Red Hat AI Inference Server 3.2

Getting started

Getting started with Red Hat AI Inference Server

Last Updated: 2025-08-07

Red Hat AI Inference Server 3.2 Getting started

Getting started with Red Hat AI Inference Server

Abstract

Learn how to work with Red Hat AI Inference Server for model serving and inferencing.

Table of Contents

PREFACE

Red Hat AI Inference Server is a container image that optimizes serving and inferencing with LLMs. Using AI Inference Server, you can serve and inference models in a way that boosts their performance while reducing their costs.

CHAPTER 1. ABOUT AI INFERENCE SERVER

AI Inference Server provides enterprise-grade stability and security, building on upstream, open source software. AI Inference Server leverages the upstream vLLM project, which provides state-of-the-art inferencing features.

For example, AI Inference Server uses continuous batching to process requests as they arrive instead of waiting for a full batch to be accumulated. It also uses tensor parallelism to distribute LLM workloads across multiple GPUs. These features provide reduced latency and higher throughput.

To reduce the cost of inferencing models, AI Inference Server uses paged attention. LLMs use a mechanism called attention to understand conversations with users. Normally, attention uses a significant amount of memory, much of which is wasted. Paged attention addresses this memory wastage by provisioning memory for LLMs similar to the way that virtual memory works for operating systems. This approach consumes less memory, which lowers costs.

To verify cost savings and performance gains with AI Inference Server, complete the following procedures:

1. Serving and inferencing with AI Inference Server
2. Validating Red Hat AI Inference Server benefits using key metrics

CHAPTER 2. PRODUCT AND VERSION COMPATIBILITY

The following table lists the supported product versions for Red Hat AI Inference Server 3.2.

Table 2.1. Product and version compatibility

Product	Supported version
Red Hat AI Inference Server	3.2.0-1754088865-hotfix-1 (CUDA), 3.2 (ROCm)
vLLM core	v0.9.2
LLM Compressor	LLM Compressor v0.6.0 is not included in the Red Hat AI Inference Server 3.2.0-1754088865-hotfix-1 container image.

CHAPTER 3. SERVING AND INFERENCING WITH AI INFERENCE SERVER

Serve and inference a large language model with Red Hat AI Inference Server.

Prerequisites

- You have installed Podman or Docker
- You have access to a Linux server with NVIDIA or AMD GPUs and are logged in as a user with root privileges
  - For NVIDIA GPUs:
    - Install NVIDIA drivers
    - Install the NVIDIA Container Toolkit
    - If your system has multiple NVIDIA GPUs that use NVswitch, you must have root access to start Fabric Manager
  - For AMD GPUs:
    - Install ROCm software
    - Verify that you can run ROCm containers
- You have access to **registry.redhat.io** and have logged in
- You have a Hugging Face account and have generated a Hugging Face token

For more information about supported quantization schemes for accelerators, see Supported hardware.

Procedure

1. Identify the correct image for your infrastructure.

AI Accelerator	AI Inference Server image
NVIDIA CUDA	**registry.redhat.io/rhaiis/vllm-cuda-rhel9:3.2.0-1754088865-hotfix-1**
AMD ROCm	**registry.redhat.io/rhaiis/vllm-rocm-rhel9:3.2.0**

2. Open a terminal on your server host, and log in to **registry.redhat.io**:

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

3. Pull the relevant image for your GPUs:

   a. To pull an NVIDIA CUDA image, run the following command:

```
$ podman pull registry.redhat.io/rhaiis/vllm-cuda-rhel9:3.2.0-1754088865-hotfix-1
```

–

b. To pull an AMD ROCm image, run the following command:

```
$ podman pull registry.redhat.io/rhaiis/vllm-rocm-rhel9:3.2.0
```

4. If your system has SELinux enabled, configure SELinux to allow device access:

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

5. Create a volume and mount it into the container. Adjust the container permissions so that the container can use it.

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

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

6. Create or append your **HF_TOKEN** Hugging Face token to the **private.env** file. Source the **private.env** file.

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

```
$ source private.env
```

7. Start the AI Inference Server container image.

a. For NVIDIA CUDA accelerators:

i. If the host system has multiple GPUs and uses NVSwitch, then start NVIDIA Fabric Manager. To detect if your system is using NVSwitch, first check if files are present in **/proc/driver/nvidia-nvswitch/devices/**, and then start NVIDIA Fabric Manager. Starting NVIDIA Fabric Manager requires root privileges.

