Cloud Native Forum

Cloud Native Forum

렌터카 서비스처럼 GPU 를 요청 쿠버네티스 DRA

할당 구조와 GPU 공유 방식, 도입 기준을 짚는
플랫폼 운영자용 기술 해설

PROJECT CODE
K8S-DRA

DATE
2026.09

PUBLIC

CONTENTS

기초 개념부터 도입 판단까지, 그리고 드라이버 내부 동작

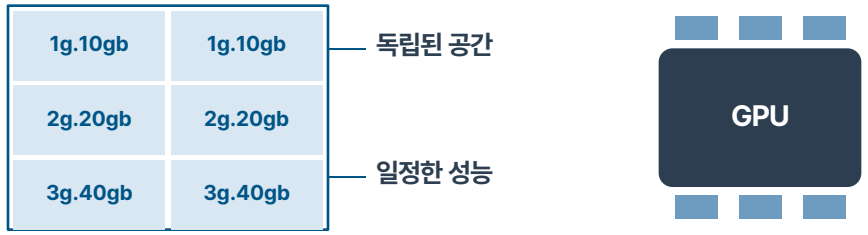
KEY FOCUS 쿠버네티스 v1.34 에서 GA 된 DRA 의 동작 원리와 운영 판단 기준

GPU를 나누는 두 기술과 배정하는 한 시스템

GPU를 나누는 두 가지 기술 · 공유 방식

비싼 GPU 한 장을 물리적으로 쪼개거나(MIG), 시간을 나눠(Time-Slicing) 함께 쓴다

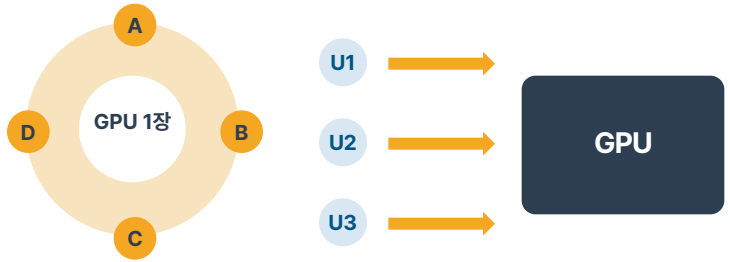
MIG — "아파트 방 쪼개기"



격리성 물리적으로 고정 분할 — 독립 공간과 일정한 성능 보장

격리성 vs 유연성

Time-Slicing — "자동차 시간제로 빌려 타기"



10ms 단위로 번갈아 — 한 장을 여러 사용자가 순서대로

유연성 짧은 시간 단위로 돌려 쓰며 공유 밀도를 높임

GPU를 배정하는 스마트한 시스템 · 관리 방식

GPU DRA — "렌터카"처럼 빌리고 반납하는 5단계

할당 → 반납 단계	렌터카 비유
1 **ResourceSlice** 드라이버가 노드의 GPU·MIG 목록과 속성을 게시	"차량 목록 등록"
2 **ResourceClaim** Pod이 GPU 모델·메모리·드라이버 조건을 적어 요청	"예약 신청서 제출"
3 **Scheduler Allocation** 조건에 맞는 장치를 골라 claim에 배정·예약 기록	"배차 확정"
4 **NodePrepareResources** kubelet·드라이버가 장치를 컨테이너에 주입	"차 키 인도"
5 **Unprepare · Deallocate** Pod 종료 시 주입 해제, claim 삭제 → 풀로 즉시 복귀	"반납 후 바로 재배차"

쿠버네티스 자원 관리의 미래

Device Plugin의 한계를 넘어 MIG·공유 GPU를 자동 매칭

세 가지 방식의 핵심 차이

구분	MIG	Time-Slicing	GPU DRA
핵심 개념	하드웨어 분할	시간 분할 공유	동적 할당 표준
쉬운 비유	아파트 방 쪼개기	자동차 시간제	렌터카 예약
주요 장점	완전한 성능 격리	높은 공유 밀도	유연한 자동 배정

아파트 · 자동차 · 렌터카 비유는 이해를 돕기 위한 단순화입니다

GPU 한 장을 나눠 쓰는 세 가지 방식

아파트 · 자동차 시간제 · 렌터카 비유로 정리

KEY INSIGHT **GPU 한 장을 아파트·자동차·렌터카에 비유하면 세 방식의 차이가 한눈에 보입니다**
MIG는 방을 쪼개고 · Time-Slicing은 시간제로 돌려 타고 · DRA는 렌터카처럼 빌렸다 반납한다

MIG

고정 분할

GPU를 하드웨어로 쪼개 독립 GPU처럼

아파트 한 채 = GPU 1장

방 A	방 B	방 C
Pod A	Pod B	Pod C

벽은 고정 — 미리 정한 평수만

아파트 한 채를 방 3개로 쪼개기
벽을 세워 방 크기를 고정한다

SM·메모리까지 물리적으로 분리된 인스턴스
1g.10gb 같은 프로파일을 미리 정해 둬야 함

장점 · 간섭 없는 완전 격리
한계 · 크기 변경은 재구성 필요

Time-Slicing

시간 분할

한 장의 GPU를 시간 단위로 번갈아

자동차 한 대 = GPU 1장

차 한 대를 한 사람씩 운전

09:00	09:10	09:20
Pod A	Pod B	Pod C

시간을 쪼개 돌아가며 운전

자동차 한 대를 시간제로 돌려 타기
09:00 A → 09:10 B → 09:20 C 순서로

GPU는 그대로 두고 실행 시간을 나눠 준다
동시에 전용으로 점유하지는 않음

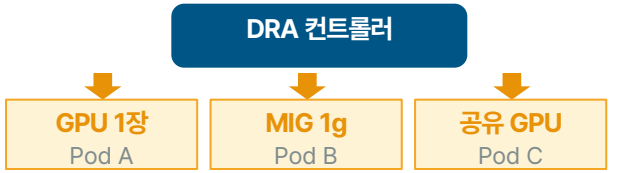
장점 · 설정이 간단하고 유휴가 적다
한계 · 메모리 격리 없고 지연이 흔들린다

