *Objective 3.4 | Review: Chapter 7*

20. C: Effective scaling must consider compute together with data movement, storage, facility capacity, and management. Adding accelerators alone can simply move the bottleneck. *Objective 2.2 | Review: Chapter 4*

21. B: CUDA provides the programming model, runtime capabilities, and libraries used by GPU-accelerated applications above the driver layer. A registry, dashboard, or scheduler partition serves a different responsibility. *Objective 1.1 | Review: Chapters 1, 6, 8*

22. D: DCGM provides a host-level framework for GPU telemetry, health, diagnostics, policy, and related management functions that can be deployed across cluster nodes and integrated into broader monitoring workflows. TensorRT and NIM address inference, while NVLink is an interconnect. *Objective 3.1 | Review: Chapters 9, 11*

23. C: Facility readiness is about the physical environment needed to support the equipment safely and continuously. Application preferences do not substitute for power, cooling, space, or environmental capacity. *Objective 2.6 | Review: Chapter 3*

24. A: Inference applies trained model parameters without updating them. It can be interactive or batch-oriented, so running overnight as a large batch does not make the workload training. *Objective 1.2 | Review: Chapter 2*

25. C: InfiniBand is a high-speed data-center fabric widely used for low-latency, RDMA-capable HPC and AI communication. The other options do not meet the stated cluster requirements. *Objective 2.9 | Review: Chapter 5*

26. D: NVIDIA Dynamo-Triton is designed for serving models from multiple frameworks through inference service interfaces. RAPIDS accelerates data science, DCGM monitors GPUs, and NCCL handles collectives. *Objective 1.6 | Review: Chapters 4, 7, 8, 9, 11*

27. B: Fast accelerators cannot sustain throughput if storage or the input pipeline cannot supply data quickly enough. The bottleneck should be investigated along the data path rather than masked with compute or scheduling changes. *Objective 2.1 | Review: Chapters 3, 4*

28. A: Slurm and Kubernetes can coexist in mixed environments because they emphasize different control models and operational responsibilities. Avoid rigid rules such as training equals Slurm and inference equals Kubernetes. *Objective 3.2 | Review: Chapter 10*

29. C: The standard hierarchy is AI as the broad field, machine learning as a subset, and deep learning as a subset of machine learning. *Objective 1.3 | Review: Chapter 2*

30. A: When accelerators wait on remotely stored training data, the relevant network requirement is the storage-to-compute data path. Healthy collective communication does not rule out a separate ingestion bottleneck. *Objective 2.7 | Review: Chapter 5*

31. C: AI progress reflects interacting advances in data, compute and algorithms. No single factor eliminates the need for the others. *Objective 1.4 | Review: Chapter 2*

32. B: GPU memory consumption relative to available memory directly measures the constrained resource in this scenario. Other metrics may matter operationally but do not reveal whether the model and batch fit in device memory. *Objective 3.3 | Review: Chapter 9*

33. C: A lower PUE indicates that a greater share of total facility energy is going to IT equipment rather than overhead such as cooling and power conversion. It is an efficiency ratio, not a compute-capacity metric. *Objective 2.3 | Review: Chapter 3*

34. B: Versioned artifacts and metadata provide lineage and reproducibility and support controlled promotion or recovery. They do not guarantee model quality or replace validation and approval controls. *Objective 1.7 | Review: Chapters 6, 8, 11*

35. D: Existing facility capacity, predictable demand, and direct control are common reasons to favor on-premises infrastructure. Cloud may still be used, but the stated priorities align strongly with on-premises operation. *Objective 2.4 | Review: Chapter 4*

36. B: CPUs are general-purpose processors optimized for diverse control and serial work, while GPUs provide high parallel throughput for suitable numerical workloads. The roles are complementary rather than mutually exclusive. *Objective 1.8 | Review: Chapter 3*

37. B: RDMA supports direct memory-oriented transfers with reduced CPU involvement and data-copy overhead. It is a networking capability, not a GPU-partitioning or collective-algorithm selection feature. *Objective 2.8 | Review: Chapter 5*

38. C: Recommendation and personalization are common AI use cases in retail and digital services. The other options describe infrastructure or operations functions. *Objective 1.5 | Review: Chapter 2*

39. A: NVIDIA vGPU is designed to provide GPU resources to virtual machines through a virtualization stack. MIG is hardware partitioning on supported GPUs and does not itself provide the full VM virtualization model. *Objective 3.4 | Review: Chapter 7*

40. D: Cluster performance depends on more than compute. If data cannot be supplied at the required rate, the storage/data-access layer becomes the bottleneck. *Objective 2.5 | Review: Chapters 4, 5*

41. B: NCCL provides topology-aware inter-GPU communication primitives, including collective operations and point-to-point send/receive. It is not a driver, virtualization mechanism, or monitoring dashboard. *Objective 1.1 | Review: Chapters 1, 6, 8*

42. A: DPUs can offload infrastructure processing such as networking, storage, and security tasks from host CPUs. They complement CPUs, GPUs, and the network rather than replacing those components. *Objective 2.10 | Review: Chapter 3*

43. C: Prometheus is commonly used to collect/store time-series metrics and Grafana to visualize them. GPU telemetry can be exposed to that pipeline by exporters such as DCGM Exporter. *Objective 3.1 | Review: Chapters 9, 11*

44. D: Training and inference have different goals and often different performance priorities, but rigid platform or resource absolutes should be avoided. Parameter updates are central to training; serving trained models is central to inference. *Objective 1.2 | Review: Chapter 2*

45. D: Scaling accelerator count only works when supporting systems and the workload can use the added resources. Network, data access, power, cooling, and software efficiency must be considered together. *Objective 2.2 | Review: Chapter 4*

46. A: NVIDIA NIM provides deployment-focused inference microservices for supported models. DCGM is monitoring, MIG is GPU partitioning, and NVLink is an interconnect. *Objective 1.6 | Review: Chapters 4, 7, 8, 9, 11*

47. D: Facility acceptance depends on physical support such as electrical and thermal capacity. Software inventory does not determine whether the room can sustain the rack. *Objective 2.6 | Review: Chapter 3*

48. A: Slurm is well suited to queued, finite jobs and policy-based resource allocation. The other scenarios emphasize continuous desired-state management or service availability, which align more naturally with orchestration. *Objective 3.2 | Review: Chapter 10*

49. D: InfiniBand is an RDMA-capable data-center fabric, while RoCE provides RDMA semantics over Ethernet. NVLink is a GPU interconnect and does not replace the external data-center network. *Objective 2.9 | Review: Chapter 5*

50. D: GPU utilization reflects the proportion of time the GPU is actively processing work during the sample interval. The other values do not measure compute activity. *Objective 3.3 | Review: Chapter 9*

Mock Exam 2

Questions: 50 | Suggested time: 60 minutes | Choose the one best answer for each question. Do not look at the answer section until the attempt is complete.

1. A virtualization team needs several virtual machines to share physical GPU capacity through a hypervisor-managed stack. Which technology is designed for this use case?

