阿里云 × CLOUD NATIVE COMPUTING FOUNDATION

云原生技术公开课

第 20 讲

GPU管理和Device Plugin工作机制

车漾　阿里巴巴高级技术专家

关注“阿里巴巴云原生”公众号
获取第一手技术资料

1 需求来源

2 GPU的容器化

3 K8S的GPU管理

4 工作原理

5 课后思考与实践

无处不在的人工智能

需求来源

为什么要用Kubernetes管理以GPU为代表的异构资源？

- **加速部署：**通过容器构建避免重复部署机器学习复杂环境
- **提升集群资源使用率：**统一调度和分配集群资源
- **保障资源独享：**利用容器隔离异构设备，避免互相影响

1 需求来源

2 GPU的容器化

3 K8S的GPU管理

4 Device Plugin原理

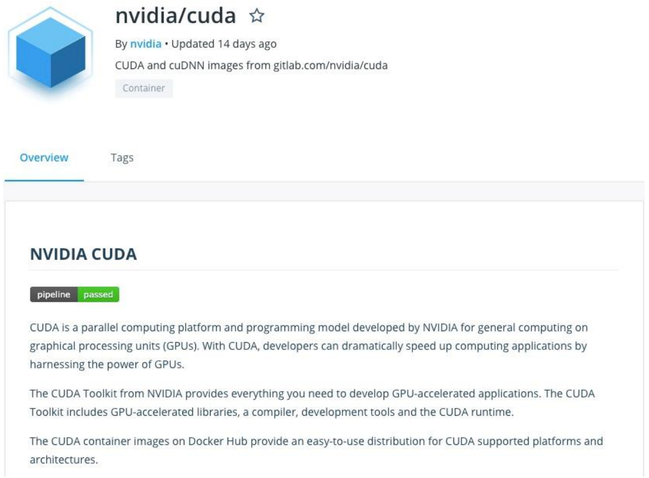
5 课后思考与实践

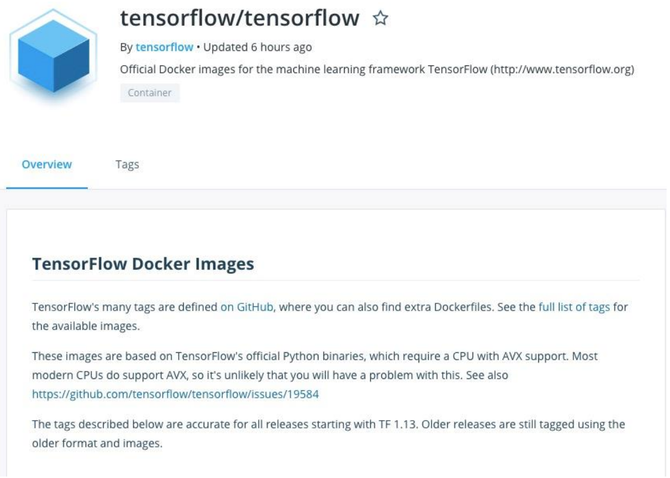
如何利用容器运行GPU程序

- 构建支持GPU容器镜像
- 利用Docker将该镜像运行起来，并且把GPU设备和依赖库映射到容器中

如何准备GPU容器镜像

- 直接使用官方深度学习容器镜像
- 基于Nvidia的CUDA镜像基础构建





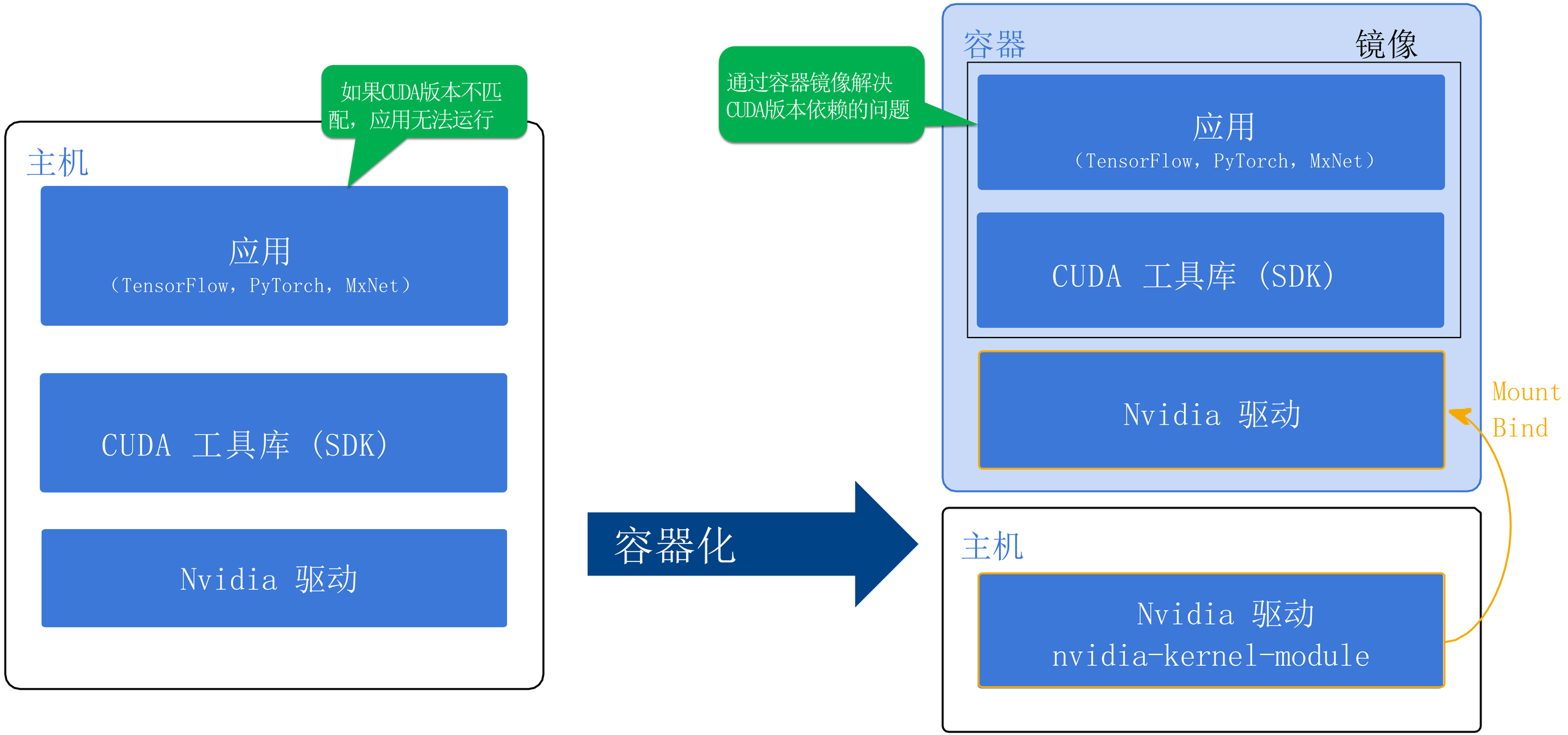

```
ARG UBUNTU_VERSION=18.04

ARG ARCH=
ARG CUDA=10.0
FROM nvidia/cuda${ARCH:+-$ARCH}:${CUDA}-base-ubuntu${UBUNTU_VERSION} as base
# ARCH and CUDA are specified again because the FROM directive resets ARGs
# (but their default value is retained if set previously)
ARG ARCH
ARG CUDA
ARG CUDNN=7.4.1.5-1

# Needed for string substitution
SHELL ["/bin/bash", "-c"]
# Pick up some TF dependencies
RUN apt-get update && apt-get install -y --no-install-recommends \
        build-essential \
        cuda-command-line-tools-${CUDA/./-} \
        cuda-cublas-${CUDA/./-} \
        cuda-cufft-${CUDA/./-} \
        cuda-curand-${CUDA/./-} \
        cuda-cusolver-${CUDA/./-} \
        cuda-cusparse-${CUDA/./-} \
        curl \
        libcudnn7=${CUDNN}+cuda${CUDA} \
        libfreetype6-dev \
        libhdf5-serial-dev \
        libzmq3-dev \
        pkg-config \
        software-properties-common \
        unzip
```

