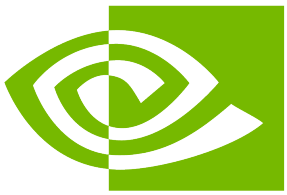

# NVIDIA cuObject client v1.0.0 Release Notes

***Release r1.16***

**NVIDIA Corporation**

**Jan 12, 2026**

# Contents

# Chapter 1. NVIDIA cuObject client v1.0.0 Release Notes

Release information for NVIDIA® cuObject client.

Table 1.1: Release information

| Version | Date | cuObject client libraries with CUDA Release |
|---|---|---|
| v1.0.0 | Jan 12, 2026 | 13.1.1 |

## 1.1. 1. Introduction

Release information for NVIDIA® cuObject client for developers and users.

cuObject is a high-performance suite of libraries designed to enable direct data transfers between GPU memory or system memory and object storage (S3-compatible) solution via RDMA. By relying on RDMA operations, rather than TCP transfer methods, cuObject both avoids using CPU kernel code for TCP processing and can bypass the CPU for data payloads. cuObject eliminates the traditional bottleneck of staging data in local scratch file systems, enabling high-throughput data ingestion for AI training and inference at scale.

### 1.1.1. Architecture Overview

cuObject consists of two primary components:

- **cuObjClient**: Provides client-side APIs for GET and PUT operations with user-defined callbacks for control path communication, while data path operations leverage RDMA for high-performance transfers.
- **cuObjServer**: Implements the RDMA-accelerated server side for S3-compatible object storage services, supporting multi-threaded concurrency, automatic connection management, and Dynamically Connected (DC) transport.

Refer to the following guides for more information about cuObject:

- cuObject Documentation

# 1.2. 2. Key Features and Changes

The following features have been added in v1.0.0:

- **Zero-Copy Data Path**: Direct RDMA transfers between client and server memory, eliminating CPU involvement in data movement.
- **GPUDirect RDMA Support**: Native support for CUDA device memory and system memory.
- **Callback-Based Architecture**: User-defined GET and PUT callbacks for control path communication.
- **Memory Registration APIs**: `cuMemObjGetDescriptor()` and `cuMemObjPutDescriptor()`.
- **RDMA Token Management**: `cuMemObjGetRDMAToken()` and `cuMemObjPutRDMAToken()` for manual descriptor control.
- **Context Management**: `getCtx()` utility for extracting user context from callbacks.
- **Thread-Safe Operations**: Concurrent GET and PUT operations on different registered buffers.

# 1.3. 3. Known Limitations

## 1.3.1. Memory and Size Constraints

- Maximum memory registration: 4 GiB per `cuMemObjGetDescriptor()` call.
- `buffer_offset` is ignored and defaults to 0 in the GET and PUT API calls.
- Concurrent GET and PUT operations on a single RDMA buffer are not supported.
- GET and PUT for GPU unregistered buffers is limited to the `per_buffer_cache_size_kb` configuration value specified in the `cufile.json` configuration.
- GET and PUT for host memory is not supported for unregistered buffers.
- The `execution` section in `cufile.json` does not apply for cuObject, which means cuObject does not use a threadpool for issuing IOs.
- The memory key can be reused after a deregister followed by a re-registration. For server error handling scenarios, either the server should guarantee that all outstanding IOs are drained or the client should call `cuFileDriverClose()` to reject any stale RDMA IO requests from corrupting newer registrations.

# 1.4. 4. Getting Started

## 1.4.1. Hardware Requirements

- x86_64 or ARM64 (Grace CPU only) CPU architecture
- Mellanox ConnectX-5 and above InfiniBand HCA or RoCE-capable Ethernet adapter

- NVIDIA GPU with CUDA Compute Capability 6.0 or higher (for GPU memory operations)
- Minimum 16 GB system RAM recommended

## 1.4.2. Software Requirements

- Linux operating system (tested on Ubuntu 22.04 and 24.04, RHEL 9 and 10)
- CUDA Toolkit 13.1.1 or later
- NVIDIA GPU Driver compatible with CUDA 13.1.1
- InfiniBand and RDMA drivers and libraries (`libibverbs`, `librdmacm`)
- CUDA runtime libraries for GPU memory support
- GPUDirect Storage (GDS) libraries (for GPU and system memory support)
- C++14 or later compatible compiler

## 1.4.3. Installation

#### 1.4.3.1 Install DOCA and RDMA Libraries

Use the DOCA Installation Guide for Linux to install DOCA.

#### 1.4.3.2 Install cuObject client Libraries

cuObject client libraries are part of CUDA Toolkit. Follow the instructions from cuObject client library downloads.

## 1.4.4. Verify Installation

```
# Check library presence
$ ldconfig -p | grep cuobj
$ ldconfig -p | grep cufile

# Verify cuObjClient installation
$ rpm -qa | grep cuobjclient    # RHEL/CentOS
$ dpkg -l | grep cuobjclient    # Ubuntu/Debian

# Verify GDS support
$ /usr/local/cuda/gds/tools/gdscheck -p
```

## 1.5. 5. Fixed Issues

This is the initial release of cuObject v1.0.0. No prior versions exist.

## 1.6. 6. Open Issues

None reported for v1.0.0 initial release.

## Copyright