A. NVIDIA vGPU

B. DCGM Exporter

C. NCCL

D. RoCE

2. A data-science team wants GPU-accelerated dataframe processing and machine-learning utilities during data preparation. Which NVIDIA software family best matches the requirement?

A. RAPIDS

B. Base Command Manager

C. NVIDIA Dynamo-Triton

D. MIG

3. A data-center design calls for remote memory transfers between hosts with reduced CPU involvement. Which concept directly describes this capability?

A. NIM

B. MIG

C. NVLink

D. RDMA

4. A model is built from a deep neural network with many learned layers. How should it be classified?

A. Machine learning but never AI

B. Deep learning, which is a subset of machine learning and AI

C. AI but not machine learning

D. A non-AI rules engine

5. A single-node inference service is expanded to many GPU servers. Which change is most important to evaluate in addition to adding accelerators?

A. Whether the service can add nodes without changing any network or data-path capacity

B. Whether every server keeps the same model name and container image tag

C. Whether one scheduler queue can hold all incoming inference requests

D. Whether networking, traffic distribution, data access, power, and management scale with compute

6. Historical GPU metrics are already being collected into Prometheus. Which tool is the most direct choice for building dashboards over those time-series metrics?

A. NCCL

B. TensorRT

C. MIG

D. Grafana

7. What is the main role of NVIDIA Container Toolkit in a containerized GPU environment?

A. Schedule Pods across nodes based on cluster policy

B. Create a new MIG layout automatically for every container

C. Make NVIDIA GPU devices and driver capabilities available inside compatible containers

D. Replace the host NVIDIA GPU driver with container-local drivers

8. A new data hall is being prepared for high-density accelerator servers. Which requirement belongs to the facility layer?

A. GPU software compatibility and framework versions

B. Adequate electrical distribution, cooling, rack/space, and environmental support

C. The model-serving API exposed to inference clients

D. The scheduler policy used to allocate GPU resources

9. A workload fine-tunes a pretrained model by updating its weights on new domain data. How should the workload be classified?

A. Virtualization

B. Inference

C. Monitoring

D. Training

10. A server spends significant CPU cycles on packet processing and storage/network infrastructure services. Which accelerator can offload those tasks?

A. A CPU cache only

B. A DPU

C. A GPU dashboard

D. A model registry

11. A Kubernetes Pod requires one GPU through an exposed extended resource. What should standard placement primarily use?

A. Model accuracy scores collected after inference

B. Live DCGM utilization as an automatic native rebalance signal

C. The Pod's declared GPU resource requirement and nodes advertising available schedulable GPU resources

D. The Grafana dashboard selected by the operator

12. Which set of factors most directly enabled recent advances in modern AI?

A. Only more software vendors

B. Growing datasets, stronger compute, and improved algorithms

C. Only faster internet access

D. Only larger storage drives

13. A model cannot fit into the memory of one GPU but can be partitioned across multiple accelerators. Which hardware characteristic becomes especially important?

A. More host CPU cores without increasing aggregate accelerator memory

B. Larger local storage while GPU memory and interconnect remain unchanged

C. Sufficient aggregate accelerator memory and fast inter-GPU communication

D. Higher management-network bandwidth with the same accelerator resources

14. A team wants to promote a model into production only after it passes evaluation, while keeping the previous version available for recovery. Which lifecycle practice supports this?

A. Versioned artifacts plus validation and controlled promotion

B. Automatic deletion of prior versions

C. Monitoring without artifact lineage

D. Deployment before validation

15. Which tradeoff is most typical when comparing cloud GPU capacity with owned on-premises capacity?

A. On-premises provides elasticity without owning facilities, while cloud provides direct hardware control

B. Cloud always has lower lifetime cost, while on-premises always has higher availability

C. Cloud can provide elasticity, while on-premises can provide greater direct control over hardware and data location

D. Cloud removes security and governance duties, while on-premises removes facility responsibilities

16. A GPU begins reporting repeated memory-related health errors even when utilization is moderate. Which monitoring evidence deserves direct attention?

A. GPU health and error counters relevant to memory reliability, such as supported ECC indicators

B. Job runtime and scheduler priority only

C. Network hostname and interface address only

D. Model artifact size and registry tag only

17. A healthcare provider uses a trained model to flag suspicious regions in medical images for clinician review. Which AI use case is represented?

A. Medical imaging assistance

B. GPU partitioning

C. Data-center switching

D. Job orchestration

18. A distributed data-parallel training workload performs frequent all-reduce operations across nodes. Which network property is especially important?

A. Only internet-facing north-south throughput

B. High-bandwidth, low-latency east-west communication

C. High latency between compute nodes

D. A larger DNS zone with unchanged fabric capacity

19. Which workload is generally better suited to a CPU than a GPU?

A. Large batched tensor calculations

B. Highly parallel vector arithmetic

C. Massively parallel matrix operations

D. Branch-heavy operating-system and control logic with limited parallelism

20. Which component is responsible for coordinating resource use and workload placement across cluster nodes rather than accelerating the numerical kernel itself?

A. The communication library used inside distributed applications

B. The shared storage layer that supplies datasets to jobs

C. The node-level NVIDIA GPU driver

D. The cluster management and scheduling layer

21. A trained model needs graph/kernel optimization and engine building for lower-latency NVIDIA GPU inference. Which tool is the most direct fit?

A. TensorRT

B. DCGM Exporter

C. RoCE

D. Base Command Manager

22. An administrator needs provisioning and management capabilities for a GPU cluster, not just GPU metric collection. Which NVIDIA product category is the better fit?

A. TensorRT

B. NCCL

C. Base Command Manager

D. MIG

23. A tightly coupled HPC/AI environment prefers a dedicated high-speed fabric designed around low latency and native RDMA. Which network option is the most natural match?

A. A management-only 1 GbE network

B. InfiniBand

C. SATA

D. Wi-Fi

24. A scheduled overnight job applies an already trained model to a billion records and writes predictions to storage. Which statement is correct?

A. It is orchestration rather than inference

B. It is training because the job is large

C. It is inference because the model parameters are not being updated

D. It is training because it runs in batch mode

25. A rack has enough electrical capacity but exhaust temperatures rise beyond the facility design target during sustained GPU load. Which requirement is limiting?

A. The scheduler queue name

B. Cooling/heat-removal capacity

C. The model registry

D. The number of user accounts

26. What is CUDA primarily in the NVIDIA software stack?

A. A monitoring dashboard for GPU fleet health

B. A virtualization licensing layer for assigning GPUs to virtual machines

C. A physical network fabric for connecting accelerated nodes

D. The parallel-computing platform and programming/runtime ecosystem used by GPU-accelerated applications

27. A research cluster uses partitions, job priorities, reservations, and resource requests to control batch work. Which platform is most directly associated with those concepts?

A. RAPIDS

B. NVIDIA Dynamo-Triton

C. Grafana

D. Slurm

28. Powerful GPUs remain idle because the training input pipeline cannot read and preprocess data fast enough. Which hardware/infrastructure requirement is being missed?

