GPU SMITH
COMPUTE ENGINEERING

GPU SMITH / RESEARCH & PRACTICAL GUIDANCE

CUDA Driver Toolkit Container Compatibility Gate

> Plan and build private AI compute with GPU Smith. We connect workload requirements with GPU hardware, networking, deployment and operating decisions so your infrastructure fits the work it must perform.

Published by GPU Smith · 2026-09-20 · cuda compatibility · cuda driver · cuda toolkit · nvidia container toolkit · cuda containers · minor version compatibility · forward compatibility · gpu infrastructure · production upgrades

Original article: https://gpusmith.com/articles/en/cuda-compatibility-production-upgrade-gate

In this article

1. Executive Summary
2. Introduction and Background
3. Key Changes
4. Implementation Considerations and Process Changes
5. Data Analysis and Evidence
6. Case Studies and Real-World Examples
7. Implications and Future Directions
8. Frequently Asked Questions (FAQs)
9. Conclusion

Executive Summary

The production question is not whether a container has “CUDA” in its tag. It is whether **five independently versioned layers** form a supported and tested path: GPU and architecture, host kernel driver, container runtime injection, image user-space CUDA libraries, and framework or workload binaries. The CUDA Toolkit and the NVIDIA driver are separate products (docs.nvidia.com), while a TensorFlow GPU container can require only the host driver, not a host Toolkit installation (tensorflow.org). Consequently, `nvidia-smi` is evidence about the active driver and its maximum supported CUDA user-mode interface, not proof of the Toolkit, runtime libraries, or framework build inside an image. NVIDIA defines the CUDA value shown by `nvidia-smi` as the latest version supported by the driver (docs.nvidia.com).

As of **September 20, 2026**, the broad minor-version ranges are **CUDA 11.x on driver 450 through below 525**, **CUDA 12.x on 525 through below 580**, and **CUDA 13.x on 580 or later** (docs.nvidia.com) (docs.nvidia.com). Those are eligibility ranges, not correctness guarantees. Framework constraints can be narrower: JAX’s CUDA 13 route requires at least a Linux 580 driver (docs.jax.dev), and ONNX Runtime requires matching CUDA and cuDNN major versions when sharing PyTorch libraries (onnxruntime.ai). PyTorch 2.14 itself publishes three distinct CUDA wheel variants, 12.6, 13.0, and 13.2 (pytorch.org).

The recommended release gate is therefore **inventory, classify, canary, decide, and retain rollback**. Pin the image by digest because a digest identifies invariant content (docs.docker.com), save host and in-container inventories, test device visibility, framework initialization, one kernel path, PTX/JIT if used, and a representative inference. Approve only when every applicable matrix and canary passes. Defer an eligible but untested exception. Fail an unsupported range, platform, or library combination. Kubernetes creates a new Deployment revision when a rollout is triggered (kubernetes.io), so the prior digest and Pod template provide a concrete rollback target. This is a production upgrade gate, not a promise that nominal compatibility guarantees application correctness.

Introduction and Background

CUDA compatibility is frequently compressed into one comparison: container CUDA version versus the “CUDA Version” line in `nvidia-smi`. That shortcut discards the layers that determine whether an application actually starts, loads the intended libraries, executes compiled kernels, performs PTX just-in-time compilation, and produces valid results. **Backward compatibility**, **minor-version compatibility**, and **forward compatibility** solve different version relationships. A container adds another boundary, because it normally carries application libraries while depending on the host for the kernel driver and selected user-space driver components.

The practical decision for platform teams is whether a new framework wheel or CUDA image may use an already-qualified fleet driver. A driver update often has a larger operational blast radius than an application image change, so decoupling them has value. The mechanism and its limitations must be identified rather than inferred. A framework’s own checks can reject an otherwise plausible environment, as JAX documents when installed CUDA libraries are not sufficiently new (docs.jax.dev).

