Read list of GPU devices from volume mounts instead of `NVIDIA_VISIBLE_DEVICES`

Kevin Klues <kklues@nvidia.com>

Last Updated:
04-Sep-2020



Table of Contents

Overview 2

Changes to k8s-device-plugin 3

Changes to the nvidia-container-toolkit 4

Discussion 5

Overview

The NVIDIA container toolkit has traditionally used an environment variable (**`NVIDIA_VISIBLE_DEVICES`**) as the interface for telling the toolkit which GPUs it should inject into a container. On the surface, this seems like a reasonable choice because every container runtime supports the notion of injecting environment variables into a container. This keeps the toolkit runtime agnostic without requiring these runtimes to agree on a new, GPU specific API (i.e. it can run under docker, CRI-O, podman, containerd, etc. without defining a new API).

However, it becomes an issue once you plug this into Kubernetes and want to ensure that the **`k8s-device-plugin`** is the only mechanism by which pods are able to gain access to a GPU. As it stands today, a pod can bypass the resource allocation done by the **`k8s-device-plugin`** and gain access to any GPU on the system, by simply setting the **`NVIDIA_VISIBLE_DEVICES`** environment variable to whichever GPUs they want access to. In fact, all NVIDIA packaged containers (e.g. **`cuda:9.0-base`**, **`cuda:10.0-base`**, etc.) already have this environment variable set to "**`all`**" by default, granting these containers access to all GPUs on the system without ever going through the **`k8s-device-plugin`**.

Although this seems like a huge security hole, some valid use cases do exist where we want to grant containers access to all GPUs on the system without going through the **`k8s-device-plugin`**. Examples of such pods include the **`k8s-device-plugin`** itself, as well as containers that do any sort of monitoring of GPU devices. However, these pods differ from “normal” containers, in that they tend to run software critical to the overall Kubernetes infrastructure and should likely be considered privileged in one way or another.

As such, the following proposal defines a mechanism that allows us to close this security hole for *unprivileged* containers, while still allowing *privileged* containers to gain access to all GPUs on the system.

The basic idea is the following:

- Provide a configuration parameter on the NVIDIA container toolkit that forces a container to have **`CAP_SYS_ADMIN`** privileges in order to pass the GPU device list via

NVIDIA_VISIBLE_DEVICES. This allows us to continue supporting *privileged* containers without any change in semantics (e.g. for monitoring containers).

- Provide a configuration parameter on the NVIDIA container toolkit that allows one to pass the GPU device list as a series of **host volume mounts** rather than passing this list via **NVIDIA_VISIBLE_DEVICES**. Since any reasonable Kubernetes deployment disallows host volume mounts for unprivileged containers, this new mechanism prevents those containers from setting up the GPU device list themselves. Now, the **k8s-device-plugin** will be the only mechanism by which they will be able to gain access to a GPU.

We acknowledge that passing the device list as a series of **host volume mounts** is not the ideal interface for achieving the stated goal. However, no better mechanism exists at the moment that doesn't add a large amount of complexity to both the **k8s-device-plugin** (on one side) and the NVIDIA container toolkit (on the other).

To close this gap, we are currently in the process of defining exactly what the "right" way to communicate a set of devices to the container runtime should be.

CDI - The Container Device Interface

Once this spec is defined, we will begin to leverage it and all of the functionality we propose in this document will become obsolete. In the meantime, however, we encourage people to adopt the solution presented here if they wish to avoid having *unprivileged* pods bypass the **k8s-device-plugin** in their Kubernetes deployments.

Changes to k8s-device-plugin

Add a flag to the plugin called:

```
--device-list-strategy=<envvar | volume-mounts>
```

With this flag set to **envvar** we will continue to operate as before, passing the device list to the container runtime via the **NVIDIA_VISIBLE_DEVICES** environment variable.

