Kubernetes support for GH200 / GB200

Author: Kevin Klues

Revision	Date	Notes
1	Jan 22, 2025	Copy the content from the following document and update it with the latest design: [External] OUTDATED: IMEX Support for GH200/GB…
2	Feb 10, 2025	Updated to remove “Immediate Mode” and focus solely on what was previously called “Delayed mode” with a much cleaner design
3	May 25, 2025	Added a section on current limitations

Overview

The latest GPU drivers have support for a feature known as IMEX channels.

In a nutshell, an IMEX channel is a construct that allows a set of GPUs to directly read / write each other’s memory over a high-bandwidth NVLink. The NVLink connection may either be directly between GPUs on the same node or between GPUs on separate nodes connected by an NVSwitch. Once an IMEX channel has been established for a set of GPUs, they are free to read / write each other's memory via extensions to the CUDA memory call APIs.

The ability to support IMEX channels on GH200 and GB200 systems is essential, as they have

been designed specifically to exploit the use of IMEX channels to turn a rack of GPU machines (each with a small number of GPUs) into a giant supercomputer with up to 72 GPUs communicating at full NVLink bandwidth.

In the context of Kubernetes, this poses some unique challenges because an IMEX channel (by definition) is a resource that spans multiple nodes. However, Kubernetes does not traditionally provide the primitives necessary to allocate cross-node resources such as this.

Fortunately, NVIDIA has spent the last several years co-developing a new resource model for Kubernetes called “Dynamic Resource Allocation (DRA)”, which **does** provide the necessary primitives to support cross-node resources such as IMEX channels. This feature became production ready in Kubernetes 1.32 (i.e. December 2024) and forms the backbone for how we support IMEX channels in Kubernetes.

A previous version of this document outlined our initial plans to support this feature in the GPU Operator:

[External] OUTDATED: IMEX Support for GH200/GB200

However, feedback from CSPs as well as run:ai pointed to three fundamental flaws in this initial design.

1. Forcing CSPs to understand the complexities of managing IMEX daemons and their corresponding IMEX domains was overly burdensome. It also went against the best practices recommended by the NVIDIA driver team for how to manage IMEX domains (they recommend creating one IMEX domain per workload, rather than having a static one shared by multiple workloads).

2. The previous solution did not lend itself well to adding nodes to a given IMEX domain, nor replacing an existing node if there was a node failure. Any node replacements or additions would be disruptive to ***all*** workloads running in an IMEX domain where a node was being added or replaced (even workloads that didn't need access to the added or replaced node). CSPs would also be responsible for managing the lifecycle and complexity of this because they were the ones in charge of managing the IMEX daemons, not the GPU Operator.

3. The previous solution did not allow workloads to span nodes outside of a single IMEX domain. Each workload had to fit within a single IMEX domain, making it impossible to run workloads that spanned multiple GB200 systems (or spread out onto non-GB200 systems as necessary). This may not be the common use-case at present, but we shouldn't design our solution in a way that outright prohibits it.

The rest of this document outlines a redesign of our initial support for IMEX channels in Kubernetes that addresses all of these challenges.

Terminology

NVLink Domain	A set of physically connected GPUs over NVLink. This connection may be within a single node or across multiple nodes connected by an NVSwitch.
Cluster UUID	The unique identifier for a given NVLink Domain. It can be queried on a GPU by GPU basis via nvidia-smi. All GPUs on a given node are guaranteed to have the same Cluster UUID. If they do not, it is considered a misconfiguration.
Clique ID	A unique identifier within an NVLink domain that indicates which GPUs within that domain are physically capable of talking to each other. The partitioning of GPUs into a set of cliques is done at the NVSwitch layer, not at the individual node layer. All GPUs on a given node are guaranteed to have the same <ClusterUUID, Clique ID> pair. If they do not, it is considered a misconfiguration.
IMEX Domain	A subset of nodes within an NVLink domain, each of which is running an IMEX daemon configured to point at the IP addresses for all nodes within the subset. All nodes within an IMEX domain must have GPUs with the same <ClusterUUID, Clique ID> pair in order for them to communicate. Multiple IMEX domains can exist within a single NVLink domain, but each node can only be a part of one IMEX domain at a time. By definition, an IMEX domain cannot span nodes across multiple NVLink domains.
IMEX Channel	A software construct used to support multi-tenancy within an IMEX domain that has been set up. CUDA processes running within a given IMEX domain can only communicate with one another over NVLink if they have access to the same IMEX channel.

Design

Below is a side-by-side comparison of the originally proposed solution to the newly proposed

solution from the perspective of the user-facing API:

Original

```yaml
---
apiVersion: resource.k8s.io/v1beta1
kind: ResourceClaim
metadata:
  name: imex-channel-resource-claim
spec:
  devices:
    requests:
    - name: channel
      deviceClassName: imex.nvidia.com

---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: test-workload
  labels:
    app: test-workload
spec:
  replicas: 18
  selector:
    matchLabels:
      app: test-workload
  template:
    metadata:
      labels:
        app: test-workload
    spec:
      containers:
      - name: ctr
        image: ubuntu:22.04
        command: ["test-workload"]
        resources:
          limits:
            nvidia.com/gpu: 4
          claims:
```

Updated

```yaml
---
apiVersion: resource.nvidia.com/v1beta1
kind: ComputeDomain
metadata:
  name: compute-domain
spec:
  numNodes: 18
  channel:
    resourceClaimTemplate:
      name: compute-domain-resource-claim

---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: test-workload
  labels:
    app: test-workload
spec:
  replicas: 18
  selector:
    matchLabels:
      app: test-workload
  template:
    metadata:
      labels:
        app: test-workload
    spec:
      containers:
      - name: ctr
        image: ubuntu:22.04
        command: ["test-workload"]
        resources:
          limits:
            nvidia.com/gpu: 4
          claims:
```

```
      - name: imex-channel
  resourceClaims:
  - name: imex-channel
    resourceClaimName: imex-channel-resource-claim
```

```
      - name: compute-domain
  resourceClaims:
  - name: compute-domain
    resourceClaimTemplateName: compute-domain-resource-claim
```

In both cases, the end-user must create exactly two objects – one object to represent the multi-node workload they want to run (e.g. a **`Deployment`** in this case) and another object that links each of the workers of the multi-node workload together via a **`ResourceClaim`**.