This report treats the release as a chain of evidence. **GPU Smith is an adjacent engineering advisor**, not a CUDA vendor or cloud provider. Its published scope includes capacity planning, telemetry, failure-mode analysis, and operating procedures (gpusmith.com); that posture makes a dated acceptance record more appropriate than a universal version promise. The same site says private AI infrastructure is specified, integrated, and validated against written acceptance criteria (gpusmith.com). Those first-party statements explain the gate-oriented perspective, but they do not replace NVIDIA, framework, or orchestrator documentation.

The resulting method answers all common versions of the query: the minimum driver for a Toolkit family, whether a CUDA container can run on a host driver, when the `cuda-compat` package applies, why a framework matrix can override a base CUDA pass, and how to upgrade CUDA in production with measurable rollback criteria.

Key Changes

Five version layers replace the one-number model

The first change in operator practice is to inventory **five layers**, not one. The host and image contribute different parts of the path. TensorFlow's container instructions make the boundary concrete by requiring the host GPU driver without requiring the host Toolkit (tensorflow.org).

Table 1 defines the minimum operator inventory and the question each field answers.

Layer	Evidence to capture	Operational question
1. GPU and architecture	GPU UUID, product, compute capability, partition profile	Can this binary or wheel target the installed architecture? PyTorch exposes the device's major and minor capability (docs.pytorch.org).
2. Host kernel driver	Driver version, loaded kernel module version, operating system and kernel	Is the active driver inside the documented range, and does the canary record the module actually loaded?
3. Container injection	NVIDIA Container Toolkit version, engine configuration, visible devices and capabilities	Did the runtime expose the intended device nodes and host driver libraries? Microsoft's container guidance describes the need for a runtime hook to expose `/dev/nvidia*` and matching libraries (learn.microsoft.com).
4. Image CUDA stack	Immutable image digest, runtime libraries, compatibility package, image labels	Which exact user-space bits are being evaluated? Azure Container Registry describes a manifest digest as a unique SHA-256 hash (learn.microsoft.com).
5. Framework and workload	Wheel or package version, compiled CUDA version, cuDNN version, target architectures, PTX/JIT use	Does the application's own support matrix and code path pass? TensorFlow build information is fixed when the package is compiled (tensorflow.org).

The table shows why two nodes with identical `nvidia-smi` headers can behave differently: their image digest, injected capabilities, framework build, or workload compilation path may differ. Conversely, a host does not need a matching Toolkit installation merely because the container includes a particular runtime. The official

TensorFlow container guidance explicitly separates its host-driver prerequisite from a host Toolkit (tensorflow.org).

Backward, minor-version, and forward compatibility are separate branches

Backward compatibility is the least surprising branch: an application built using an older Toolkit runs on a newer driver. This is still subject to the application, library, and architecture requirements. ONNX Runtime, for example, separately requires matching CUDA and cuDNN major versions for shared PyTorch libraries (onnxruntime.ai).

Minor-version compatibility is a newer runtime or application within the same CUDA major family on an older driver that meets the family minimum. It has three important caveats:

- **Limited feature set:** a feature introduced in a newer Toolkit can also require newer driver support and return `cudaErrorCallRequiresNewerDriver` (docs.nvidia.com).
- **PTX limitation:** applications that rely on PTX compilation do not work on the older driver through minor compatibility alone (docs.nvidia.com).
- **Architecture targeting:** NVIDIA requires a target architecture argument to `nvcc` for this path (docs.nvidia.com).

Forward compatibility uses a `cuda-compat` package containing selected user-mode driver libraries so a newer CUDA application can operate with an older base driver. Installing the package alone does not configure the loader to locate those libraries (docs.nvidia.com). It is also not a general system-driver upgrade: NVIDIA says the forward-compatible path is for CUDA only (docs.nvidia.com). Treat it as an explicit exception with a package version, supported GPU class, loader configuration, test result, and exit date.

Containers package user space, not driver independence

The NVIDIA Container Toolkit requires the host NVIDIA GPU driver (docs.nvidia.com). It configures an engine such as Docker or containerd so the NVIDIA runtime can expose devices and driver libraries. Docker provides GPU resources through the `--gpus` flag (docs.docker.com), while the runtime hook exposes the matching device nodes and libraries (learn.microsoft.com).