TensorFlow GPU的DockerFile

GPU容器镜像原理


如果CUDA版本不匹配，应用无法运行
主机
应用
（TensorFlow，PyTorch，MxNet）
CUDA 工具库（SDK）
Nvidia 驱动
容器化
通过容器镜像解决CUDA版本依赖的问题
容器
镜像
应用
（TensorFlow，PyTorch，MxNet）
CUDA 工具库（SDK）
Nvidia 驱动
Mount
Bind
主机
Nvidia 驱动
nvidia-kernel-module

如何利用容器运行GPU程序

利用Docker对于Device和Mount Volume的支持运行GPU程序

```
docker run -it                    \
--volume=nvidia_driver_xxx.xx:/usr/local/nvidia:ro          \
--device=/dev/nvidiactl                  \
--device=/dev/nvidia-uvm            \
--device=/dev/nvidia-uvm-tools           \
--device=/dev/nvidia0            \
nvidia/cuda nvidia-smi
```

```
"Devices": [
    {
        "PathOnHost": "/dev/nvidiactl",
        "PathInContainer": "/dev/nvidiactl",
        "CgroupPermissions": "rwm"
    },
    {
        "PathOnHost": "/dev/nvidia-uvm",
        "PathInContainer": "/dev/nvidia-uvm",
        "CgroupPermissions": "rwm"
    },
    {
        "PathOnHost": "/dev/nvidia-uvm-tools",
        "PathInContainer": "/dev/nvidia-uvm-tools",
        "CgroupPermissions": "rwm"
    },
    {
        "PathOnHost": "/dev/nvidia0",
        "PathInContainer": "/dev/nvidia0",
        "CgroupPermissions": "rwm"
    }
],
```

Docker inspect

```
"Mounts": [
    {
        "Type": "bind",
        "Source": "/usr/lib64/libnvidia-ml.so",
        "Destination": "/usr/lib/x86_64-linux-gnu/libnvidia-ml.so.410.79",
        "Mode": "",
        "RW": true,
        "Propagation": "rprivate"
    }
```

1 需求来源

2 GPU的容器化

3 K8S的GPU管理

4 Device Plugin原理

5 课后思考与实践

如何部署GPU Kubernetes

1.安装 NVIDIA 驱动

```
$ sudo yum install -y gcc kernel-devel-$(uname -r)
    $ sudo /bin/sh ./NVIDIA-Linux-x86_64*.run
```

2.安装 NVIDIA Docker2

```
    $ sudo yum install nvidia-docker2
    $ sudo pkill -SIGHUP dockerd
```

3.部署Nvidia Device Plugin

```
$ kubectl create -f nvidia-device-plugin.yml
```

```
bash# cat daemon.json
{
    "default-runtime": "nvidia",
    "runtimes": {
        "nvidia": {
            "path": "/usr/bin/nvidia-container-runtime",
            "runtimeArgs": []
        }
    }
}
```

daemon.json

```
bash# kubectl get po -l component=nvidia-device-plugin
NAME                                          READY     STATUS
nvidia-device-plugin-cn-shanghai.192.168.0.85   1/1       Running
```

验证部署GPU Kubernetes结果

```
$ kubectl describe node gpu-node-01

apiVersion: v1
kind: Node
metadata:
    name: node-1
...
Status:
    Capacity:
         cpu:                          4
           memory:                     15234152Ki
           nvidia.com/gpu:     2
```

在Kubernetes中使用GPU的yaml样例

```
apiVersion: v1
kind: Pod
metadata:
    name: cuda-vector-add
spec:
    restartPolicy: OnFailure
    containers:
        - name: cuda-vector-add
          image: ”nvidia/cuda-vector-add:v0.1"
                    resources:
            limits:
               nvidia.com/gpu: 1
               memory: ”512Mi"
                     cpu: "250m"
```

