KubeCon | CloudNativeCon

North America 2023

KubeCon | CloudNativeCon

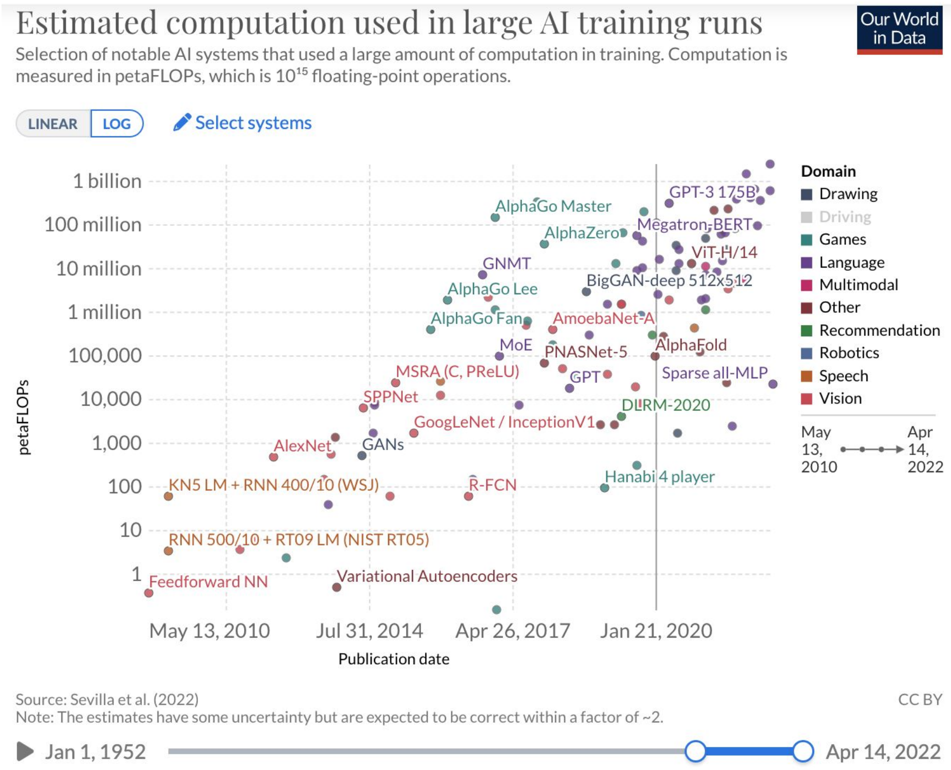
North America 2023

The Hidden Heroes Behind AI: Making Sense of GPUs and TPUs in K8s

David Porter, Google & Evan Lezar, NVIDIA

Intro

- Devices and accelerators are becoming increasingly important to serve new types of workloads.
- We need accelerator devices for all of our new ML models!
  - GPUs, TPUs, FPGAs, etc
- We will cover:
  - How kubernetes integrates with devices
  - How device plugin and device allocation works
  - GPUs & TPUs on k8s
  - Operating clusters with GPUs/TPUs
  - Future of devices in k8s



Devices in k8s

- What is a **D**evice?
  - Something a user wants access to for a specific purpose
  - A collection of device nodes, libraries, and utilities that are required to use the device

- Exposed as countable extended resources in Kubernetes

- Require per-node Device Plugins

Devices in k8s

- A Device Plugin:
  - Registers a resource name with the Kubelet (e.g. `nvidia.com/gpu`)
  - Lists opaque IDs of allocatable devices and provides allocation hints
  - On allocation returns required edits to the container spec
    - Device Nodes
    - Mounts
    - Environment Variables
    - Annotations
    - **CDI Device names** (alpha in 1.28+, beta in 1.29+)