GPU DRA

동적 할당

필요할 때 배정하고 끝나면 회수

렌터카 영업소 = DRA



조건에 맞춰 배차 → 반납 시 즉시 회수

조건에 맞는 차를 내주는 렌터카 예약
반납하면 다음 사람에게 바로 재배차

Pod이 조건을 적으면 스케줄러가 장치를 고른다
MIG·공유 모두 쿠버네티스 자원으로 관리

장점 · 조건 기반 배정과 자동 반환
한계 · v1.34 GA, 드라이버 지원 필요

렌터카 5단계 차량 등록(ResourceSlice) → 예약(ResourceClaim) → 배차(Allocation) → 인도(NodePrepare) → 반납(Deallocate)

CHAPTER 01

GPU 개수만 셀 수 있는 device plugin 모델

정수 요청이 남기는 유휴 용량과 수동 배치 규칙

KEY INSIGHT **정수 요청에는 장치 종류·메모리·연결 구조가 담기지 않음 —**
2~4GB만 쓰는 추론 작업도 80GB A100 한 장을 통째로 점유

요청 형식은 정수 하나

```
resources:
  limits:
    nvidia.com/gpu: 1        ← 종류·메모리 정보 없음
```

스케줄러가 보는 노드 GPU

GPU	GPU	GPU	GPU
단위 1	단위 1	단위 1	단위 1

A100 80GB와 T4 16GB도 똑같이 나눌 수 없는 한 단위로 취급

MIG 우회 방식

nvidia.com/mig-1g.10gb 같은 리소스 이름을
프로파일마다 미리 정의해야 하는 경직된 구조

Time-slicing 복제본 슬롯

용량 보장이 아닌 접근 권한 — 두 파드가 같은 GPU를
쓴다는 사실이 API에 드러나지 않음

정수 요청으로 표현할 수 없는 요구

메모리 조건

"메모리 40GB 이상인 GPU만" 배정 요청

분할 프로파일 조건

"MIG 1g.10gb 인스턴스만" 배정 요청

명시적 공유

"이 파드와 저 파드가 같은 GPU를 함께 사용"

연결 구조 조건

"GPU와 NIC가 같은 PCIe 루트에 연결된 조합"

Source: wikidocs GPU 공유 기술 가이드 9장 · AI on EKS · Google Cloud Blog (2026.03)

예약한 GPU가 유휴 상태로 남는 스케줄링 비효율

KEY INSIGHT **두 수치는 측정 대상이 달라 직접 비교 불가 —**
공통 원인은 할당 단위가 GPU 한 장이라 남는 용량을 다른 작업이 쓰지 못하는 구조

고수요 AI 클러스터의 GPU 활용률

40% 미만으로 흔히 관찰되는 활용률

출처: AWS AI on EKS — 수요가 높은 클러스터에서 관찰된 활용률

운영 클러스터에서 실제로 쓰이는 GPU 자원

평균 5%

출처: Cast AI 2026 쿠버네티스 최적화 연구 (saaro 인용)

비효율이 드러나는 증상

큐 기아
Queue starvation

짧은 추론 작업이 오래 걸리는
학습 작업 뒤에서 대기

자원 단편화
Resource fragmentation

노드마다 쓸 수 없는 크기로
흩어진 GPU 메모리

토폴로지 무인식
Topology blindness

NVLink로 묶이지 않은 GPU에
멀티 GPU 작업이 배치되어
통신 성능 저하

과다 프로비저닝
Overprovisioning

스케줄링 비효율을 GPU 추가
구매로 메우는 비용 증가

Source: AWS AI on EKS (DRA 가이드) · saaro (2026.08, Cast AI 연구 인용) · Rafay (2025.08)

두 수치는 측정 범위와 방법이 달라 한 축에 놓고 비교하지 않음

KEY INSIGHT **GPU 구성 정보가 팀별 배포 파일에 중복 기록됨 —**
장비를 증설하거나 교체할 때마다 여러 팀의 매니페스트를 함께 고치는 운영 비용 발생

파드 매니페스트에 한 겹씩 쌓이는 배치 설정

① nodeSelector
NodeGroupType: g6-mng 처럼 GPU 노드 그룹 직접 지정

② affinity
instance-type 후보를 규칙으로 나열해 모델 선택

③ taint · toleration
GPU 노드에서 일반 파드를 막고 GPU 파드만 허용

④ 웹훅 · 스케줄러 확장
GPU 구성별 변환, 연결된 장치 묶음 배정

↓

네 겹을 모두 통과해야 닿는 GPU 노드

그 결과 생기는 운영 부담

노드 구성을 앱 운영자가 직접 조사
어느 노드에 어떤 GPU가 있는지 먼저 파악한 뒤
라벨·affinity 규칙을 매니페스트에 작성

연결된 장치 묶음은 기본 스케줄러로 배정 불가
AWS Neuron 가속기는 서로 연결된 장치 할당에
별도 스케줄러 확장(Scheduler Extension) 필요