A. A storage and data-ingestion path with sufficient throughput

B. More GPU memory even though the accelerators are idle waiting for input

C. A larger scheduler queue without any change to the data path

D. More management-network bandwidth without improving data ingestion

29. Which hierarchy correctly relates AI, machine learning, and deep learning?

A. AI is the broad field; machine learning is a subset of AI; deep learning is a subset of machine learning

B. Deep learning is broader than AI and contains machine learning

C. Machine learning excludes systems that learn patterns from data

D. AI and machine learning are identical terms in every technical context

30. Why is RDMA attractive for high-performance data-center communication?

A. It guarantees linear application scaling regardless of software or network topology

B. It can reduce CPU involvement and data-copy overhead for suitable memory-to-memory transfers

C. It partitions one physical GPU into isolated compute and memory instances

D. It is a physical cable type that replaces Ethernet and InfiniBand

31. A neural-network approach existed conceptually before it became practical to train at modern scale. Which broader lesson best explains how such ideas can become widely usable?

A. Algorithmic ideas often become practical when sufficient data and scalable compute make them trainable and deployable

B. An algorithmic idea guarantees adoption regardless of available data or hardware

C. More data removes the need for improved algorithms or computational capability

D. Facility cooling alone determines whether an AI method becomes technically viable

32. A GPU is drawing near its configured power limit and clocks fall under sustained load. Which measurements should be correlated first?

A. Power draw/limit and clock behavior

B. User account age

C. Grafana theme and dashboard title

D. Model version and filename

33. Which problem cannot be corrected simply by changing a CUDA or framework setting?

A. A container is missing a required user-space library

B. The data hall lacks enough heat-removal capacity for the planned GPU racks

C. A model uses an unsupported framework operator

D. A job requests a GPU resource that the scheduler does not expose

34. What is the primary purpose of a model registry or equivalent artifact-tracking mechanism in an MLOps process?

A. Coordinate GPU placement and job priorities across cluster nodes

B. Replace model evaluation and approval before deployment

C. Record artifact versions and metadata to support lineage and controlled lifecycle operations

D. Guarantee that every registered model meets production accuracy targets

35. Which situation most strongly favors keeping an AI workload on premises?

A. Short-term demand that changes rapidly and no owned facility capacity

B. A temporary experiment that needs capacity immediately

C. Highly uncertain demand where elasticity is the dominant requirement

D. Strict data-location requirements with predictable demand and suitable existing facility capacity

36. Why can GPUs outperform CPUs on many AI tensor operations?

A. GPUs eliminate the need for host CPUs in an AI server

B. CPUs cannot execute floating-point or vector operations

C. GPUs expose many parallel execution resources suited to repeated numerical operations over large datasets

D. GPUs always outperform CPUs because their clock frequency is higher

37. A training cluster reads large datasets from remote shared storage. GPU utilization drops whenever many nodes begin reading simultaneously. Which networking requirement is most relevant?

A. More public internet bandwidth for external users

B. Higher out-of-band management bandwidth for BMC traffic

C. Lower storage-to-compute bandwidth to reduce the amount of data in flight

D. Sufficient storage-to-compute bandwidth and congestion control on the data path

38. A bank analyzes transaction patterns to identify potentially fraudulent activity. Which AI use case is this?

A. Rack cooling

B. Container orchestration

C. GPU virtualization

D. Fraud and anomaly detection

39. Which MIG characteristic is most relevant when several workloads share one supported physical GPU?

A. MIG uses only software time-slicing and provides no dedicated memory resources

B. MIG requires every workload to run inside a separate virtual machine

C. MIG creates separate GPU instances with isolated compute and memory resources

D. MIG is a network transport used to provide RDMA between GPU nodes

40. A cluster has sufficient GPUs and shared storage, but multi-node jobs spend most of their time waiting for peer-to-peer traffic across an oversubscribed fabric. Which cluster component needs attention?

A. Additional local GPU memory only

B. A second model registry for job metadata

C. More scheduler queues without changing the data path

D. The inter-node network fabric, including its topology and available bandwidth

41. A service must host models from several supported frameworks and expose them for inference through HTTP/gRPC-style interfaces. Which NVIDIA technology best matches?

A. BlueField DPU

B. NVIDIA Dynamo-Triton

C. DCGM

D. MIG

42. Which statement best describes the relationship between a DPU and a GPU in an AI server?

A. The GPU primarily offloads networking, storage, and security services from the CPU

B. The DPU and GPU are interchangeable names for the same accelerator role

C. The DPU can offload infrastructure services, while the GPU focuses on accelerated application computation

D. The DPU replaces the need for a network adapter or data-center fabric

43. An operator wants a quick local snapshot of GPU temperature, utilization, memory use, and processes before opening a broader monitoring dashboard. Which tool is most appropriate?

A. NGC

B. NIM

C. nvidia-smi

D. TensorRT

44. Which statement avoids an overly rigid training-versus-inference rule?

A. Training often emphasizes model-update throughput and communication; inference priorities vary with latency and serving-throughput needs

B. Training has no latency-sensitive workflows, so latency never matters during training

C. Inference is always interactive and therefore cannot use batch processing

D. Training belongs only on Slurm and inference belongs only on Kubernetes

45. A distributed training cluster is expanded from 16 to 128 GPUs. Which factor is most likely to become more important as scale increases?

A. The color of rack labels

B. Network topology and collective-communication efficiency between accelerators/nodes

C. The number of file extensions used by the model

D. The desktop resolution of the management workstation

46. Which statement best describes NCCL relative to InfiniBand or Ethernet?

A. NCCL is communication software that uses supported underlying interconnects or transports; it is not the physical fabric

B. NCCL is the Ethernet or InfiniBand cable-and-switch layer used by the cluster

C. NCCL is the host GPU driver that initializes network adapters

D. NCCL is a hypervisor feature for assigning GPU resources to virtual machines

47. A Kubernetes cluster has NVIDIA drivers on GPU nodes, but Pods cannot request an NVIDIA GPU because no GPU resource is advertised to Kubernetes. Which integration component is missing?

A. DCGM Exporter

B. NCCL

C. The NVIDIA device plugin that exposes GPUs as schedulable resources

D. Grafana

48. Which statement about PUE is correct at a high level?

A. A higher PUE means a larger share of facility energy reaches IT equipment

B. PUE is the number of scheduled jobs per GPU in a data center

C. PUE measures the effective memory bandwidth of installed GPUs

D. A value closer to 1.0 indicates less facility overhead relative to IT equipment energy

49. Which statement best compares InfiniBand with an Ethernet-based RoCE design for AI networking?

A. InfiniBand is a storage protocol and is not used for accelerator clusters

B. InfiniBand provides an RDMA-capable fabric, while RoCE carries RDMA semantics over Ethernet

C. RoCE is a local GPU interconnect that replaces the data-center network

D. Ethernet-based designs cannot support low-overhead RDMA communication

50. A GPU shows moderate compute utilization but memory usage is almost at capacity. What is the best interpretation?

A. Compute utilization must reach 100% before memory can constrain a workload

B. GPU memory capacity can be the bottleneck even when compute utilization is moderate

C. Temperature is the only meaningful metric when utilization is below 100%

D. High memory usage means the workload must be training rather than inference

Mock Exam 2 answers and explanations

Review the explanation even when you selected the correct option. Record the objective when your reasoning was uncertain, incomplete, or based on elimination alone.