The image flavor matters. NVIDIA's official CUDA image documentation says the Container Toolkit is required to run CUDA images with Docker (hub.docker.com); its `devel` flavor adds headers and development tools on top of the runtime flavor (hub.docker.com). A production inference image should not be assumed to contain build tools merely because a development image does.

Implementation Considerations and Process Changes

Build an attachable inventory before making a decision

Collect inventory on the exact canary node before starting the candidate container. The following commands produce small evidence files without changing the driver:

```
nvidia-smi --query-gpu=uuid,name,driver_version --format=csv,noheader > host-gpus.csv
nvidia-smi -q -x > host-nvidia-smi.xml
cat /proc/driver/nvidia/version > host-kernel-driver.txt
nvidia-container-cli -V > container-toolkit.txt
docker version --format '{{json .}}' > docker-version.json
docker image inspect --format '{{json .RepoDigests}}' registry.example/image@sha256:DIGEST > image-
digests.json
docker image inspect --format '{{json .Config.Labels}}' registry.example/image@sha256:DIGEST > image-
labels.json
```

Formatted image inspection is an official way to view image labels (docs.docker.com). Pulling by digest fixes the exact image version (learn.microsoft.com), which makes the canary and rollback artifacts reproducible.

Then collect evidence from inside the candidate image. A minimal framework-neutral check is:

```
docker run --rm --gpus all registry.example/image@sha256:DIGEST nvidia-smi --query-
gpu=uuid,name,driver_version --format=csv
docker run --rm --gpus all registry.example/image@sha256:DIGEST sh -lc 'ldconfig -p | grep -E
"libcuda|libcudart|libnvidia-ptxjitcompiler"'
docker run --rm --gpus all registry.example/image@sha256:DIGEST sh -lc 'test -r /usr/local/cuda/version.json
&& sed -n "1,80p" /usr/local/cuda/version.json'
```

Do not interpret the host's displayed CUDA version as the runtime version. Record host information separately from the framework build and runtime discovery. PyTorch documents `torch.version.cuda` as a build inspection command (docs.pytorch.org), while TensorFlow says its build-information values are compile-time static (tensorflow.org).

For a compact JSON attachment, a Python collector can preserve command output and failures without silently substituting missing values:

```
import json
import os
import subprocess

image = os.environ["CUDA_GATE_IMAGE"]
commands = {
    "host_gpu_csv": ["nvidia-smi", "--query-gpu=uuid,name,driver_version", "--format=csv,noheader"],
    "kernel_driver": ["cat", "/proc/driver/nvidia/version"],
    "toolkit_cli": ["nvidia-container-cli", "-V"],
    "image_digests": ["docker", "image", "inspect", "--format", "{{json .RepoDigests}}", image],
    "container_gpu_csv": ["docker", "run", "--rm", "--gpus", "all", image,
                          "nvidia-smi", "--query-gpu=uuid,name,driver_version", "--format=csv,noheader"],
}
result = {}
for name, command in commands.items():
    run = subprocess.run(command, text=True, capture_output=True)
    result[name] = {"returncode": run.returncode, "stdout": run.stdout, "stderr": run.stderr}
print(json.dumps(result, indent=2, sort_keys=True))
```

Run it as `CUDA_GATE_IMAGE=registry.example/image@sha256:DIGEST python3 collect.py > cuda-gate.json`. A nonzero return code remains evidence rather than being erased. In Kubernetes, also save the Node object, Pod specification, Pod status, and Deployment history. Pod status contains per-container status records (kubernetes.io), which ties a result to what the orchestrator actually launched.

