WSL2 Üzerinde Minikube + Kubernetes Ortamında HAMi (vGPU) Denemesi: Yaptıklarım, Teşhis Süreci, Bulgular ve Mimari Çatışmalar

ysf

25 Aralık 2025

1 Amaç ve Kapsam

Bu raporda, WSL2 üzerinde kurulu Minikube/Kubernetes ortamımda HAMi (vGPU) kullanarak GPU paylaşımı yapmayı denerken karşılaştığım problemi; attığım adımları; girdiğim komutları ve aldığım çıktıları; ulaştığım teknik bulguları ve özellikle WSL2 üzerinde HAMi çalıştırmanın getirdiği zorlukları birinci ağızdan özetliyorum.

Odak problem: HAMi ile GPU limiti vermeye çalıştığım pod içinde `nvidia-smi` komutunun **segmentation fault** (exit code 139) ile çökmesi ve bunun arkasındaki mimari nedenlerin anlaşılması.

2 Ortam Bilgisi (Gözlemsel)

Bu çalışma sırasında gözlemlediğim yığın ve bileşenler:

- WSL2 tabanlı Linux ortamı. Kernel loglarında `microsoft-standard-WSL2` ibaresi bulunuyor.
- Minikube üzerinde Kubernetes.
- GPU: `NVIDIA GeForce GTX 1650`.
- Test için kullandığım konteyner imajı: `nvidia/cuda:12.2.0-base-ubuntu22.04`.
- `nvidia-smi` çıktısında driver ve CUDA sürümü (pod içinde gözlem): `Driver Version: 591.59`, `CUDA Version: 13.1`.

3 Problem Tanımı

HAMi ile oluşturduğum pod içinde `nvidia-smi` çalıştırmak istediğimde süreç çöküyor ve aşağıdaki gibi exit code 139 dönüyor:

```
kubectl get pods
NAME READY STATUS RESTARTS AGE
gpu-final-test 1/1 Running 0 14m
hami-retry-pod 1/1 Running 0 16s

kubectl exec -it hami-retry-pod -- nvidia-smi
command terminated with exit code 139
```

Exit code 139 pratikte SIGSEGV (segmentation fault) anlamına geliyor: yani `nvidia-smi` süreç olarak başlıyor ancak bir noktada bellek erişim ihlali ile çöküyor.

4 Teşhis Süreci: Girdi/Çıktılar ve İncelemeler

4.1 1) Pod içinde `nvidia-smi` bağımlılıklarını kontrol ettim (`ldd`)

İlk olarak pod içinde `nvidia-smi` dosyasının nerede olduğunu, sürüm bilgisini ve dinamik kütüphane bağımlılıklarını kontrol ettim:

```
kubectl exec -it hami-retry-pod -- sh -lc \
'which nvidia-smi; nvidia-smi --version || true; ldd $(which nvidia-smi) || true'

/usr/bin/nvidia-smi
Segmentation fault (core dumped)
        linux-vdso.so.1 (...)
        /usr/local/vgpu/libvgpu.so (...)
        ...
        libcuda.so.1 => /usr/lib/x86_64-linux-gnu/libcuda.so.1 (...)
        /lib64/ld-linux-x86-64.so.2 (...)
```

Burada kritik gözlemim şuydu:

- `nvidia-smi` çalışır çalışmaz çöküyor (Segmentation fault).
- Dinamik bağımlılıklar arasında `/usr/local/vgpu/libvgpu.so` görünüyor.

Bu, HAMi'nin bir *hook/intercept* yaklaşımı ile bazı çağrıları araya girerek kontrol ettiğini düşündürdü; çünkü klasik CUDA pod'larında bu kütüphane görünmüyordu.