가속기를 미리 준비해 두는 정적 구조
파드를 스케줄링하기 전에 필요한 가속기가
노드에 이미 올라와 있어야 배치 가능

관리자와 워크로드 운영자의 수작업 조율 — 선언형 스케줄링과 어긋남

Source: Google Cloud Blog (2026.03) · Rafay (2025.08) · AWS Neuron DRA 문서 · AI on EKS

CHAPTER 02

장치 속성을 스케줄러가 직접 읽는 DRA 할당 구조

resource.k8s.io API 오브젝트와 드라이버 · 스케줄러 · kubelet 의 역할

02 PersistentVolume 구조를 그대로 옮겨 온 장치 요청 모델

KEY INSIGHT **PVC 를 써 본 운영자라면 학습 비용이 작음 —**
요청과 재고를 분리한 덕분에 워크로드가 노드 구성을 몰라도 장치를 받음

맡는 역할	스토리지 — PersistentVolume 모델	장치 — DRA 모델 (resource.k8s.io)
자원 종류와 선택 기준 정의	StorageClass	DeviceClass
워크로드의 자원 요청	PersistentVolumeClaim	ResourceClaim
파드마다 요청 자동 생성	StatefulSet volumeClaimTemplates	ResourceClaimTemplate
실제로 쓸 수 있는 자원 목록	PersistentVolume	ResourceSlice

스토리지 흐름: 사용자가 PVC 작성 → 프로비저너가 조건 확인 → PV 를 PVC 에 연결

DRA 흐름: **운영자가 Claim 작성** → **kube-scheduler 가 조건 대조** → **ResourceSlice 장치를 Claim 에 할당** 노드 구성은 몰라도 됨

Source: wikidocs GPU 공유 기술 가이드 9장 (KubeCon NA 2024 발표 인용) · Rafay (2025.08)

02 드라이버와 관리자, 워크로드 운영자가 나눠 만드는 핵심 오브젝트

KEY INSIGHT **책임이 오브젝트 단위로 분리됨 —**
장치 교체나 드라이버 업그레이드가 워크로드 매니페스트 수정으로 번지지 않음

드라이버가 만드는 것

ResourceSlice

노드별로 게시하는 장치 목록

속성(attributes): 제품명 · 아키텍처
용량(capacity): GPU 메모리

NVIDIA · AWS Neuron 드라이버가
노드에서 장치를 찾아 자동 등록

관리자가 정하는 것 (드라이버 기본값 포함)

DeviceClass

장치 분류와 기본 선택 기준

드라이버 기본 생성:

gpu.nvidia.com · mig.nvidia.com
compute-domain.nvidia.com

high-memory-gpu 같은 클래스 추가 정의

워크로드 운영자가 작성하는 것

ResourceClaimTemplate

파드마다 Claim 을 만들어 주는 틀
resourceclaim-controller 가 생성 담당

파드별 생성

ResourceClaim

DeviceClass 를 겨냥한 장치 요청
파드가 resourceClaims 필드에서 이름으로 참조

재고 조회 · 선택 기준 참조 · 요청 대조

kube-scheduler

ResourceClaim 의 요구와 ResourceSlice 의 미할당 장치를 대조해 배치할 노드 결정

파드 배치 결정
노드 kubelet 이 장치 준비

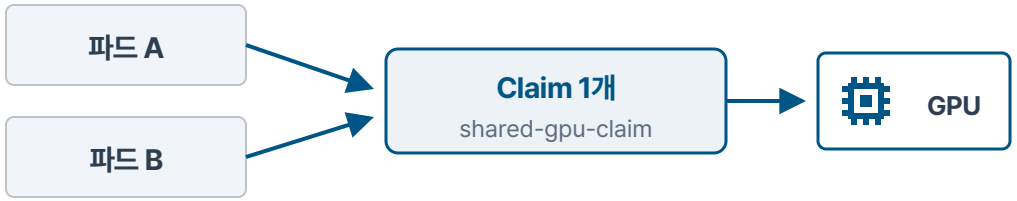
Source: Kubernetes 공식 문서 How DRA Works · wikidocs 9장 · AI on EKS · Google Cloud Blog (2026.03)

02 공유 여부로 갈리는 두 가지 Claim 참조 방식

KEY INSIGHT **추론 서비스와 분산 학습의 장치 사용 형태를 매니페스트에서 바로 구분 —**
어느 파드들이 한 장치를 함께 쓰는지 운영자가 추적 가능