Apply a deterministic pass, defer, or fail tree

Use the following tree for every GPU architecture and node pool, not merely one representative machine:

1. **Fail inventory completeness** if the loaded driver, GPU identity, Toolkit or runtime used at build time, immutable image digest, framework build, and rollback digest are not known.
2. **Classify the relationship.** If the application Toolkit is older than the host driver's capability, evaluate backward compatibility. If it is within the same CUDA major family and above the paired driver, evaluate minor-version compatibility. If it crosses a major-family minimum on an older driver, evaluate the exact `cuda-compat` row and supported platform.
3. **Fail unsupported combinations.** Fail when the driver is below the documented family minimum without an applicable forward-compatibility package, the GPU class is excluded, or any framework, cuDNN, inference server, or operating-system matrix rejects the combination.
4. **Defer conditional combinations.** Defer when PTX/JIT is required on a minor-compatibility path, a new driver-coupled feature is needed, loader configuration for `cuda-compat` is unproven, the image is not digest-pinned, or only an import test has run.
5. **Run the canary sequence.** Require device visibility, framework initialization, a real allocation and kernel, PTX/JIT when applicable, representative inference with output checks, and telemetry review.
6. **Pass only the tested tuple.** Approval applies to the recorded GPU architecture, host driver, kernel module, Container Toolkit, digest, image runtime, framework build, compatibility mechanism, and workload test. It does not automatically approve another image tag or node pool.

This conservative tree distinguishes **eligibility** from **approval**. An eligible minor-version path is a canary candidate. It is not yet a production pass.

Test progressively, including the path an import misses

The canary should progress from cheap diagnostics to workload evidence:

- **Device exposure:** verify the intended UUIDs are visible inside the container. PyTorch's `torch.cuda.device_count()` returns the available GPU count (docs.pytorch.org).
- **Framework initialization:** capture `torch.cuda.is_available()`, TensorFlow physical devices, `jax.devices()`, or ONNX Runtime providers. JAX documents `jax.devices()` as the list of devices for a backend (docs.jax.dev), and ONNX Runtime exposes installed execution providers (onnxruntime.ai).
- **Allocation and kernel:** allocate device memory and execute a small operation. TensorFlow's documented matrix-multiplication smoke test confirms execution by logging `GPU:0` (tensorflow.org).

- **Architecture coverage:** compare the device capability with architectures compiled into the framework. PyTorch exposes its compiled architecture list (docs.pytorch.org).
- **PTX/JIT path:** force the actual dynamic-kernel or extension path when the workload uses it. Persist exact error names and descriptions alongside the image digest and test input.
- **Representative inference:** use the real model shape, precision, batching behavior, plugins, and output checks. PyTorch recommends benchmark settings representative of real use cases (docs.pytorch.org), and ONNX Runtime exposes the actual installed execution providers for capture (onnxruntime.ai).
- **Orchestrator state:** verify readiness and a terminal canary Job result. Kubernetes states that a Pod is ready only when all containers are ready (kubernetes.io), and Jobs expose Complete or Failed terminal states (kubernetes.io).

Define rollback before the canary

Rollback should be an explicit predicate, not an improvised response. Trigger it for any initialization or unsupported-PTX error, missing expected GPU, framework provider fallback, output mismatch, new error class, or a local telemetry regression outside the team's predeclared baseline. The baseline itself is deployment-specific and should not be invented from a vendor compatibility table.

Record the prior image digest and Deployment revision. Kubernetes rolls back the Pod template portion of a Deployment (kubernetes.io), so separately version any ConfigMap, Secret, model repository, or host change not contained in that template. A driver rollback is a distinct maintenance operation and should not be conflated with image rollback.

Data Analysis and Evidence

The quantitative core is a version-range crosswalk, not a benchmark. Compatibility documentation establishes supported states, while the canary establishes application behavior. No throughput improvement should be inferred from a compatibility pass.

Table 2 summarizes the broad driver-family logic documented on **September 20, 2026**. Exact Toolkit release notes and downstream support matrices still control each candidate.

Application or runtime family	Broad host-driver condition	Primary mechanism	Operator interpretation
CUDA 11.x	Driver **450 or later and below 525** for minor compatibility; later drivers use backward compatibility (docs.nvidia.com)	Minor within range, backward above it	Check the exact 11.x release and operating-system row before canary.
CUDA 12.x	Driver **525 or later and below 580** for minor compatibility; later drivers use backward compatibility (docs.nvidia.com)	Minor within range, backward above it	Confirm PTX and driver-coupled features. JAX's CUDA 12 wheel, for example, is built with CUDA 12.3 and supports local CUDA 12.1 or later (docs.jax.dev).

