Red Hat AI Inference Server 3.0

开始使用

Red Hat AI Inference Server 入门

Last Updated: 2025-07-20

Red Hat AI Inference Server 3.0 开始使用

Red Hat AI Inference Server 入门

Legal Notice

Abstract

了解如何使用红帽 AI Inference Server for model serving and inferencing。

Table of Contents

前言

Red Hat AI Inference Server 是一个容器镜像，它通过 LLMs 优化服务和推断。使用 AI Inference Server，您可以以增强其性能并降低成本的方式提供和推测模型。

第 1 章 关于 AI INFERENCE SERVER

AI Inference 服务器提供企业级稳定性和安全性，基于上游开源软件进行构建。AI Inference 服务器利用上游 vLLM 项目，它提供第一流推测功能。

例如，AI Inference 服务器使用持续批处理来处理请求，而不必等待完整批处理被累积。它还使用 10sor parallelism 在多个 GPU 之间分发 LLM 工作负载。这些功能提高了延迟和更高的吞吐量。

要降低推断模型的成本，AI Inference 服务器使用页面关注。LLMs 使用一种称为注意的机制来理解与用户对话。通常，请注意，使用大量内存。通过为 LLM 置备内存，如虚拟内存可用于操作系统的方式，页面对此内存的注意。这个方法消耗较少的内存，这会降低成本。

要验证 AI Inference Server 节约成本和性能提高，请完成以下步骤：

1. 使用 AI Inference Server 提供和推断
2. 使用关键指标验证 Red Hat AI Inference 服务器的好处

第 2 章 产品和版本兼容性

下表列出了 Red Hat AI Inference Server 3.0 支持的产品版本。

表 2.1. 产品和版本兼容性

产品	支持的版本
Red Hat AI Inference Server	3.0
vLLM core	0.8.4
LLM Compressor	0.5.1 技术预览

第 3 章 使用 AI INFERENCE SERVER 提供和推断

使用红帽 AI Inference Server 提供大型语言模型和推测。

先决条件

- 已安装 Podman 或 Docker
- 您可以使用 NVIDIA 或 AMD GPU 访问 Linux 服务器，并以具有 root 权限的用户身份登录
  - 对于 NVIDIA GPU：
    - 安装 NVIDIA 驱动程序
    - 安装 NVIDIA Container Toolkit
    - 如果您的系统有多个使用 NVswitch 的 NVIDIA GPU，则必须具有启动 Fabric Manager 的 root 访问权限
  - 对于 AMD GPU：
    - 安装 ROCm 软件
    - 验证您是否可以运行 ROCm 容器
      - 您可以访问 **registry.redhat.io** 并已登录
      - 您有一个 Hugging Face 帐户，并生成了一个 Hugging Face 令牌

注意

AMD GPU 仅支持 FP8 (W8A8)和 GGUF 量化方案。如需更多信息，请参阅支持的硬件。

流程

1. 使用下表，识别您的基础架构的正确镜像。

GPU	AI Inference Server 镜像
NVIDIA CUDA (T4, A100, L4, L40S, H100, H200)	**registry.redhat.io/rhaiis/vllm-cuda-rhel9:3.0.0**
AMD ROCm (MI210, MI300X)	**registry.redhat.io/rhaiis/vllm-rocm-rhel9:3.0.0**

2. 在服务器主机上打开一个终端，并登录到 **registry.redhat.io** ：

```
$ podman login registry.redhat.io
```

3. 为您的 GPU 拉取相关镜像：

```
$ podman pull registry.redhat.io/rhaiis/vllm-<gpu_type>-rhel9:3.0.0
```

4. 如果您的系统启用了 SELinux，请将 SELinux 配置为允许设备访问：

```
$ sudo setsebool -P container_use_devices 1
```

5. 创建卷并将其挂载到容器中。调整容器权限，以便容器可以使用它。

```
$ mkdir -p rhaiis-cache
```

```
$ chmod g+rwX rhaiis-cache
```