ResourceClaim 직접 참조

resourceClaimName



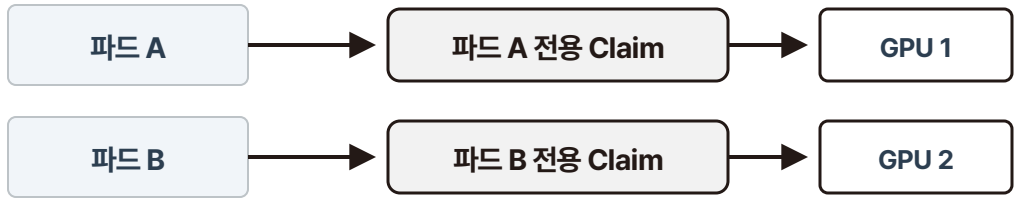
두 파드가 같은 Claim 을 가리키면 같은 장치를 공유

생성 주체	운영자가 미리 만들어 두는 오브젝트
장치 사용	여러 파드가 할당된 장치 하나를 공유
수명	파드가 종료되어도 Claim 은 남음

권장 용도
추론 서비스처럼 한 GPU를 나눠 쓰는 워크로드

ResourceClaimTemplate 참조

resourceClaimTemplateName



템플릿이 파드마다 Claim 을 따로 찍어 내어 전용 장치 배정

생성 주체	쿠버네티스가 파드마다 자동 생성
장치 사용	파드별로 전용 장치 할당
수명	파드가 종료되면 Claim 도 함께 삭제

권장 용도
분산 학습처럼 복제본마다 GPU가 필요한 워크로드

공통 제약 — ResourceClaim 은 참조하는 파드와 같은 네임스페이스에만 둘 수 있음 (네임스페이스 간 참조 미지원)

Source: Rafay (2025.08) · Cast AI Docs · AWS AI on EKS (네임스페이스 요구 사항)

KEY INSIGHT **세대가 섞인 장비도 하나의 스케줄링 영역으로 운영 가능 —**
장비를 증설해도 워크로드 매니페스트 수정이 필요 없음

속성 조건

ResourceSlice 에 게시된 제품명·
MIG 프로파일 같은 값 비교

```
deviceClassName: mig.nvidia.com
selectors:
  - cel:
      expression: device.attributes['gpu.nvidia.com'].profile == '1g.10gb'
```

용량 조건

GPU 메모리 하한처럼 수량 비교,
속성 조건과 한 식으로 결합

```
- cel:
    expression: |
      device.attributes['gpu.nvidia.com'].architecture == 'Volta' &&
      device.capacity['gpu.nvidia.com'].memory.isGreaterThan(quantity('10Gi'))
```

장치 사이 조건

요청한 장치들이 같은 연결 그룹 값을
갖도록 묶어서 할당

```
count: 4
constraints:
  - requests: ["neurons"]
    matchAttribute: resource.aws.com/devicegroup4_id
```

복수 장치 조건 사례

AWS Neuron: 서로 연결된 가속기 4개

GKE: GPU 와 NIC 를 같은 PCIe 루트
컴플렉스에 연결된 조합으로 요청

할당 수량 지정

ExactCount	개수를 정확히 지정
All	조건에 맞는 장치 전부 할당
firstAvailable	우선순위대로 대안 목록 시도

Source: Cast AI Docs · saaro · AWS Neuron DRA 문서 · Google Cloud Blog · wikidocs 9장

02 스케줄러가 Claim에 결과를 기록한 뒤 노드로 넘기는 할당 흐름

KEY INSIGHT **할당 결과가 API 오브젝트에 남음 —**
장애 시 ResourceClaim 상태만 확인해도 어느 단계에서 멈췄는지 추적 가능

1 ResourceSlice 게시

드라이버

비슷한 장치 묶음(pool) 단위로 노드의 장치 목록 등록

2 Claim 확인

컨트롤 플레인

템플릿 참조 시 resourceclaim-controller 가 Claim 생성

지정한 Claim 이 없으면 파드 배포 중단

3 장치 필터링

kube-scheduler

파드를 실행할 수 있는 노드이면서 Claim 조건에 맞는 미할당 장치가 있는지 확인

4 할당 기록

kube-scheduler

first-fit 방식, pool 과 Slice 이름의 사전순 평가

status.allocation 에 결과 기록

5 노드 준비

kubelet + 드라이버

선택된 노드에서 gRPC 로 장치 준비 요청
CDI 로 컨테이너에 장치 연결

스케줄러를 거치지 않은 파드는 Claim 미할당 상태로 멈춤

spec.nodeName 을 직접 지정하면 1~4단계를 건너뜀 — 필요한 Claim 이 할당·예약되지 않아 kubelet 이 실행을 거부하고 주기적으로 재확인

멈춘 파드가 노드의 CPU · 메모리를 계속 점유

특정 노드가 필요할 때

스케줄링 흐름을 유지하는 nodeSelector 사용

```
nodeSelector:
  kubernetes.io/hostname:
    name-of-the-intended-node
```

Source: Kubernetes 공식 문서 How DRA Works (2026.06) · Rafay (2025.08) · KEP-3063 README

컨트롤러 협상 방식을 걷어 내고 GA에 오른 구조화 파라미터

KEY INSIGHT **자원 정보를 쿠버네티스가 직접 읽는 방식으로 표준이 수렴 —**
Cluster Autoscaler 가 증설 효과를 계산할 수 있어야 GPU 노드 자동 확장이 성립

v1.26 alpha
KEP-3063 최초 도입
드라이버 컨트롤러와 스케줄러가 협상하는 방식으로 출발

v1.31 alpha
v1alpha3 API 개편
즉시 할당 방식과 CRD 파라미터 지원을 제거

v1.32 beta
KEP-3063 철회
구조화 파라미터 (KEP-4381) 방식이 기준 기능으로 전환

v1.34 GA
stable · 기본 활성화
`resource.k8s.io/v1`
GKE · AKS · EKS 지원 기준 버전

v1.35
AI Conformance
쿠버네티스 AI 적합성 프로그램의 첫 필수 (MUST) 요건으로 지정

v1.36
확장 기능 추가
Binding Conditions beta
Node Allocatable alpha
장치 상태 보고

철회 — 컨트롤 플레인 컨트롤러 방식 (KEP-3063)

