CDW Documentation

NVIDIA NCP-AIO Training Plan

NVIDIA NCP-AIO Training Plan

Certification: NVIDIA-Certified Professional: AI Operations (NCP-AIO)
Format: 30 multiple-choice questions + 3 hands-on lab exercises, 120 minutes total
Cost: $500 USD, online proctored
Recommended experience: 2–3 years of data center work with NVIDIA hardware.
Plan length: 8 weeks @ ~8–10 hours/week (≈70 hours total)

Topics Areas	% of Exam	Topics Covered
Installation and Deployment	31%	* Describe the Mission Control toolkit * Use BCM’s Base View interface to monitor cluster performance, resource utilization, and node health in real time. * Manage job scheduling and resource allocation using BCM’s workload manager (e.g., SLURM or Kubernetes) * Apply patches, update firmware, and synchronize software images across cluster nodes using BCM * Administer user accounts, roles, and permissions to ensure secure access to the cluster using BCM * Configure and monitor network settings for cluster nodes, DPUs, and switches using BCM * Diagnose and resolve cluster issues, such as job failures, node outages, or resource bottlenecks, using BCM. * Use BCM to organize and configure compute nodes into categories based on hardware or workload requirements. * Using BCM, maintain documentation and generate reports on cluster usage, performance, and issues. * Install and initialize Kubernetes on NVIDIA hosts using BCM * Deploy DOCA Services on DPU Arm * Install Run:ai * Install Slurm
Administration	23%	* Administer Slurm cluster. * Describe data center architecture for AI Workloads * Administer Run:ai * Administer Kubernetes * Configure MIG
Workload Management	23%	* Deploy inference workloads with Kubernetes * Deploy inference workloads with Run:ai * Deploy training workloads with Slurm * Deploy training workloads with Run:ai * Use system management tools to troubleshoot issues\ * Allocate resources between teams with Run:ai, Slurm and Kubernetes * Deploy containers from NGC
Troubleshooting and Optimization	23%	* Troubleshoot Docker * Troubleshoot the fabric manager service for NVLink and NVSwitch systems * Troubleshoot Base Command Manager\\. * Troubleshoot Magnum IO components * Troubleshoot storage performance * Troubleshoot the deployment of a container from NGC

How to Use This Plan

Each week is built around a tight cluster of related exam tasks. Every module has four parts:

1. **Objectives** — the exact NCP-AIO tasks the week maps to.
2. **Concepts to master** — the technical content you need fluency in.
3. **Resources** — official NVIDIA docs, courses, and open-source references.
4. **Hands-on lab** — a practical exercise. The exam has a live cluster lab section, so command-line muscle memory matters more than reading.
5. **Self-check** — questions and tasks to confirm you can perform, not just recognize.

If you do not have hardware, NVIDIA LaunchPad provides free, time-boxed access to real NVIDIA stacks (BCM, DGX, BlueField, Run:ai). Pair it with a small cloud GPU instance (one A100 or L40S is enough for MIG and single-node practice).

The exam's four domains and approximate weights:

Domain	Weight
Administration	~30%
Workload Management	~20%
Installation & Deployment	~30%
Troubleshooting	~20%

Pre-Work (Days 1–3, before Week 1)

Before starting, confirm your baseline. If any of these feel shaky, address them first.

- **Linux command-line fluency:** `systemd`, `journalctl`, `ssh`, `scp`, package management on Ubuntu and RHEL families.
- **Networking basics:** IP, routing, VLANs, `ip`, `ss`, `tcpdump`, basic InfiniBand vocabulary (`ibstat`, subnet manager).
- **Containers:** Docker daemon and CLI, image vs container, registries, `docker logs`, `docker exec`.
- **GPUs at the OS level:** `nvidia-smi`, NVIDIA driver vs CUDA toolkit vs Container Toolkit (the three are not the same).

Pre-work resources:

- NVIDIA Academy: *AI Infrastructure and Operations Fundamentals* (self-paced, ~7 hours). Aligns vocabulary.
- *NVIDIA NCP-AIO Exam Study Guide* (PDF, document number 3770900). Read cover-to-cover and copy the blueprint into your notes.
- Linux Foundation: *Introduction to Linux* (free) if Linux fundamentals are weak.

Set up your environment now:

- Sign up for NVIDIA LaunchPad and request the *Base Command Manager* and *Run:ai* labs (they queue).
- Provision one cloud instance with an NVIDIA GPU (A100 ideal, L40S/T4 acceptable for non-MIG topics).
- Create a study notes repository — one markdown file per week, plus a `commands.md` you keep adding to. Re-running your own past commands during the lab portion of the exam is the fastest way to recover under pressure.

Note on DC Lab Sim (dclabsim.com / Seanbo5386/dc-lab-sim)

If you have DC Lab Sim installed, treat it as a high-value but partial supplement to this plan. Important context:

- It is built for NCP-AII, not AIO. Its 32 scenarios map to AII's 5 domains (Systems Bring-Up 31%, Physical Layer 5%, Control Plane 19%, Cluster Test & Verification 33%, Troubleshooting 12%) — different weights and emphasis than AIO.
- It is not affiliated with NVIDIA and outputs are approximations.

Where it directly supports this AIO plan:

- **Week 4 (Slurm):** drill the `sinfo`, `squeue`, `scontrol`, `sbatch`, and `srun` simulators until invocation is reflexive.
- **Week 7 (MIG, NGC, Docker):** drill `nvidia-smi -mig` workflows, `docker`, and `enroot` commands.
- **Week 8 (Troubleshooting):** the largest overlap. Drill `nvidia-smi` (especially `-q`, `dmon`, `nvlink -s`, `topo -m`), `dcgmi diag`, `ibstat`, `iblinkinfo`, and `ipmitool`. Run as many of the 32 scenarios as you can — even those framed for AII bring-up will sharpen your diagnostic instinct.

Where it is silent on AIO content (do not rely on the simulator for these — use LaunchPad, real labs, and docs):

- Run:ai (`runai` CLI, projects, fractional GPUs, multi-tenant quotas)
- Kubernetes (`kubectl`, GPU Operator, Triton, NIM)
- BCM Base View administration, user/role/profile management, reports
- Mission Control toolkit
- Multi-tenant resource allocation across Run:ai, Slurm, and K8s

In the weeks below, look for **Simulator drills** callouts where it earns its keep.

Your Lab Environment Mapping

You have three complementary environments:

Primary: BCM-managed hybrid cluster.

- BCM head node in your home lab.
- BCM Cloud Director in AWS (BCM provisioning, DHCP/DNS, image cache for cloud nodes).

- A small dedicated AWS instance as the Kubernetes control plane (BCM-deployed via `cm-kubernetes-setup`).
- 3 AWS GPU worker nodes, sized appropriately per task (`p4d.24xlarge` for A100, `p5.48xlarge` for H100, smaller g5/g6 for cheap practice).
- *No Run:ai license — Run:ai installation cannot be performed on this cluster.*

This is where Weeks 2, 3, 4, 5, and 7 happen.

Secondary: External A100 cluster with Run:ai pre-installed.

- Not BCM-managed; not on your network.
- You have user-level access: can submit workloads (`runai submit`), use fractional GPU and (potentially) MIG profile requests, observe your own workloads, and navigate whatever project/department structure your role exposes.
- You likely cannot install, configure node roles, create projects/departments/quotas, or change cluster-level settings. Worth requesting at minimum read-only admin access from the cluster owner — even observer-level visibility into cluster-wide utilization closes a substantial admin-task gap for exam prep.