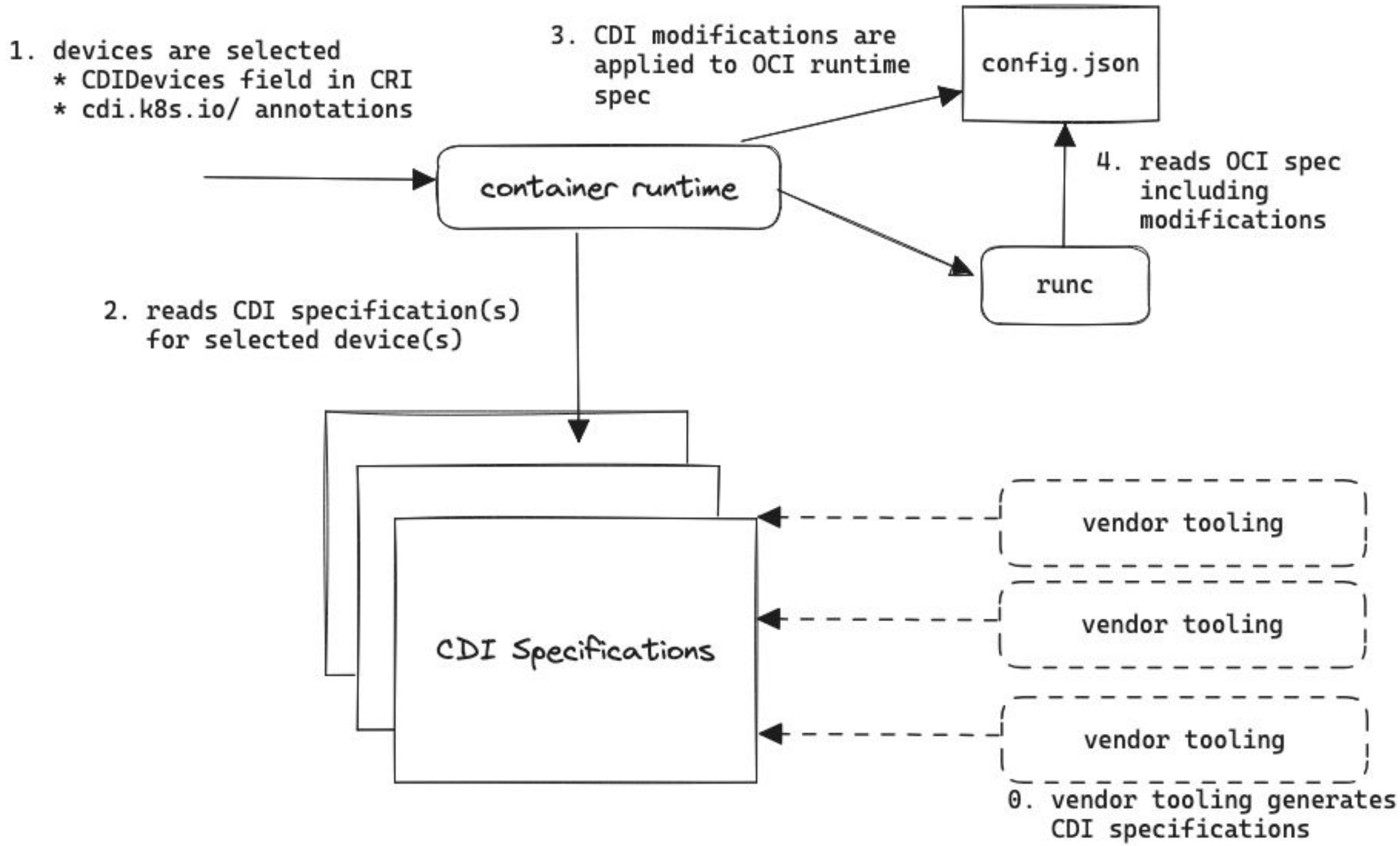
An aside on CDI

- The Container Device Interface (CDI)
  - A CNCF-sponsored project under TAG Runtime
  - A declarative specification for defining what a **D**evice is
    - device nodes, mounts, environments variables, hooks
    - maps to OCI runtime spec modifications