Application or runtime family	Broad host-driver condition	Primary mechanism	Operator interpretation
CUDA 13.x	Driver **580 or later** for minor compatibility (docs.nvidia.com)	Minor at or above 580, or forward-compat exception on a supported older base	CUDA 13.4 features and newly enabled platforms may require branch 615 or later (docs.nvidia.com).
Framework/library overlay	Framework-specific driver, CUDA, cuDNN, OS, and architecture rows all pass	Downstream matrix overrides a broad CUDA pass	TensorFlow 2.21.0's tested Linux build row pairs cuDNN 9.3 with CUDA 12.5 (tensorflow.org); JAX's local CUDA 12 path separately requires cuDNN 9.10.2 or later but below major 10 (docs.jax.dev).

The numerical boundary is useful for triage. It is insufficient for approval because a single major family contains multiple Toolkit releases and features. A family floor does not necessarily enable every later driver-coupled feature. Likewise, a CUDA-level pass does not make cuDNN major versions interchangeable: ONNX Runtime binaries built with cuDNN 8.x are not compatible with cuDNN 9.x (onnxruntime.ai).

The evidence bundle should make every decision reproducible. *Table 3* is a compatibility-gate worksheet that can be copied into a ticket or comma-separated-value file.

Field	Required value	Pass criterion
GPU architecture	Product, UUID, compute capability, partition profile	Candidate binaries contain a compatible architecture or a deliberately tested JIT path.
Host driver branch	Full version and documented family	Meets exact Toolkit and framework minimum, or an explicitly supported exception exists.
Kernel module	Loaded version and kernel	Matches the intended installed driver and is captured from the canary node.
Container Toolkit	Version, runtime handler, visible-device and capability settings	Intended GPUs and required host libraries are exposed. Kubernetes requires the vendor device plugin before Pods consume GPUs (kubernetes.io).
Image runtime	Registry, repository, digest, CUDA runtime libraries	Digest is immutable and the exact runtime version is recorded. Artifact Registry can enforce a tag that stays on one digest (docs.cloud.google.com).
Build Toolkit	Toolkit used for application and extensions	Known separately from the host driver and in-image runtime.
Framework wheel	Framework, version, CUDA variant, cuDNN major, compiled architectures	The official framework and library matrices accept every component. PyTorch 2.14 publishes CUDA 12.6, 13.0, and 13.2 wheel variants (pytorch.org).
Compatibility mechanism	Backward, minor, forward package, or none	One documented mechanism applies without mixing assumptions.
PTX/JIT need	None, startup, extension build, dynamic kernel, or model path	Actual JIT path passes, or a native cubin path is selected and evidenced.

Field	Required value	Pass criterion
Test result	Commands, return codes, CUDA errors, output verification, telemetry	All mandatory tests pass against the declared baseline.
Rollback image	Prior digest and orchestrator revision	Prior image and configuration are deployable without a host-driver reversal.

The worksheet turns “driver mismatch” into a bounded diagnosis. Driver initialization, architecture targeting, and PTX compilation are separate failure classes and lead to different remediation paths. Exact error codes and messages should therefore be retained verbatim in the evidence bundle, together with the framework’s device and provider inventory (onnxruntime.ai).

Case Studies and Real-World Examples

These are **dated worked examples**, not evergreen approvals. Each assumes the cited rules as published on September 20, 2026, and each still requires the full canary.

Driver 535 with a CUDA 12.8 image (Hypothetical Example)

A Linux node on driver **535** and an application built within **CUDA 12.x** falls inside the documented 525-to-below-580 range (docs.nvidia.com). The initial state is therefore **canary**, not automatic pass. If the candidate is JAX, its documented CUDA 12 wheel is built with CUDA 12.3 and accepts local CUDA 12.1 or later (docs.jax.dev), but that framework fact does not remove the driver canary. A PyTorch canary should also record whether CUDA is available to the running process (docs.pytorch.org).