通过nvidia.com/gpu指定
使用GPU的数量

查看运行结果

```
$ kubectl apply -f gpupod.yaml
pod/gpu created

$ kubectl exec -it gpu -- nvidia-smi
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 410.79       Driver Version: 410.79       CUDA Version: 10.1     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla T4            On   | 00000000:00:07.0 Off |                    0 |
| N/A   33C    P8    15W /  70W |      0MiB / 15079MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

1 需求来源

2 GPU的容器化

3 K8S的GPU管理

4 工作原理

5 课后思考与实践

通过扩展的方式管理GPU资源

Extended Resources

- 通过自定义资源扩展，允许用户分配和使用non-Kubernetes-built-in的计算资源。

Device Plugin Framework

- 允许第三方设备提供商以插件外置的方式对设备的调度和全生命周期管理。

Extended Resources的上报

```
# 执行 PACTH 操作
$ curl --header "Content-Type: application/json-patch+json" \
--request PATCH \
--data '[{"op": "add", "path": "/status/capacity/nvidia.com/gpu", "value": "1"}]' \
https://localhost:6443/api/v1/nodes/<your-node-name>/status
```

```
$ kubectl describe node <your-node-name>
apiVersion: v1
kind: Node
...
Status:
    Capacity:
        nvidia.com/gpu:     1
```

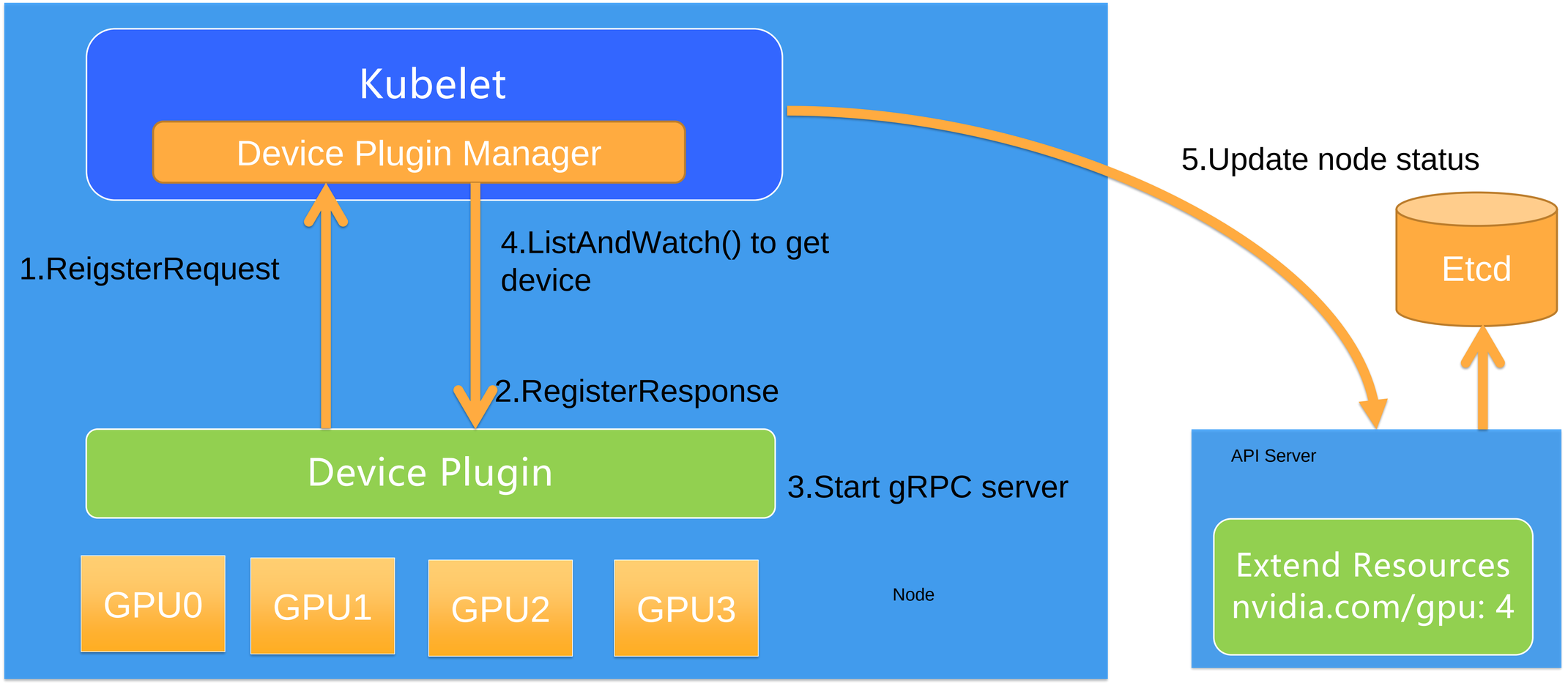
Device Plugin的工作机制

- 资源的上报和监控