6. 创建或附加 **HF_TOKEN** Hugging Face 令牌到 **private.env** 文件。提供 **private.env** 文件。

```
$ echo "export HF_TOKEN=<your_HF_token>" > private.env
```

```
$ source private.env
```

7. 启动 AI Inference Server 容器镜像。

   a. 对于 NVIDIA CUDA 加速器：

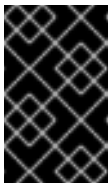
      i. 如果主机系统有多个 GPU 并使用 NVSwitch，则启动 NVIDIA Fabric Manager。要检测您的系统是否使用 NVSwitch，请首先检查 **/proc/driver/nvidia-nvswitch/devices/** 中是否存在文件，然后启动 NVIDIA Fabric Manager。启动 NVIDIA Fabric Manager 需要 root 特权。

```
$ ls /proc/driver/nvidia-nvswitch/devices/
```

输出示例

```
0000:0c:09.0  0000:0c:0a.0  0000:0c:0b.0  0000:0c:0c.0  0000:0c:0d.0
0000:0c:0e.0
```

```
$ systemctl start nvidia-fabricmanager
```



重要

只有在使用多个 GPU 的系统上需要使用 NVswitch 的系统才需要 NVIDIA Fabric Manager。如需更多信息，请参阅 NVIDIA 服务器架构。

      ii. 运行以下命令，检查 Red Hat AI Inference Server 容器是否可以访问主机上的 NVIDIA GPU：

```
$ podman run --rm -it \
--security-opt=label=disable \
--device nvidia.com/gpu=all \
nvcr.io/nvidia/cuda:12.4.1-base-ubi9 \
nvidia-smi
```

输出示例

```
+-----------------------------------------------------------------------------------------+
| NVIDIA-SMI 570.124.06             Driver Version: 570.124.06     CUDA
Version: 12.8     |
|-----------------------------------------+------------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncorr.
ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util
Compute M. |
|                                         |                        |               MIG M. |
|=========================================+========================+======================
=====+=======================|
|   0  NVIDIA A100-SXM4-80GB          Off |   00000000:08:01.0 Off |
0 |
| N/A   32C    P0             64W /  400W |       1MiB /  81920MiB |      0%
Default |
|                                         |                        |           Disabled |
+-----------------------------------------+------------------------+----------------------+
|   1  NVIDIA A100-SXM4-80GB          Off |   00000000:08:02.0 Off |
0 |
| N/A   29C    P0             63W /  400W |       1MiB /  81920MiB |      0%
Default |
|                                         |                        |           Disabled |
+-----------------------------------------+------------------------+----------------------+

+-----------------------------------------------------------------------------------------+
| Processes:                                                                              |
|  GPU   GI   CI              PID   Type   Process name                        GPU
Memory |
|        ID   ID                                                               Usage      |
|=========================================================================================
============================|
|  No running processes found                                                             |
+-----------------------------------------------------------------------------------------+
```

iii. 启动容器。

```
$ podman run --rm -it \
--device nvidia.com/gpu=all \
--security-opt=label=disable \ 1
--shm-size=4g -p 8000:8000 \ 2
--userns=keep-id:uid=1001 \ 3
--env "HUGGING_FACE_HUB_TOKEN=$HF_TOKEN" \ 4
--env "HF_HUB_OFFLINE=0" \
--env=VLLM_NO_USAGE_STATS=1 \
-v ./rhaiis-cache:/opt/app-root/src/.cache:Z \ 5
registry.redhat.io/rhaiis/vllm-cuda-rhel9:3.0.0 \
--model RedHatAI/Llama-3.2-1B-Instruct-FP8 \
--tensor-parallel-size 2 6
```

1 启用 SELinux 的系统需要。**--security-opt=label=disable** 防止 SELinux 重新标记卷挂载中的文件。如果您选择不使用此参数，您的容器可能无法成功运行。

2 如果您在共享内存时遇到问题，请将 **-shm-size** 增加到 **8GB**。

3. 将主机 UID 映射到容器中 vLLM 进程的有效 UID。您也可以 pass--**user=0**，但这比 the- **userns** 选项不太安全。set-- **user=0** 在容器内以 root 身份运行 vLLM。

