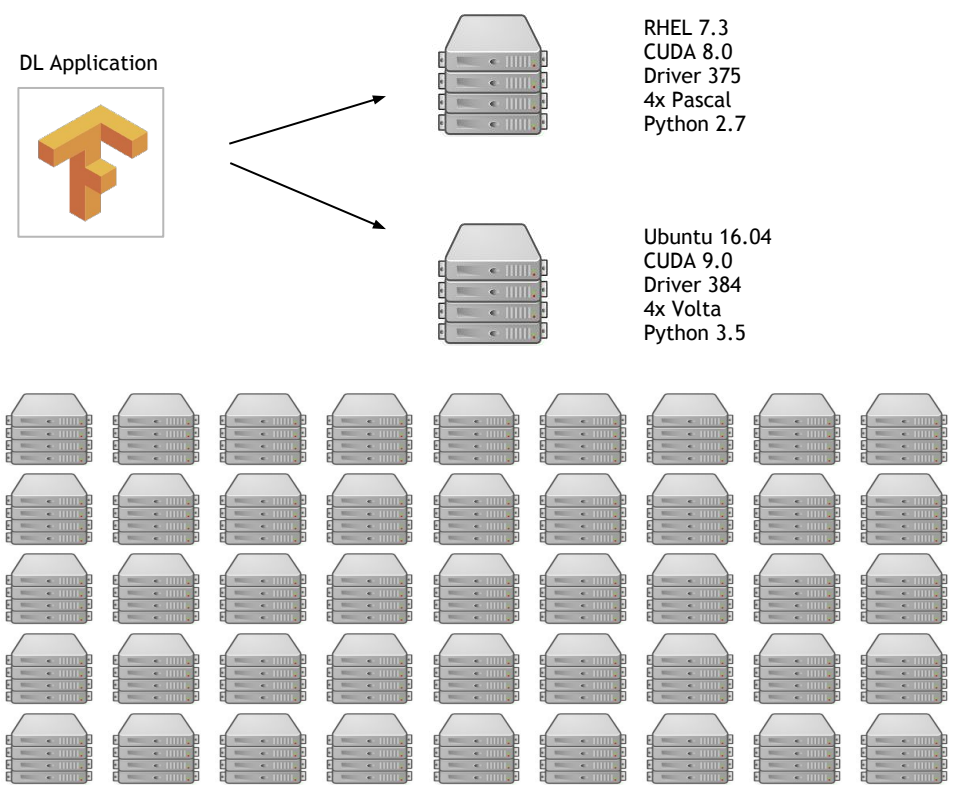
The Path to GPU as a Service in Kubernetes

Renaud Gaubert <@RenaudWasTaken>, Lead Kubernetes Engineer

May 03, 2018

RUNNING A GPU APPLICATION

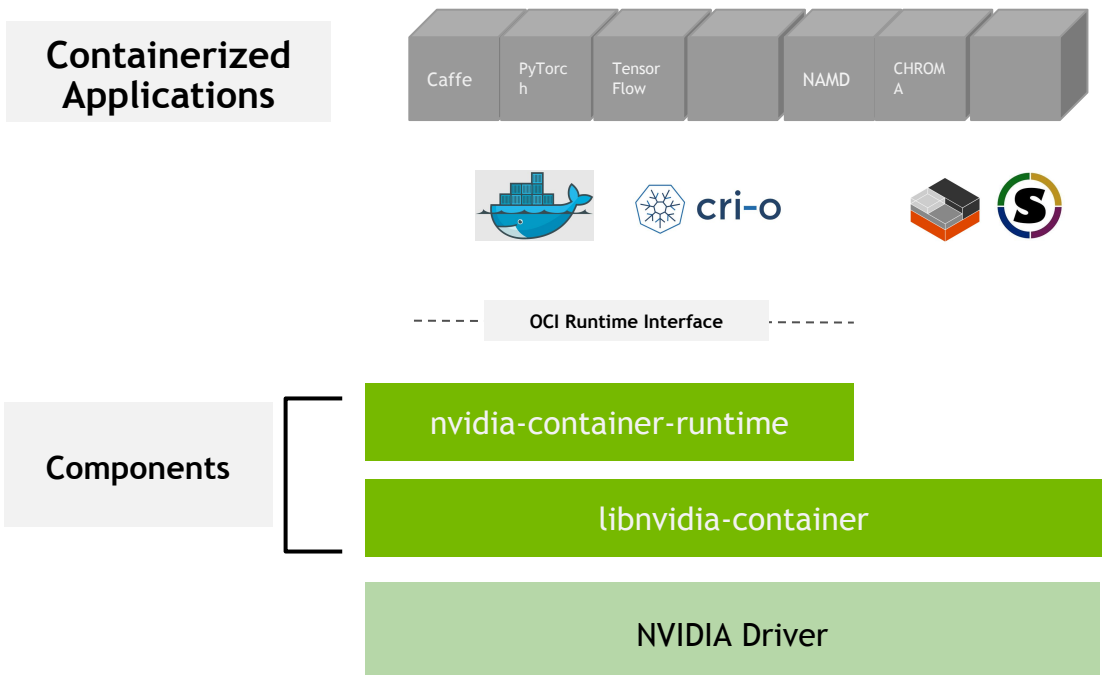
Customers using DL



- “This framework requires installing 6 dependencies from sources”
- “I want to train my model on the cluster but it’s running RHEL 7”
- “Some machines in the cluster have different NVIDIA hardware & drivers”
- “How do I deploy a DL model/application at scale”
- “How do I deploy a fault tolerant inference service”