- 容器的调度和运行

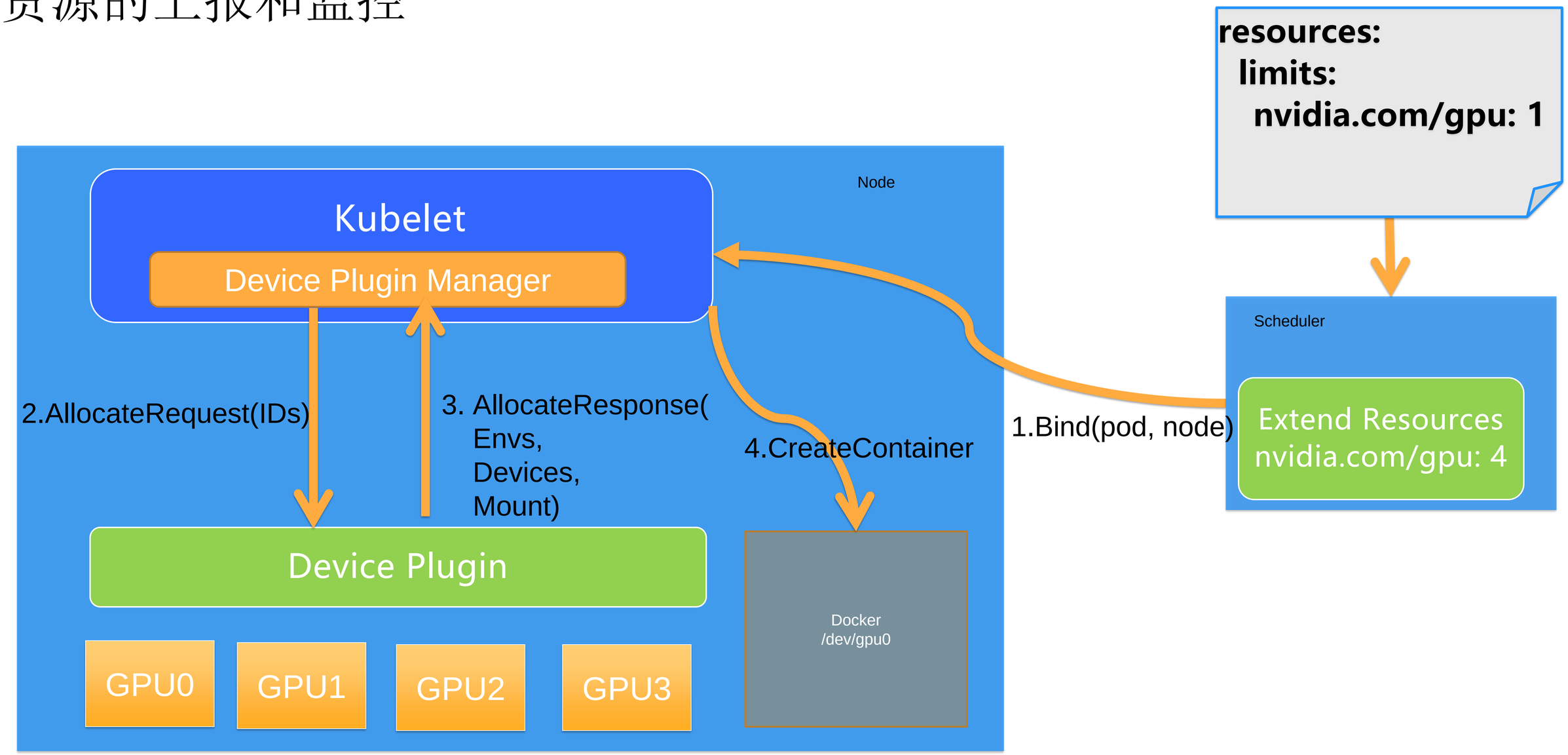
```
// DevicePluginServer is the server API for DevicePlugin service.
type DevicePluginServer interface {
  // ListAndWatch returns a stream of List of Devices
  // Whenever a Device state change or a Device disappears, ListAndWatch
  // returns the new list
  ListAndWatch(*Empty, DevicePlugin_ListAndWatchServer) error
  // Allocate is called during container creation so that the Device
  // Plugin can run device specific operations and instruct Kubelet
  // of the steps to make the Device available in the container
  Allocate(context.Context, *AllocateRequest) (*AllocateResponse, error)
}

```

资源的上报和监控


Kubelet
Device Plugin Manager
1.ReigsterRequest
4.ListAndWatch() to get device
2.RegisterResponse
Device Plugin
3.Start gRPC server
GPU0
GPU1
GPU2
GPU3
Node
5.Update node status
Etcd
API Server
Extend Resources
nvidia.com/gpu: 4

资源的上报和监控


resources:
limits:
nvidia.com/gpu: 1
Node
Kubelet
Device Plugin Manager
2.AllocateRequest(IDs)
3. AllocateResponse(
Envs,
Devices,
Mount)
4.CreateContainer
1.Bind(pod, node)
Scheduler
Extend Resources
nvidia.com/gpu: 4
Device Plugin