4. 使用 Hugging Face API 访问令牌设置和导出 **HF_TOKEN**

5. 启用 SELinux 的系统需要。在 Debian 或 Ubuntu 操作系统上，或者在不使用 SELinux 的情况下使用 Docker 时，**:Z** 后缀不可用。

6. 在多个 GPU 上运行 AI Inference Server 容器时，set **-tensor-parallel-size** 与 GPU 的数量匹配。

b. 对于 AMD ROCm 加速器：

i. 使用 **amd-smi static -a** 验证容器是否可以访问主机系统 GPU：

```
$ podman run -ti --rm --pull=newer \
--security-opt=label=disable \
--device=/dev/kfd --device=/dev/dri \
--group-add keep-groups \ 1
--entrypoint="" \
registry.redhat.io/rhaiis/vllm-rocm-rhel9:3.0.0 \
amd-smi static -a
```

1. 您必须属于 AMD 系统上的视频和呈现组才能使用 GPU。要访问 GPU，您必须将 **--group-add=keep-groups** 补充组选项传给容器。

ii. 启动容器：

```
podman run --rm -it \
--device /dev/kfd --device /dev/dri \
--security-opt=label=disable \ 1
--group-add keep-groups \
--shm-size=4GB -p 8000:8000 \ 2
--env "HUGGING_FACE_HUB_TOKEN=$HF_TOKEN" \
--env "HF_HUB_OFFLINE=0" \
--env=VLLM_NO_USAGE_STATS=1 \
-v ./rhaiis-cache:/opt/app-root/src/.cache \
registry.redhat.io/rhaiis/vllm-rocm-rhel9:3.0.0 \
--model RedHatAI/Llama-3.2-1B-Instruct-FP8 \
--tensor-parallel-size 2 3
```

1. **--security-opt=label=disable** 防止 SELinux 重新标记卷挂载中的文件。如果您选择不使用此参数，您的容器可能无法成功运行。

2. 如果您在共享内存时遇到问题，请将 **-shm-size** 增加到 **8GB**。

3. 在多个 GPU 上运行 AI Inference Server 容器时，set **-tensor-parallel-size** 与 GPU 的数量匹配。

8. 在终端中的单独标签页中，使用 API 向模型发出请求。

```
curl -X POST -H "Content-Type: application/json" -d '{
    "prompt": "What is the capital of France?",
    "max_tokens": 50
}' http://<your_server_ip>:8000/v1/completions | jq
```

输出示例

```
{
    "id": "cmpl-b84aeda1d5a4485c9cb9ed4a13072fca",
    "object": "text_completion",
    "created": 1746555421,
    "model": "RedHatAI/Llama-3.2-1B-Instruct-FP8",
    "choices": [
        {
            "index": 0,
            "text": " Paris.\nThe capital of France is Paris.",
            "logprobs": null,
            "finish_reason": "stop",
            "stop_reason": null,
            "prompt_logprobs": null
        }
    ],
    "usage": {
        "prompt_tokens": 8,
        "total_tokens": 18,
        "completion_tokens": 10,
        "prompt_tokens_details": null
    }
}
```

第 4 章 使用关键指标验证 RED HAT AI INFERENCE 服务器的好处

使用以下指标评估 AI Inference Server 提供的 LLM 模型的性能：

- **第一次令牌(TTFT)** ：模型提供其响应的第一个令牌所需的时间？
- **各个输出令牌(TPOT)** 的时间：模型需要多久才能向已发送请求的每个用户提供输出令牌？
- **延迟** ：模型生成完整响应所需的时间？
- **吞吐量** ：在所有用户和请求中，模型可以同时生成多少个输出令牌？