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

Example output

```
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
```

IMPORTANT

NVIDIA Fabric Manager is only required on systems with multiple GPUs that use NVswitch. For more information, see NVIDIA Server Architectures.

ii. Check that the Red Hat AI Inference Server container can access NVIDIA GPUs on the host by running the following command:

```
$ podman run --rm -it \
--security-opt=label=disable \
```

```
--device nvidia.com/gpu=all \
nvcr.io/nvidia/cuda:12.4.1-base-ubi9 \
nvidia-smi
```

Example output

```
+-----------------------------------------------------------------------------------------+
| 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. Start the container.

```
$ 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-rhel9:3.2.0 \
--model RedHatAI/Llama-3.2-1B-Instruct-FP8 \
--tensor-parallel-size 2 (6)
```

1. Required for systems where SELinux is enabled. **--security-opt=label=disable** prevents SELinux from relabeling files in the volume mount. If you choose not to use this argument, your container might not successfully run.

2. If you experience an issue with shared memory, increase **--shm-size** to **8GB**.

3. Maps the host UID to the effective UID of the vLLM process in the container. You can also pass **--user=0**, but this less secure than the **--userns** option. Setting **--**

4. Set and export **HF_TOKEN** with your Hugging Face API access token

5. Required for systems where SELinux is enabled. On Debian or Ubuntu operating systems, or when using Docker without SELinux, the **:Z** suffix is not available.

6. Set **--tensor-parallel-size** to match the number of GPUs when running the AI Inference Server container on multiple GPUs.

b. For AMD ROCm accelerators:

i. Use **amd-smi static -a** to verify that the container can access the host system GPUs:

```
$ 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-rhel9:3.2.0 \
amd-smi static -a
```

1. You must belong to both the video and render groups on AMD systems to use the GPUs. To access GPUs, you must pass the **--group-add=keep-groups** supplementary groups option into the container.

ii. Start the container:

```
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-rhel9:3.2.0 \
--model RedHatAI/Llama-3.2-1B-Instruct-FP8 \
--tensor-parallel-size 2 3
```

1. **--security-opt=label=disable** prevents SELinux from relabeling files in the volume mount. If you choose not to use this argument, your container might not successfully run.

2. If you experience an issue with shared memory, increase **--shm-size** to **8GB**.

3. Set **--tensor-parallel-size** to match the number of GPUs when running the AI Inference Server container on multiple GPUs.

8. In a separate tab in your terminal, make a request to your model with the API.

–

```
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
```

Example output


```
{
    "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
    }
}
```

CHAPTER 4. VALIDATING RED HAT AI INFERENCE SERVER BENEFITS USING KEY METRICS

Use the following metrics to evaluate the performance of the LLM model being served with AI Inference Server:

- **Time to first token (TTFT)**: How long does it take for the model to provide the first token of its response?
- **Time per output token (TPOT)**: How long does it take for the model to provide an output token to each user, who has sent a request?
- **Latency**: How long does it take for the model to generate a complete response?
- **Throughput**: How many output tokens can a model produce simultaneously, across all users and requests?

Complete the procedure below to run a benchmark test that shows how AI Inference Server, and other inference servers, perform according to these metrics.

Prerequisites

- AI Inference Server container image
- GitHub account
- Python 3.9 or higher

Procedure

1. On your host system, start an AI Inference Server container and serve a model.