1. A: NVIDIA vGPU provides GPU capability to virtual machines through supported virtualization platforms. NCCL, DCGM Exporter and RoCE address communication or monitoring instead. *Objective 3.4 | Review: Chapter 7*

2. A: RAPIDS provides GPU-accelerated libraries for data-science workflows such as dataframe processing and related analytics. The other options serve inference, cluster management, or GPU partitioning. *Objective 1.6 | Review: Chapters 4, 7, 8, 9, 11*

3. D: RDMA stands for Remote Direct Memory Access and enables suitable memory-to-memory transfers with reduced CPU involvement. RoCE and InfiniBand can provide RDMA-capable networking, while NIM, MIG, and NVLink serve different roles. *Objective 2.8 | Review: Chapter 5*

4. B: Deep learning uses multi-layer neural networks and sits within machine learning, which itself sits within the broader field of AI. *Objective 1.3 | Review: Chapter 2*

5. D: Scaling GPU infrastructure requires the supporting data path, facility, and operations layers to scale too. Accelerator count alone is not a complete capacity plan. *Objective 2.2 | Review: Chapter 4*

6. D: Grafana visualizes data from sources such as Prometheus. NCCL, TensorRT and MIG have communication, inference-optimization and partitioning roles. *Objective 3.1 | Review: Chapters 9, 11*

7. C: NVIDIA Container Toolkit integrates container runtimes with NVIDIA GPU devices and driver capabilities available from the host. It does not replace the host driver, act as the scheduler, or automatically create a MIG layout for each container. *Objective 1.1 | Review: Chapters 1, 6, 8*

8. B: Facility requirements are physical and environmental: electrical distribution, cooling, rack/space, and related support. Frameworks, serving APIs, and scheduling policy belong to software or operations layers. *Objective 2.6 | Review: Chapter 3*

9. D: Fine-tuning changes model parameters, so it is a form of training. Inference applies trained parameters without updating them. *Objective 1.2 | Review: Chapter 2*

10. B: DPUs are designed to offload and accelerate data-center infrastructure services such as networking, storage and security processing. *Objective 2.10 | Review: Chapter 3*

11. C: Standard Kubernetes GPU scheduling uses the declared GPU resource requirement and the GPU resources advertised by nodes through the device-plugin mechanism. Telemetry does not automatically become a native utilization-based placement policy. *Objective 3.2 | Review: Chapter 10*

12. B: Modern AI progress reflects the combination of data, computational capability and algorithmic improvements. *Objective 1.4 | Review: Chapter 2*

13. C: Large-model training may require multiple GPUs for aggregate memory capacity and efficient communication among those accelerators. Host CPU count, local storage, or management-network bandwidth does not solve that specific constraint. *Objective 2.1 | Review: Chapters 3, 4*

14. A: Versioning, validation and controlled promotion support traceable releases and recovery. Monitoring alone does not provide release lineage. *Objective 1.7 | Review: Chapters 6, 8, 11*

15. C: Elasticity and direct control are common differentiators. Cost, availability, facility responsibility, and governance depend on context and should not be stated as universal absolutes. *Objective 2.4 | Review: Chapter 4*

16. A: GPU health and error counters can reveal memory-reliability problems that utilization alone will not show. The exact counters available depend on the GPU, but the evidence should match the reported memory-health symptom. *Objective 3.3 | Review: Chapter 9*

17. A: Applying AI to assist image interpretation is a healthcare AI use case. The other choices are infrastructure or operations technologies. *Objective 1.5 | Review: Chapter 2*

18. B: Frequent collective exchanges depend on fast node-to-node communication. East-west bandwidth and latency are therefore central requirements. *Objective 2.7 | Review: Chapter 5*

19. D: CPUs are optimized for versatile control flow and serial/general-purpose work. GPUs excel when computation can be expressed with high parallelism. *Objective 1.8 | Review: Chapter 3*

20. D: Cluster management and scheduling coordinate resources and workload placement across nodes. Communication libraries, storage, and device drivers are important cluster components but do not provide the overall placement/control function. *Objective 2.5 | Review: Chapters 4, 5*

21. A: TensorRT optimizes trained models for high-performance inference. The other choices concern monitoring, cluster management or networking. *Objective 1.6 | Review: Chapters 4, 7, 8, 9, 11*

22. C: Base Command Manager is positioned for cluster provisioning and management. TensorRT, NCCL and MIG serve inference, communication and partitioning roles. *Objective 3.1 | Review: Chapters 9, 11*

23. B: InfiniBand is a high-speed RDMA-capable data-center fabric commonly used for tightly coupled HPC and AI communication. *Objective 2.9 | Review: Chapter 5*

24. C: Batch execution does not make a workload training. Applying fixed trained parameters to inputs is inference. *Objective 1.2 | Review: Chapter 2*

25. B: The symptom is thermal and appears at sustained facility load, so heat-removal capacity is the relevant constraint. *Objective 2.3 | Review: Chapter 3*

26. D: CUDA is NVIDIA's parallel-computing platform and programming model/runtime ecosystem for GPU-accelerated applications. It operates with the GPU driver and is not a monitoring, virtualization, or physical networking technology. *Objective 1.1 | Review: Chapters 1, 6, 8*

27. D: Partitions, priorities, reservations and queued jobs are core Slurm scheduling concepts. The other choices are visualization, inference serving or data-science libraries. *Objective 3.2 | Review: Chapter 10*

28. A: Training infrastructure must feed accelerators at the required rate. Storage and ingestion performance are part of the hardware/data-path requirement, and increasing unrelated compute or control-plane capacity does not fix the bottleneck. *Objective 2.1 | Review: Chapters 3, 4*

29. A: The hierarchy distinguishes the broad AI field from machine learning and the deep-learning subset based on multi-layer neural-network approaches. *Objective 1.3 | Review: Chapter 2*

30. B: RDMA is a communication mechanism that can reduce CPU involvement and copying for suitable remote-memory transfers. It does not guarantee application scaling, create GPU partitions, or replace the underlying network technology. *Objective 2.8 | Review: Chapter 5*

31. A: Modern AI progress reflects interaction among algorithms, data, and compute. An idea can become practical at much larger scale when hardware and data availability catch up with the algorithmic approach. *Objective 1.4 | Review: Chapter 2*

32. A: Power and clock telemetry directly match the suspected power-limiting behavior. Unrelated metadata does not explain the operating state. *Objective 3.3 | Review: Chapter 9*

33. B: Insufficient facility cooling is a physical infrastructure constraint. Software configuration can address software or scheduling problems, but it cannot create missing heat-removal capacity. *Objective 2.6 | Review: Chapter 3*