完成以下步骤，运行一个基准测试，其中显示了 AI Inference Server 和其他 inference 服务器如何根据这些指标执行。

先决条件

- AI Inference Server 容器镜像
- GitHub 帐户
- Python 3.9 或更高版本

流程

1. 在您的主机系统上，启动一个 AI Inference Server 容器并提供模型。

```
$ podman run --rm -it --device nvidia.com/gpu=all \
--shm-size=4GB -p 8000:8000 \
--env "HUGGING_FACE_HUB_TOKEN=$HF_TOKEN" \
--env "HF_HUB_OFFLINE=0" \
-v ./rhaiis-cache:/opt/app-root/src/.cache \
--security-opt=label=disable \
registry.redhat.io/rhaiis/vllm-cuda-rhel9:3.0.0 \
--model RedHatAI/Llama-3.2-1B-Instruct-FP8
```

2. 在一个单独的终端选项卡中，安装基准工具依赖项。

```
$ pip install vllm pandas datasets
```

3. 克隆 vLLM Git 存储库 ：

```
$ git clone https://github.com/vllm-project/vllm.git
```

4. 运行 **./vllm/benchmarks/benchmark_serving.py** 脚本。

```
$ python vllm/benchmarks/benchmark_serving.py --backend vllm --model RedHatAI/Llama-3.2-1B-Instruct-FP8 --num-prompts 100 --dataset-name random  --random-input 1024 --random-output 512 --port 8000
```

验证

结果显示 AI Inference 服务器如何根据密钥服务器指标执行：

```
============ Serving Benchmark Result ============
Successful requests:                     100
Benchmark duration (s):                  4.61
Total input tokens:                      102300
Total generated tokens:                  40493
Request throughput (req/s):              21.67
Output token throughput (tok/s):         8775.85
Total Token throughput (tok/s):          30946.83
---------------Time to First Token----------------
Mean TTFT (ms):                          193.61
Median TTFT (ms):                        193.82
P99 TTFT (ms):                           303.90
-----Time per Output Token (excl. 1st token)------
Mean TPOT (ms):                          9.06
Median TPOT (ms):                        8.57
P99 TPOT (ms):                           13.57
---------------Inter-token Latency----------------
Mean ITL (ms):                           8.54
Median ITL (ms):                         8.49
P99 ITL (ms):                            13.14
==================================================
```

尝试更改此基准的参数，然后再次运行它。注意如何将 **vllm** 作为后端与其他选项进行比较。吞吐量应该始终更高，但延迟应该较低。

- 其它选项是： **tgi**,**lmdeploy**,**deepspeed-mii**,**openai**, 和 **openai-chat**

- **--dataset-name 的其它选项有： sharegpt,burstgpt,sonnet,random,hf**

其他资源

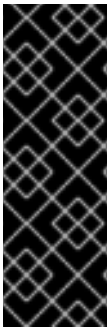
- **vLLM 文档**

- **LLM Inference Performance Engineering: 最佳实践, Mosaic AI research，它解释了吞吐量和延迟等指标**

第 5 章 故障排除

Red Hat AI Inference Server 3.0 的以下故障排除信息描述了与模型加载、内存、模型响应质量、网络和 GPU 驱动程序相关的常见问题。在可用的情况下，描述了常见问题的临时解决方案。

vLLM 中的大多数常见问题与安装、型号加载、内存管理和 GPU 通信相关。大多数问题可以通过使用正确配置的环境来解决，确保兼容硬件和软件版本，并遵循推荐的配置实践。



重要

对于持久问题，请导出 VLLM_LOGGING_LEVEL=DEBUG 以启用调试日志，然后检查日志。

```
$ export VLLM_LOGGING_LEVEL=DEBUG
```

5.1. 模型加载错误

- 当您在没有指定用户命名空间的情况下运行 Red Hat AI Inference Server 容器镜像时，会返回一个未识别的模型错误。

```
podman run --rm -it \
--device nvidia.com/gpu=all \
--security-opt=label=disable \
--shm-size=4GB -p 8000:8000 \
--env "HUGGING_FACE_HUB_TOKEN=$HF_TOKEN" \
--env "HF_HUB_OFFLINE=0" \
--env=VLLM_NO_USAGE_STATS=1 \
-v ./rhaiis-cache:/opt/app-root/src/.cache \
registry.redhat.io/rhaiis/vllm-cuda-rhel9:3.0.0 \
--model RedHatAI/Llama-3.2-1B-Instruct-FP8
```