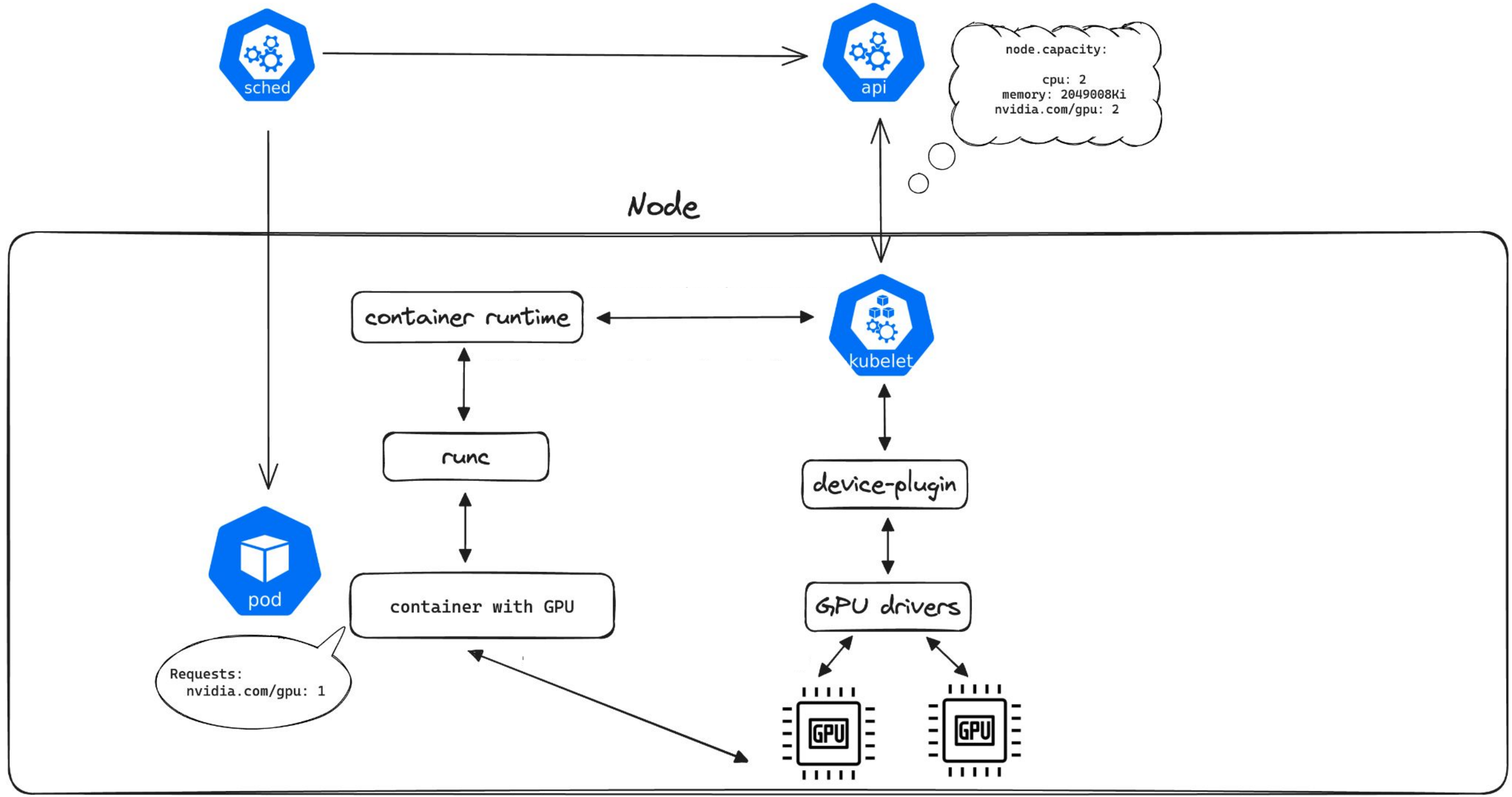
- A CDI **D**evice has a locally-unique fully-qualified device name
  `{{vendor}}/{{class}}={{name}}` → **`nvidia.com/gpu=GPU-1234`**

- The CRI includes `CDIDevice` field in `ContainerConfig` message (since v0.27.0)

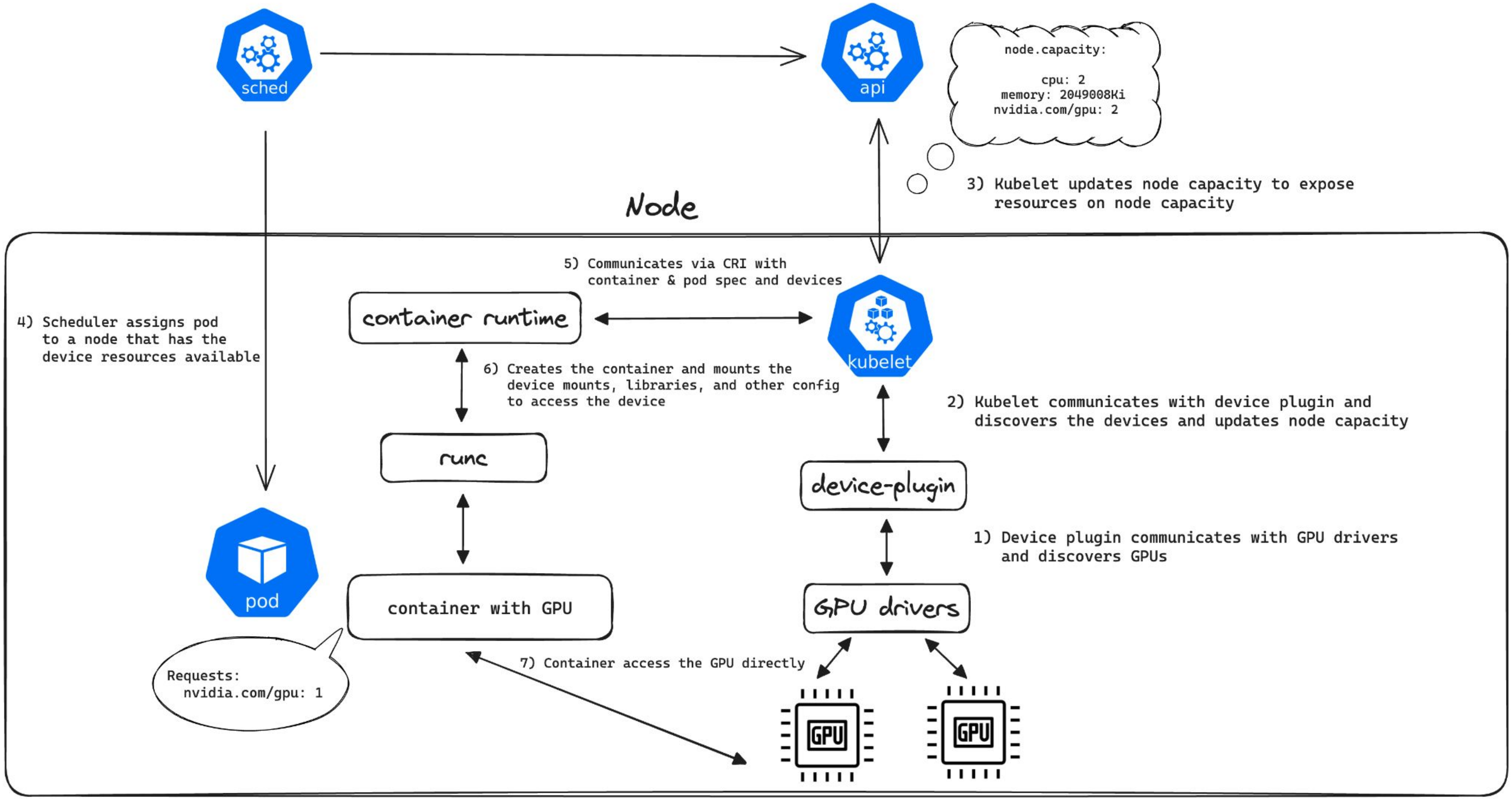
An aside on CDI (workflow)