34. C: A registry or equivalent tracking mechanism supports versioning, metadata, and lineage for controlled lifecycle operations. It does not guarantee model quality, replace evaluation, or schedule GPU jobs. *Objective 1.7 | Review: Chapters 6, 8, 11*

35. D: On-premises infrastructure can align well with direct control, data-location requirements, and stable demand when the organization already has suitable facilities. The other scenarios emphasize elasticity or temporary capacity. *Objective 2.4 | Review: Chapter 4*

36. C: AI tensor work often exposes large amounts of parallelism that GPUs are built to exploit. Clock frequency alone does not explain the advantage, and CPUs remain essential for general-purpose and control work. *Objective 1.8 | Review: Chapter 3*

37. D: This symptom points to the network path carrying training data from shared storage to compute nodes. The storage-to-compute path needs enough throughput and appropriate congestion behavior to keep GPUs fed. *Objective 2.7 | Review: Chapter 5*

38. D: Fraud detection is a common financial-services AI application based on identifying suspicious patterns or anomalies. *Objective 1.5 | Review: Chapter 2*

39. C: MIG partitions supported GPUs into separate GPU instances with isolated compute and memory resources. It is not a network transport and does not inherently require each workload to run in a separate VM. *Objective 3.4 | Review: Chapter 7*

40. D: An accelerated cluster depends on a balanced set of compute, network, storage/data access, and management components. Here the inter-node fabric is the cluster component limiting multi-node work. *Objective 2.5 | Review: Chapters 4, 5*

41. B: NVIDIA Dynamo-Triton is an inference-serving platform that supports multiple model frameworks and service interfaces. The other choices address partitioning, monitoring, or infrastructure offload. *Objective 1.6 | Review: Chapters 4, 7, 8, 9, 11*

42. C: DPUs and GPUs have complementary roles. DPUs focus on infrastructure-service offload, while GPUs focus on accelerated application computation such as AI workloads. *Objective 2.10 | Review: Chapter 3*

43. C: nvidia-smi is the direct local inspection tool for a host. NIM, TensorRT and NGC have deployment, optimization and distribution roles. *Objective 3.1 | Review: Chapters 9, 11*

44. A: Training and inference have different goals and common performance tendencies, but deployment platform and performance priorities vary. Avoid equating either workload with one scheduler, one orchestrator, or one fixed metric. *Objective 1.2 | Review: Chapter 2*

45. B: At larger scale, communication topology and efficiency can increasingly limit distributed training performance. Supporting infrastructure must scale with compute. *Objective 2.2 | Review: Chapter 4*

46. A: NCCL provides GPU communication primitives and relies on supported underlying interconnects or network transports. It should not be confused with the physical data-center fabric, a device driver, or virtualization software. *Objective 1.1 | Review: Chapters 1, 6, 8*

47. C: Kubernetes relies on a vendor device plugin to advertise GPUs as schedulable extended resources. The NVIDIA GPU Operator can manage deployment of this component, but monitoring, visualization, and communication tools do not expose the resource to the scheduler. *Objective 3.2 | Review: Chapter 10*

48. D: PUE compares total facility energy with IT-equipment energy. Values closer to 1.0 indicate lower non-IT facility overhead; PUE is not a GPU performance or scheduling metric. *Objective 2.3 | Review: Chapter 3*

49. B: InfiniBand is an RDMA-capable data-center fabric, while RoCE provides RDMA semantics over Ethernet. Both can appear in high-performance AI environments; the choice depends on architecture and operational requirements. *Objective 2.9 | Review: Chapter 5*

50. B: GPU monitoring requires multiple metrics. Memory capacity can constrain a workload even when compute utilization is moderate, so utilization alone is not enough to diagnose capacity pressure. *Objective 3.3 | Review: Chapter 9*

Post-exam review and weak-area retest

A mock score is useful only if it changes the next revision decision. Review the result by objective, not just by total percentage. The published domain weights help prioritize time, but any objective you cannot explain reliably should return to the revision queue.

- I can explain all 22 objective statements in my own words and point to the chapter that owns each concept.
- I can distinguish the major comparison pairs in Table A.3 without relying on product-name memorization.
- I can identify whether a scenario is primarily about AI knowledge, infrastructure, or operations, while recognizing cross-domain handoffs.
- I can explain why each incorrect mock answer is wrong, not only why the correct option is right.
- I have re-read the chapter sections connected to every guessed or missed question.
- I have reattempted weak objectives after a delay rather than immediately memorizing the answer key.
- I have checked NVIDIA's official certification page for any blueprint, logistics, or policy changes before the real exam.

Objective / topic	**Evidence of gap**	**Next action**	**Checkpoint**
Example: 3.2	Confused job scheduling with continuous orchestration	Re-read Chapter 10 comparison and explain one Slurm and one Kubernetes scenario	Retest with 3 new questions

Table A.4: Weak-area tracker for mock-exam review

Once the weak-area tracker is empty or reduced to a small number of clearly defined topics, return to the official blueprint one final time. Appendix B provides current reference links for version-sensitive product and platform details.

Appendix B

Referenced Resources and Further Reading

This appendix collects the primary documentation used to verify version-sensitive statements in the book and to support deeper study after the main chapters. It is intentionally selective. The goal is not to reproduce every link mentioned in the course, but to point you toward the most authoritative starting points for the technologies that appear in the NCA-AIIO objectives and in this guide.

The resource set was rechecked in October 2026. Product documentation, support matrices, certification policies, and software releases can change, so use the listed landing pages as the starting point and confirm the current release notes or support matrix before applying version-specific guidance.

Note

Resource note: NVIDIA documentation is preferred for NVIDIA products. For open-source technologies such as Kubernetes, Slurm, Prometheus, and Grafana, the project documentation is used as the primary source. URLs are printed in brackets so they remain visible in print and ebook exports.

Using this appendix

Use these resources in three ways: to verify facts that may change after publication, to explore a topic beyond the associate-level depth of this book, and to return to the source when a product name, support boundary, or operational workflow has changed. Stable landing pages are preferred over version-pinned pages. When the same resource supports several chapters, later sections cross-reference the earlier entry instead of repeating the URL.

Chapter 1: Navigating NCA-AIIO and the NVIDIA Technology Stack

- **NCA-AIIO certification page:** Use this as the source of truth for current exam logistics, certification level, blueprint weights, validity, and the link to the latest study guide. [`https://www.nvidia.com/en-us/learn/certification/ai-infrastructure-operations-associate/`]
- **NCA-AIIO exam study guide:** Review the current objective wording, domain weights, recommended training, and NVIDIA's suggested reading list before final exam preparation. [`https://dam-cdn.nvd.orangelogic.com/AssetLink/x874j05hy3m3r2sor84kpvp70750m468.pdf`]

Chapter 2: Understanding AI Workloads, Training, and Inference

- **TensorRT documentation:** Use this for deeper reading on inference optimization, engine generation, precision choices, and runtime behavior after a model has been trained. [`https://docs.nvidia.com/deeplearning/tensorrt/latest/`]
- For current objective wording and the training-versus-inference certification boundary, also use the NCA-AIIO certification page and study guide listed under *Chapter 1*. For serving-oriented inference resources, see the Dynamo-Triton and NIM entries under *Chapter 8*.