输出示例

```
ValueError: Unrecognized model in RedHatAI/Llama-3.2-1B-Instruct-FP8. Should have
a model_type key in its config.json
```

要解决这个问题，pass-- userns=keep-id:uid=1001 作为 Podman 参数，以确保容器使用 root 用户运行。

- 当 Red Hat AI Inference Server 下载模型时，下载会失败或卡住。为防止模型下载挂起，请先使用 huggingface-cli 下载模型。例如：

```
$ huggingface-cli download <MODEL_ID> --local-dir <DOWNLOAD_PATH>
```

在提供模型时，将本地模型路径传递给 vLLM，以防止再次下载模型。

- 当 Red Hat AI Inference Server 从磁盘加载模型时，该过程有时会挂起。大型模型会消耗内存，如果内存运行较低，系统会因为它在 RAM 和磁盘之间进行交换数据而减慢。网络文件系统速度缓慢或缺少可用内存可能会触发过度交换。在集群节点之间共享文件系统的集群中会出现这种情况。

在可能的情况下，将模型保存在本地磁盘中，以防止在模型加载过程中出现缓慢的问题。确定系统有足够的 CPU 内存。

确定您的系统有足够的 CPU 容量来处理模型。

- 有时，Red Hat AI Inference Server 无法检查模型。日志中会报告错误。例如：

```
#...
  File "vllm/model_executor/models/registry.py", line xxx, in \_raise_for_unsupported
    raise ValueError(
ValueError: Model architectures [''] failed to be inspected. Please check the logs for
more details.
```

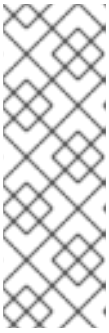
当 vLLM 无法导入模型文件时，这个错误通常与 vLLM 构建中缺少依赖项或过时的二进制文件相关。

- 不支持一些模型架构。请参阅验证的模型 列表。例如，以下错误表示您试图使用的模型不被支持：

```
Traceback (most recent call last):
#...
  File "vllm/model_executor/models/registry.py", line xxx, in inspect_model_cls
```

```
  for arch in architectures:
TypeError: 'NoneType' object is not iterable
```

```
#...
 File "vllm/model_executor/models/registry.py", line xxx, in \_raise_for_unsupported
  raise ValueError(
ValueError: Model architectures [''] are not supported for now. Supported
architectures:
#...
```



注意

有些架构，如 DeepSeekV2VL 需要使用- hf_overrides 标志明确指定架构，例如：

```
--hf_overrides '{\"architectures\": [\"DeepseekVLV2ForCausalLM\"]}
```

- 当您加载 8 位浮点(FP8)模型时，某些硬件有时会发生运行时错误。FP8 需要 GPU 硬件加速。当您加载 FP8 模型（如 deepseek-r1 或带有 F8_E4M3 10sor 类型的型号）时会出现错误。例如：

```
triton.compiler.errors.CompilationError: at 1:0:
def \_per_token_group_quant_fp8(
\^
ValueError("type fp8e4nv not supported in this architecture. The supported fp8 dtypes
are ('fp8e4b15', 'fp8e5')")
[rank0]:[W502 11:12:56.323757996 ProcessGroupNCCL.cpp:1496] Warning: WARNING:
destroy_process_group() was not called before program exit, which can leak
resources. For more info, please see
https://pytorch.org/docs/stable/distributed.html#shutdown (function operator())
```

注意

查看 Getting started 以确保支持您的特定加速器。目前支持的 FP8 模型支持的加速器包括：

  - NVIDIA CUDA T4、A100、L4、L40S、H100 和 H200 GPU
  - AMD ROCm MI300X GPU

-

有时，当提供与主机系统相关的运行时错误时。例如，您可能会在日志中看到错误，如下所示：

```
INFO 05-07 19:15:17 [config.py:1901] Chunked prefill is enabled with
max_num_batched_tokens=2048.
OMP: Error #179: Function Can't open SHM failed:
OMP: System error #0: Success
Traceback (most recent call last):
  File "/opt/app-root/bin/vllm", line 8, in <module>
    sys.exit(main())
..........................    raise RuntimeError("Engine core initialization failed. "
RuntimeError: Engine core initialization failed. See root cause above.
```

