How to use GPUs with Device Plugin in OpenShift 3.11

November 15, 2018
By Zvonko Kaiser

Host Preparation

NVIDIA drivers for RHEL must be installed on the host as a prerequisite for using GPUs with OpenShift. Let's prepare the host by installing NVIDIA drivers and NVIDIA container enablement. The following procedures will make containerized GPU workloads possible in OpenShift, leveraging the Device Plugin feature in OpenShift 3.10.

Part 1: NVIDIA Driver Installation

NVIDIA drivers are compiled from source. The build process requires the kernel-devel package to be installed.

```
# yum -y install kernel-devel-`uname -r`
```

The `nvidia-driver` package requires DKMS package. DKMS is not supported or packaged by Red Hat. Work is underway to remove the NVIDIA driver requirement on DKMS for Red Hat distributions. DKMS can be installed from the EPEL repository.

First install the epel repository.

```
# yum install -y
https://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rpm
```

The newest NVIDIA drivers are located in the following repository.

```
# yum install -y
https://developer.download.nvidia.com/compute/cuda/repos/rhel7/x86_64/cuda-r
epo-rhel7-10.0.130-1.x86_64.rpm
```

Auxiliary tools and libraries are contained in the following packages. This will also install the nvidia-kmod package, which includes the NVIDIA kernel modules.

```
# yum -y install nvidia-driver nvidia-driver-cuda nvidia-modprobe
```

Remove the nouveau kernel module, (otherwise the nvidia kernel module will not load). The installation of the NVIDIA driver package will blacklist the driver in the kernel command line

(nouveau.modeset=0 rd.driver.blacklist=nouveau video=vesa:off), so that the nouveau driver will not be loaded on subsequent reboots.

```
# modprobe -r nouveau
```

Load the NVIDIA and the unified memory kernel modules.

```
# nvidia-modprobe && nvidia-modprobe -u
```

Verify the installation and the drivers are working. Extracting the name of the GPU can later be used to label the node in OpenShift.

```
# nvidia-smi --query-gpu=gpu_name --format=csv,noheader --id=0 | sed -e 's/ /-/g'
Tesla-V100-SXM2-16GB
```

Adding the nvidia-container-runtime-hook

The version of docker shipped by Red Hat includes support for OCI runtime hooks. Because of this, we only need to install the nvidia-container-runtime-hook package. On other distributions of docker, additional steps may be necessary. See NVIDIA's documentation for more information.

The next step is to install libnvidia-container and the nvidia-container-runtime repository.

```
# curl -s -L
https://nvidia.github.io/nvidia-container-runtime/centos7/nvidia-container-runtime.r
epo | tee /etc/yum.repos.d/nvidia-container-runtime.repo
```

The next step will install an OCI prestart hook. The prestart hook is responsible for making NVIDIA libraries and binaries available in a container (by bind-mounting them in from the host). Without the hook, users would have to include libraries and binaries into each and every container image that might use a GPU. Hooks simplify management of container images by ensuring only a single copy of libraries and binaries are required. The prestart hook is triggered by the presence of certain environment variables in the container: **`NVIDIA_DRIVER_CAPABILITES=compute,utility.`**

```
# yum -y install nvidia-container-runtime-hook
```

This package will install also the config/activation files for docker/podman/cri-o. Beware that the hook json in the package will only work with cri-o >= 1.12 for crio-1.11 use the following json file:

```
# cat <<'EOF' >> /usr/share/containers/oci/hooks.d/oci-nvidia-hook.json
{
   "hasbindmounts": true,
   "hook": "/usr/bin/nvidia-container-runtime-hook",
   "stage": [ "prestart" ]
}
EOF
```

Adding the SELinux policy module

To run NVIDIA containers contained and not privileged, we have to install an SELinux policy tailored for running CUDA GPU workloads. The policy creates a new SELinux type (`nvidia_container_t`) with which the container will be running.

Furthermore, we can drop all capabilities and prevent privilege escalation. See the invocation below to have a glimpse how to start a NVIDIA container.

First install the SELinux policy module on all GPU worker nodes:

```
# wget
https://raw.githubusercontent.com/zvonkok/origin-ci-gpu/master/selinux/nvidia-contai
ner.pp

# semodule -i nvidia-container.pp
```

Check and restore the labels on the node

The new SELinux policy heavily relies on the correct labeling of the host. Therefore we have to make sure that the files that are needed have the correct SELinux label.

1. Restorecon all files that the prestart hook will need

```
# nvidia-container-cli -k list | restorecon -v -f -
```

2. Restorecon all accessed devices

```
# restorecon -Rv /dev
```

3. Restorecon all files that the DevicePlugin will need

```
# restorecon -Rv /var/lib/kubelet
```

Everything is now set up for running a GPU-enabled container.