Chapter 3: Building the AI Data Center Foundation

- **NVIDIA DGX Systems documentation:** Use the system guides for current hardware, firmware, service, and platform information across the DGX family. Choose the guide for the system you are actually operating. [`https://docs.nvidia.com/dgx-systems/index.html`]
- **NVIDIA DGX SuperPOD documentation:** Use the deployment, administration, reference architecture, and data center design material to explore how compute, storage, networking, facility requirements, and cluster management come together at scale. [`https://docs.nvidia.com/dgx-superpod/index.html`]
- **NVIDIA DOCA Framework:** Use this for current BlueField and DOCA architecture, software packages, device roles, and the relationship between host software and the BlueField platform. [`https://docs.nvidia.com/doca/sdk/doca-framework/`]

Chapter 4: Scaling with NVIDIA Accelerated Computing Platforms

- **NVIDIA DGX Platform documentation:** Use this as the main hub for DGX platform software, BaseOS, DGX systems, DGX SuperPOD, DGX BasePOD, and related administration resources. [`https://docs.nvidia.com/dgx/`]
- **NVIDIA DGX BasePOD documentation:** Use the reference architectures and deployment guides for prescriptive scale-out AI infrastructure below the SuperPOD level. [`https://docs.nvidia.com/dgx-basepod/index.html`]
- **NVIDIA AI Enterprise documentation:** Use this to understand the current supported enterprise software platform across application and infrastructure layers, including planning, deployment, support, and release branches. [`https://docs.nvidia.com/ai-enterprise/index.html`]

Chapter 5: Moving Data Across Accelerated AI Infrastructure

- **NVIDIA Networking documentation:** Start here for current ConnectX adapters, BlueField DPUs and SuperNICs, Ethernet and InfiniBand switches, cables, drivers, and deployment planning resources. [`https://docs.nvidia.com/networking/`]
- **GPUDirect RDMA documentation:** Use this for the direct GPU-to-peer-device data path, platform requirements, PCIe considerations, and current API guidance. [`https://docs.nvidia.com/cuda/gpudirect-rdma/index.html`]
- **NCCL documentation:** Use this for current collective communication concepts, installation, supported communication primitives, topology-aware behavior, and release notes. [`https://docs.nvidia.com/deeplearning/nccl/`]

Chapter 6: Understanding the NVIDIA GPU Software Stack

- **CUDA Toolkit documentation:** Use the documentation hub for installation guides, the programming guide, release notes, libraries, compiler documentation, and samples. [`https://docs.nvidia.com/cuda/`]
- **CUDA Compatibility documentation:** Use this when checking driver and toolkit compatibility, backward compatibility, minor-version compatibility, and forward-compatibility requirements. [`https://docs.nvidia.com/deploy/cuda-compatibility/latest/`]
- **GPU Deployment and Management documentation:** Use this hub for driver and toolkit compatibility, nvidia-smi, NVML, GPU management interfaces, health diagnostics, and operational references. [`https://docs.nvidia.com/deploy/index.html`]
- **NVIDIA Container Toolkit documentation:** Use this for current GPU-container runtime integration, supported platforms, installation, Container Device Interface support, and runtime configuration. [`https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/latest/`]

Chapter 7: Virtualizing Accelerated Infrastructure with vGPU and MIG

- **NVIDIA vGPU Software documentation:** Use the current release documentation and support information for vGPU software, supported products, hypervisors, guest operating systems, configuration, and troubleshooting. [`https://docs.nvidia.com/vgpu/latest/`]
- **NVIDIA MIG User Guide:** Use this for supported GPUs, MIG concepts, profiles, deployment considerations, configuration, device naming, and container or Kubernetes integration. [`https://docs.nvidia.com/datacenter/tesla/mig-user-guide/`]

Chapter 8: Mapping NVIDIA Software to the AI Lifecycle

- **NVIDIA AI Enterprise documentation:** Use this as the enterprise software-platform map across application development and infrastructure management. The *Chapter 4* entry remains the same canonical landing page. [`https://docs.nvidia.com/ai-enterprise/index.html`]
- **NVIDIA NGC documentation:** Use this for NGC accounts, catalog content, containers, models, Helm charts, private registries, APIs, and current platform guidance. [`https://docs.nvidia.com/ngc/index.html`]
- **NVIDIA RAPIDS documentation:** Use this for GPU-accelerated data science libraries, installation guidance, user manuals, and release notes. [`https://docs.rapids.ai/`]
- **NVIDIA NeMo documentation:** Use this to explore the current NeMo suite for model and agent development, data preparation, customization, evaluation, and deployment workflows. [`https://docs.nvidia.com/nemo/`]
- **TensorRT documentation:** Use this for the current inference optimization SDK, model compilation, precision options, engine creation, runtime APIs, and deployment guidance. [`https://docs.nvidia.com/deeplearning/tensorrt/latest/`]
- **NVIDIA Dynamo-Triton (formerly Triton Inference Server) documentation:** Use this for model serving architecture, supported backends, model repositories, scheduling and batching, APIs, metrics, and deployment guidance. [`https://docs.nvidia.com/deeplearning/triton-inference-server/user-guide/docs/index.html`]
- **NVIDIA NIM documentation:** Use this for current NIM microservices, deployment guides, model-specific prerequisites, support matrices, APIs, and release notes. [`https://docs.nvidia.com/nim/`]

Chapter 9: Monitoring and Managing GPU Infrastructure

- For nvidia-smi and NVML, use the GPU Deployment and Management documentation listed under *Chapter 6*.
- **NVIDIA DCGM documentation:** Use this for DCGM concepts, installation, diagnostics, telemetry fields, command-line reference, DCGM Exporter, Prometheus integration, and release notes. [`https://docs.nvidia.com/datacenter/dcgm/latest/`]
- **NVIDIA Base Command Manager documentation:** Use this for current cluster provisioning, infrastructure monitoring, administration, workload management integration, release notes, and support information. [`https://docs.nvidia.com/base-command-manager/`]
- **Prometheus documentation:** Use the project documentation for the Prometheus data model, scraping, exporters, PromQL, alerting, and monitoring architecture. [`https://prometheus.io/docs/introduction/overview/`]
- **Grafana documentation:** Use the project documentation for dashboards, visualization, alerting, data sources, provisioning, and operational administration. [`https://grafana.com/docs/grafana/latest/introduction/`]

Chapter 10: Scheduling and Orchestrating Accelerated Workloads

- **Slurm GRES documentation:** Use this as the primary reference for Generic RESources, including GPUs, device files, and GPU resource configuration in Slurm. [`https://slurm.schedmd.com/gres.html`]
- **Kubernetes GPU scheduling documentation:** Use this for the Kubernetes device-plugin model, nvidia.com/gpu-style extended resources, GPU requests and limits, node labels, and GPU scheduling constraints. [`https://kubernetes.io/docs/tasks/manage-gpus/scheduling-gpus/`]
- **NVIDIA GPU Operator documentation:** Use this for the current GPU Operator architecture, installation, platform support, drivers, Container Toolkit, device plugin, MIG, DCGM Exporter, and GPUDirect integration. [`https://docs.nvidia.com/datacenter/cloud-native/gpu-operator/latest/`]

Chapter 11: Recognizing MLOps Practices in the AI Lifecycle

Chapter 11 deliberately treats MLOps at recognition level rather than introducing a separate enterprise toolchain. For current lifecycle software, use the AI Enterprise, NGC, RAPIDS, NeMo, TensorRT, Dynamo-Triton, and NIM resources listed under *Chapter 8*. For infrastructure telemetry and monitoring feedback, use the DCGM, Prometheus, and Grafana resources listed under *Chapter 9*. These primary sources are more useful than duplicating the same URLs here.