4.2 2) Kıyas için "normal GPU pod" üzerinde `nvidia-smi` çalıştırdım

Aynı cluster içinde bir kıyas pod'u (`gpu-final-test`) vardı. Burada `nvidia-smi` sorunsuz çalıştı:

```
kubectl exec -it gpu-final-test -- sh -lc 'nvidia-smi; echo "exit=$?"'
Thu Dec 25 16:02:25 2025
+-----------------------------------------------------------------------------------------+

| NVIDIA-SMI 590.48.01 Driver Version: 591.59 CUDA Version: 13.1 |
...
exit=0
```

Ayrıca bu pod içinde `/usr/local/vgpu` klasörü yoktu ve `ldd` çıktısında `libvgpu.so` görünmüyordu:

```
kubectl exec -it gpu-final-test -- sh -lc '
echo "== ldd nvidia-smi =="; ldd /usr/bin/nvidia-smi || true;
echo "== vgpu =="; ls -lah /usr/local/vgpu 2>/dev/null || echo "no /usr/local/vgpu";
'
== ldd nvidia-smi ==
        linux-vdso.so.1 (...)
        libpthread.so.0 => /usr/lib/x86_64-linux-gnu/libpthread.so.0 (...)
        ...
== vgpu ==
no /usr/local/vgpu
```

Bu kıyas, sorunun "GPU tamamen bozuk" olmadığını; daha çok HAMi'nin devreye aldığı mekanizmayla ilgili olduğunu gösterdi.

4.3 3) Kernel logları: WSL crash capture ve segfault kayıtlarını aldım

Segfault detayını kernel loglarında aradım:

```
minikube ssh -- 'sudo dmesg -T | tail -n 200 | egrep -i "segfault|nvidia|gpu|uvm|nvml|
    xid"'

[Thu Dec 25 15:09:00 2025] nvidia-smi[45126]: segfault at ... in libc.so.6 ...
[Thu Dec 25 15:09:00 2025] CPU: ... Comm: nvidia-smi ...
[Thu Dec 25 15:09:00 2025] ... microsoft-standard-WSL2 ...
```

Burada iki önemli nokta vardı:

- Segfault kernel tarafında doğrulanıyordu.
- Loglarda WSL2'nin crash-capture mekanizması görünüyordu; yani olay WSL2 çekirdeği üstünde gerçekleşiyordu.

4.4 4) `/etc/ld.so.preload` üzerinden zorunlu preload'u doğruladım

En kritik doğrulama: `LD_PRELOAD` boş olmasına rağmen, `/etc/ld.so.preload` dosyası üzerinden `libvgpu.so` zorunlu yükleniyordu.

```
kubectl exec -it hami-retry-pod -- sh -lc '
echo "LD_PRELOAD=$LD_PRELOAD";
echo "== /etc/ld.so.preload ==";
ls -l /etc/ld.so.preload 2>/dev/null || true;
cat /etc/ld.so.preload 2>/dev/null || echo "(none)";
'
LD_PRELOAD=
== /etc/ld.so.preload ==
-rw-r--r-- 1 root root 26 Dec 25 15:52 /etc/ld.so.preload
/usr/local/vgpu/libvgpu.so
```

Benim yorumum: Bu ayar, dinamik bağlayıcıya her process başlatımında ilgili kütüphaneyi öncelikle yükletiyor. Dolayısıyla `nvidia-smi` dahil her binary, bu kütüphane tarafından etkilenebiliyor.

4.5 5) `LD_DEBUG=libs` ile kitaplık yükleme zincirini izledim

Sorunun "hangi kütüphaneyi yüklerken" patladığını anlamak için dinamik yükleyici debug çıktısına baktım. Çıktının kritik kısmı:

```
LD_DEBUG=libs nvidia-smi 2>&1 | head -n 120
...
calling init: /usr/local/vgpu/libvgpu.so
...
find library=libnvidia-ml.so.1 [0]; searching
 trying file=/usr/lib/x86_64-linux-gnu/libnvidia-ml.so.1
 calling init: /usr/lib/x86_64-linux-gnu/libnvidia-ml.so.1
...
calling init: /usr/lib/wsl/drivers/.../libnvidia-ml.so.1
Segmentation fault (core dumped)
```

Benim açımdan burada mimari açıdan çok önemli bir durum vardı:

- HAMi `libvgpu.so` init oluyor.
- Sonrasında NVML (`libnvidia-ml.so.1`) yükleniyor.
- WSL2'ye özgü `/usr/lib/wsl/...` benzeri bir yoldan başka bir `libnvidia-ml.so.1` daha init oluyor.
- Ardından segfault gerçekleşiyor.