With this flag set to **volume-mounts** we will:

- Set up a series of host volume mounts from:
  /dev/null → /var/run/nvidia-container-devices/<GPU-UUID>

- Set **NVIDIA_VISIBLE_DEVICES=/var/run/nvidia-container-devices** so that users know to look under this path for the full device list

An example of the mounts set up for 3 full GPUs and 3 MIG devices can be seen below:

```
root@aa4a3875e996:/# find /var/run/nvidia-container-devices/ -type f
/var/run/nvidia-container-devices/GPU-edfee158-11c1-52b8-0517-92f30e7fac88
/var/run/nvidia-container-devices/GPU-f22fb098-d1b3-3806-2655-ba25f02229c1
/var/run/nvidia-container-devices/GPU-f613f823-1032-b3ec-a876-50f2e35e6f9e
/var/run/nvidia-container-devices/MIG-GPU-82a956cb-4353-571f-83bf-4b9e76bf44d2/9/0
/var/run/nvidia-container-devices/MIG-GPU-82a956cb-4353-571f-83bf-4b9e76bf44d2/2/0
/var/run/nvidia-container-devices/MIG-GPU-82a956cb-4353-571f-83bf-4b9e76bf44d2/3/0
```

Changes to the `nvidia-container-toolkit`

1) Add a configuration parameter to only honour **`NVIDIA_VISIBLE_DEVICES`** on containers started with **`CAP_SYS_ADMIN`** privileges.

   Call this parameter:
   `accept-nvidia-visible-devices-envvar-when-unprivileged`

   Set its default value to **`true`** for backwards compatibility

   Note: We only look for this capability in the ***`bounding`*** capability set. This means a container must be started with these privileges, but the entry point of the container can be run as a different user without these privileges set.

2) Add a configuration parameter to receive the list of devices we should inject into a container under a specific folder inside the container

   Call this parameter:
   `accept-nvidia-visible-devices-as-volume-mounts`

   Set its default value to **`false`** for backwards compatibility

   The place to look for these volume mounts is then hard-coded as:
   `container_path = /var/run/nvidia-container-devices`

   With this parameter set, we will parse the OCI spec to see if there are any injected files under **`<container_path>`** and assume they are device UUIDs that we should use to build up the device list to pass to **`libnvidia-container`** instead of reading the **`NVIDIA_VISIBLE_DEVICES`** environment variable.

   If no such mounts exist and either:

1) The container is privileged, or
2) **`accept-nvidia-visible-devices-when-unprivileged == true`**

Then fallback to reading **`NVIDIA_VISIBLE_DEVICES`** to set the device list

These semantics imply that requesting GPUs via the pod spec (which will set up the mounts) always takes precedence over setting **`NVIDIA_VISIBLE_DEVICES`** (regardless of privileges).

This is how things operate today, and will continue to be the semantics going forward. If a privileged user wants to ensure that his setting of **`NVIDIA_VISIBLE_DEVICES`** is honored, he simply avoids requesting GPUs via the pod spec (i.e. don't set **`nvidia.com/gpu`** in the pod spec).

Discussion

The one caveat with this proposal is that a user could still bypass the Kubernetes resource model if **`hostPath`** volume mounts have *not* been disabled by an administrator in the cluster (they are enabled by default). A user would simply need to set up his own mounts under **`/var/run/nvidia-container-devices`** instead of relying on the device plugin to do it for him. That said, it is standard practice for an administrator to disable **`hostPath`** volume mounts, and any reasonable production deployment should already be doing this.

Another approach we considered to solve these problems relies on "inferring" the set of GPU devices to inject into a container by inspecting the set of **`/dev`** nodes being passed to the container. This is still something we could consider if people aren't happy with the current proposal, but it is much more complicated and potentially error prone. The current proposal simply maps what we are already doing today onto a set of volume mounts rather than an environment variable, making it much simpler and easier to verify its correctness. Given that both solutions are temporary workarounds until CDI is available has prompted us to lean on the side of simplicity. Please let us know if you disagree or would like to discuss further.