This is where Week 6 hands-on workload submission and as much administration as your role permits happen.

Tertiary: Two DGX Sparks running MicroK8s, connected back-to-back via QSFP112 in RoCE mode. GB10 Grace Blackwell, 128 GB unified memory each, ConnectX-7 at 200 Gb/s. The Spark's CX-7 is firmware-locked to Ethernet (per the official DGX Spark User Guide: *"The DGX Spark CX-7 ports support ethernet configuration only"*). The Sparks give you real RDMA, GPUDirect RDMA, and NCCL diagnostics over RoCE — useful for cheap K8s iteration in Week 5 and Week 8 RoCE-flavored troubleshooting. **Explicitly not supported by Run:ai** (DGX Spark is on the Run:ai unsupported list per official docs).

This covers the bulk of the plan with real hardware. Below is what fits where, and where to substitute.

What to run where

Plan content	Run on	Notes
Week 2–3 BCM (Base View, `cmsh`, categories, images, users, reports)	BCM head node + AWS-deployed compute nodes	Best possible setup. Use cheap `g5.xlarge` or `g6.xlarge` nodes for image and category practice; bring up p4d/p5 only for the workload-heavy exercises.
Week 4 Slurm install + multi-node training	BCM + AWS GPU nodes	Provision two GPU nodes, install Slurm with Enroot/Pyxis through BCM's Slurm role. Cross-node NCCL works over AWS EFA.
Week 5 Kubernetes admin + inference	**Primary:** BCM-managed AWS K8s cluster (deployed via `cm-kubernetes-setup`). **Optional alternative:** two-Spark MicroK8s for cheap, fast iteration on basic `kubectl`/GPU Operator concepts.	Do the exam-relevant K8s administration work — BCM-deployed K8s, GPU Operator manual install, Triton/NIM inference deployments, ResourceQuotas — on the BCM AWS cluster. Use Sparks only for tinkering on concepts you can test cheaply.

Plan content	Run on	Notes
Week 6 Run:ai workload submission, project navigation, fractional GPU, MIG profile requests	**External A100 Run:ai cluster**	Submit workloads via `runai submit`, navigate UI within your role's visibility, exercise fractional GPU and (if cluster admin has enabled it) Dynamic MIG profile requests. Request observer/department-admin access from the cluster owner if possible — closes most of the admin-task gap.
Week 6 Run:ai install + cluster-level admin (creating departments, projects, quotas, node roles)	**LaunchPad Run:ai lab** + documentation study	The install task is on the exam. You cannot install Run:ai on your BCM AWS cluster (no license) and you cannot install on the A100 cluster (already installed by someone else). LaunchPad is the only hands-on install venue. Document the install flow as your own runbook from BCM/Run:ai docs first, then execute it once on LaunchPad.
Week 7 MIG (low-level `nvidia-smi mig`, GPU Operator MIG Manager mode switching)	BCM-managed AWS K8s cluster (size at least one worker as `p4d.24xlarge` for A100 or `p5.48xlarge` for H100)	Needs shell access to the GPU node and admin access to the K8s cluster — both of which you have on your BCM AWS cluster.
Week 7 Run:ai-orchestrated MIG (`–mig-profile` workload requests)	External A100 Run:ai cluster — **only if** the cluster admin has enabled Dynamic MIG on a node	Otherwise covered on LaunchPad alongside the Run:ai install lab.
Week 7 NGC container deployment	Anywhere	NGC steps work anywhere. The BCM AWS cluster is convenient since you're already there.
Week 8 `nvidia-smi` / `dcgmi` / Docker troubleshooting	Either Spark or AWS	Both work. Spark is closer at hand.
Week 8 RDMA diagnostics, NCCL over RDMA, GPUDirect RDMA	**Two-Spark RoCE link** for the diagnostic command set; **AWS EFA** on the BCM cluster for a different RDMA flavor	Sparks: real `ibstat`/`iblinkinfo`/`ib_write_bw` output, NCCL reporting `NET/IB` over RoCE. AWS EFA: NCCL works (`NCCL_PROTO`, EFA-specific debug), but `ibstat` won't show `Link layer: InfiniBand` because EFA is its own transport. Neither is native IB; for that you need LaunchPad.

Gaps your environment cannot cover

For these, plan to use NVIDIA LaunchPad (request the labs early — they queue), or supplement with the simulator and documentation walkthroughs.

1. **DOCA Services on DPU Arm (Week 7).** No BlueField DPU in Spark or in standard AWS GPU instances. LaunchPad's BlueField environment is the realistic option — request it in advance. Without LaunchPad, this becomes a paper exercise from the DOCA documentation; the DC Lab Sim simulator will not help here either.

1. **Native InfiniBand experience (Weeks 4 and 8).** Neither environment gives you a true switched IB fabric. AWS uses EFA — a different RDMA transport that NCCL works over but that doesn't surface `Link layer: InfiniBand` in `ibstat`, has no Subnet Manager, and uses

different diagnostic tools (`fi_info`, `aws-efa-installer` checks). DGX Spark is RoCE — RDMA over Ethernet, also not native IB. For LIDs, OpenSM, `ibping`, `ibhosts`, and `ibnetdiscover` topology discovery, you need LaunchPad DGX environments with real Quantum-2 IB. Cover the IB-specific topics from documentation; for hands-on, LaunchPad is the answer.

1. **Fabric Manager (Week 8).** Fabric Manager is the NVLink/NVSwitch daemon. Neither AWS GPU nodes nor Sparks have NVSwitch — Spark has on-die NVLink C2C between Grace and Blackwell (no switched fabric), and standard AWS GPU instances aren't NVSwitch-equipped (DGX Cloud SuperPOD-class instances are, but those aren't what you're spinning up). For real Fabric Manager troubleshooting drills you need a DGX A100/H100/H200/GB200 — LaunchPad.

ARM caveat for Spark labs

DGX Spark uses an ARM Grace CPU. Most modern NGC containers ship multi-arch images, but verify before each lab — `docker manifest inspect <image>` shows whether `linux/arm64` is present. The symptom of a missing ARM variant is `exec format error` at container start. When in doubt for x86-only containers, fall back to your AWS-deployed nodes.

One-time prep checklist for your environment

Before Week 1, confirm each of these works once so you don't lose a study session to setup:

BCM and AWS K8s cluster (primary environment):

- [ ] `cmsh` on the BCM head node returns a prompt; you can list categories and nodes.
- [ ] You can deploy and tear down at least one AWS GPU instance via BCM and see it appear in Base View.
- [ ] BCM Cloud Director is healthy in AWS; cloud nodes provision through it without WAN-routed image transfer.
- [ ] `cm-kubernetes-setup` wizard has run successfully; `kubectl get nodes` from the K8s control-plane node shows control plane and all GPU workers `Ready`.
- [ ] All `kube-system` and `gpu-operator` namespace pods are `Running`. `kubectl describe node <gpu-worker>` shows `nvidia.com/gpu` under Capacity and Allocatable.
- [ ] A trivial CUDA pod (`nvcr.io/nvidia/cuda:12.x-base` requesting `nvidia.com/gpu: 1`) schedules, runs `nvidia-smi`, and exits cleanly.
- [ ] Default `StorageClass` is set; a test PVC reaches `Bound` state.
- [ ] If you'll be testing low-level MIG: at least one GPU worker is sized as `p4d.24xlarge` (A100) or `p5.48xlarge` (H100). G-series instances do not support MIG.