NVIDIA CONTAINER RUNTIME

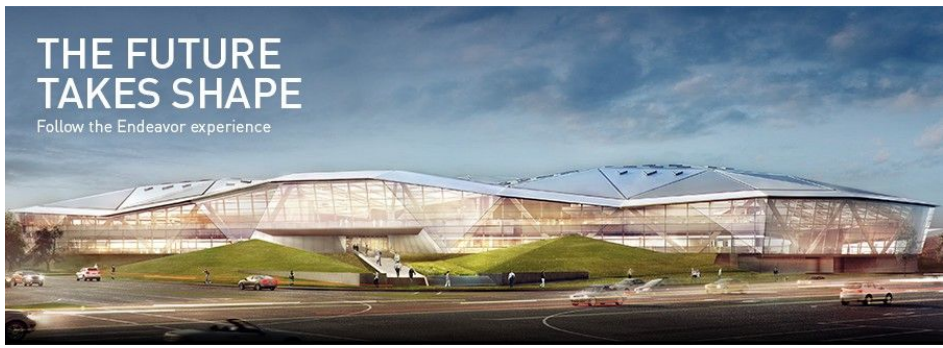
Enables GPU support in various container runtimes



- Integrates Linux container internals instead of wrapping specific runtimes (e.g. Docker)
- Includes runtime library, headers, CLI tools
- Backward compatibility with NVIDIA-Docker 1.0
- Support new use-cases - HPC, DL, Graphics

Involvement in the Community

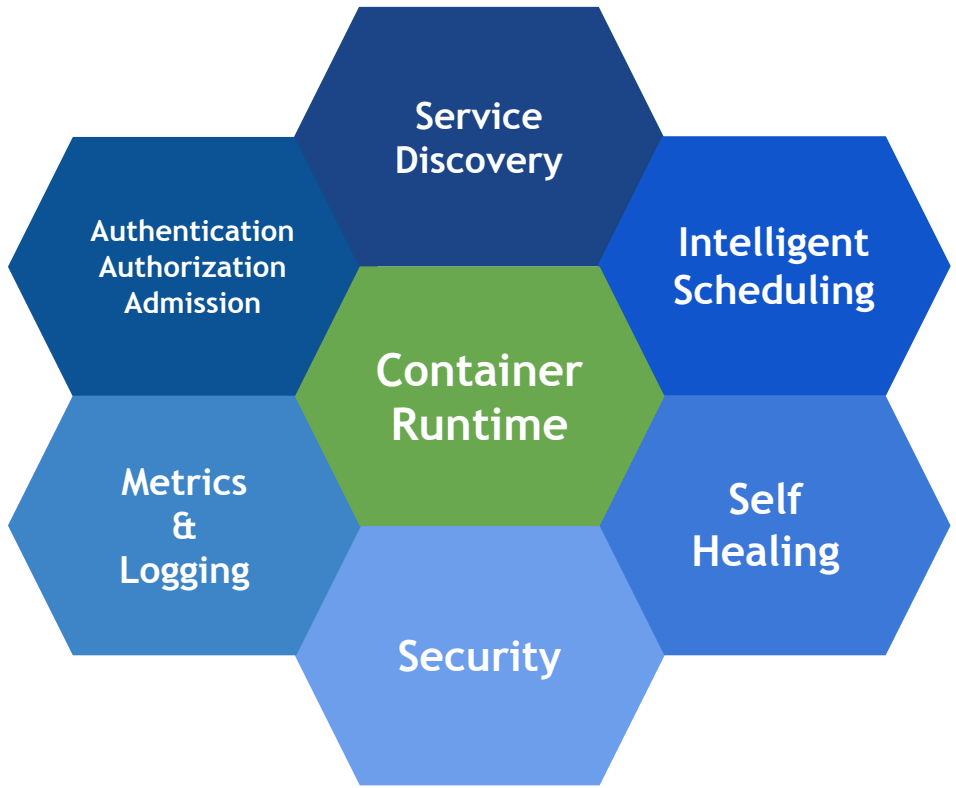
- **February 2017:** Involvement in the community discussions
- **Spring 2017**: Face 2 Face meeting
- **Summer 2017**: Design doc merged in Summer
- **Kubernetes 1.8**: Alpha Feature available
- **Kubernetes 1.10:** Beta Feature available
- **Spring 2018**: Face to Face meeting hosted at NVIDIA



CONTAINER ORCHESTRATION: KUBERNETES

Kubernetes is a portable, extensible open-source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation

- Supports auto scaling, self-healing, multi-region clusters
- Upstream GPU support still beta and provisioning a GPU cluster can be challenging



KUBERNETES ROADMAP FOR NVIDIA GPUs

Alpha GPU Support (--accelerators)

- ✓ Experimental support (1.6 supports one GPU / node)
- ✓ Manually mount the volumes in your pod spec
- ✓ No GPU Monitoring or Health check

Device Plugin system

- ✓ NVIDIA Device Plugin (alpha)
- ✓ Uses new NVIDIA Container Runtime

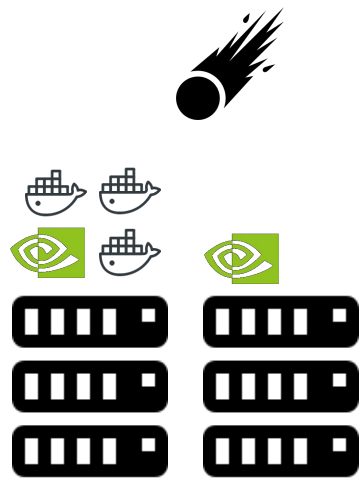
GPU Enhancements

- ✓ GPU Health Checks and Monitoring
- ✓ Heterogeneous GPU Support
- ✓ Support for MPS
- ✓ GPU Topology aware
- ✓ Device Plugin support for GPU lifecycle management
- ✓ Representing resources in the cluster