Bu zincir, WSL2'nin sürücü/kitaplık yerleşiminin native Linux'tan farklı olmasının ve HAMi'nin hook mekanizmasının aynı süreçte birleşmesinin stabilite riski oluşturduğunu düşündürdü.

4.6 6) Kubernetes tarafında iki farklı GPU device plugin'in birlikte çalıştığını tespit ettim

Cluster'da aynı anda hem HAMi device plugin hem NVIDIA device plugin çalışıyordu:

```
kubectl get ds -A | egrep -i "nvidia|device-plugin|hami"
kube-system hami-device-plugin 1 1 1 1 ...
kube-system nvidia-device-plugin-daemonset 1 1 1 1 ...
```

Benim yorumum: Tek node'lu Minikube senaryosunda iki farklı device plugin'in aynı anda GPU kaynak ilanı (`nvidia.com/gpu`) ve container mount/ENV düzenini etkilemesi çakışma riskini artırıyor. Bu noktada özellikle WSL2'nin kendine özgü sürücü kitaplığı düzeniyle birleşince hata ayıklama daha da zorlaşıyor.

4.7 7) DaemonSet ölçekleme (`kubectl scale`) denemesi ve sonuç

NVIDIA device plugin'i devre dışı bırakmak için `scale` denedim; ancak şu hatayı aldım:

```
kubectl -n kube-system scale daemonset.apps/nvidia-device-plugin-daemonset --replicas=0
Error from server (NotFound): the server could not find the requested resource
```

Sonradan anladığım nokta: DaemonSet'ler Deployment gibi `replicas` ile ölçeklenmediği ve `scale` alt kaynağını desteklemediği için bu komut NotFound hatası verebiliyor.

4.8 8) NVIDIA device plugin DaemonSet'i sildim ve pod'u yeniden oluşturdum

Çakışmayı azaltmak için NVIDIA device plugin'i sildim:

```
kubectl -n kube-system delete daemonset nvidia-device-plugin-daemonset
daemonset.apps "nvidia-device-plugin-daemonset" deleted from kube-system namespace
```

Ardından test pod'umu (HAMi ile) yeniden oluşturmak için kendi yazdığım pod manifestini kullandım:

```
cat hami-retry.yaml
apiVersion: v1
kind: Pod
metadata:
  name: hami-retry-pod
  annotations:
    scheduling.k8s.io/group-name: "hami-group"
spec:
  schedulerName: hami-scheduler
  containers:
    - name: cuda-container
      image: nvidia/cuda:12.2.0-base-ubuntu22.04
      command: ["bash", "-c", "sleep 86400"]
      resources:
        limits:
          nvidia.com/gpu: 1
          nvidia.com/gpumem: 2000
```

Uygulama sonucu:

```
kubectl apply -f hami-retry.yaml
pod/hami-retry-pod created
```

Bu aşamadan sonra pod durumunda iki farklı hata gözledim:

```
kubectl get pods -w
hami-retry-pod 0/1 UnexpectedAdmissionError 0 5s
```

Daha sonra:

```
kubectl get pods -w
hami-retry-pod 0/1 OutOfnvidia.com/gpu 0 6s
```

Benim yorumum:

- `UnexpectedAdmissionError` admission webhook katmanında (mutating/validating) reddedilme olasılığını düşündürdü.
- `OutOfnvidia.com/gpu` ise cluster'da `nvidia.com/gpu` kaynağının allocatable olarak görülmemesi veya yeterli olmaması anlamına geldi.
- NVIDIA device plugin'i kaldırdıktan sonra `nvidia.com/gpu` kaynağını kimin/ nasıl expose ettiği konusu kritik hale geldi.