GPU0
GPU1
GPU2
GPU3
Docker
/dev/gpu0

Device Plugin生命周期总结

- Device Plugin启动时，以grpc的形式通过/var/lib/kubelet/device-plugins/kubelet.sock向Kubelet注册设备id（比如nvidia.com/gpu）
- Kubelet将会把设备数量以Node状态上报到API server中，后续调度器会根据这些信息进行调度
- Kubelet同时会建立一个到Device Plugin的listAndWatch长连接，当插件检测到某个设备不健康的时候，就会主动通知Kubelet
- 当用户的应用请求GPU资源后，Device Plugin根据Kubelet在Allocate请求分配好的设备id定位到对应的**设备路径**和**驱动文件**
- Kubelet根据Device plugin提供的信息创建对应容器

Device Plugin机制的缺陷

- 资源上报信息有限导致调度精细度不足
- 调度发生在Kubelet层面，缺乏全局调度视角
- 调度策略简单，并且无法配置，无法应对复杂场景

社区的异构资源调度方案

- Nvidia GPU Device Plugin: https://github.com/NVIDIA/k8s-device-plugin
- GPU Share Device Plugin : https://github.com/aliyunContainerService/gpushare-device-plugin
- RDMA Device Plugin : https://github.com/Mellanox/k8s-rdma-sriov-dev-plugin
- FPGA Device Plugin : https://github.com/Xilinx/FPGA_as_a_Service/tree/master/k8s-fpga-device-plugin/trunk

1 需求来源

2 GPU的容器化

3 K8S的GPU管理

4 Device Plugin原理

5 课后思考与实践

课后总结

GPU的容器化：

- 构建GPU容器
- 直接在Docker上运行GPU容器

利用Kubernetes管理GPU资源：

- 如何在Kubernetes支持GPU调度
- 如何验证Kubernetes下的GPU配置
- 调度GPU容器的方法

Device Plugin的工作机制：

- 现有工作流程
- 目前的缺陷
- 社区常见的Device Plugin

关注“阿里巴巴云原生”公众号
获取第一手技术资料