Kubernetes User Perspective

Motivations and Opportunity for Growth

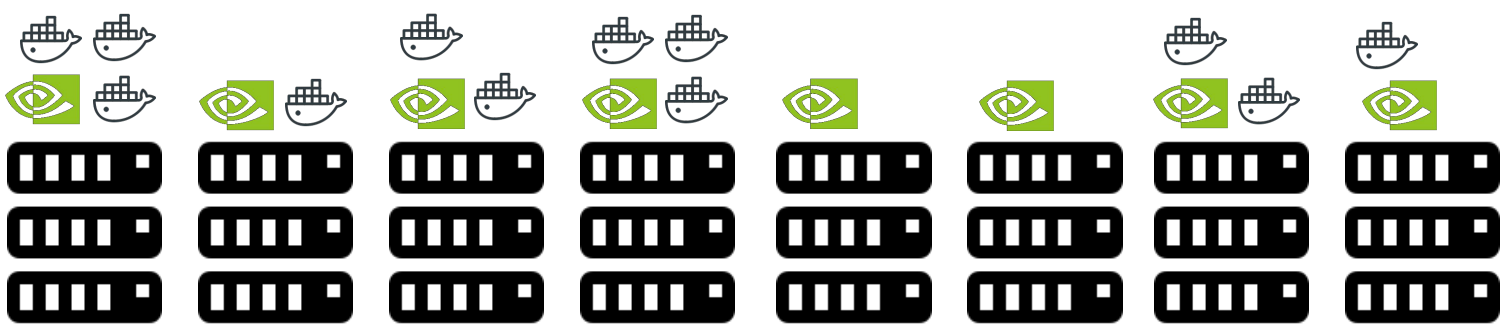
- In 1.6 - 1.7 Kubernetes had experimental GPU support
  - 1.6 supports one GPU per node
  - Manually mount the driver volumes
- Fragmented ecosystem for GPU support
- No GPU metrics or health checks



NVIDIA Device Plugin

- In 1.8 we introduced the device plugin system
  - You deploy a Daemonset in your cluster for it to be GPU aware
- Reports to the cluster and setup the GPU resources
- Exposes the GPU resource inside your containers

```
$ kubectl create -f raw.github/.../NVIDIA/.../v1.10/device-plugin.yml
```



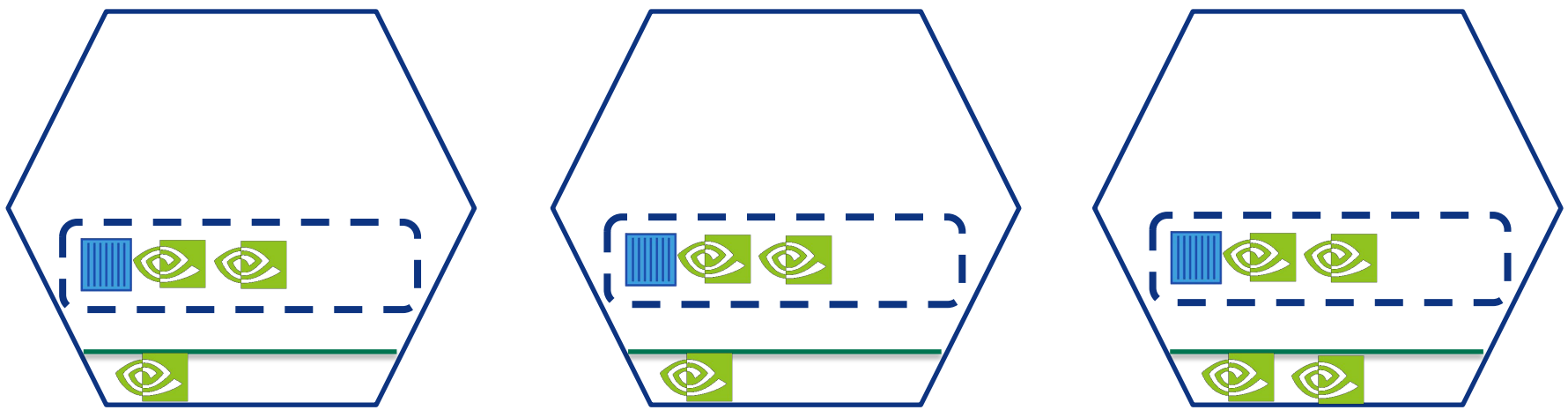
GPU Pod

```
apiVersion: v1
kind: Pod
metadata:
  name: gpu-pod
spec:
  containers:
    - name: digits-container
      image: nvidia/digits:6.0
      resources:
        limits:
          nvidia.com/gpu: 2 # requesting 2 GPUs
```

Updating for GPU Services

Good luck doing your blue green update!