External A100 Run:ai cluster (secondary environment):

- [ ] You can `runai login` and the CLI is configured to point at the cluster.
- [ ] `runai list projects` shows at least one project you have submit rights on.
- [ ] A trivial test workload (`runai submit test –image nvcr.io/nvidia/cuda:12.x-base -g 1 – nvidia-smi`) schedules, runs, and completes — you can read the logs with `runai logs test`.
- [ ] You've asked the cluster owner whether observer/department-admin access is available; if it is, you have it.
- [ ] You've asked the cluster owner whether Dynamic MIG is enabled on any node — affects

whether you can do –`mig-profile` workload requests there or need LaunchPad for that exercise.

LaunchPad (for install + admin tasks you can't do locally):

- [ ] You have requested a Run:ai LaunchPad lab slot. These queue — request well before Week 6.
- [ ] You have also requested DOCA/BlueField, native IB, and Fabric Manager labs as needed for Weeks 7 and 8.

Sparks (tertiary environment):

- [ ] On both Sparks: `microk8s status` shows ready; `microk8s kubectl get nodes` shows both nodes; `microk8s kubectl describe node` shows GPU resources advertised.
- [ ] On both Sparks: `ibstat` shows the connected port `State: Active`, `Physical state: LinkUp`, `Link layer: Ethernet`. (Spark CX-7 is locked to Ethernet — that's expected, not a problem.) `ibdev2netdev` confirms which `roce`/`enp` interfaces are in use.
- [ ] A baseline `ib_write_bw` test between the two Sparks completes and reports a sane bandwidth number, confirming RDMA over RoCE is working.
- [ ] A baseline `nccl-tests` `all_reduce_perf` between the two Sparks completes with `NCCL_DEBUG=INFO` showing `NET/IB` transport selected (NCCL labels RoCE-mode RDMA as `IB` — that's expected).

General:

- [ ] You have an NGC API key configured (`ngc config set`).

Week 1 — Foundations: Data Center Architecture & Mission Control

Exam tasks covered:

- Describe data center architecture for AI workloads
- Describe the Mission Control toolkit

This week is conceptual. You are building the mental map that every later task hangs from.

Concepts to master

- **AI factory architecture:** the four planes — compute (GPU servers like DGX/HGX), networking (InfiniBand for east-west, Ethernet for north-south, NVLink/NVSwitch for intra-node), storage (high-throughput parallel file systems, often with GPUDirect Storage), and management (BCM, schedulers, observability).
- **Reference architectures:** DGX BasePOD and DGX SuperPOD. Know what they are, the rough scale of each, and which NVIDIA-validated networking and storage partners ship with them.
- **NVIDIA Mission Control:** the unified software stack for managing AI factories. Know its scope — it bundles BCM, Run:ai, observability, and lifecycle automation under one operations layer for full-stack management of GPU clusters. Be able to name what sits inside it and what role each piece plays.

- **Workload classes:** training (long-running, multi-node, bandwidth-sensitive, often Slurm) vs. inference (short-lived, latency-sensitive, often Kubernetes) vs. mixed (Run:ai for fractional and dynamic allocation).
- **Networking topologies:** rail-optimized fat-tree designs, the role of leaf/spine for storage and management, the role of NVLink/NVSwitch for tight GPU coupling within a node or NVLink domain.

Resources

- NVIDIA: *DGX BasePOD Reference Architecture* white paper.
- NVIDIA: *DGX SuperPOD Reference Architecture* white paper (current Hopper/Blackwell generation).
- NVIDIA blog and product pages on Mission Control.
- NVIDIA *Magnum IO* overview page (you'll go deeper on Magnum IO in Week 8 troubleshooting).

Hands-on lab

This week is mostly reading, but produce two artifacts:

1. A one-page diagram of a generic AI factory showing all four planes, with the names of the NVIDIA components that live on each.
2. A table mapping each Mission Control component to the operational task it owns (provisioning, scheduling, observability, lifecycle).

Self-check

- In one sentence each, define BCM, Run:ai, DCGM, NVSM, and the GPU Operator. (You will refine these all term, but write the first version now.)
- Why is InfiniBand commonly used for the east-west fabric in training clusters? What problem does NVLink/NVSwitch solve that InfiniBand does not?
- Name three things Mission Control gives you that a hand-rolled BCM + Slurm install does not.

Week 2 — BCM Fundamentals: Base View, Nodes, and Networks

Exam tasks covered:

- Use BCM's Base View interface to monitor cluster performance, resource utilization, and node health in real time
- Configure and monitor network settings for cluster nodes, DPUs, and switches using BCM
- Use BCM to organize and configure compute nodes into categories based on hardware or workload requirements

Concepts to master

- **BCM architecture:** head node(s), provisioning network, internal/external networks, compute nodes, software images, categories, node groups. Understand the difference between **categories** (hardware/role definitions, e.g., `dgx-a100`, `cpu-only`) and **node groups** (administrative groupings).
- **Image management:** the difference between a software image, a category-assigned image, and the live system on a node. Know how `cmsh` (CLI) and Base View (web GUI) both expose the same model.
- **Base View navigation:** Overview dashboard, Nodes view, Monitoring view (metric thresholds, health checks), Workload view. Know where GPU metrics, network counters, and node health surface.
- **Network configuration in BCM:** how networks are objects, how interfaces on nodes are bound to networks, how DPU and switch objects are represented, the role of `ipmi` and out-of-band networks for power and console.
- **Health checks and metrics:** built-in BCM health checks vs. custom checks, the metric collection cadence, alerting via Base View.

Resources

- *NVIDIA Base Command Manager Administrator Manual* — the canonical reference. Read at least the chapters on “Configuring Networks,” “Configuring Categories,” “Monitoring,” and “Cluster Management with Base View.”
- *BCM Installation Manual* — skim for context; you'll execute pieces of it in Week 3.
- LaunchPad: *Base Command Manager Cluster Administration* lab.

Hands-on lab

In LaunchPad or a local BCM install:

1. Open Base View. Locate the cluster overview, drill into a single compute node, and find: GPU utilization, memory pressure, NIC throughput, and the most recent failed health check.
2. In `cmsh`: list categories (`category list`), show the image bound to your default category (`category use default; show`), and list nodes assigned to it.
3. Create a new category called `gpu-training`, clone the image of an existing category into it, and assign one node to the new category. Reboot the node and confirm it provisions with the new image.
4. Add a custom metric threshold (e.g., GPU temperature > 80°C raises a warning) and verify the alert appears in Base View.
5. Inspect the network objects (`network list`). Identify which network the management interface uses vs. the high-speed fabric. If a DPU or BlueField device is present, show its interface objects.

Self-check

- What is the difference between `cmsh` and `cmgui`/Base View? When would you reach for one over the other?
- A user reports a node “looks down” in Base View but `ssh` works. Where do you check next, and

what does that ambiguity usually mean?
- You need to roll out a kernel parameter change to 50 compute nodes. Walk through how you'd do it via image management instead of node-by-node SSH, and explain why that's the BCM way.