5 Bulguların Özeti (Ne Öğrendim?)

Bu çalışma sonucunda şu net bulgulara ulaştım:

1. GPU sürücüsü/erişimi temel seviyede çalışıyor: "normal" GPU pod'unda (`gpu-final-test`) `nvidia-smi` başarılı (exit 0).
2. Sadece HAMi ile ilişkili pod'da `nvidia-smi` segfault üretiyor (exit 139).
3. HAMi tarafında `/etc/ld.so.preload` ile `/usr/local/vgpu/libvgpu.so` zorunlu preload ediliyor.
4. Dinamik yükleyici debug çıktısında NVML (`libnvidia-ml`) tarafında WSL2'ye özgü kütüphane yolu devreye giriyor ve segfault bu zincirde tetikleniyor.
5. Kubernetes tarafında aynı anda iki GPU device plugin (HAMi + NVIDIA) çalışırken çakışma riski yüksek; kaldırma sonrası bu sefer kaynak ilanı/scheduling hataları (`OutOfnvidia.com/gpu`) gündeme geldi.

6 WSL2 Üzerinde HAMi Çalıştırmanın Zorlukları: Mimari Çatışma Analizi

Benim deneyimime göre WSL2 üzerinde HAMi çalıştırmayı zorlaştıran başlıca mimari noktalar şunlar oldu:

6.1 1) WSL2 GPU sürücü/kitaplık yerleşimi native Linux'tan farklı

WSL2'de CUDA çalıştırma modeli Windows tarafındaki NVIDIA sürücüsü ile WSL entegrasyonuna dayanıyor. Bu, native Linux'taki standart "kernel modülü + klasik `/usr/lib` düzeni" beklentilerinden farklı bir kitaplık yükleme dinamiği oluşturabiliyor (özellikle NVML tarafında). Ben bunu `LD_DEBUG=libs` çıktısında `/usr/lib/wsl/...` benzeri yolun devreye girmesinden doğrudan gözlemledim.

6.2 2) HAMi'nin `libvgpu.so` preload/hook yaklaşımı süreç başlangıcını kırılganlaştırıyor

HAMi'nin kullandığı yaklaşımda `libvgpu.so` `/etc/ld.so.preload` ile süreçlere zorla enjekte edildiği için, `nvidia-smi` gibi yardımcı araçlar bile bu hook katmanının etkisi altına giriyor. Benim senaryomda bu, `nvidia-smi` çalışır çalışmaz segfault ile sonuçlandı.

6.3 3) Kubernetes device plugin katmanı ile birden fazla bileşenin aynı kaynağı yönetmesi çakışma üretiyor

Benim cluster'ımda hem HAMi device plugin hem NVIDIA device plugin aynı anda çalışıyordu. Tek node'lu Minikube üzerinde bu durum:

- GPU kaynağının (`nvidia.com/gpu`) ilanı,
- container'lara yapılan mount/ENV enjeksiyonları,
- scheduling davranışı

gibi alanlarda çakışmayı kolaylaştırdı. Bu çakışmayı azaltmak için NVIDIA device plugin'i kaldırdığımda, bu sefer kaynak ilanı/scheduling tarafında farklı problemler (`OutOfnvidia.com/gpu`) ortaya çıktı. Yani WSL2 + Minikube + vGPU hedefi birleşince hata ayıklama döngüsü uzadı.

7 Sonuç ve Yeni Hedef

Bu denemede, WSL2 üzerinde Minikube/Kubernetes ortamında HAMi ile vGPU paylaşımı hedeflediğim senaryoda:

- HAMi'nin preload ettiği `libvgpu.so` ile,
- WSL2'nin GPU kitaplık yükleme düzeni (NVML dahil) ve
- Kubernetes device plugin/scheduling katmanının

aynı anda devreye girmesinin stabiliteyi ciddi biçimde zorlaştırdığını gördüm.