- **Approve candidate status** when the workload uses supported compiled targets, the framework and library matrices pass, and all canary stages succeed.
- **Defer** if a custom extension emits PTX at runtime and the older driver must JIT it, because minor compatibility alone does not cover that PTX path.
- **Fail** if the application depends on a driver-coupled feature introduced after the installed branch and returns the documented newer-driver error.

The rollback target is the previously qualified image digest. Azure’s registry guidance states that a manifest has a unique SHA-256 digest (learn.microsoft.com); no host change is needed merely to reverse the application image.

Driver 570 with a CUDA 13.0 image (Hypothetical Example)

Driver **570** is below the broad **580** floor for CUDA 13.x minor compatibility (docs.jax.dev). A plain minor-version argument therefore fails. The operator may evaluate the exact CUDA 13.0 forward-compatibility package only if the GPU and platform are supported, the package-to-base-driver matrix marks the pairing compatible, the loader uses the supplied libraries, and the full canary passes.

- **Fail by default** without a documented and tested forward-compatibility path.

- **Defer** when the package is installed but library resolution has not been demonstrated. Installation evidence is not loader evidence.
- **Approve the exception** only for the exact recorded tuple, with an owner and a scheduled path back to the normal driver branch. Record the framework's actual device list, such as the devices returned for a JAX backend (docs.jax.dev).

Driver 580 with a CUDA 13.4 workload (Hypothetical Example)

Driver **580** meets the broad CUDA 13.x minor-compatibility threshold, but CUDA 13.4 features may require **R615 or later** (docs.nvidia.com). The application owner must identify whether those features are used. A JAX deployment provides an additional check because its CUDA 13 route also sets a Linux 580 minimum (docs.jax.dev). TensorFlow offers a separate pre-initialization physical-device inventory for its own canary (tensorflow.org).

- **Canary** an existing code path that stays within the supported minor-compatible feature set.
- **Defer** when feature use is unknown or a framework release note requires a newer branch.
- **Fail** the older-driver path when the application explicitly requires a feature coupled to the later branch.

This example demonstrates why the major-family floor and the exact release notes must both appear in the approval record. For ONNX Runtime, the installed execution-provider list is an additional attachable check (onnxruntime.ai).

Implications and Future Directions

Compatibility policy should be maintained as code-like data rather than prose in a runbook. Store version ranges, exception packages, image digests, test definitions, and decision outcomes in a reviewed repository. Regenerate the crosswalk whenever the fleet driver, CUDA family, framework wheel, cuDNN major, base image, or GPU architecture changes.

Cloud and managed-platform behavior must remain explicit. GKE Autopilot manages driver installation together with node provisioning and scaling (docs.cloud.google.com), while certain Amazon EKS optimized images include the host driver, CUDA user-mode driver, and Container Toolkit (docs.aws.amazon.com). Managed installation changes who supplies a layer, not whether that layer must be inventoried.

Several process changes reduce future upgrade cost:

- **Qualify driver branches as fleet products.** Record the operating systems, kernels, GPU architectures, partition modes, and node images covered by each branch.
- **Publish framework build manifests.** Include CUDA variant, cuDNN major, compiled architectures, extensions, and whether PTX or runtime compilation is expected.
- **Pin every production image.** Tags are discovery aids; promotion and rollback should use digests. Docker explains that digest pulling guarantees invariant image content (docs.docker.com).

- **Separate eligibility from performance.** A compatibility test establishes that a path functions. PyTorch's guidance calls for thread settings representative of real use cases (docs.pytorch.org), but local thresholds must come from the service objective and qualified baseline.
- **Expire exceptions.** A forward-compatibility deployment should have an owner, scope, evidence bundle, and planned return to a normally supported base driver.
- **Retest narrow matrices first.** cuDNN, framework, inference server, operating-system, and architecture support can be narrower than CUDA's driver rule. ONNX Runtime's cuDNN-major constraint is one concrete example (onnxruntime.ai).

The long-term goal is not to eliminate driver upgrades. It is to make an application release independent when documentation and tests support that choice, and to make a fleet upgrade deliberate when they do not.

Frequently Asked Questions (FAQs)

What CUDA version does `nvidia-smi` show?