Week 3 — BCM Advanced: Lifecycle, Users, Reports, and Troubleshooting

Exam tasks covered:

- Apply patches, update firmware, and synchronize software images across cluster nodes using BCM
- Administer user accounts, roles, and permissions to ensure secure access to the cluster using BCM
- Diagnose and resolve cluster issues, such as job failures, node outages, or resource bottlenecks, using BCM
- Use BCM to maintain documentation and generate reports on cluster usage, performance, and issues
- Troubleshoot Base Command Manager

This week is where BCM stops feeling like a GUI and starts feeling like an operational platform.

Concepts to master

- **Image lifecycle:** how to create, clone, modify, and grab images. The difference between modifying a live node vs. modifying its image. The “grab” workflow (capturing a known-good live node back into an image) vs. the “provision” workflow (pushing an image to a node).
- **Patching and firmware:** running OS package updates inside an image (chroot or `cm-chroot-sw-img`), then rebooting nodes to pick up the new image. Coordinating firmware updates with BCM where applicable (DGX firmware, BMC firmware, NIC firmware) — when BCM orchestrates and when you drop to vendor tooling like `nvfwupd` or `mlxfwmanager`.
- **Users, roles, and profiles:** BCM's user objects, the distinction between cluster Linux users and BCM admin users, profiles that bound what an admin can do (e.g., a profile that can reboot nodes but not modify networks). Mapping LDAP/AD where applicable.
- **Reports and monitoring history:** generating cluster utilization reports, exporting metric history, the metric retention policy, where logs live (`/var/log/cmdaemon` and friends).
- **BCM troubleshooting fundamentals:** the role of `cmd` (CMDaemon) on the head node, `cmd` on compute nodes, the message bus between them. What it means when a node shows `INSTALLER_CALLINGINIT` vs. `UP` vs. `DOWN` vs. `CLOSED`. The `request` and `event` log streams.

Resources

- *BCM Administrator Manual* chapters on User Management, Cluster Monitoring, Software Image Management, and Troubleshooting.
- BCM release notes for the version you're studying — know what changed recently.
- The output of `cmsh -c “main showprofile”` for a real installation; reading a real profile

teaches you what the permission model can express.

Hands-on lab

1. Inside a software image, install a new package (e.g., `htop`) using `cm-chroot-sw-img`. Reboot a node assigned to that image and confirm the package is present.
2. Create a new BCM admin user with a custom profile that can reboot nodes and view monitoring data, but cannot modify networks or images. Log in as that user and verify the restrictions hold.
3. Generate a cluster utilization report covering the last 7 days. Export it as PDF or CSV.
4. Stop `cmd` on a single compute node. Watch how Base View represents the loss. Restart it. Inspect `/var/log/cmdaemon` on both head and compute to see the reconnect.
5. Force a node into `CLOSED` state and bring it back. Document the exact steps in your `commands.md`.

Self-check

- A node provisions but immediately drops back to `INSTALLING`. List five things you'd check, in order.
- The head node's `cmd` is healthy, but Base View shows half the cluster as down. What single networking fault explains this most often?
- A user complains their job died and blames the cluster. What BCM artifacts (logs, metrics, events) do you look at to determine whether the node, the scheduler, or the user's job is at fault?

Week 4 — Slurm: Install, Administer, and Run Training Workloads

Exam tasks covered:

- Install Slurm
- Administer Slurm cluster
- Manage job scheduling and resource allocation using BCM's workload manager (Slurm)
- Deploy training workloads with Slurm

Slurm is the historical heavyweight in HPC and remains the default choice for multi-node training. The exam expects fluency in submission, scheduling, and GRES configuration for GPUs.

Concepts to master

- **Slurm architecture:** `slurmctld` (controller, on the head node or a dedicated scheduler node), `slurmd` (one per compute node), `slurmdbd` (accounting database), `munge` for auth.
- **Installation via BCM:** the BCM-shipped Slurm role and how it integrates Slurm config into the image. Enroot (rootless container runtime) and Pyxis (Slurm SPANK plugin that runs Enroot images as job steps) — the standard way to run NGC containers under Slurm without Docker.
- **GRES and GPUs:** `gres.conf`, `Gres=gpu:8` in node definitions, requesting with

`--gres=gpu:N` or `--gpus=N`, and the difference between those flags. CUDA device visibility is set by Slurm via `CUDA_VISIBLE_DEVICES`.
- **Partitions, QOS, and accounting:** partitions slice the cluster (e.g., `train`, `inference`, `debug`), QOS layers priority/limits on top, accounts/users let you do showback and quota. Fairshare and the multifactor priority plugin.
- **Scheduler tuning:** the backfill scheduler — what it does, why it matters for multi-GPU job throughput, and how `SchedulerParameters` knobs affect packing.
- **Submission patterns:** `sbatch` script structure, `srun` for interactive and step launches, `salloc` for interactive allocations, `--ntasks-per-node`, `--cpus-per-task`, and how those interact with NCCL and `mpirun`/`srun` on multi-node jobs.

Resources

- Official Slurm documentation, especially the GRES, Multi-Cluster, and Accounting pages.
- BCM Administrator Manual section on Slurm integration.
- NVIDIA Pyxis and Enroot GitHub repos (read the README).
- NVIDIA NeMo Framework Launcher examples — they show the canonical multi-node training submit script.

Hands-on lab

1. Install Slurm into your BCM cluster. Confirm `sinfo`, `squeue`, and `scontrol show nodes` work end-to-end.
2. Configure GRES on at least one GPU node. Submit a job that requests `--gres=gpu:1` and prints `nvidia-smi` from inside the allocation. Confirm only the requested GPUs are visible.
3. Create two partitions, `interactive` (short, high priority) and `train` (long, lower priority). Configure a QOS that limits the `interactive` partition to jobs ≤ 1 hour.
4. Pull an NGC PyTorch container with `enroot import`. Submit a single-node training job using `srun --container-image` (Pyxis) that runs a tiny model for one epoch. Confirm GPU utilization in `nvidia-smi`.
5. Run a 2-node NCCL all-reduce test under Slurm. Confirm bandwidth numbers via `nccl-tests`. This is the muscle memory you want for the lab portion of the exam.
6. Practice troubleshooting: kill `slurmd` on one node, observe `sinfo` showing the node as down, inspect `slurmctld` logs, restart, verify recovery.

Simulator drills (DC Lab Sim)

Before moving to the next week, run these in DC Lab Sim until each feels automatic:

- `sinfo` — read partition state, identify drained vs. down vs. allocated nodes.
- `squeue` and `squeue -u <user>` — interpret job state codes (PD, R, CG, CD, F).
- `scontrol show node <nodename>`, `scontrol show job <jobid>`, `scontrol update`.
- `sbatch` and `srun` invocations with `--gres=gpu:N`, `--ntasks-per-node`, `--time`.
- Any simulator scenario involving a stuck or pending job — practice reading reasons in `squeue -l` and translating them to fixes.

Self-check

- A user submits `–gres=gpu:8` and the job stays `PENDING` forever. List four causes and the diagnostic command for each.
- Explain in one paragraph why Pyxis exists and what life is like on a Slurm cluster without it.
- Why is `scontrol reconfigure` not always sufficient after editing `slurm.conf`, and when do you need a full `systemctl restart slurmctld`?