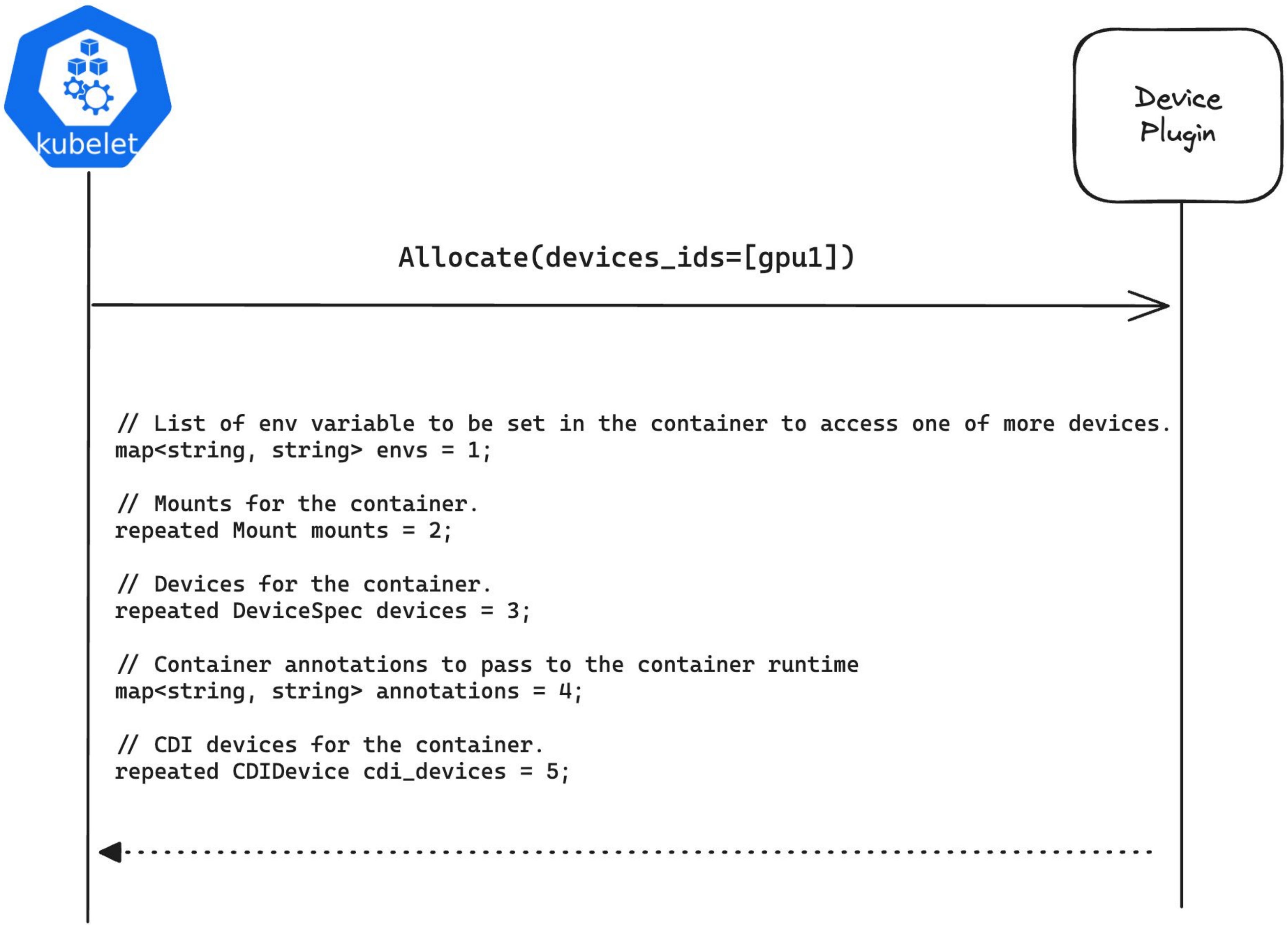
Overall workflow



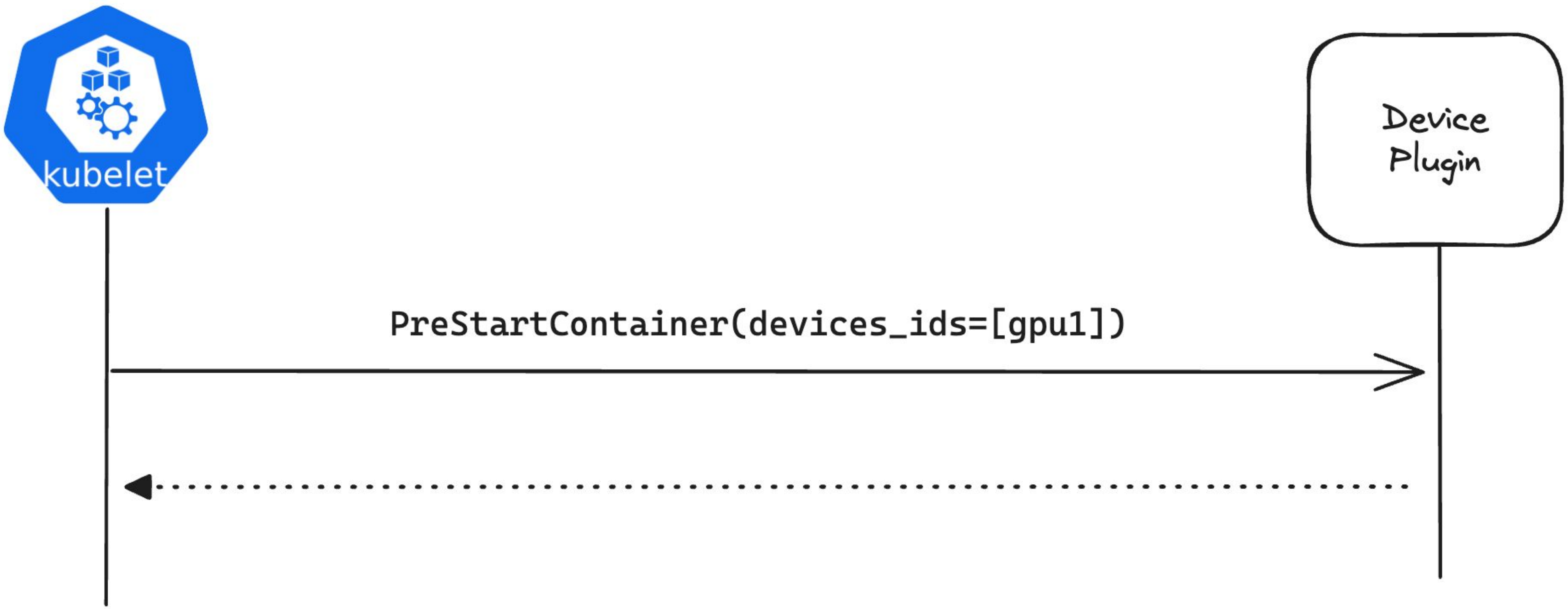
Overall workflow



Device Allocation

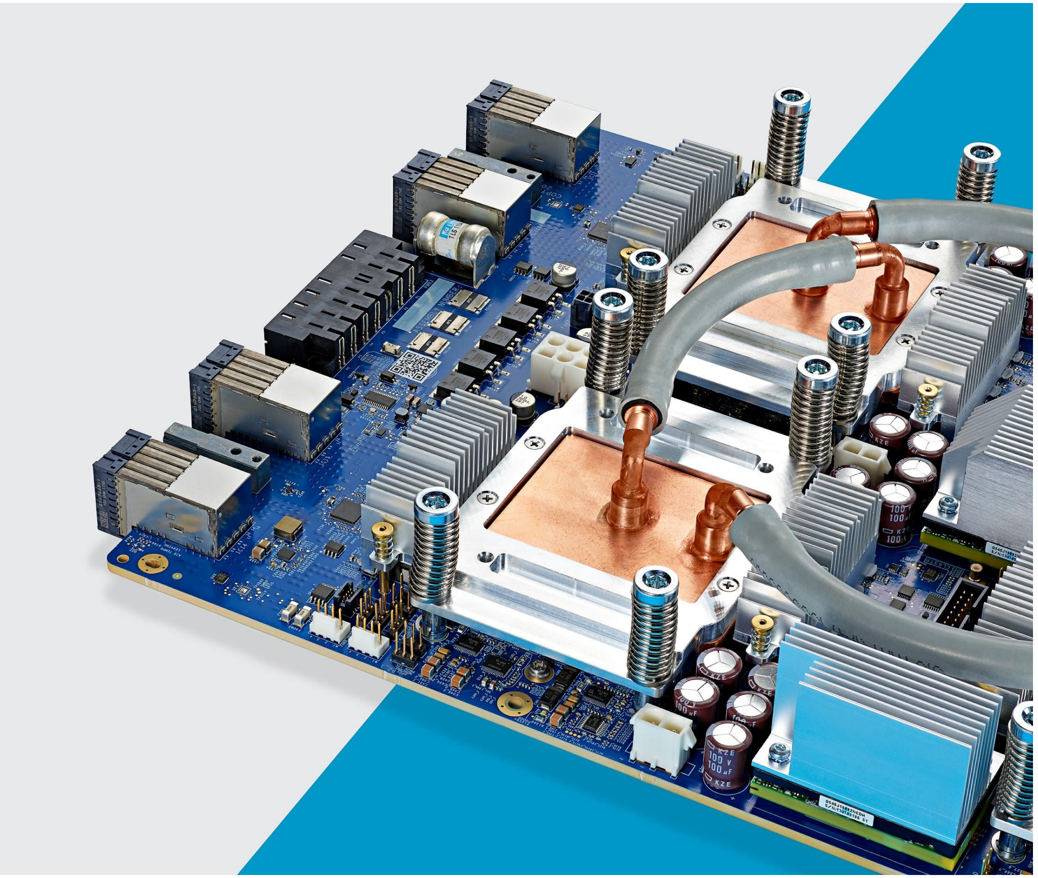


Device Start



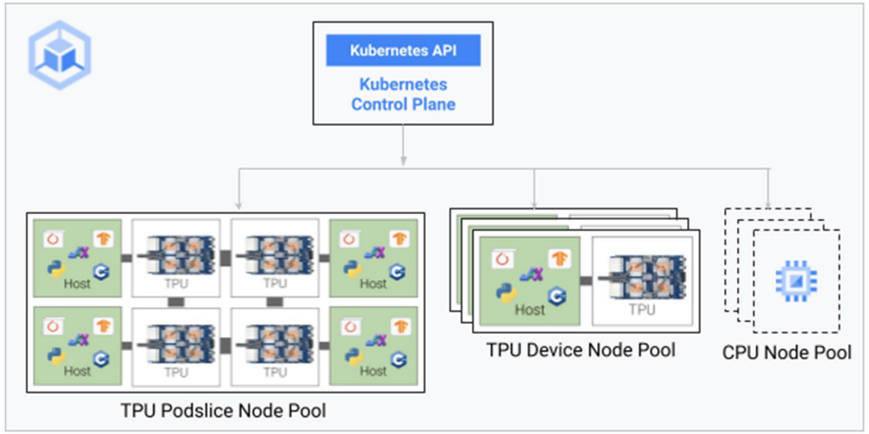
Introducing TPUs

- Special built accelerators for inference and training
- Optimized for training and inference of large AI models, including LLMs and GenAI models
- Two flavors of TPUs
  - TPU device - independent device (No special network connections to other nodes/TPUs)
  - TPU Slices - Groups of TPUs boards linked together with high-speed interconnect (ICI links) across multiple VMs
- Supports existing common ML frameworks