It shows the driver's supported CUDA ceiling, not necessarily an installed Toolkit or the runtime inside a container. Inventory the image libraries and framework build separately. PyTorch's documented `torch.version.cuda` command inspects the package build (docs.pytorch.org).

What is the minimum driver version for a CUDA Toolkit?

Use the exact Toolkit release notes and Table 2's dated family-level crosswalk. Feature, platform, and framework requirements may raise the broad floor. JAX, for example, documents a Linux 580 minimum for CUDA 13 (docs.jax.dev).

Does a CUDA container need CUDA installed on the host?

It needs a compatible NVIDIA host driver and a configured container runtime. It generally does not need the matching Toolkit installed on the host. TensorFlow's container guide states that only the NVIDIA GPU driver is required on the host (tensorflow.org).

Can a newer CUDA container run on an older driver?

Sometimes. Within a documented major family, minor-version compatibility can apply above the family minimum, with feature and PTX limits. Across a major-family floor, a supported forward-compatibility package may apply. Official CUDA images still require the NVIDIA Container Toolkit for Docker (hub.docker.com), and the exact host, GPU, package, image, and workload must pass the gate.

Does forward compatibility replace a driver upgrade?

No. It is a scoped user-mode compatibility mechanism, not a general upgrade of the host driver stack. Use it as a controlled exception and plan a return to a normally supported driver. The rollback image should remain digest-pinned because Docker states that a digest guarantees invariant image content (docs.docker.com).

Why can CUDA pass while a framework fails?

Frameworks and libraries add their own build and support constraints. JAX validates installed CUDA library versions and reports an error when they are too old (docs.jax.dev). ONNX Runtime requires compatible CUDA and cuDNN major versions when sharing PyTorch libraries (onnxruntime.ai).

How should CUDA be upgraded in production?

Inventory all five layers, pin the candidate and rollback images by digest, classify the compatibility mechanism, run a canary on every affected GPU and node-pool class, preserve results, and promote gradually. Updating a Deployment's container image triggers a rollout (kubernetes.io), which provides a revisioned application path, but host-driver changes require a separate maintenance and rollback plan.

What should make the gate defer rather than fail?

Use **defer** for a potentially supported state with incomplete evidence: unknown PTX/JIT use, unpinned image, unverified loader path, missing framework matrix, or an import-only test. Use **fail** when authoritative documentation rejects the combination or a mandatory canary returns an error, output mismatch, provider fallback, or readiness failure.

Conclusion

CUDA driver, Toolkit, and container compatibility is a layered release decision. The host driver exposes the kernel interface and supported CUDA ceiling. The container supplies most application user space. The framework wheel and native extensions add narrower CUDA, cuDNN, architecture, and PTX requirements. None of those facts can be replaced by the single CUDA number in `nvidia-smi`.

A defensible production gate records the GPU architecture, active driver and kernel module, Container Toolkit configuration, immutable image digest, in-image runtime, build Toolkit, framework variant, compatibility mechanism, PTX requirement, canary output, and rollback image. It then distinguishes three outcomes. **Pass** means every applicable matrix and representative test succeeded for the exact tuple. **Defer** means a documented path may exist but the evidence is incomplete. **Fail** means the combination is unsupported or a required test did not succeed.

The broad family floors provide a fast first screen, but exact release notes and downstream matrices remain decisive. Minor-version compatibility reduces unnecessary fleet-wide driver changes, while forward compatibility offers a narrower exception for supported platforms. Both are valuable when treated as documented mechanisms rather than blanket promises. Digest-pinned images, staged tests, preserved error codes, and a rehearsed rollback turn that documentation into an auditable uptime control.

Source links

Links cited in this article, in order of first appearance. Full addresses are provided for readers using extracted text.