Week 5 — Kubernetes: Install via BCM, Administer, Run Inference

Exam tasks covered:

- Install and initialize Kubernetes on NVIDIA hosts using BCM
- Administer Kubernetes
- Manage job scheduling and resource allocation using BCM's workload manager (Kubernetes)
- Deploy inference workloads with Kubernetes

For inference and most modern serving stacks, Kubernetes is the default. NVIDIA's GPU Operator is the connective tissue between vanilla K8s and a GPU node.

Concepts to master

- **K8s on BCM:** the BCM Kubernetes role, how BCM lays down `kubeadm`-based clusters, and how nodes are flipped between Slurm and K8s roles (or run both via co-existence).
- **Pre-init prerequisites: swap must be off** (a common exam-style gotcha), kernel modules loaded, container runtime configured (containerd is standard on modern NVIDIA stacks).
- **NVIDIA GPU Operator:** what it is — a meta-operator that deploys the driver, container toolkit, device plugin, DCGM exporter, MIG manager, and more, all from one Helm chart. The difference between the public NGC version and the NVIDIA AI Enterprise version (signed images, support contract, validated upgrade paths).
- **GPU scheduling in K8s:** the `nvidia.com/gpu` resource, requesting whole GPUs, requesting MIG slices via the MIG manager, the `nvidia.com/mig-1g.5gb`-style resources.
- **Inference patterns:** Triton Inference Server deployment via Helm, NIM microservices, autoscaling on GPU utilization, model repositories backed by shared storage.
- **K8s administration basics:** `kubectl` muscle memory (`get`, `describe`, `logs`, `exec`, `top`), namespaces and RBAC, quotas, taints and tolerations to dedicate GPU nodes.

Resources

- BCM Administrator Manual, Kubernetes chapter.
- NVIDIA GPU Operator documentation on `docs.nvidia.com`.
- NVIDIA Triton Inference Server docs and the NIM documentation.
- The Kubernetes upstream docs on Resource Management and Scheduling.

Hands-on lab

> **Your environment:** Run the exam-relevant work — BCM-deployed K8s, manual GPU Operator install, Triton/NIM, ResourceQuota — on your **BCM-managed AWS K8s cluster** (deployed via `cm-kubernetes-setup`). Your Sparks/MicroK8s are still useful for cheap iteration on basic concepts (kubectl reps, simple pod manifests), but the AWS cluster is what matches the exam's BCM-centric framing. The same AWS cluster will host Run:ai in Week 6 and MIG in Week 7, so investing in it here pays off.

1. Use BCM to bring up a K8s control plane on one node and a worker on another. Confirm `kubectl get nodes` shows both `Ready`.
2. Install the GPU Operator via Helm. Watch all the pods come up in `gpu-operator` namespace. Verify with `kubectl describe node` that GPU resources are advertised.
3. Deploy a sample CUDA pod (`nvidia/cuda:12.x-base`) requesting `nvidia.com/gpu: 1`. Confirm `nvidia-smi` works from inside.
4. Deploy Triton Inference Server with a sample model. Issue an inference request from a client pod. Inspect Triton's metrics endpoint.
5. Create a namespace `inference-team-a`, set a `ResourceQuota` capping it at 4 GPUs, and verify a pod requesting 5 is rejected.
6. Drain a GPU node, confirm the workloads reschedule, then `uncordon` it.

Self-check

- A pod requests `nvidia.com/gpu: 1` and stays `Pending`. Walk through the diagnostic order: scheduler events → node resources → device plugin health → driver health.
- Explain in your own words what each major GPU Operator component does and what fails if it's missing.
- When would you choose K8s over Slurm for a workload, and when the reverse?

Week 6 — Run:ai: Install, Administer, and Multi-Tenant Resource Allocation

Exam tasks covered:

- Install Run:ai
- Administer Run:ai
- Deploy training workloads with Run:ai
- Deploy inference workloads with Run:ai
- Allocate resources between teams with Run:ai, Slurm, and Kubernetes

Run:ai is the orchestration layer that NVIDIA acquired in 2024 and integrated tightly into the Mission Control stack. It sits on top of Kubernetes and provides the multi-tenant GPU sharing and fractional GPU features that vanilla K8s does not.

Concepts to master

- **Run:ai architecture:** the control plane (SaaS or self-hosted), the cluster agent that runs in your K8s cluster, the scheduler (Run:ai's scheduler replaces or augments the default K8s scheduler for projects under its control).
- **Projects, departments, and quotas:** the Run:ai resource hierarchy. Projects own quotas (guaranteed GPUs and over-quota allowance). Departments group projects. Fairshare and over-quota use govern how idle capacity is shared.
- **Workload types:** training jobs, interactive workspaces (Jupyter etc.), inference deployments. The Run:ai CLI (`runai submit`) and the web UI both produce K8s objects via Run:ai's CRDs.
- **Fractional GPUs and dynamic GPU memory allocation:** Run:ai's signature feature. Multiple workloads share a GPU by memory partition or time-slice, beyond what raw MIG provides.
- **Multi-tenant resource allocation across the three schedulers:** the operational reality. Slurm partitions + QOS for HPC-style fairness, K8s namespaces + ResourceQuotas for service-style isolation, Run:ai projects + quotas for AI-team-style fairness layered on K8s. Know which tool solves which problem.
- **Run:ai administration:** adding users, mapping to identity providers, creating projects, setting quotas, monitoring cluster utilization in the Run:ai UI.

Resources

- Run:ai documentation: `run-ai-docs.nvidia.com` — start with the Workloads, CLI Reference, and (separately) the *Install Using Base Command Manager* page.
- BCM 11 release notes — note the `cm-kubernetes-setup` wizard improvements that streamline Run:ai control plane installation.
- Run:ai Dynamic MIG quickstart — directly applicable to Week 7 if your access cluster supports it.
- LaunchPad: any *Run:ai* lab. **Required** for hands-on install practice and for any cluster-admin tasks your role on the external A100 cluster doesn't permit.
- KAI Scheduler GitHub repo (`github.com/NVIDIA/KAI-Scheduler`) — optional supplementary reading for the underlying scheduling concepts (gang, fairshare, preemption, reclaim). Not a substitute for Run:ai itself but useful for the *why* behind Run:ai's scheduling behavior.

Hands-on lab

> **Your environment for this week is split:**
> - **External A100 Run:ai cluster** (user-level access): workload submission, project navigation, fractional GPU, possibly Dynamic MIG profile requests.
> - **LaunchPad Run:ai lab**: install task end-to-end, plus any admin tasks (creating departments, projects, quotas, configuring node roles) that your role on the A100 cluster doesn't allow.
> - **Documentation study**: read the BCM-Run:ai install flow as a runbook before you touch LaunchPad, so the LaunchPad session is execution rather than first-time learning.