  - Pytorch & JAX/Tensorflow via XLA (compiler framework)



TPUs in GKE

- Kubernetes helps and manage many TPU workloads spread large cluster of TPU machines
- GKE developed a TPU device plugin to expose TPUs as a k8s resource to pods
  - `requests = { google.com/tpu: N }`
- Scheduling
  - TPU device - similar to GPUs (1:1 mapping from container to TPU)
  - TPU slices - require using a Job to create N pods per TPU node; each pod requests all TPUs on the node.
    - Gang scheduling concept; all pods and TPU must be up and running at same time
- Software stack: libtpu.so; xla; and [tensorflow/pytorch/jax]



GKE control plane version **1.26.1.gke-1500 or later**

TPU Workloads in k8s

KubeCon CloudNativeCon North America 2023

```yaml
apiVersion: v1
kind: Service
metadata:
  name: headless-svc
spec:
  clusterIP: None
  selector:
    job-name: tpu-job-podslice
---
apiVersion: batch/v1
kind: Job
metadata:
  name: tpu-job-podslice
spec:
  backoffLimit: 0
  # completions and parallelism should be the
  # number of cores divided by 8 (e.g. 4 for a v4-32)
  completions: 4
  parallelism: 4
  completionMode: Indexed
  template:
    spec:
      hostNetwork: false
      subdomain: headless-svc
      restartPolicy: Never
      nodeSelector:
        cloud.google.com/gke-nodepool: tpu-v4-32
```