스케줄러 ⇄ PodSchedulingContext ⇄ 드라이버 컨트롤러

철회 이유

· 파라미터가 불투명해 오토스케일러가 할당 효과를 모의 계산 불가
· 스케줄러와 드라이버가 API 서버를 여러 번 오가는 협상

채택 — 구조화 파라미터 방식 (KEP-4381)

· 드라이버는 ResourceSlice 에 속성과 용량만 게시
· 할당 판단은 kube-scheduler 가 직접 수행
· 오토스케일러가 같은 정보로 증설 효과 계산

대가: 새 하드웨어 요구는 쿠버네티스의 파라미터 모델 확장이 먼저 필요

Source: KEP-3063 README · Google Cloud Blog (2026.03) · Kubernetes 공식 문서 · saaro (2026.08)

일부 소스의 v1.35 GA 표기 대신 쿠버네티스 공식 블로그 기준 v1.34 GA 채택

CHAPTER 03

요청 단위로 고르는 GPU 공유 기술과 토폴로지 조건

Time-slicing · MPS · MIG 선택 기준, 고속 연결 구조, v1.36 확장 기능

03 격리 범위와 동시 실행 가능 여부가 다른 GPU 공유 기술

KEY INSIGHT **처리량을 높일수록 작업 사이 보호 수준이 낮아짐 —**
Time-slicing 은 한 작업의 메모리 고갈이 같은 장치의 다른 작업까지 중단

비교 항목	전용 할당 Exclusive	Time-slicing 시간 분할	MPS Multi-Process Service	MIG Multi-Instance GPU
격리 방식	공유 없이 GPU 한 장 전체	시간 단위로 번갈아 실행	데몬이 CUDA 프로세스와 GPU 드라이버 사이를 중계	하드웨어 수준 인스턴스 분할
메모리 격리	완전 격리	격리 없음 같은 메모리 공간을 함께 사용	프로세스별 메모리 분리	연산·메모리·장애까지 격리 메모리 대역폭도 물리적 분할
동시 실행	작업 하나가 GPU 독점	작업마다 차례로 GPU 점유 문맥 전환 오버헤드 발생	여러 커널이 병렬 실행, 문맥 전환 부담 감소	인스턴스마다 보장된 자원 사용 인스턴스 사이 독립 실행
적합한 작업	대규모 모델 학습, 공유를 고려하지 않은 기존 앱	개발·테스트 환경, 사용 시간대가 다른 추론	GPU 연산의 50% 미만을 쓰는 소형 추론 여러 개	멀티 테넌트 환경, 규정 준수가 필요한 작업
요구 조건	별도 조건 없음 — 활용률은 가장 낮음	메모리 격리가 필요 없는 애플리케이션에 한정	NVIDIA DRA 드라이버의 MPSSupport 기능 활성화	A100 · H100 이상 GPU와 GPU Operator MIG Manager

Source: AWS AI on EKS (GPU 공유 전략 심층 분석) · Cast AI Docs (MPS 설정)

03 Claim마다 따로 지정하는 GPU 공유 설정

KEY INSIGHT **한 클러스터 안에서 추론은 MPS, 개발 환경은 Time-slicing 으로 혼용 가능 —**
공유 방식 변경이 클러스터 전역 설정 변경 없이 매니페스트 교체로 완료

device plugin
클러스터 전역 ConfigMap 으로 공유 방식 고정

DRA
ResourceClaim 의 config.opaque 에 요청마다 지정

워크로드별 권장 공유 방식

워크로드 유형	권장 방식
소형 추론 작업	**Time-slicing 또는 MPS**
동시에 도는 소형 모델	**MPS**
멀티 테넌트	**MIG**
대규모 모델 학습	**전용 할당**
개발 · 테스트 환경	**Time-slicing**

kind: ResourceClaim

```
spec:
  devices:
    requests:
    - name: mps-gpu
      exactly: { deviceClassName: gpu.nvidia.com }
    config:
    - requests: ["mps-gpu"]
      opaque:
        driver: gpu.nvidia.com
        parameters:
          kind: GpuConfig
          sharing: { strategy: MPS }        ← 요청마다 지정
          mpsConfig: { defaultActiveThreadPercentage: 33,
                       defaultPinnedDeviceMemoryLimit: 1Gi }
```

MPS — 드라이버 MPSSupport 기능 활성화 필요
같은 GPU를 공유하는 복제본은 모두 같은 노드에 배치

도구마다 다른 지원 범위
NIM Operator 는 Time-slicing 만 지원 · Cast AI 자동 확장은 MIG 미지원

Source: AWS AI on EKS (전략 선택 가이드) · Cast AI Docs (MPS 예제) · NVIDIA NIM Operator 문서