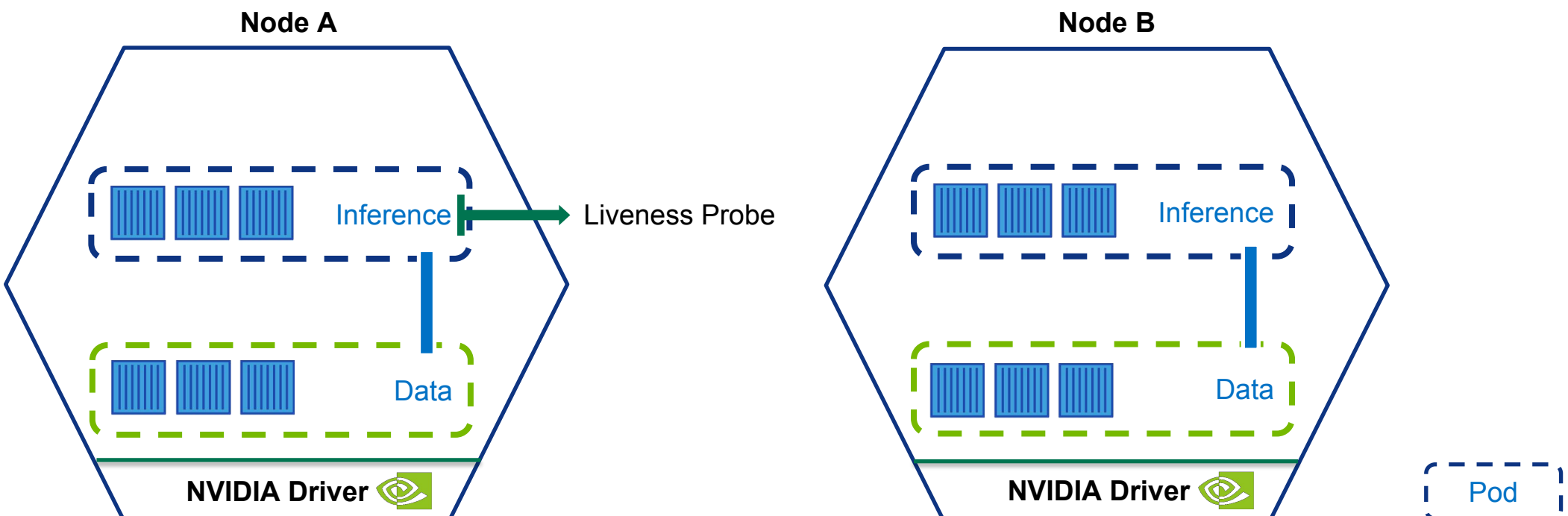
```
$ kubectl edit deployment/gpu-deployment
```



Resiliency for GPU Services

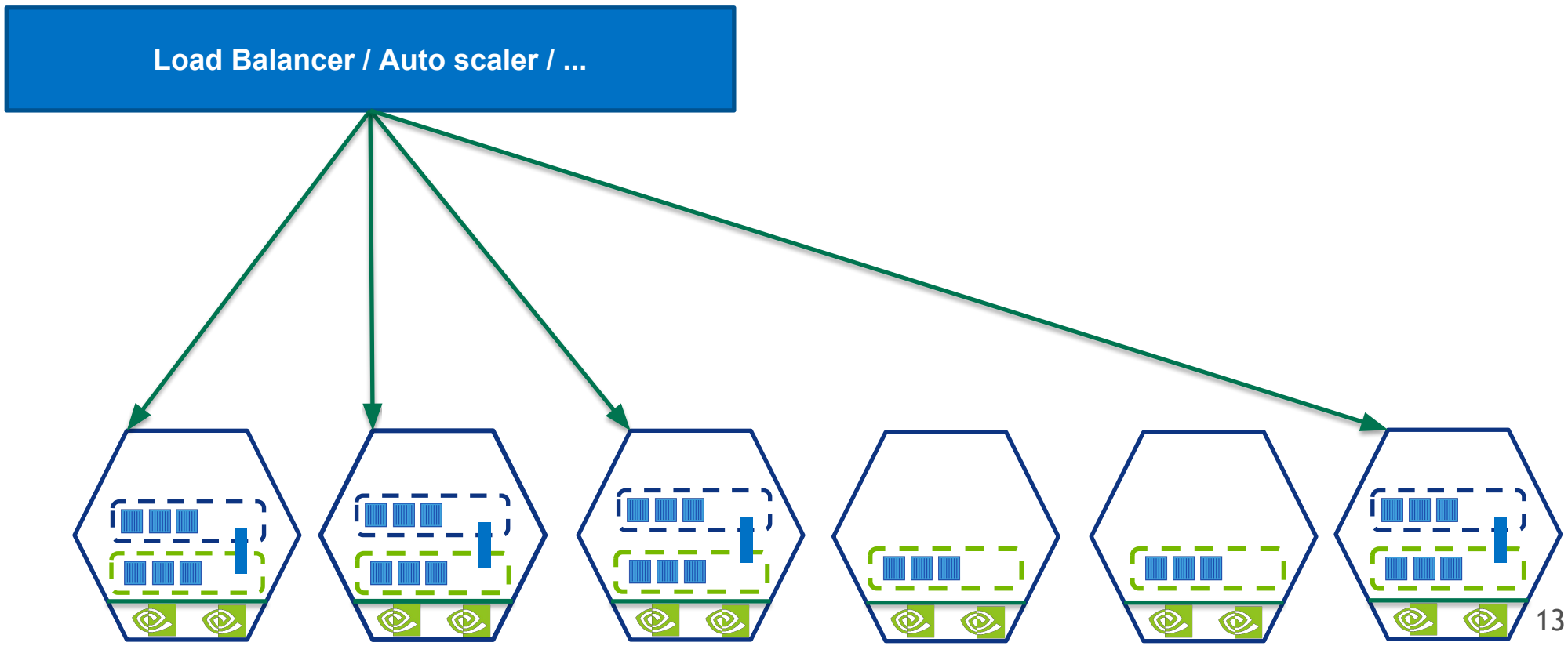
When deploying an inference service some things are slow:

- Provisioning a GPU instance
- Loading the data



Scaling for GPU Services

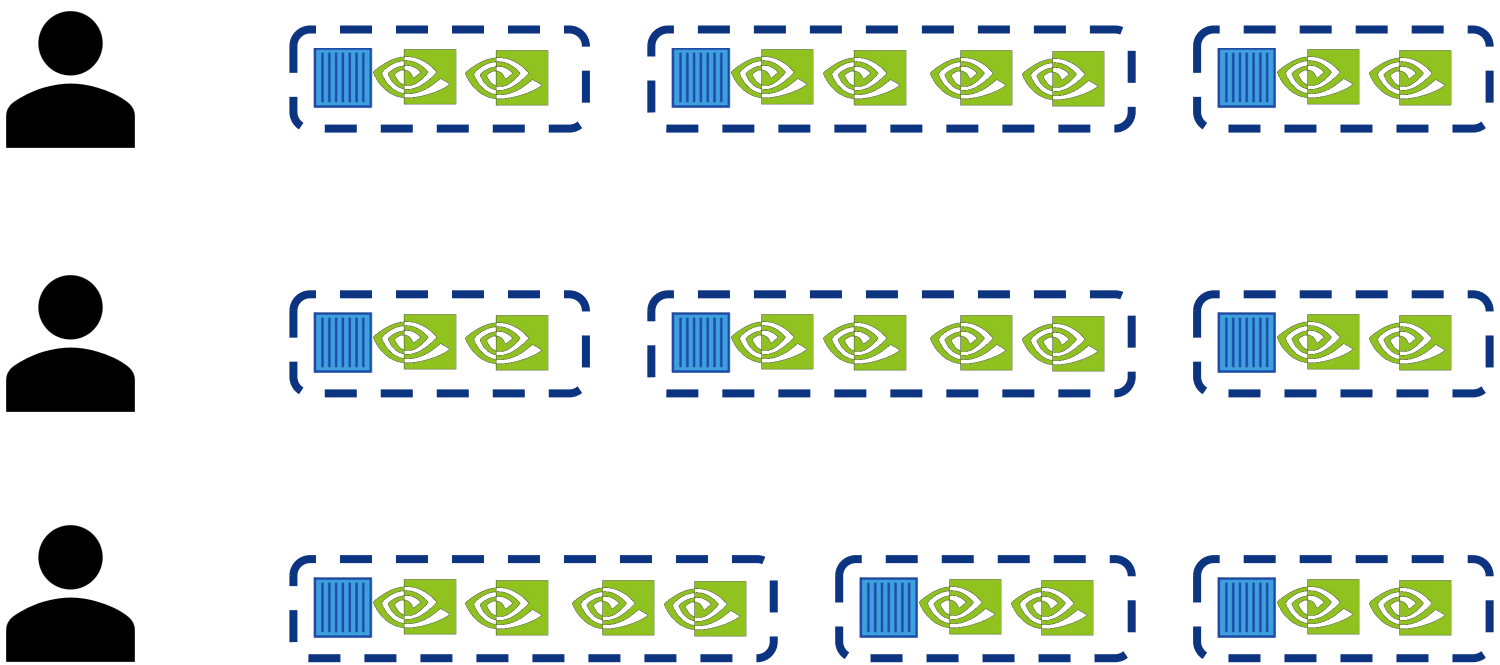
- Scale based on QPS not GPU load
- Use Inter-pod affinity to land on nodes that have your data pod
- Keep some spare nodes