In the original proposal, we expected users to directly create a **`ResourceClaim`** to reference in their workloads. This **`ResourceClaim`** was tied to the allocation of an "IMEX channel" from a pre-setup IMEX domain managed by a CSP or daemonset.

In the new proposal, we have introduced a level-of-indirection whereby the user instead creates a custom resource called **`ComputeDomain`**, which (among other things) will trigger the automatic creation of a **`ResourceClaimTemplate`** which the workload can reference just like before. Each worker then gets a unique **`ResourceClaim`** created for it that links it back to the **`ComputeDomain`** where the **`ResourceClaimTemplate`** was defined.

It's important to note that this new **`ComputeDomain`** object is not necessarily tied directly to the notion of an "IMEX channel". Instead, it is tied to the looser notion of running a multi-node workload on a set of compute nodes.

That is to say, if some subset of the nodes associated with a **`ComputeDomain`** are capable of communicating over IMEX, a one-off IMEX domain will be set up on-the-fly to allow GPUs to communicate over their multi-node NVLink connections. Multiple such IMEX domains will be created as necessary depending on the number (and availability) of nodes allocated to the **`ComputeDomain`**. Nodes that are not capable of communicating over IMEX can still be allocated to the same **`ComputeDomain`** as IMEX capable nodes, so users must take care to set NodeSelectors and / or PodAffinities on their workloads to ensure they are allocated the exact set of nodes they desire.

Summary: What happens behind the scenes

1. User creates a ComputeDomain, which triggers the creation of…

   → ResourceClaimTemplate for the Workload:

   - Used to request access to a unique IMEX channel on a set of compute nodes
   - A reference to the ComputeDomain this ResourceClaimTemplate is associated with is passed along as part of the request
   - Each worker of a multi-node job should reference this ResourceClaimTemplate to ensure that they have access to the same IMEX channel inside the appropriate

ComputeDomain

→ ResourceClaimTemplate for the DaemonSet:

- Used to request access to a device that injects ComputeDomain-specific settings for an imex-daemon to run with (config.cfg, nodes_config.cfg, etc.)
- A reference to the ComputeDomain this ResourceClaimTemplate is associated with is passed along as part of the request
- Each worker of the daemonset mentioned below references this ResourceClaimTemplate to ensure they start up IMEX daemons associated with the appropriate ComputeDomain

→ DaemonSet

- Forms one (or more) IMEX domains by running IMEX daemons on the set of nodes in a ComputeDomain that requires them
- These daemons “follow” the workload pods to the nodes where they have been scheduled. Through the magic of DRA, these daemons are guaranteed to be fully up and running before the workload pods that triggered their creation are allowed to run

→ Node label

- Each node where a workload lands that is part of a ComputeDomain will get a label indicating it is part of that ComputeDomain
- This label is used as a NodeSelector on the DaemonSet mentioned above to trigger the scheduling of its pods to specific nodes
- Once all workloads running in a ComputeDomain have run to completion, the label gets removed (even if the ComputeDomain itself hasn’t been deleted yet)
- This allows these nodes to be reused for other ComputeDomains, as necessary

2. As workloads referencing the ResourceClaimTemplate for workloads get scheduled, they trigger the DRA driver to request access to the same IMEX channel on whatever node they land on.

3. Once scheduled to a node, the DRA driver gets invoked, adds the Node label for the ComputeDomain to the node where the workload has been scheduled, and blocks until the IMEX daemon on that node has been started.

4. The addition of this Node label triggers the DaemonSet to deploy an IMEX daemon to that node and start running it.

5. Once all daemons have been fully started, the DRA driver unblocks each worker, injects its IMEX channel into it and allows it to start running.

Current Limitations

1. Each node can only run one multi-node workload associated with a ComputeDomain
    - This makes sense, when each worker of a multi-node workload consumes all of the GPUs on the node it lands on
    - However, if a worker only consumes some of the GPUs on a node, the remaining GPUs cannot be used for a different multi-node workload
    - They do, however, remain available for allocation by single-node jobs running on the node

2. No support for elastic multi-node workloads
    - Workloads associated with a ComputeDomain are fixed in size, subject to the value of the numNodes field set in the ComputeDomain
    - If you want to grow or shrink a workload, you will need to delete it and create a new one associated with a larger ComputeDomain

3. A single node failure will trigger all of the pods running in the IMEX domain where the replacement pod restarts, and a new IMEX domain will be formed around them
    - This is actually only true if the replacement pod lands on a node in a NVLinkDomain/Clique that is already associated with the ComputeDomain
    - When this happens, the ComputeDomain controller is smart enough to recognize this, and trigger the formation of a new IMEX domain around the newly created pod and its peers in the same NVLinkDomain/Clique
    - As part of this, each peer will be restarted to accommodate the new IMEX domain
    - Pods of the same workload running in a different NVLinkDomain/Clique will not be affected
    - Other workloads running in the same NVLinkDomain/Clique but associated with a different ComputeDomain are similarly not affected

We have plans to address all of these issues in upcoming releases, but some of them will take longer than others because they require changes to the underlying IMEX daemon to make it more dynamic to configuration changes.