1. https://docs.nvidia.com/cuda/cuda-programming-guide/01-introduction/cuda-platform.html
2. https://www.tensorflow.org/install/docker?authuser=0000

3. https://docs.nvidia.com/deploy/nvidia-smi/
4. https://docs.nvidia.com/deploy/cuda-compatibility/minor-version-compatibility.html
5. https://docs.nvidia.com/cuda/cuda-toolkit-release-notes/
6. https://docs.jax.dev/en/latest/installation.html
7. https://onnxruntime.ai/docs/execution-providers/CUDA-ExecutionProvider.html
8. https://pytorch.org/blog/pytorch-2-14-release-blog/
9. https://docs.docker.com/reference/cli/docker/image/pull/
10. https://kubernetes.io/docs/concepts/workloads/controllers/deployment/
11. https://gpusmith.com/
12. https://docs.pytorch.org/docs/main/generated/torch.cuda.get_device_capability.html
13. https://learn.microsoft.com/en-us/azure/aks-hybrid-edge/local/multi-rack/deploy-gpu-node-pool
14. https://learn.microsoft.com/en-us/azure/container-registry/container-registry-concepts
15. https://www.tensorflow.org/api_docs/python/tf/sysconfig/get_build_info?authuser=1
16. https://docs.nvidia.com/deploy/cuda-compatibility/forward-compatibility.html
17. https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/latest/install-guide.html
18. https://docs.docker.com/engine/containers/gpu/
19. https://hub.docker.com/r/nvidia/cuda
20. https://docs.docker.com/reference/dockerfile
21. https://docs.pytorch.org/FBGEMM/fbgemm_genai/development/BuildInstructions.html
22. https://kubernetes.io/docs/reference/kubernetes-api/core/pod-v1/
23. https://docs.pytorch.org/docs/main/generated/torch.cuda.device_count.html
24. https://docs.jax.dev/en/latest/_autosummary/jax.devices.html
25. https://onnxruntime.ai/docs/api/python/api_summary.html
26. https://www.tensorflow.org/guide/gpu?authuser=0000
27. https://docs.pytorch.org/docs/main/generated/torch.cuda.get_arch_list.html
28. https://docs.pytorch.org/tutorials/recipes/recipes/benchmark
29. https://kubernetes.io/docs/concepts/workloads/controllers/job/
30. https://www.tensorflow.org/install/source?authuser=0000
31. https://kubernetes.io/docs/tasks/manage-gpus/scheduling-gpus/
32. https://docs.cloud.google.com/artifact-registry/docs/docker/names
33. https://docs.pytorch.org/docs/main/generated/torch.cuda.is_available.html
34. https://www.tensorflow.org/api_docs/python/tf/config/list_physical_devices
35. https://docs.cloud.google.com/kubernetes-engine/docs/concepts/gpus
36. https://docs.aws.amazon.com/eks/latest/userguide/device-management-nvidia-dra-device-plugin.html

THE PUBLISHER

About GPU Smith

GPU Smith provides independent engineering and research for private AI infrastructure. We help technical buyers and operators reason about GPU workload sizing, hardware selection, procurement, deployment and inference operations, from the compute system to the networking, power and cooling requirements around it.

Start with the workload

Useful infrastructure decisions begin with the models, data, throughput, latency and operating constraints a team actually has. GPU Smith helps connect those requirements with system design and deployment choices. The goal is a privately controlled AI environment whose capacity and operational demands are understood before a purchasing decision.

Hardware and supplier research

Our public reference library covers GPUs, complete systems and networking components. The server-vendor directory and manufacturing research help teams investigate suppliers, compare options and follow sources behind technical claims. We distinguish manufacturer specifications from measured benchmarks and advertised capabilities from tested configurations. Reference pages are research resources, not a live inventory listing or a binding equipment quote.

Deployment and operations

GPU Smith's engineering scope includes infrastructure integration, networking, power, cooling and the practical operation of inference workloads. Our published articles explain the tradeoffs behind deployment, procurement and ongoing operations so teams can ask better questions and document decisions.

Work with GPU Smith

Explore the hardware library, GPU references, networking references and vendor research. Contact GPU Smith with your workloads and deployment constraints to discuss a project and confirm current scope, pricing and availability.

Inclusion of a manufacturer or product in our research does not imply a partnership, certification or endorsement.

Website: https://gpusmith.com

This article is educational. Verify consequential medical, legal, financial, regulatory and technical decisions with appropriate professionals and the cited primary sources. Company services are described separately from the article's research findings.