Operating a GPU Cluster

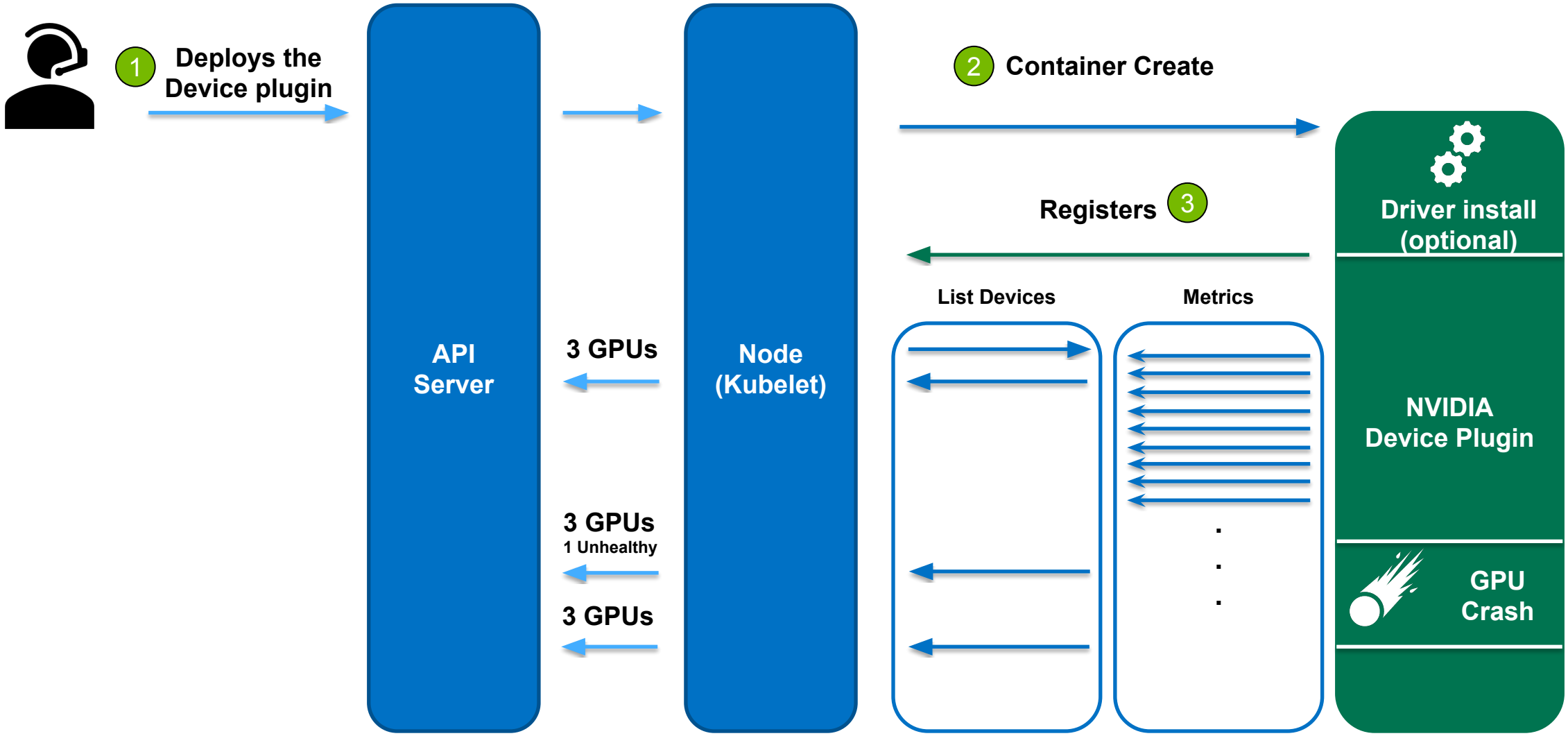
In order User queues

Possible Solutions: Use an operator? A custom scheduler?