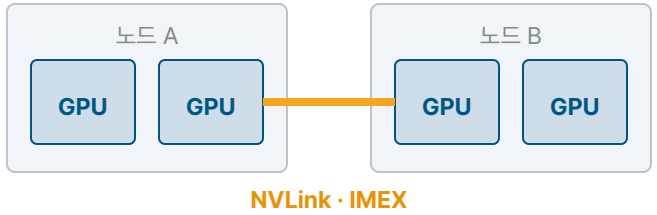
코드는 Cast AI 예제를 흐름 중심으로 줄여 표기

03 NVLink와 PCIe 연결 구조를 배치에 반영하는 토폴로지 제약

KEY INSIGHT

최신 멀티 노드 GPU 시스템은 DRA 없이 핵심 기능 사용 불가 —
GB200 계열 도입 계획이 있으면 DRA 전환이 선행 조건

NVIDIA ComputeDomain



IMEX — 노드 사이 GPU 메모리를 직접
주고받게 하는 NVIDIA 서비스

고속 연결로 묶인 GPU 그룹을
하나의 요청 대상으로 취급

EC2 P6e-GB200 UltraServer 는 DRA 필수
device plugin 방식으로는 동작하지 않음

요청 대상 DeviceClass
compute-domain.nvidia.com

AWS Neuron 장치 그룹

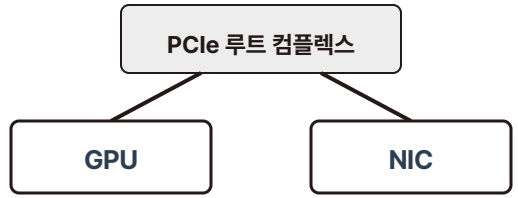
devicegroup4 · 8 · 16_id 속성 기반
서로 연결된 가속기 묶음 요청

기존 스케줄러 확장 없이
기본 스케줄러가 처리

networkNodeLayer1~3 속성
같은 스파인 스위치 아래 노드까지 선택

constraints 조건
matchAttribute: .../devicegroup4_id

GPU–NIC PCIe 정렬



GPU 와 네트워크 카드를 같은 PCIe
루트에 연결된 조합으로 요청

장치 사이 경로를 줄여 분산 학습의
지연 감소와 처리량 확보

ResourceSlice 게시 속성
PCIe 루트 컴플렉스 · NUMA 노드

요청 방법
한 Claim 에 GPU·NIC 요청 + 장치 사이 조건

Source: AWS AI on EKS (IMEX · ComputeDomain) · AWS Neuron DRA 문서 · Google Cloud Blog (2026.03)

GA 이후 alpha · beta 단계로 추가되는 DRA 확장 기능

KEY INSIGHT **확장 기능 대부분이 feature gate 대상 —**
클러스터 버전과 활성화 여부를 기능별로 확인하지 않으면 매니페스트에 적은 조건이 무시됨

Partitionable devices
alpha/beta
동적 MIG 분할

물리 GPU 한 장을 나눌 수 있는 장치로 게시,
수요에 맞춰 MIG 분할을 동적으로 선택

GPU 가 비어 있을 때만 바꾸던
정적 MIG 재구성 제약 완화

Consumable capacity
alpha/beta
용량 분할 공유

장치 메모리 같은 용량을 소비 가능한
자원으로 취급

여러 Claim 이 한 장치의 용량을
나눠 쓰는 분할 공유를 API 수준에서 지원

Device taints
alpha/beta
이상 장치 배제

문제가 생긴 GPU에 taint 를 걸어
배치 대상에서 제외

toleration 을 가진 특정 워크로드만
허용하는 장치 단위 제어

Binding Conditions
v1.36 beta
준비 완료 후 바인딩

패브릭 연결 GPU 준비 완료 전
파드 바인딩 대기 (기본 활성)

기본 600초 초과 또는 실패 조건 발생 시
할당을 비우고 재스케줄

Node Allocatable
v1.36 alpha
호스트 자원 반영

GPU 가 쓰는 호스트 메모리를
스케줄러 자원 계산에 포함

cgroup 과 OOM 점수까지 DRA 할당량 반영,
feature gate 기본 비활성

장치 상태 보고
v1.36
장애 원인 구분

GPU 장애가 장치 문제인지
애플리케이션 문제인지 구분

노드에 직접 들어가지 않고
ResourceSlice 에서 확인

Source: Kubernetes 공식 문서 How DRA Works (2026.06) · wikidocs 9장 · saaro (2026.08)

alpha/beta 표기 기능은 버전마다 단계가 달라 사용 전 해당 버전의 feature gate 문서 확인

CHAPTER 04

쿠버네티스 버전과 드라이버 조건으로 가르는 DRA 도입 판단

플랫폼별 지원 현황, device plugin 과의 공존 제약, 점검과 장애 대응

KEY INSIGHT **같은 DRA라도 드라이버와 노드 이미지 조건이 플랫폼마다 다름 —**
우리 클러스터 조합에서 쓰려는 GPU 종류와 공유 방식이 실제 지원되는지 먼저 확인해야

플랫폼 · 도구	최소 버전	지원 드라이버 · 장치	제약 사항
GKE	DRA 정식 지원	NVIDIA GPU 드라이버 Google TPU 용 DRA 드라이버	TPU 드라이버를 Google 이 커뮤니티에 기증 KubeCon EU 2026 발표
Amazon EKS	**1.34 이상**	NVIDIA GPU AWS Neuron (노드 AMI 1.34.2 이상)	**P6e-GB200 UltraServer 는 DRA 필수** Bottlerocket 노드 미지원 (Cast AI 기준)
AKS	**1.34 이상**	NVIDIA DRA 드라이버 25.8.1 A10 vGPU 분할 VM (1/6 · 1/3 · 1/2)	GRID 드라이버를 쓰는 A10 노드는 IMEX 관련 기능 비활성화 필요
Red Hat OpenShift	**4.21** (쿠버네티스 1.34)	DRA GA, NVIDIA DRA 드라이버 운영 사례	소스에 별도 제약 기재 없음 (saaro 인용)
NVIDIA NIM Operator	**1.34 이상**	전체 GPU · MIG 공유 방식은 Time-slicing 만 지원	**기술 프리뷰 단계** 완전 지원 전이라 운영 환경 배포 비권장
Cast AI 자동 확장	**1.34 이상**	EKS · GKE · AKS 의 NVIDIA GPU 에이전트 v0.109.0 이상	**DRA 경로에서 MIG 미지원** device plugin 과 같은 노드 공존 불가