Verify SELinux and prestart hook functionality

To verify correct operation of driver and container enablement, try running a cuda-vector-add container. We can run the container with docker or podman.

```
# podman run  --user 1000:1000 --security-opt=no-new-privileges --cap-drop=ALL \
              --security-opt label=type:nvidia_container_t    \
  docker.io/mirrorgooglecontainers/cuda-vector-add:v0.1
```

```
# docker run  --user 1000:1000 --security-opt=no-new-privileges --cap-drop=ALL \
              --security-opt label=type:nvidia_container_t    \
  docker.io/mirrorgooglecontainers/cuda-vector-add:v0.1
```

If the test passes, the drivers, hooks and the container runtime are functioning correctly and we can move on to configuring OpenShift.

Part 2: OpenShift 3.11 with the GPU Device Plugin

After successful installation of OpenShift 3.11, the first step is to install the NVIDIA Device Plugin. To schedule the Device Plugin on nodes that include GPUs, label the node as follows:

```
# oc label node <node-with-gpu> openshift.com/gpu-accelerator=true
node "<node-with-gpu>" labeled
```

This label will be used in the next step.

Deploy the NVIDIA Device Plugin Daemonset

The next step is to deploy the NVIDIA Device Plugin Note that the NVIDIA Device Plugin (and more generally, any hardware manufacturer's plugin) is supported by the vendor, and is not shipped or supported by Red Hat.

Clone the following repository, there are several yaml files for future use.

```
# git clone https://github.com/redhat-performance/openshift-psap.git
# cd openshift-psap/blog/gpu/device-plugin
```

Here is an example daemonset (`device-plugin/nvidia-device-plugin.yml`) which will use the label we created in the last step so that the plugin pods will only run where GPU hardware is available.

Now create the NVIDIA Device Plugin daemonset.

```
# oc create -f nvidia-device-plugin.yml
```

Lets verify the correct execution of the Device Plugin. You can see there is only one running, since only one node was labeled in the previous step.

```
# oc get pods -n kube-system
NAME                                   READY     STATUS     RESTARTS   AGE
nvidia-device-plugin-daemonset-s9ngg   1/1       Running    0          1m
```

Once the pod is running, let's have a look at the logs.

```
# oc logs nvidia-device-plugin-daemonset-7tvb6 -n kube-system
2018/07/12 12:29:38 Loading NVML
2018/07/12 12:29:38 Fetching devices.
2018/07/12 12:29:38 Starting FS watcher.
2018/07/12 12:29:38 Starting OS watcher.
2018/07/12 12:29:38 Starting to serve on /var/lib/kubelet/device-plugins/nvidia.sock
2018/07/12 12:29:38 Registered device plugin with Kubelet
```

At this point the node itself will advertise the nvidia.com/gpu [extended resource](#) in it's capacity:

```
# oc describe node <gpu-node> | egrep 'Capacity|Allocatable|gpu'

Capacity:
 nvidia.com/gpu:  2
Allocatable:
 nvidia.com/gpu:  2
```

Nodes that do not have GPUs installed will not advertise GPU capacity.

Deploy a pod that requires a GPU

Let's run a GPU-enabled container on the cluster. We can use the cuda-vector-add image that was used in the Host Preparation step. Use the following file (`device-plugin/cuda-vector-add.yaml`) as a pod description for running the

cuda-vector-add image in OpenShift. Note the last line requests one NVIDIA GPU from OpenShift. The OpenShift scheduler will see this and schedule the pod to a node that has a free GPU. Once the pod create request arrives at a node, the Kubelet will coordinate with the Device Plugin to start the pod with a GPU resource.

First create a project that will group all of our GPU work.

```
# oc new-project nvidia
```

The next step is to create and start the pod.

```
# oc create -f cuda-vector-add.yaml
```

After a couple of seconds the container finishes.

```
# oc get pods
NAME                                   READY     STATUS            RESTARTS   AGE
cuda-vector-add                        0/1       Completed         0          3s
nvidia-device-plugin-daemonset-s9ngg   1/1       Running           0          9m
```

Let's have a look at the logs for any errors. We are looking for Test PASSED in the pod logs.

```
# oc logs cuda-vector-add
[Vector addition of 50000 elements]
Copy input data from the host memory to the CUDA device
CUDA kernel launch with 196 blocks of 256 threads
Copy output data from the CUDA device to the host memory
Test PASSED
Done
```

This output is the same as when we ran the container directly using podman or docker. If you see a permission denied error, check to see that you have the correct SELinux label.

Troubleshooting

SELinux

The following table shows the correct labels of the files needed to have a working GPU container.

File	SELinux label
`/dev/nvidia*`	`xserver_misc_device_t`
`/usr/bin/nvidia-*`	`xserver_exec_t`
`/var/lib/kubelet/*/*`	`container_file_t`