Track 1 — Workload submission and use (on the external A100 Run:ai cluster):

1. `runai login` and confirm `runai list projects` shows your assigned projects. Submit a test training job (`runai submit hello –image nvcr.io/nvidia/pytorch:24.x -g 1 – python -c “import torch; print(torch.cuda.is_available())”`). Read `runai`

`logs hello`.
2. Submit an interactive workspace (Jupyter) using `runai submit –interactive –jupyter`. Connect to it via the URL Run:ai prints. Confirm GPU access from inside the notebook.
3. **Fractional GPU exercise.** Submit two workloads each requesting `–gpu-memory 20G` against a single 80GB A100. Confirm both run concurrently. Inspect with `runai describe job <name>` to see how Run:ai represents the partial allocation.
4. **Dynamic MIG exercise (if available).** If the cluster admin has enabled Dynamic MIG on a node, submit a workload with `–mig-profile 2g.20gb` and confirm Run:ai configures the slice. If Dynamic MIG isn't enabled on this cluster, skip this step here and do it on LaunchPad in Track 2.
5. Submit an inference workload (`runai submit inference-test –image <triton-image> –port 8000:8000 –service-type loadbalancer`). Send a sample inference request. Observe the Run:ai inference workload type's lifecycle (deployment vs. job).
6. Navigate the Run:ai UI within your role's visibility: your projects, your job history, your quota usage. Note what you *can't* see — that's your gap to fill in Track 2.

Track 2 — Install and admin (on LaunchPad):

1. Before the LaunchPad session, write a runbook from the BCM and Run:ai docs covering: prerequisites (K8s version, GPU Operator version, Ingress, Prometheus, cert-manager), the `cm-kubernetes-setup` wizard's Run:ai installer step, the SaaS tenant configuration (URL, client ID, client secret), post-install verification (`kubectl -n runai get pods`, cluster registration in the SaaS UI). The act of writing this from documentation is most of the learning.
2. In the LaunchPad Run:ai lab, execute your runbook end-to-end. Where the lab environment differs from BCM, note the differences in your runbook. The exam tests the BCM-installer flow specifically.
3. In LaunchPad, create two departments (`research`, `production`) and two projects under each. Set guaranteed GPU quotas and over-quota allowances per project.
4. **Cross-team allocation exercise.** Submit a job under `research/team-a` that exceeds its guaranteed quota; have `production/team-b` claim its own quota; observe the over-quota job preempted. This is the “Allocate resources between teams with Run:ai” exam task in concrete form.
5. Configure a node for Dynamic MIG: `runai-adm set node-role –dynamic-mig-enabled <node-name>`. Confirm GPU Operator MIG strategy is `mixed`. Submit MIG-profile workloads.

Track 3 — Cross-scheduler comparison:

1. Take a single training job spec and submit equivalent versions to: (a) Slurm on your BCM-managed AWS Slurm cluster from Week 4, (b) plain Kubernetes on the BCM K8s cluster (as a non-Run:ai pod), © Run:ai on the external A100 cluster. Document the resource allocation flow in each. The exam directly tests “Allocate resources between teams with Run:ai, Slurm, and Kubernetes.”

> **Cost note:** The BCM AWS GPU worker isn't strictly needed during Week 6 unless you want to do step 12(b). Consider keeping it down to control cost; bring it up for Week 7's MIG mechanics work.

Self-check

- A team consistently runs over their guaranteed quota even when other teams need GPUs. What's misconfigured?
- Why can't vanilla Kubernetes do fractional GPU sharing without Run:ai (or a similar tool)?

- A research team wants Jupyter notebooks for exploration and long-running multi-node training. How do you structure their Run:ai project, and what features do you enable for each use case?
- On a single A100 node, when does Run:ai use fractional GPU vs Dynamic MIG, and what config setting forces one or the other? (Answer involves `runai-adm set node-role –dynamic-mig-enabled` and the GPU Operator MIG strategy.)
- Walk through the BCM-driven Run:ai install from memory: what does `cm-kubernetes-setup` need (FQDN, tenant, secret), what does it deploy, what should you verify post-install? If you can't answer this fluently, your runbook needs more time before LaunchPad.

Week 7 — GPU, DPU, and Container Specifics: MIG, DOCA, and NGC

Exam tasks covered:

- Configure MIG
- Deploy DOCA Services on DPU Arm
- Deploy containers from NGC

This week covers three skills that show up across multiple exam domains.

Concepts to master

- **MIG (Multi-Instance GPU):** which GPUs support it (A100, H100, H200, B200 — the data-center heavy hitters; consumer and L40-class do not). Profiles like `1g.5gb`, `2g.10gb`, `3g.20gb`, `7g.40gb`. Enabling MIG (`nvidia-smi -mig 1`), creating GPU instances and compute instances, and the exposure of MIG slices to Kubernetes via the GPU Operator MIG Manager (single-strategy vs. mixed-strategy).
- **DOCA on BlueField DPU Arm:** BlueField-3 has Arm cores running an OS independent of the host. DOCA is NVIDIA's SDK and services framework for that environment. Deploying DOCA services means installing them on the DPU's Arm OS — typically via container or native package — to provide network offload, storage offload, security, or telemetry functions. Know the DOCA Service framework, container deployment on the DPU, and the firmware/driver pairing between host and DPU.
- **NGC catalog and CLI:** containers, models, Helm charts, resources. The `ngc` CLI for authentication (API key), pulling images, and pushing to private registries. NGC private registries vs. the public catalog. Image signing and the NVIDIA AI Enterprise variants of common containers (validated, supported).
- **Running NGC containers:** under Docker (NVIDIA Container Toolkit), under Slurm via Enroot/Pyxis, under K8s via standard pod specs with `nvidia.com/gpu` resources, under Run:ai via job specs.

Resources

- *NVIDIA Multi-Instance GPU User Guide* (current driver release).
- NVIDIA DOCA documentation portal — start with “DOCA Services” and “BlueField Software.”
- NGC documentation: the catalog overview, the CLI reference, and the private registry guide.

- NVIDIA Container Toolkit documentation.

Hands-on lab

> **Your environment for MIG:** Same BCM-managed AWS K8s cluster you set up in Week 5, with at least one worker sized as `p4d.24xlarge` (A100) or `p5.48xlarge` (H100). This week's MIG lab focuses on the lower-level mechanics — `nvidia-smi mig` directly, GPU Operator MIG Manager, non-Run:ai MIG-profile pods. The Run:ai-orchestrated MIG flow (if applicable) was covered in Week 6 on the external A100 cluster or LaunchPad. Step 3 (DOCA) requires a BlueField environment via LaunchPad. Steps 4–6 (NGC) work anywhere; the BCM AWS cluster is convenient since you're already there.

1. On the A100 or H100 worker, drop down to `nvidia-smi` directly. Enable MIG (`nvidia-smi -mig 1`), list profiles (`nvidia-smi mig -lgip`), create three `2g.20gb` GPU instances, then create compute instances inside them. Verify with `nvidia-smi`. Disable MIG and reset.
2. With the GPU Operator MIG Manager installed, switch the node from full-GPU mode to MIG mixed-strategy by labeling the node (`nvidia.com/mig.config=all-2g.20gb` or similar). Watch the MIG Manager pod reconfigure the GPU. Submit a non-Run:ai pod requesting `nvidia.com/mig-2g.20gb` and confirm it lands.
3. If LaunchPad has a BlueField environment available, log into the DPU's Arm OS, list installed DOCA services, and deploy a sample DOCA service container. If no DPU is available, work through the DOCA Services tutorial on `docs.nvidia.com` end-to-end as a paper exercise.
4. Configure the `ngc` CLI with your API key. Pull a PyTorch container, run it under Docker on a BCM-managed AWS node with `–gpus all`, and confirm GPU access.
5. Pull an NGC Helm chart for Triton, customize values, and deploy to your BCM-managed K8s cluster.
6. Practice the same NGC container running under all three runtimes: Docker (on a BCM AWS node), Enroot/Slurm (on your Week 4 Slurm cluster), Kubernetes (on the BCM K8s cluster). Note differences in invocation.