1 K8s Service name

2 K8s Job name

3 IndexedJob to identify each pod with a unique ID

4 The node pool created with the given type and topology

```yaml
      containers:
      - name: tpu-job
        image: python:3.8
        ports:
        - containerPort: 8471
        securityContext:
          privileged: true
        env:
        - name: TPU_WORKER_ID
          valueFrom:
            fieldRef:
              fieldPath:
metadata.annotations['batch.kubernetes.io/job-completion-index']
        - name: TPU_WORKER_HOST_NAMES
          value:
tpu-job-podslice-0.headless-svc,tpu-job-podslice-1.headless-svc,tpu
-job-podslice-2.headless-svc,tpu-job-podslice-3.headless-svc
        command:
        - bash
        - -c
        - |
          pip install 'jax[tpu]' -f
https://storage.googleapis.com/jax-releases/libtpu_releases.html
          python -c 'import jax; print("TPU cores:",
jax.device_count())'
        resources:
          requests:
            google.com/tpu: 4
          limits:
            google.com/tpu: 4
```

5 ENV variable for job index to use in your app (0..3)

6 ENV var: job hostname

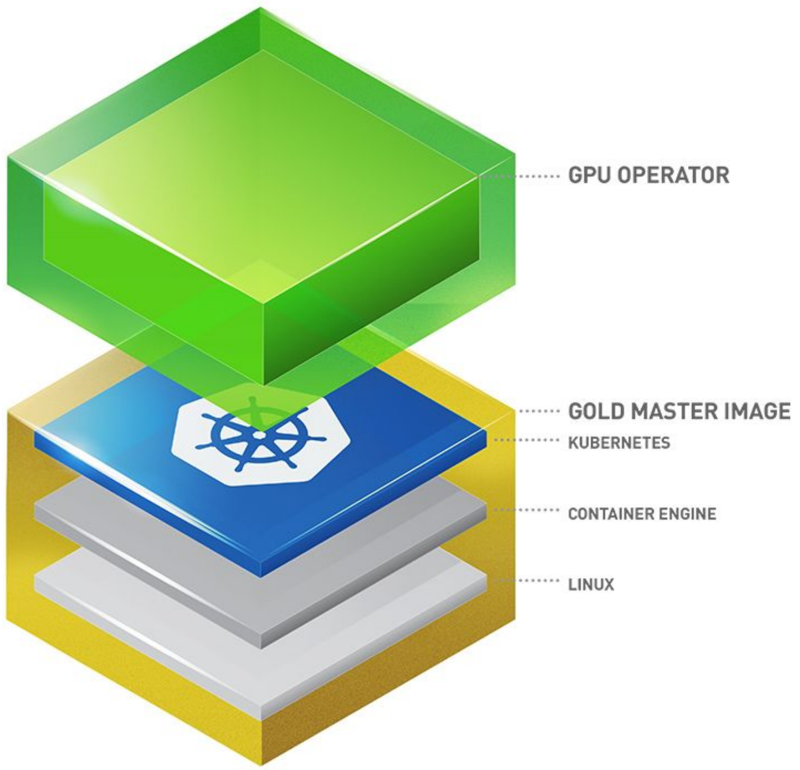
7 Install libtpu

8 Set requests

TPUs Demo

Operating clusters with GPUs/TPUs [1]

- Create a cluster and provision nodes with the resources
- Install the corresponding device plugin and drivers
  - Cloud providers usually pre/install manage this part!
  - Consider using the NVIDIA GPU Operator



Operating clusters with GPUs/TPUs [2]

- Clusters have many nodes -- we need to label them!
  - Label nodes based on the resources they contain
    - Cloud providers like GKE automatically label your nodes
    - Otherwise, see NVIDIA gpu-feature-discovery project that does so
- Consider to taint accelerator nodes to avoid other workloads to be scheduled there
- Schedule workload with node affinity based on node labels and requests to device resources
- For GPU, utilization is important!
  - Consider resource sharing schemes: MIG, Time slicing, NVIDIA MPS

Accelerator Monitoring

KubeCon | CloudNativeCon
North America 2023

Cloud Provider Built in Metrics [e.g. GKE]

`node/accelerator/duty_cycle` BETA
Accelerator duty cycle with node

GAUGE, DOUBLE, percent k8s_node	*Percent of time over the past sample period (10s) during which the accelerator was actively processing. Sampled every 60 seconds.* `make`: Make of the accelerator. `accelerator_id`: ID of the accelerator. `model`: Model of the accelerator.