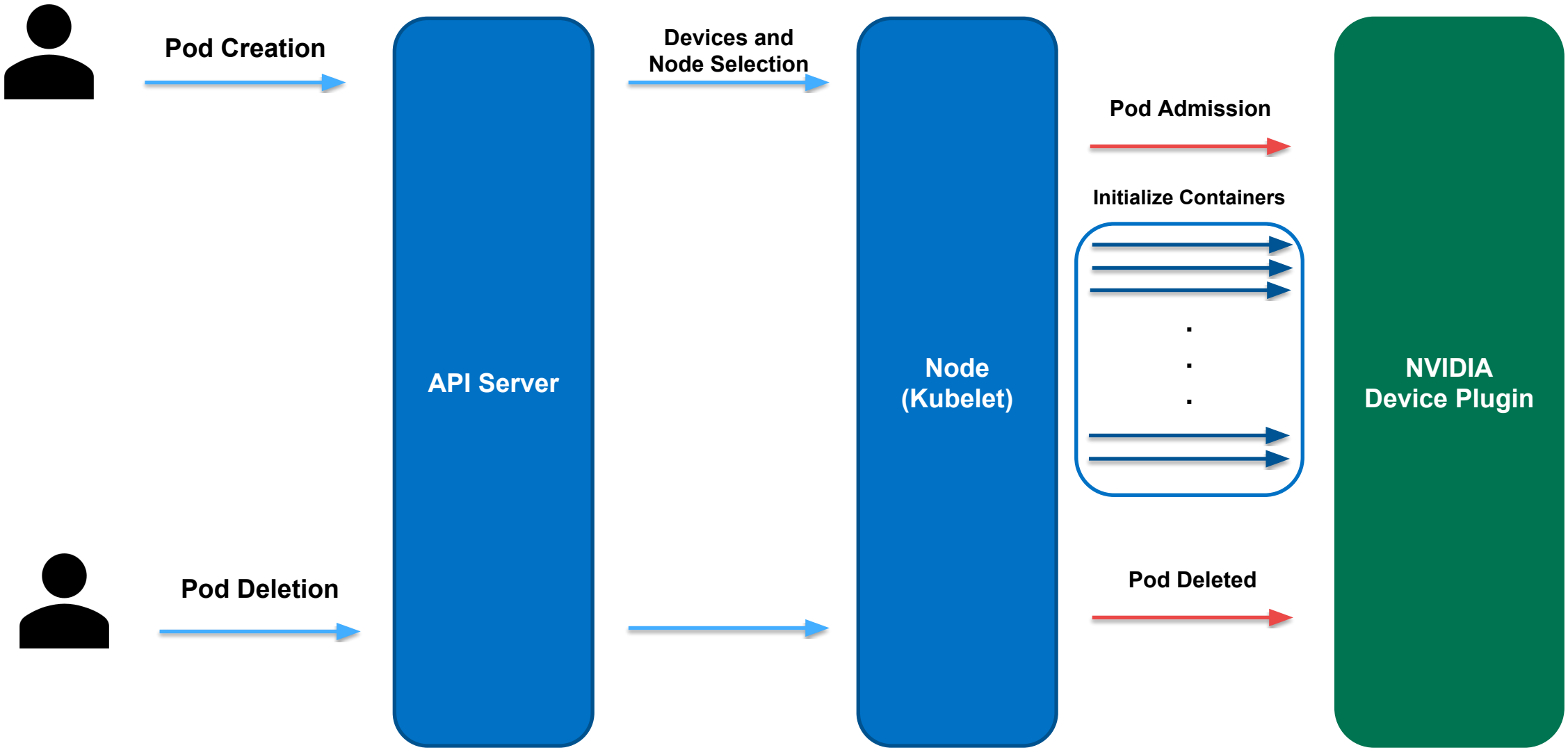


Kubernetes Developer Perspective

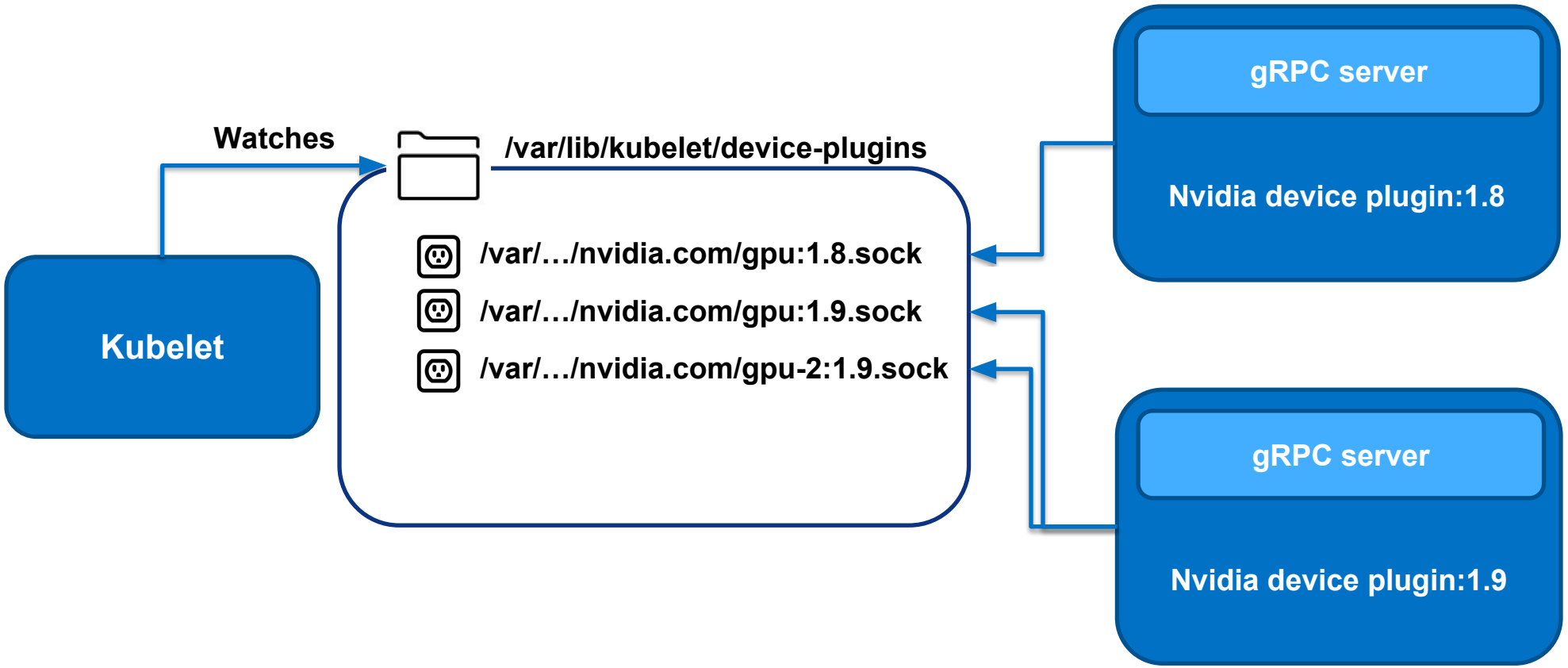
Device Plugin Lifecycle



Device Plugin Lifecycle



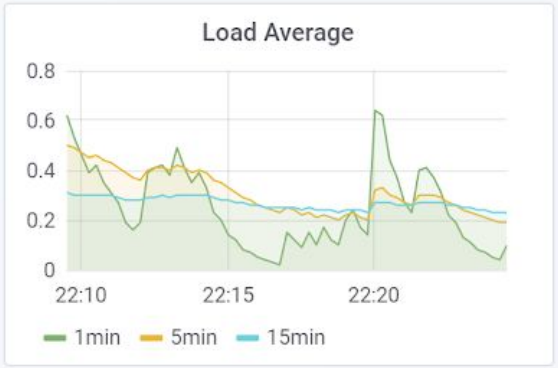
Zero Downtime Registration

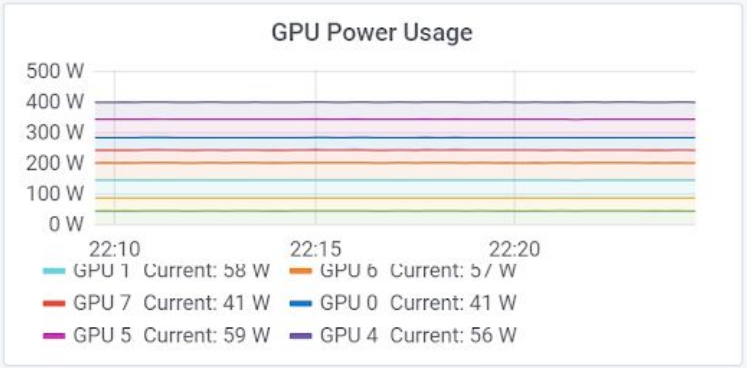


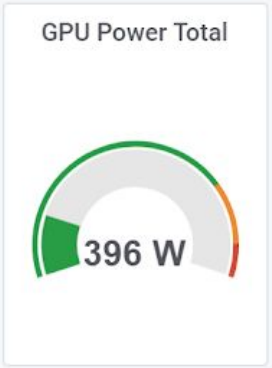
Current Challenges and Challenges Ahead

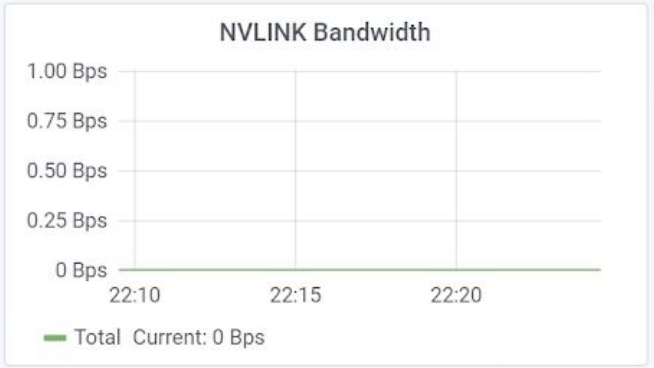
GPU Monitoring

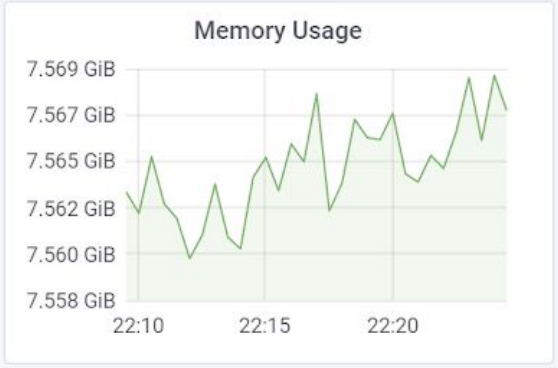
Upstream	Enhancements
Reports basic metrics (GPU utilization, Memory usage)	• Identify badly behaved tasks and unhealthy GPUs • Detect power inefficiencies • Identify bottlenecks and throttling

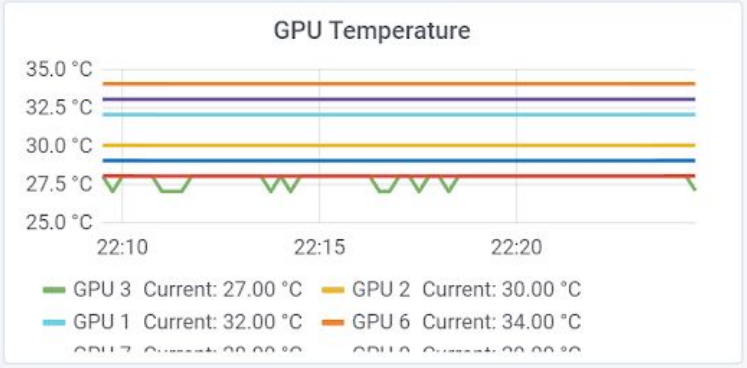


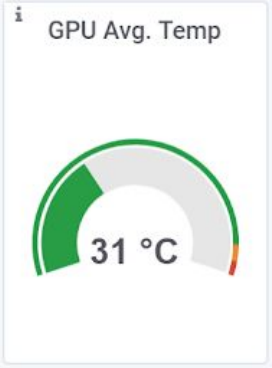


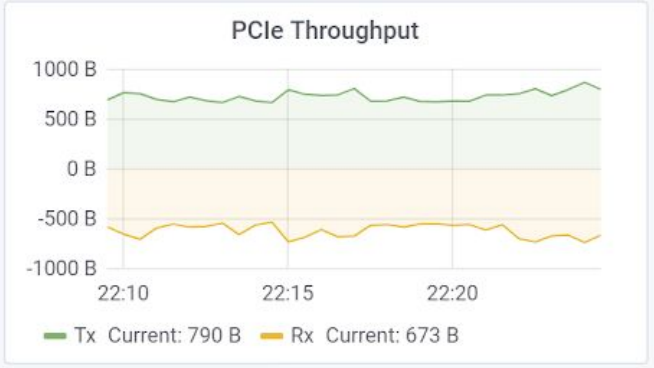












Current Challenges

In Core Kubernetes

Feature	Upstream	Enhancements
Heterogeneous Cluster Support	Manually label nodes with GPU attributes Taints, Tolerations, …	Specify GPU requirements for your pod (Minimum Memory, NVLINK connection, ECC Enabled, …)
Additional Runtimes Support	Docker and CRIO were not fully supported: • The NVIDIA runtime is used for all images • Images that did not request GPUs might have all GPUs exposed	Full Docker and CRIO support

Challenges Ahead

In Core Kubernetes

Feature	Enhancements
GPU Sharing	• Support for MPS • Make use of the full Volta capabilities for inference services • How is this exposed to the user?
GPU Topology Aware	• Users are exposed to more and more complex topologies • Make sure that users get the best performance possible • How is this exposed to the user?

Technical Challenges Ahead

For GPUs in Containers

At NVIDIA there are still a lot of challenges that we look forward to solve:

- Attach GPUs to containers dynamically
- Different Architectures (ARM, PowerPC, …)
- Container OS
- Virtualization
- Graphics, video encoding, …
- Support new GPUs and new GPU features
- …

Thank You!

Renaud Gaubert