MSAP.ai 플랫폼별 DRA 지원 조건(최소 버전 · 드라이버 · 공유 방식)을 한 화면에서 점검하고 GPU 배정·반환을 정책으로 관리

Source: Google Cloud Blog · AKS Blog · Cast AI Docs · AWS Neuron 문서 · NIM Operator 문서 · saaro
NVIDIA GPU DRA 드라이버는 쿠버네티스 SIG 로 이관 — 벤더 단독이 아닌 커뮤니티 관리 드라이버

같은 노드에서 함께 돌 수 없는 device plugin과 DRA 드라이버

KEY INSIGHT **노드 풀 단위로 전환 경계를 긋는 방식이 안전 —**
한 노드를 두 방식이 동시에 관리하면 같은 GPU가 이중으로 할당될 위험

하나의 클러스터 — 공존 가능

device plugin 노드 풀	DRA 노드 풀
기존 워크로드 유지	새 워크로드부터 이동
nvidia.com/gpu: 1	**nvidia.com/gpu.dra=true**
정수 요청으로 GPU 할당 DRA 노드는 affinity 로 제외	DRA kubelet 플러그인만 실행 ResourceClaim 으로 GPU 할당

전환 경계를 긋는 설정

기존 device plugin 의 nodeAffinity 에 조건 추가
nvidia.com/gpu.dra NotIn "true"

GKE 는 노드 라벨로 기본 플러그인 제외
gke-no-default-nvidia-gpu-device-plugin

gpuResourcesEnabledOverride=true 는 충돌 검사 생략
기존 플러그인이 없는 DRA 전용 노드에서만 사용

위험이 낮은 곳부터 옮기는 전환 순서

1. **GPU 할당은 device plugin 유지**
   멀티 노드 NVLink 용 ComputeDomain 만 DRA 로 활성화
   `resources.gpus.enabled=false`

2. **개발 환경과 추론 서비스를 DRA 노드 풀로 이동**
   DRA 의 GPU 할당이 완전히 지원된 뒤 시작
   공유 설정과 CEL 조건의 운영 경험 확보
   NVIDIA DRA 드라이버 GPU 기능의 지원 단계는 버전별로 확인

3. **핵심 학습 작업 이동**
   중단 비용이 큰 학습 작업은 마지막 단계에서 전환
   NVIDIA 와 쿠버네티스는 DRA 를 device plugin 의 최종 대체 방식으로 설계

Source: Cast AI Docs · AWS Neuron DRA 문서 (FAQ) · AWS AI on EKS (공존 전략) · AKS Engineering Blog

KEY INSIGHT **DeviceClass · ResourceSlice 부재는 드라이버 단계 결함 —**
점검을 드라이버 쪽부터 시작하면 원인 범위가 워크로드까지 번지기 전에 좁혀짐

드라이버 → 노드 → 워크로드 순

	점검 항목	확인 명령	정상 결과
01	**API 활성 · DeviceClass** 1.34 미만은 리소스 유형 없음 오류	`kubectl get deviceclasses`	gpu.nvidia.com 표시
02	**ResourceSlice 게시** 드라이버가 장치를 찾았는지 확인	`kubectl get resourceslices`	GPU 노드마다 1개 이상
03	**드라이버 파드** 컨트롤러와 kubelet 플러그인	`kubectl get pods -n nvidia-dra-driver-gpu`	두 파드 모두 Running
04	**노드 라벨** kubelet 플러그인 배치 조건	`kubectl get nodes -l nvidia.com/gpu.present=true`	대상 GPU 노드가 모두 조회됨
05	**CDI 런타임** GPU Operator 사용 시 (NIM 문서)	`kubectl get clusterpolicy cluster-policy -o json | jq .spec.cdi`	default · enabled 모두 true
06	**요청 상태** 워크로드 배포 후 확인	`kubectl get resourceclaim`	**STATE 가 allocated,reserved**

Source: AKS Engineering Blog (2026.03) · Cast AI Docs · NVIDIA NIM Operator 문서 (Troubleshooting)

네임스페이스 이름은 설치 방식에 따라 다름 — Cast AI 예제는 nvidia, AKS 예제는 nvidia-dra-driver-gpu

Pending에 멈춘 DRA 파드의 원인별 확인 경로

KEY INSIGHT **ResourceClaim 이 allocated,reserved 에 도달했는지가 분기점 —**
도달 전 실패는 요청 조건과 장치 재고, 도달 후 실패는 노드 쪽 장치 준비에서 원인 확인