您可以在启动 vllm 时传递 **--shm-size=2g** 参数来解决这个问题。

5.2. 内存优化

- 如果模型太大以使用单个 GPU 运行，则会出现内存不足(OOM)错误。使用内存优化选项，如 quantization, tensor parallelism, 或减少精度来减少内存消耗。如需更多信息，请参阅保留内存。

5.3. 生成的模型响应质量

- 在某些情况下，所生成的模型响应的质量可能会在更新后去除去。

在新版本中更新了默认抽样参数源。对于 vLLM 版本 0.8.4 及更高版本，默认的抽样参数来自模型创建器提供的 generation_config.json 文件。在大多数情况下，这会导致更高的质量响应，因为模型创建者可能知道哪个抽样参数最适合其模型。然而，在某些情况下，模型创建者提供的默认值可能会导致性能下降。

如果您遇到这个问题，请使用-- generation-config vllm 服务器参数使用旧默认值提供模型。

重要

如果应用 -generation-config vllm 服务器参数改进了模型输出，请继续使用 vLLM 默认值，并利用 Hugging Face 上的模型创建者更新其默认的 generation_config.json，以便生成更好的质量的生成。

5.4. CUDA 加速器错误

- 使用 CUDA 加速器运行模型时，您可能会遇到 self.graph.replay （） 错误。

  如果 vLLM 崩溃，且错误追踪捕获 vllm/worker/model_runner .py 模块中的 self.graph.replay （） 方法的某种错误，则很可能是 CUDAGraph 类中出现的 CUDA 错误。

  要识别导致错误的特定 CUDA 操作，请将 --enforce-eager 服务器参数添加到 vllm 命令行，以禁用 CUDAGraph 优化并隔离有问题的 CUDA 操作。

- 您可能会遇到由不正确的硬件或驱动程序设置导致的加速器和 CPU 通信问题。

  对于某些类型的 NVIDIA GPU，多 GPU 系统需要 NVIDIA Fabric Manager。nvidia-fabricmanager 软件包和相关 systemd 服务可能无法安装，或者软件包可能没有在运行。

  运行 诊断 Python 脚本，以检查 NVIDIA Collective Communications Library (NCCL)和 Gloo 库组件是否正确通信。

  在 NVIDIA 系统中，运行以下命令来检查光纤管理器状态：

  ```
  $ systemctl status nvidia-fabricmanager
  ```

  在成功配置的系统上，服务应处于活动状态并在运行且无错误。

- 在 NVIDIA Multi -Instance GPU (MIG)硬件上运行带有十个并行性且将--tensor-parallel-size 设置为大于 1 的 vLLM 会导致在初始模型加载或形成检查阶段造成 AssertionError。这通常在启动 vLLM 时作为第一个错误之一。

5.5. 网络错误

- 您可能会遇到复杂网络配置的网络错误。

  要排除网络问题，搜索列出不正确的 IP 地址的 DEBUG 语句的日志，例如：

```
DEBUG 06-10 21:32:17 parallel_state.py:88] world_size=8 rank=0 local_rank=0
distributed_init_method=tcp://<incorrect_ip_address>:54641 backend=nccl
```

要更正此问题，请使用 VLLM_HOST_IP 环境变量设置正确的 IP 地址，例如：

```
$ export VLLM_HOST_IP=<correct_ip_address>
```

指定与 NCCL 和 Gloo 的 IP 地址关联的网络接口：

```
$ export NCCL_SOCKET_IFNAME=<your_network_interface>
```

```
$ export GLOO_SOCKET_IFNAME=<your_network_interface>
```

5.6. PYTHON 多处理错误

- 您可能会遇到 Python 多处理警告或运行时错误。这可能是由没有为 Python 多处理正确构建的代码导致。以下是控制台警告示例：

```
WARNING 12-11 14:50:37 multiproc_worker_utils.py:281] CUDA was previously
  initialized. We must use the `spawn` multiprocessing start method. Setting
  VLLM_WORKER_MULTIPROC_METHOD to 'spawn'. See
  https://docs.vllm.ai/en/latest/getting_started/troubleshooting.html#python-
multiprocessing
  for more information.
```