Simulator drills (DC Lab Sim)

- `nvidia-smi -mig 1` / `-mig 0` enable/disable cycle, `nvidia-smi mig -lgip` (list GPU instance profiles), `nvidia-smi mig -cgi` and `-cci` to create instances.
- `docker run –gpus all` and `–gpus device=0,1` patterns.
- `enroot import`, `enroot create`, `enroot start` end-to-end, including importing a docker image.
- If the simulator includes any `ngc` CLI scenarios, drill the auth and pull flow.

Self-check

- What's the difference between a GPU instance and a compute instance in MIG, and why does the distinction matter?
- A DOCA service on a DPU appears installed but cannot reach the host network. Where do you start troubleshooting — the DPU OS, the host driver, or the firmware pairing?
- A teammate downloads an image from NGC and reports it “doesn't work in our cluster.” What questions do you ask first?

Week 8 — Troubleshooting Deep Dive

Exam tasks covered:

- Use system management tools to troubleshoot issues
- Troubleshoot Docker
- Troubleshoot the fabric manager service for NVLink and NVSwitch systems
- Troubleshoot Magnum IO components
- Troubleshoot storage performance
- Troubleshoot the deployment of a container from NGC

The exam's troubleshooting block expects fluent, decision-tree-style diagnosis. This week is about converting all the previous weeks' tooling into reflexive diagnostic patterns.

Concepts to master

- **System management tooling:**
  - `nvidia-smi` — the basics, plus the underused flags: `-q`, `dmon`, `pmon`, `nvlink -s`, `topo -m`, `–query-gpu` for scripting.
  - **DCGM** (Data Center GPU Manager) — `dcgmi diag`, health checks, the DCGM exporter for Prometheus. Distinguish it from `nvidia-smi` (DCGM is the supported tool for cluster-scale telemetry and validation).
  - **NVSM** (NVIDIA System Management) — DGX-specific health and inventory.
  - `nvidia-bug-report.sh` — when escalating, NVIDIA support will ask for this output.
- **Xid errors:** the canonical GPU error code system. Memorize the most common codes (Xid 13, 31, 43, 48, 63, 64, 74, 79, 92, 119) and what hardware vs. software class each implies. Reading dmesg for Xid is a core operational skill.
- **Fabric Manager:** required for systems with NVSwitch (DGX A100, H100, H200, GB200 NVL). It must be running for GPUs to use NVLink fully across the node. Common failures: service not started, version mismatch with driver, node restart didn't bring it back, NVLink errors visible in `nvidia-smi nvlink -s`.
- **Magnum IO components:** the umbrella term for NVIDIA's IO stack — NCCL (collective comms), GPUDirect RDMA, GPUDirect Storage, UCX, NVSHMEM. Troubleshooting usually means diagnosing one of: `NCCL_DEBUG=INFO` to surface topology and transport choice, `ibstat` and `ibdiagnet` for InfiniBand health, GPUDirect Storage probe utilities for storage path validation.
- **Storage performance:** distinguishing “slow storage” from “slow IO pattern.” Tools: `fio` for raw bandwidth, `gdscheck` and `gds_stats` for GPUDirect Storage path, file system-specific tools (Lustre `lfs check`, Weka diagnostics, etc.).
- **Docker troubleshooting:** `docker logs`, `docker inspect`, `docker events`, container runtime logs (`journalctl -u containerd`), the NVIDIA Container Toolkit logs at `/var/log/nvidia-container-toolkit.log`, and common failures: missing toolkit, missing driver mounts, capability mismatches.
- **NGC container deployment failures:** auth (expired API key, wrong org), network (proxy/firewall blocking pulls), driver version too old for the container's CUDA version (the most common single cause), missing toolkit, OOM at startup.

Resources

- *NVIDIA Xid Error Reference Guide* (in the data center driver docs).
- DCGM documentation, especially `dcgmi diag` and the diagnostics levels.
- NVIDIA NCCL Troubleshooting Guide.
- NVIDIA Container Toolkit troubleshooting page.
- NVIDIA Magnum IO and GPUDirect Storage docs.

Hands-on lab — build your decision trees

> **Your environment:** You have two venues. The **BCM-managed AWS cluster** (your Run:ai/MIG/inference home) is best for cluster-scale troubleshooting — multi-node NCCL with EFA, BCM-detected node faults, GPU Operator pod failures, real Xid errors on A100/H100. The **two-Spark RoCE link** is best for the IB-style diagnostic command set — `ibstat`, `iblinkinfo`, `ib_write_bw`/`ib_read_bw`, `ibdev2netdev`. Pick the venue per scenario:
>
> - Scenario 1 (NCCL hang): try on both. Sparks give you `ibstat`/`iblinkinfo` output to read; AWS gives you a multi-node EFA-flavored hang to diagnose with NCCL debug environment variables.
> - Scenarios 2 and 4 (`nvidia-smi ERR!`, container `No devices found`): either venue.
> - Scenario 3 (NGC pull failures): AWS — you can manipulate VPC routes, security groups, and registry endpoints.
> - Scenario 5 (storage benchmarks): AWS — variety of backends (EFS, FSx for Lustre, EBS) gives you real IO-path problems to solve.
> - Scenario 6 (Fabric Manager): LaunchPad DGX only. Neither of your environments has NVSwitch.

Bonus drills your RoCE-connected Sparks support (do these — they're high-yield):

- Run a 2-node NCCL all-reduce between Sparks with `NCCL_DEBUG=INFO` `NCCL_DEBUG_SUBSYS=NET,GRAPH`. Read the topology output, identify which transport NCCL chose, and force the alternative with `NCCL_IB_DISABLE=1` to see the bandwidth difference between RDMA and TCP.
- Simulate a partial fabric failure: down one of the two interfaces with `ip link set <iface> down`. Watch NCCL behavior, recover, document the symptom-to-cause path.
- Use `ib_write_bw` and `ib_read_bw` from the perftest package between the two Sparks. These are the same tools you'd use on a true IB fabric for storage and RDMA bandwidth troubleshooting.
- Verify GPUDirect RDMA is in use: `nvidia-smi` plus the NCCL debug output should confirm GPU memory is reaching the NIC without a host bounce.

What these drills do not cover (study from documentation): Subnet Manager workflows, LID assignment and routing, `ibping`/`ibhosts`/`ibnetdiscover` topology discovery, partition keys, IB-specific QoS. The DGX Spark cannot be configured for native IB, so these topics are paper-only on this hardware.

Pick each scenario below and write a one-page runbook. Then practice executing it.

1. **“Distributed training hangs on AllReduce.”** Where do you look first? `NCCL_DEBUG=INFO`, then `ibstat`/`ibdiagnet`, then Fabric Manager status, then NVLink status, then check whether one node is stuck.
2. **“`nvidia-smi` returns ERR! on one GPU.”** Walk through Xid in dmesg, GPU reset, driver

reinstall, hardware replacement.
3. **“NGC container fails to pull on three nodes but succeeds on five.”** Auth, network, NGC org/team, image tag pinning.
4. **“Container starts but `nvidia-smi` inside reports `No devices found.`“** Toolkit configuration, runtime selection (containerd default runtime), –gpus/device plugin annotation, driver visibility.
5. **“Storage benchmark shows 1 GB/s when the spec says 50 GB/s.”** Single client vs. parallel, GPUDirect Storage enabled, mount options, file system config, network path.
6. **“Fabric Manager fails to start after a driver upgrade.”** Version pinning, package state, log inspection, service order on boot.