`node/accelerator/memory_bandwidth_utilization` BETA
Memory bandwidth utilization

GAUGE, DOUBLE, percent k8s_node	*Current percentage of the accelerator memory bandwidth that is being used. Computed by dividing the memory bandwidth used over a sample period by the maximum supported bandwidth over the same sample period. Sampled every 60 seconds. After sampling, data is not visible for up to 120 seconds.* `make`: Make of the accelerator. `accelerator_id`: ID of the accelerator. `model`: Model of the accelerator. `tpu_topology`: Topology of the TPU accelerator.

`node/accelerator/memory_total` BETA
Accelerator memory total with node

GAUGE, INT64, bytes k8s_node	*Total accelerator memory in bytes. Sampled every 60 seconds.* `make`: Make of the accelerator. `accelerator_id`: ID of the accelerator. `model`: Model of the accelerator.

`node/accelerator/memory_used` BETA
Accelerator memory used with node

GAUGE, INT64, bytes k8s_node	*Total accelerator memory allocated in bytes. Sampled every 60 seconds.* `make`: Make of the accelerator. `accelerator_id`: ID of the accelerator. `model`: Model of the accelerator.

`node/accelerator/tensorcore_utilization` BETA
Tensorcore utilization

GAUGE, DOUBLE, percent k8s_node	*Current percentage of the Tensorcore that is utilized. Computed by dividing the Tensorcore operations that were performed over a sample period by the supported number of Tensorcore operations over the same sample period. Sampled every 60 seconds. After sampling, data is not visible for up to 120 seconds.* `make`: Make of the accelerator. `accelerator_id`: ID of the accelerator. `model`: Model of the accelerator. `tpu_topology`: Topology of the TPU accelerator.

NVIDIA Prometheus DCGM exporter

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: nvidia-dcgm-exporter-metrics
…
data:
  counters.csv: |
    # Utilization (the sample period varies depending on the product),,
    DCGM_FI_DEV_GPU_UTIL, gauge, GPU utilization (in %).
    DCGM_FI_DEV_MEM_COPY_UTIL, gauge, Memory utilization (in %).

    # Utilization of IP blocks,,
    DCGM_FI_PROF_SM_ACTIVE, gauge,
    DCGM_FI_PROF_SM_OCCUPANCY, gauge,
    DCGM_FI_PROF_PIPE_TENSOR_ACTIVE, gauge,
    DCGM_FI_PROF_PIPE_FP64_ACTIVE, gauge,
    DCGM_FI_PROF_PIPE_FP32_ACTIVE, gauge,
    DCGM_FI_PROF_PIPE_FP16_ACTIVE, gauge,

    # Memory usage,,
    DCGM_FI_DEV_FB_FREE, gauge,
    DCGM_FI_DEV_FB_USED, gauge,
    DCGM_FI_DEV_FB_TOTAL, gauge,

    # PCIE,,
    DCGM_FI_PROF_PCIE_TX_BYTES, gauge,
    DCGM_FI_PROF_PCIE_RX_BYTES, gauge,

    # NVLink,,
    DCGM_FI_PROF_NVLINK_TX_BYTES, gauge,
    DCGM_FI_PROF_NVLINK_RX_BYTES, gauge,
```

Future

- Dynamic Resource Allocation (DRA)
  - New way of requesting resources available (as an alpha feature) since Kubernetes 1.26+
  - An ***alternative*** to counting-based interface in the Device plugin
  - Puts full control of the API to request resources in the hands of 3rd-party developers
  - Key concepts (in-tree API → vendor-specific API):
    - `ResourceClass` → `ClassParameters`
    - `ResourceClaim` → `ClaimParameters`
  - Uses CDI behind the scenes to define and/or select devices

Future

- DRA enables
  - Support for multiple device types per node
  - Sharing of devices across containers and pods
  - Selection of resources based on constraints such as available memory
  - Dynamic provisioning of resources such as MIG devices
  - Support for enhanced features such as MPS

- Better control allows for right-sizing the device for the application

- Ongoing discussions around Scheduler and Autoscaler implications

How you can help

- What problems do you have with using accelerators?
- Is there anything limiting you in the existing device plugin model?

Links

KubeCon sessions

- Unlocking the Full Potential of GPUs for AI Workloads on Kubernetes - Kevin Klues, NVIDIA
- Reducing AI Job Cold Start Time from 15 Mins to 1 Min - Tao He, Google
- On-Demand Systems and Scaled Training Using the JobSet API - Abdullah Gharaibeh, Google & Vanessa Sochat, Lawrence Livermore National Laboratory

Other

- Dynamic Resource Allocation KEP
- The Container Device Interface
- Device Plugin KEP
- Improving GPU Utilization in Kubernetes

Slack

- @elezar
- @bobbypage