以下是 Python 运行时错误示例：

```
RuntimeError:
    An attempt has been made to start a new process before the
    current process has finished its bootstrapping phase.

    This probably means that you are not using fork to start your
    child processes and you have forgotten to use the proper idiom
    in the main module:

      if __name__ = "__main__":
        freeze_support()
        ...

    The "freeze_support()" line can be omitted if the program
    is not going to be frozen to produce an executable.

    To fix this issue, refer to the "Safe importing of main module"
    section in https://docs.python.org/3/library/multiprocessing.html
```

要解决运行时错误，请更新您的 Python 代码，以保护 if__name__ = "__main__": 块后面的 vllm 的使用，例如：

```
if __name__ = "__main__":
    import vllm

    llm = vllm.LLM(...)
```

5.7. GPU 驱动程序或设备直通问题

- 当您运行 Red Hat AI Inference Server 容器镜像时，有时不明确设备传递错误是由 GPU 驱动程序或工具（如 NVIDIA Container Toolkit ）导致的。

  - 检查主机机器上安装的 NVIDIA Container 工具包是否可以看到主机 GPU：

    ```
    $ nvidia-ctk cdi list
    ```

    输出示例

    ```
    #...
    nvidia.com/gpu=GPU-0fe9bb20-207e-90bf-71a7-677e4627d9a1
    nvidia.com/gpu=GPU-10eff114-f824-a804-e7b7-e07e3f8ebc26
    nvidia.com/gpu=GPU-39af96b4-f115-9b6d-5be9-68af3abd0e52
    nvidia.com/gpu=GPU-3a711e90-a1c5-3d32-a2cd-0abeaa3df073
    nvidia.com/gpu=GPU-6f5f6d46-3fc1-8266-5baf-582a4de11937
    nvidia.com/gpu=GPU-da30e69a-7ba3-dc81-8a8b-e9b3c30aa593
    nvidia.com/gpu=GPU-dc3c1c36-841b-bb2e-4481-381f614e6667
    nvidia.com/gpu=GPU-e85ffe36-1642-47c2-644e-76f8a0f02ba7
    nvidia.com/gpu=all
    ```

  - 确保在主机上创建了 NVIDIA 加速器配置：

    ```
    $ sudo nvidia-ctk cdi generate --output=/etc/cdi/nvidia.yaml
    ```

  - 运行以下命令，检查 Red Hat AI Inference Server 容器是否可以访问主机上的 NVIDIA GPU：

```
$ podman run --rm -it --security-opt=label=disable --device nvidia.com/gpu=all
nvcr.io/nvidia/cuda:12.4.1-base-ubi9 nvidia-smi
```

输出示例

```
+-----------------------------------------------------------------------------------------+
| NVIDIA-SMI 570.124.06             Driver Version: 570.124.06     CUDA Version: 12.8
|
|-----------------------------------------+------------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute
M. |
|                                         |                        |               MIG M. |
|=========================================+========================+======================
==================|
|   0  NVIDIA A100-SXM4-80GB          Off |   00000000:08:01.0 Off |                    0 |
| N/A   32C    P0             64W /  400W |       1MiB /  81920MiB |      0%      Default |
|                                         |                        |             Disabled |
+-----------------------------------------+------------------------+----------------------+
|   1  NVIDIA A100-SXM4-80GB          Off |   00000000:08:02.0 Off |                    0 |
| N/A   29C    P0             63W /  400W |       1MiB /  81920MiB |      0%      Default |
|                                         |                        |             Disabled |
+-----------------------------------------+------------------------+----------------------+

+-----------------------------------------------------------------------------------------+
| Processes:                                                                              |
|  GPU   GI   CI              PID   Type   Process name                        GPU Memory |
|        ID   ID                                                               Usage      |
|=========================================================================================
==================|
|  No running processes found                                                             |
+-----------------------------------------------------------------------------------------+
```