Chapter 12 and Appendix A: Certification Preparation and Mock Exams

- Before scheduling or taking the exam, recheck the NCA-AIIO certification page and the current study guide listed under *Chapter 1*. Do not rely on the logistics printed in the book if the official page has changed.
- **NVIDIA Certification Programs and policies:** Use the current certification FAQ and policy information for registration, cancellation and rescheduling, remote-proctoring rules, scoring, retakes, and certification renewal. [`https://www.nvidia.com/en-us/learn/certification/`]

Keeping the resource list current

The documentation set will continue to evolve after publication. When a link redirects, begin from the product landing page rather than an archived release page. When a command, support matrix, or compatibility claim matters to an operational decision, check the current release notes and support documentation for the exact software and hardware versions in use. For certification logistics and objective wording, the NCA-AIIO certification page and its linked study guide always take precedence over the static text in this book.



https://www.packtpub.com

Subscribe to our online digital library for full access to over 7,000 books and videos, as well as industry leading tools to help you plan your personal development and advance your career. For more information, please visit our website.

Why subscribe?

- Spend less time learning and more time coding with practical eBooks and Videos from over 4,000 industry professionals
- Improve your learning with Skill Plans built especially for you
- Get a free eBook or video every month
- Fully searchable for easy access to vital information
- Copy and paste, print, and bookmark content

At https://www.packtpub.com, you can also read a collection of free technical articles, sign up for a range of free newsletters, and receive exclusive discounts and offers on Packt books and eBooks.

Other Books You May Enjoy

If you enjoyed this book, you may be interested in these other books by Packt:

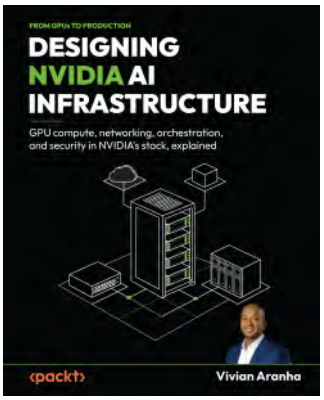


GPU-Accelerated Computing with Python 3 and CUDA

Niels Cautaerts, Hossein Ghorbanfekr

ISBN: 978-1-80324-542-3

- Understand GPU execution, parallelism, and the CUDA programming model
- Write, launch, and debug custom CUDA kernels in Python with CUDA
- Profile GPU code with NVIDIA Nsight and optimize memory access
- Use CUDA streams and async execution to overlap compute and transfers
- Apply JAX, CuPy, and RAPIDS to numerical computing and machine learning
- Scale GPU workloads across devices using Dask and multi-GPU strategies
- Accelerate PDE solvers, simulations, and image processing on the GPU
- Build, train, and run a transformer model from scratch on the GPU



Designing NVIDIA AI Infrastructure

Vivian Aranha

ISBN: 978-1-80808-013-5

- Understand what MIG and vGPU isolate and what they don't
- Distinguish RBAC, network policy, and encryption's separate roles
- See how storage, NVLink, and InfiniBand affect GPU utilization
- Recognize where Kubernetes tools' responsibilities stop
- Understand how GDPR, HIPAA, and FedRAMP shape AI infrastructure controls and evidence
- Use GPU profiling and telemetry data to investigate bottlenecks
- Learn how NGC, Triton, and ensembles fit a serving pipeline
- Compare on-prem, cloud, and hybrid AI cluster trade-offs

Packt is searching for authors like you

If you're interested in becoming an author for Packt, please visit `https://authors.packt.com` and apply today. We have worked with thousands of developers and tech professionals, just like you, to help them share their insight with the global tech community. You can make a general application, apply for a specific hot topic that we are recruiting an author for, or submit your own idea.

Share your thoughts

Now you've finished *NVIDIA NCA-AIIO Certification Study Guide*, we'd love to hear your thoughts! Scan the QR code below to go straight to the Amazon review page for this book and share your feedback or leave a review on the site that you purchased it from.



`https://packt.link/r/1808818474`

Your review is important to us and the tech community and will help us make sure we're delivering excellent quality content.

Index

A

AI Infrastructure
- reviewing 208 – 210

AI Operations
- reviewing 210, 211

AI data center
- network zones 95
- traffic 93

AI factory **153**
- architecture, connecting to outcome 154

AI use cases
- automotive and autonomous systems 39
- finance and banking 40
- healthcare and life-science workloads 40
- manufacturing 40, 41
- retail 40
- video analytics 40

AI workflow **144, 145**

AI-centric data center **51**
- compute 52
- facility and support infrastructure 51
- networking 52
- storage 52, 53

application programming interfaces (APIs) **149**

artificial intelligence (AI) **31**

B

Base Command Manager (BCM)
- positioning 171

Baseboard Management Controller (BMC) connection **93**

BlueField
- using for infrastructure offload and isolation 80

BlueField DPU **80**

backend **149**

bandwidth **101**

baseline knowledge check **16**
- ratings, turning into priorities 18

C

CPU and GPU architecture
- comparisons for AI workloads 57 – 59

CUDA **121**
- installation guide 122
- installation path 122
- placing, between driver and higher-level software 121

CUDA Cores **61, 62**

CUDA-X **146**
- reference link 146

Clara **152**

ConnectX
- using for high-performance network path 80

central processing units (CPUs) **49**

cloud infrastructure **82**

code **195**

combined deployment patterns **83**

comparisons and terminology traps **212**

compute traffic **93, 94**

configuration **195**

container orchestrator **179**

continuous delivery **196**

continuous integration (CI) **196**
control-heavy pattern
recognizing 58
converged Ethernet **98, 99**
cooling and thermal limits
checking 54
cross-domain scenario **12**
current blueprint and exam logistics
rechecking 204, 205

D

DCGM Exporter **170**
DGX SuperPOD **77**
positioning 78, 79
DGX, as integrated AI system **75**
component map 75, 76
stable roles, separating from generation-specific details 76, 77
DPU responsibilities **59 – 61**
data **34, 195**
data model
parameters 34
data processing unit (DPU) **49, 107**
data-parallel pattern
recognizing 58
deep learning **31**
direct memory access (DMA) **105, 106**
durable concepts **13**