```
$ 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-rhel9:3.2.0 \
--model RedHatAI/Llama-3.2-1B-Instruct-FP8
```

2. In a separate terminal tab, install the benchmark tool dependencies.

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

3. Clone the vLLM Git repository:

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

4. Run the **./vllm/benchmarks/benchmark_serving.py** script.

```
$ 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
```

–

Verification

The results show how AI Inference Server performs according to key server metrics:

```
============ 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
==================================================
```

Try changing the parameters of this benchmark and running it again. Notice how **vllm** as a backend compares to other options. Throughput should be consistently higher, while latency should be lower.

- Other options for **--backend** are: **tgi**, **lmdeploy**, **deepspeed-mii**, **openai**, and **openai-chat**
- Other options for **--dataset-name** are: **sharegpt**, **burstgpt**, **sonnet**, **random**, **hf**

Additional resources

- vLLM documentation
- LLM Inference Performance Engineering: Best Practices , by Mosaic AI Research, which explains metrics such as throughput and latency

CHAPTER 5. TROUBLESHOOTING

The following troubleshooting information for Red Hat AI Inference Server 3.2 describes common problems related to model loading, memory, model response quality, networking, and GPU drivers. Where available, workarounds for common issues are described.

Most common issues in vLLM relate to installation, model loading, memory management, and GPU communication. Most problems can be resolved by using a correctly configured environment, ensuring compatible hardware and software versions, and following the recommended configuration practices.

IMPORTANT

For persistent issues, export **VLLM_LOGGING_LEVEL=DEBUG** to enable debug logging and then check the logs.

```
$ export VLLM_LOGGING_LEVEL=DEBUG
```

5.1. MODEL LOADING ERRORS

- When you run the Red Hat AI Inference Server container image without specifying a user namespace, an unrecognized model error is returned.

  ```
  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-rhel9:3.2.0 \
  --model RedHatAI/Llama-3.2-1B-Instruct-FP8
  ```

  Example output

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

  To resolve this error, pass **--userns=keep-id:uid=1001** as a Podman parameter to ensure that the container runs with the root user.

- Sometimes when Red Hat AI Inference Server downloads the model, the download fails or gets stuck. To prevent the model download from hanging, first download the model using the **huggingface-cli**. For example:

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

  When serving the model, pass the local model path to vLLM to prevent the model from being downloaded again.

- When Red Hat AI Inference Server loads a model from disk, the process sometimes hangs. Large models consume memory, and if memory runs low, the system slows down as it swaps data between RAM and disk. Slow network file system speeds or a lack of available memory can

trigger excessive swapping. This can happen in clusters where file systems are shared between cluster nodes.
Where possible, store the model in a local disk to prevent slow down during model loading. Ensure that the system has sufficient CPU memory available.

Ensure that your system has enough CPU capacity to handle the model.

- Sometimes, Red Hat AI Inference Server fails to inspect the model. Errors are reported in the log. For example:

```
#...
  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.
```

  The error occurs when vLLM fails to import the model file, which is usually related to missing dependencies or outdated binaries in the vLLM build.

- Some model architectures are not supported. Refer to the list of Validated models. For example, the following errors indicate that the model you are trying to use is not supported:

```
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:
#...
```

  NOTE

  Some architectures such as **DeepSeekV2VL** require the architecture to be explicitly specified using the **--hf_overrides** flag, for example:

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

- Sometimes a runtime error occurs for certain hardware when you load 8-bit floating point (FP8) models. FP8 requires GPU hardware acceleration. Errors occur when you load FP8 models like **deepseek-r1** or models tagged with the **F8_E4M3** tensor type. For example:

```
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())
```

> **NOTE**
>
> Review Getting started to ensure your specific accelerator is supported. Accelerators that are currently supported for FP8 models include:
>
> - NVIDIA CUDA T4, A100, L4, L40S, H100, and H200 GPUs
> - AMD ROCm MI300X GPUs

- Sometimes when serving a model a runtime error occurs that is related to the host system. For example, you might see errors in the log like this:

```
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.
```

You can work around this issue by passing the **--shm-size=2g** argument when starting **vllm**.

5.2. MEMORY OPTIMIZATION

- If the model is too large to run with a single GPU, you will get out-of-memory (OOM) errors. Use memory optimization options such as quantization, tensor parallelism, or reduced precision to reduce the memory consumption. For more information, see Conserving memory.

5.3. GENERATED MODEL RESPONSE QUALITY

- In some scenarios, the quality of the generated model responses might deteriorate after an update.
  Default sampling parameters source have been updated in newer versions. For vLLM version 0.8.4 and higher, the default sampling parameters come from the **generation_config.json** file that is provided by the model creator. In most cases, this should lead to higher quality responses, because the model creator is likely to know which sampling parameters are best for their model. However, in some cases the defaults provided by the model creator can lead to degraded performance.

  If you experience this problem, try serving the model with the old defaults by using the **--generation-config vllm** server argument.

IMPORTANT

If applying the **--generation-config vllm** server argument improves the model output, continue to use the vLLM defaults and petition the model creator on Hugging Face to update their default **generation_config.json** so that it produces better quality generations.

5.4. CUDA ACCELERATOR ERRORS

- You might experience a **self.graph.replay()** error when running a model using CUDA accelerators.
  If vLLM crashes and the error trace captures the error somewhere around the **self.graph.replay()** method in the **vllm/worker/model_runner.py** module, this is most likely a CUDA error that occurs inside the **CUDAGraph** class.

  To identify the particular CUDA operation that causes the error, add the **--enforce-eager** server argument to the **vllm** command line to disable **CUDAGraph** optimization and isolate the problematic CUDA operation.

- You might experience accelerator and CPU communication problems that are caused by incorrect hardware or driver settings.
  NVIDIA Fabric Manager is required for multi-GPU systems for some types of NVIDIA GPUs. The **nvidia-fabricmanager** package and associated systemd service might not be installed or the package might not be running.

  Run the diagnostic Python script to check whether the NVIDIA Collective Communications Library (NCCL) and Gloo library components are communicating correctly.

  On an NVIDIA system, check the fabric manager status by running the following command:

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

  On successfully configured systems, the service should be active and running with no errors.

- Running vLLM with tensor parallelism enabled and setting **--tensor-parallel-size** to be greater than 1 on NVIDIA Multi-Instance GPU (MIG) hardware causes an **AssertionError** during the initial model loading or shape checking phase. This typically occurs as one of the first errors when starting vLLM.

5.5. NETWORKING ERRORS

- You might experience network errors with complicated network configurations.
  To troubleshoot network issues, search the logs for DEBUG statements where an incorrect IP address is listed, for example:

  ```
  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
  ```

  To correct the issue, set the correct IP address with the **VLLM_HOST_IP** environment variable, for example:

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

Specify the network interface that is tied to the IP address for NCCL and Gloo:

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

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

5.6. PYTHON MULTIPROCESSING ERRORS

- You might experience Python multiprocessing warnings or runtime errors. This can be caused by code that is not properly structured for Python multiprocessing. The following is an example console warning:

```
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.
```

The following is an example Python runtime error:

```
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
```

To resolve the runtime error, update your Python code to guard the usage of **vllm** behind an **if__name__ = "__main__":** block, for example:

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

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

5.7. GPU DRIVER OR DEVICE PASS-THROUGH ISSUES

- When you run the Red Hat AI Inference Server container image, sometimes it is unclear whether device pass-through errors are being caused by GPU drivers or tools such as the NVIDIA Container Toolkit.
  - Check that the NVIDIA Container toolkit that is installed on the host machine can see the

host GPUs:

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

Example output

```
#...
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
```

- Ensure that the NVIDIA accelerator configuration has been created on the host machine:

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

- Check that the Red Hat AI Inference Server container can access NVIDIA GPUs on the host by running the following command:

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

Example output

```
+-----------------------------------------------------------------------------------------+
| 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                                              |
+-----------------------------------------------------------------------------+
```