Bu nedenle, bu noktadan sonra WSL2 üzerinde devam etmeme kararı aldım. **Yeni hedefim: aynı çalışmayı native Linux (bare-metal veya tam Linux VM) üzerinde tekrar kurarak HAMi davranışını daha deterministik bir ortamda yeniden doğrulamak ve orada ilerlemek.**

A Ek A: Bu Çalışmada Kullandığım Temel Komutlar

Amaç	Komut
Pod listesini görmek	`kubectl get pods`
Pod içinde komut çalıştırmak	`kubectl exec -it <pod> - <cmd>`
`nvidia-smi` çalıştırmak	`kubectl exec -it <pod> - nvidia-smi`
Bağımlılık analizi	`kubectl exec -it <pod> - ldd /usr/bin/nvidia-smi`
Kernel log inceleme	`minikube ssh - 'sudo dmesg -T | ...'`
Preload kontrolü	`kubectl exec -it <pod> - cat /etc/ld.so.preload`
Dinamik yükleyici debug	`kubectl exec -it <pod> - sh -lc 'LD_DEBUG=libs nvidia-smi'`
DaemonSet listeleme	`kubectl get ds -A`
DaemonSet silme	`kubectl -n kube-system delete ds <name>`
Pod manifest uygulama	`kubectl apply -f <file.yaml>`

B Kaynaklar

Kaynaklar

[1] Project HAMi Resmi Dokümantasyon, https://project-hami.io/docs/.

[2] HAMi "Build" Dokümantasyonu (bileşenler ve build çıktıları), https://project-hami.io/docs/developers/build/.

[3] Koordinator: "Device Scheduling - GPU Share With HAMi" (HAMi-core/libvgpu hakkında genel açıklamalar), https://koordinator.sh/docs/next/user-manuals/device-scheduling-gpu-share-with-hami.

[4] Project-HAMi/HAMi Issue örneği: `/etc/ld.so.preload` ve `libvgpu.so` ile ilgili problemler, https://github.com/Project-HAMi/HAMi/issues/1346.

[5] Project-HAMi/HAMi Issue örneği: `CUDA_DISABLE_CONTROL` davranışı ve `/etc/ld.so.preload` ilişkisi, https://github.com/Project-HAMi/HAMi/issues/1050.

[6] NVIDIA Kubernetes Device Plugin (DaemonSet olarak GPU'ları node'larda expose eder), https://github.com/NVIDIA/k8s-device-plugin.

[7] NVIDIA NGC: Kubernetes Device Plugin container açıklaması, https://catalog.ngc.nvidia.com/orgs/nvidia/containers/k8s-device-plugin.

[8] NVIDIA "CUDA on WSL User Guide", https://docs.nvidia.com/cuda/wsl-user-guide/index.html.

[9] Microsoft: "Enable NVIDIA CUDA on WSL 2", https://learn.microsoft.com/en-us/windows/ai/directml/gpu-cuda-in-wsl.

[10] `ld.so(8)` Linux man page (dinamik bağlayıcı ve yükleme davranışı), https://man7.org/linux/man-pages/man8/ld.so.8.html.

[11] `/etc/ld.so.preload` açıklaması (dinamik yükleyiciye ek kütüphane yükletme), https://superuser.com/questions/1183037/what-is-does-ld-so-preload-do.

[12] Kubernetes Resmi Doküman: Device Plugins konsepti, https://kubernetes.io/docs/concepts/extend-kubernetes/compute-storage-net/device-plugins/.

[13] DaemonSet'lerde `kubectl scale` NotFound örnek tartışma (scale subresource/replicas ilişkisi), https://stackoverflow.com/questions/60927311/how-to-scale-daemon-set-about-kubernetes-using-kubectl.

[14] Kubernetes Issue: Scale subresource olmayan nesnelerin "scalable" olmaması (DaemonSet/HPA bağlamı), https://github.com/kubernetes/kubernetes/issues/48588.

[15] Kubernetes Resmi Doküman: DaemonSet, https://kubernetes.io/docs/concepts/workloads/controllers/daemonset/.

[16] NVIDIA Developer Forums: WSL2 üzerinde `nvidia-smi` yolu ve WSL kütüphane yerleşimi tartışması, https://forums.developer.nvidia.com/t/nvidia-smi-through-wsl2/180310.