E

Essential AI Knowledge
reviewing 206
Ethernet **97 – 99**
traffic, converging 98
electrical capacity
checking 54
end-to-end data path
tracing 109
environment **195**
event-driven retraining **197**
exam domains, NCA-AIIO **4**
AI Infrastructure 5
AI Operations 6
Essential AI Knowledge 5
exam scenarios
traps, avoiding 41, 42
external/client boundary (context) **94, 95**

F

first-response reasoning **172**
five-question classification routine **43**

G

GPU Instance (GI) **134**
GPU sharing **130, 131**
GPU software dependency chain **116, 117**
operating system, placing at base 118
reading 118, 119
GPU solution categories
to integrated platforms 72 – 74
GPU virtualization
foundations 131, 132
GPUDirect RDMA **106 – 109**
GPUDirect Storage **108, 109**
Generic Resources (GRES) mechanism **182**
Grafana
to visualize retained telemetry 170
generative AI **31, 32**
goals and GPU measures
monitoring 160 – 164
graphics processing unit (GPU) **49**

H

high-performance computing (HPC) **99, 179**

I

InfiniBand **98, 99**

in-band management traffic **93, 96**
independent out-of-band path **96**
inference **35, 36**
infrastructure signals **37**
inference pressure 38, 39
larger datasets 38
larger models 38

J

job scheduler **179**

K

Kubernetes **179**
essentials 183, 184
GPU resources, requesting 184
health 183
scaling 183
Service 183

L

latency **101**
learning system **14**
accelerated AI infrastructure, designing 15
AI and workload foundations, establishing 15
AI infrastructure, operating 16
AI infrastructure, optimizing 16
NVIDIA software and virtualization stack, building 15
recurring chapter elements, using 16
lineage **196**

M

MIG profile
reading 134
ML framework **146**
Merlin **153**
Multi-Instance GPU (MIG) **133 – 135**
inspection evidence 138
licensing 136
profiles 136 – 138
support 136
target GPU, identifying 136, 137
versus NVIDIA vGPU 135
machine learning (ML) **31**
machine learning operations (MLOps) **194**
managed feedback loop 195
management traffic **95**
microservice **150**
mixed environments
operating 187
mixed-domain practice set **217**
mock exams attempt **231, 232**
model artifacts **195**
model registry **196**
modern AI adoption
drivers 28, 29
monitoring **198**

N

NCA-AIIO **3**
current exam snapshot 3
exam blueprint 6
exam domains 4
study guide 6
weights 6
NGC Catalog **152**
NGC Private Registry **152**
NIM **150**
for packaged inference capability 150
NVIDIA AI Enterprise **151**
NVIDIA CUDA Deep Neural Network (cuDNN) **146**
reference link 146
NVIDIA CUDA DeepNeural Network (cuDNN)
library 122
NVIDIA Collective Communications Library (NCCL) **122, 123, 146, 182**
collective operations 123 – 125
framework integration 125
version-aware

reference link 146
separating, from data path 123
topology awareness, using 125
without overclaiming automation

NVIDIA DOCA 80

NVIDIA Data Center GPU Manager (DCGM) 167
entities 167, 168
fields 167, 168
groups 167, 168
passive health monitoring, separating from active diagnostics 168, 169
watches 167, 168

NVIDIA Dynamo-Triton 150
for model serving 149
reference link 149

NVIDIA GPU Cloud (NGC) 152, 196
distribution, keeping separate from support 152
selected domain-specific software 152

NVIDIA driver 119
current driver 119
documentation, consulting
driver success, keeping separate from application success 119
operating system, connecting to GPU 119

NVIDIA integration points 186
Kubernetes and NVIDIA GPU enablement 186
Slurm and NVIDIA GPU resources 186

NVIDIA reference architectures 81

NVIDIA tools
mapping, to lifecycle operations 197, 198

NVIDIA vGPU 133 – 135
host-to-guest path 133
inspection evidence 138
licensing 136
support 136
versus MIG 135

NVLink 102 – 105

NVMe over Fabrics (NVMe-oF) 109

NVSwitch 103 – 105
NVLink fabric, building 103

NeMo Curator 148
for training-data curation 148
reference link 148

NeMo Framework 147
for model development 148

Non-Volatile Memory Express (NVMe) 62, 74, 101

network interface card (NIC) 101

nvidia-smi 171, 172

nvidia-smi command-line utility 120
fields 120, 121

nvidia-smi, for immediate inspection 165
local observation, repeating 166
selective script-friendly 165
snapshot, capturing
snapshot, turning into next action 166

O

OpenSM 109

objective-to-chapter map 227

on-premises infrastructure 82

operational roles 7, 8

optimization 35

out-of-band management traffic 93, 94

P

PCI Express (PCIe) 101 – 103

Pod 183

Prometheus
for collection and retained telemetry 170

plan 149

platform generation changes
recognizing 81

power usage effectiveness (PUE) **55**
calculation 56
using, without overinterpreting 57
product-name confusion
controlling, with lifecycle responsibility 208
production scenario **199**

Q

Quality of Service (QOS) **181**

R

RAPIDS
for data processing and analytics 147, 148
references 147
RDMA over Converged Ethernet (RoCE) **80, 97 – 99**
RT Cores **61, 62**
rack and physical-space constraints
checking 53
readiness planning and targeted revision **216**
final two-week pass 217
measurable evidence 216
mock exams 217
weak-area tracker 216
realistic study and practice plan **20, 21**
certification scenarios 22
weak-area tracker 21
relationships
building 207
remote direct memory access (RDMA) **80, 105, 106, 182**
responsibility-first scenario method **188**

S

Secure Shell (SSH) session **93**
Slurm
essentials 181 – 183
versus Kubernetes 184, 185
Slurm Workload Manager **181**
scenario questions **214, 215**
scheduling
versus orchestration 179, 180
six-layer NVIDIA technology stack **8**
applications and vertical solutions 10
as orientation model 11
connecting to AI lifecycle 11
core libraries 10
data movement and I/O acceleration 10
infrastructure support 11
monitoring and management 10
OS, driver, and virtualization 10
physical layer 9
software development kits (SDKs) **151**
solid-state drive (SSD) **101**
storage roles for AI workloads **62**
capacity 62
complementary roles 63, 64
latency 62
local NVMe, for active data 63
network file services 63
object storage 63
shared and parallel storage 63
storage-tiering pattern 63
throughput 62
storage traffic **93, 94**

T

Tensor Cores **61, 62**
TensorRT **150**
build and runtime roles 149
reference link 149
tasks
classifying, by domain and operational owner 18, 19
suggested classification 19, 20
throughput **101**
traffic
compute traffic 93, 94

external/client boundary (context) 94, 95
in AI data center 93
in-band management traffic 93, 96
management traffic 95
out-of-band management traffic 93, 94
storage traffic 93, 94

training 34

transformers, at associate level 36
attention 36
next-word prediction example 36
prediction loop 37

V

version-aware reading habit 14

versioning 195

W

workload-management system 179