Simulator drills (DC Lab Sim) — heaviest overlap of the plan

This is the week where DC Lab Sim earns its keep. Drill until each command's flags are reflex:

- **`nvidia-smi` deep dive:** `nvidia-smi`, `nvidia-smi -q`, `nvidia-smi -q -d ECC`, `nvidia-smi dmon`, `nvidia-smi pmon`, `nvidia-smi nvlink -s`, `nvidia-smi nvlink -e`, `nvidia-smi topo -m`, `nvidia-smi –query-gpu=…` for scripted output.
- **`dcgmi`:** `dcgmi discovery -l`, `dcgmi diag -r 1` (quick), `dcgmi diag -r 2` (medium), `dcgmi diag -r 3` (long), `dcgmi health`, `dcgmi stats`.
- **InfiniBand:** `ibstat`, `iblinkinfo`, `ibhosts`, `ibnetdiscover`. Practice reading link state, rate, and counters.
- **`ipmitool`:** `ipmitool sel list` (reading the system event log is a foundational troubleshooting skill), `ipmitool sensor`, `ipmitool chassis status`. Even though IPMI leans more AII, the diagnostic mindset transfers.
- **Run every troubleshooting scenario** the simulator offers. Even AII-flavored bring-up scenarios sharpen the same diagnostic instincts you'll need on AIO troubleshooting questions.
- After each scenario, write a one-paragraph postmortem in your notes: symptom → first command → what it told you → next step → resolution. This converts simulator reps into real recall.

Caveat for AIO: the simulator's troubleshooting scenarios are framed around AII's domains (cabling, transceivers, burn-in failures). For AIO-specific troubleshooting like Run:ai job preemption, K8s scheduler events, or BCM `cmd` daemon failures, you'll need real LaunchPad time or a self-built lab — the simulator won't cover those.

Self-check

- For each of these tools, in one sentence: when do you reach for `nvidia-smi`, `dcgmi`, `nvsm`, `nvidia-bug-report.sh`?
- A user reports “GPU is broken.” What ten data points do you collect before you agree?
- Magnum IO is a marketing umbrella, but for troubleshooting it decomposes into specific components. Name them and the diagnostic command you'd run for each.

Final Week — Mock Exam, Lab Drills, and Test-Day Logistics

By this point you've covered every task on the blueprint. The last stretch is about reflexes, gaps, and exam mechanics.

Mock exam regimen

- Take at least one full-length practice exam under timed conditions (third-party banks like the ones on Udemy or Preporato are reasonable; treat the questions as patterns, not gospel).
- Score yourself by domain. Anything below 70% in a domain → revisit that week's notes and lab.
- Re-do the live labs cold from your `commands.md`. The exam includes hands-on labs in a real cluster environment; muscle memory on `cmsh`, `sbatch`, `kubectl`, and `runai submit` is what saves you when nerves hit.

Lab drill targets (do all of these in under 15 minutes each)

- Bring a node from `DOWN` back to `UP` in BCM (assume image is fine, network is fine, `cmd` needs a kick).
- Submit a 2-node, 16-GPU NCCL test under Slurm and read the bandwidth.
- Deploy a Triton inference pod on K8s, expose it via service, and curl an inference.
- Create a Run:ai project, set quota, submit a training workload, observe it consuming guaranteed GPUs.
- Enable MIG on a GPU, create three slices, assign them to pods.
- Pull and run an NGC container under Docker, Slurm, and K8s.

Test-day logistics

- Online proctored via Certiverse. Allow 30 minutes for environment setup before the start.
- Close everything except the secure browser. Disable all notifications.
- 120 minutes for 30 multiple-choice + 3 hands-on labs. The lab provisions automatically when you start. Budget roughly 60 minutes for MCQs, 60 for labs — but the timer is unified, so use whichever order fits you. Many candidates do the labs first while the cluster is fresh.
- Read every multiple-choice question twice. The plausible-but-wrong answer is usually adjacent to the right answer in tooling (e.g., `nvidia-smi` vs. `dcgmi`, GPU Operator vs. Container Toolkit).
- For lab questions: read the entire prompt before touching anything. Identify what “done” looks like. The labs are scored on outcome state, not on the path you took.

Two-week buffer rule

If your mock-exam scores are consistently below 75%, push the exam date out two weeks rather than schedule and hope. The recertification cost is the exam fee again.

Quick-Reference: Task → Week Map

If you ever lose track of where a topic lives in this plan:

Exam task	Week
Describe Mission Control toolkit	1
Describe data center architecture for AI workloads	1
BCM Base View monitoring	2
BCM network config (nodes, DPUs, switches)	2
BCM compute node categories	2
BCM patches, firmware, image sync	3
BCM users, roles, permissions	3
BCM cluster issue diagnosis	3
BCM documentation and reports	3
Troubleshoot BCM	3
Install Slurm	4
Administer Slurm cluster	4
Manage scheduling/allocation via BCM (Slurm)	4
Deploy training workloads with Slurm	4
Install/initialize Kubernetes via BCM	5
Administer Kubernetes	5
Manage scheduling/allocation via BCM (K8s)	5
Deploy inference workloads with Kubernetes	5
Install Run:ai	6
Administer Run:ai	6
Deploy training workloads with Run:ai	6
Deploy inference workloads with Run:ai	6
Resource allocation across Run:ai/Slurm/K8s	6
Configure MIG	7
Deploy DOCA Services on DPU Arm	7
Deploy containers from NGC	7
Use system management tools to troubleshoot	8
Troubleshoot Docker	8
Troubleshoot Fabric Manager (NVLink/NVSwitch)	8
Troubleshoot Magnum IO	8
Troubleshoot storage performance	8
Troubleshoot NGC container deployment	8

Core Resources at a Glance

- **Official:** *NVIDIA NCP-AIO Exam Study Guide* (PDF, doc number 3770900) — read first, read again before the exam.
- **NVIDIA Academy:** *AI Infrastructure & Operations Fundamentals* (self-paced, free) and the *AI Operations Professional Workshop* (multi-day, instructor-led).

- **NVIDIA LaunchPad:** free time-boxed labs for BCM, Run:ai, DGX, BlueField. Queue early; some labs have wait lists.
- **DC Lab Sim** (dclabsim.com / Seanbo5386/dc-lab-sim): browser-based command simulator built for NCP-AII. Strong supplement for the foundational tooling you'll see on AIO too — Slurm CLI (Week 4), MIG/Docker/Enroot (Week 7), and the troubleshooting commands in Week 8. Does not cover AIO-specific topics like Run:ai, Kubernetes admin, BCM Base View admin, or Mission Control.
- **Documentation portals:**
  - `docs.nvidia.com/base-command-manager`
  - `docs.nvidia.com/ngc`
  - `docs.nvidia.com/datacenter/dcgm`
  - `docs.nvidia.com/doca`
  - `docs.run.ai`
  - Slurm: `slurm.schedmd.com/documentation.html`
  - Kubernetes upstream + NVIDIA GPU Operator docs

Good luck. The exam is hands-on by design — if you've actually done every lab in this plan, the test will feel familiar rather than scary.

AI Cloud Managed Services Policies and Procedures