할당 전 실패 — 요청 조건과 장치 재고 확인

DeviceClass 없음 `kubectl get deviceclasses`

드라이버 재설치 후 컨트롤러 로그에서 오류 확인

ResourceSlice 없음 `kubectl get nodes -l nvidia.com/gpu.dra=true`

대상 노드 존재 여부와 kubelet 플러그인 파드 로그 확인

Claim 미할당 `kubectl describe pod <파드 이름>`

이벤트에서 CEL 조건이 지나치게 좁은지,
조건에 맞는 미할당 장치 재고가 있는지 확인

할당 후 실패 — 노드 쪽 장치 준비 확인

Claim 할당 후에도 파드 대기

· spec.nodeName 을 직접 지정해 스케줄러를 우회했는지
· Binding Conditions 대기 시간(기본 600초)을 넘겼는지

`kubectl get resourceclaim <이름> -o yaml`

MPS 로 GPU를 공유하는 복제본

같은 GPU를 쓰는 복제본이 다른 노드로 흩어져
배치되지 않았는지 확인

`kubectl get pods -n <네임스페이스> -o wide`

Source: Cast AI Docs (Troubleshooting) · NIM Operator 문서 · Kubernetes 공식 문서 · AWS AI on EKS

THANK YOU

GPU를 속성으로 요청하는 방식으로 옮겨 갈 때 정할 DRA 전환 기준

v1.34 이상 버전 확인 · 노드 풀 단위 전환 경계 · 워크로드별 공유 기술 선택

Cloud Native Forum

Cloud Native Forum

https://cncf.co.kr

hello@cncf.co.kr

PUBLIC

APPENDIX 05

드라이버는 무엇을 하고, 장치는 어떻게 컨테이너에 꽂히는가

컴포넌트 구조 · Pod 기동 흐름 · CDI · API 오브젝트 · 구현체 비교

DRA 드라이버는 배정과 준비, 두 컴포넌트로 나뉜다

KEY INSIGHT **장치를 "누구에게 줄지 정하는 쪽"과 "노드에서 실제로 꽂아 주는 쪽"이 분리되어 있다 —**
그래서 실패도 배정 단계와 준비 단계 두 곳에서 따로 일어난다

컨트롤러 (controller)

배정 담당

역할	ResourceClaim에 맞는 노드와 장치를 확정한다
호출	Allocate · Deallocate
시점	스케줄러가 노드를 고른 뒤(또는 즉시 모드에서 먼저)
다루는 것	클레임 상태와 장치 재고

비유 · 렌터카 예약 센터 — 어느 차를 내줄지 결정하고 예약을 기록

kubelet 플러그인

준비 담당

역할	노드에서 장치를 준비해 컨테이너에 넘긴다
호출	NodePrepareResources · NodeUnprepareResources
시점	Pod 기동 직전과 종료 직후
산출물	CDI Spec 파일 (/var/run/cdi)

비유 · 차고 직원 — 키를 건네고, 반납 처리까지 담당

두 조각이 만나는 지점

API Server (ResourceClaim 저장) → **컨트롤러** (장치 배정 기록) → **kubelet** (준비 요청) → **kubelet 플러그인** (CDI 파일 생성) → **containerd** (컨테이너에 주입)

Source: KEP-3063 (sig-node) · kubernetes.io Blog (2022.12) · bells17, KEP 持ち寄り会 발표자료 (2024)

05 Pod이 기동하기까지 여섯 단계

KEY INSIGHT **요청은 클레임으로 적히고, 배정은 컨트롤러가, 준비는 kubelet이, 주입은 런타임이 맡는다 —**
종료할 때는 준비의 역순으로 정리되어 장치가 재고로 돌아간다

01 Pod 생성

ResourceClaimTemplate이 있으면 claim 컨트롤러가 ResourceClaim을 만든다

02 스케줄 대기

클레임이 아직 미할당이면 Pod은 Pending으로 남는다

03 노드 선정

스케줄러가 후보 노드를 고르고 클레임 조건과 맞춰 본다

다음 줄로

04 장치 배정

컨트롤러가 노드와 장치를 확정해 클레임에 기록한다 (Allocate)

05 노드 준비

kubelet이 NodePrepareResources를 호출, 플러그인이 CDI 파일을 쓴다

06 컨테이너 기동

런타임이 CDI 내용대로 컨테이너 설정에 장치를 주입한다

종료는 역순으로

Pod 종료
컨테이너 정리

→

NodeUnprepareResources
CDI 파일·장치 준비 해제

→

Deallocate
클레임의 배정 기록 삭제

→

장치 재고 복귀
다음 클레임이 사용 가능

Source: KEP-3063 (sig-node) 문서의 기동 흐름 · bells17, KEP 持ち寄り会 발표자료 (2024)

KEY INSIGHT **노드를 먼저 고르고 배정하면 낭비가 적고, 먼저 배정하면 확정은 빠르지만 자원이 묶인다 —**
alpha 시절의 두 모드는 1.34 GA에서 스케줄러가 재고를 직접 보는 방식으로 합쳐졌다

WaitForFirstConsumer

노드 먼저

Pod 생성 — 클레임 미할당

↓

노드 선정 — 스케줄러가 후보 결정

↓

장치 배정 — 고른 노드에 맞춰 Allocate

장점 노드·토폴로지에 맞는 장치를 고를 수 있어 낭비가 적다

한계 배정이 확정되기까지 단계가 길어 기동이 늦다

Immediate

배정 먼저

Pod 생성 — 클레임 생성 즉시

↓

장치 배정 — 노드와 장치를 먼저 확정

↓

노드 배치 — 확정된 노드로 스케줄

장점 배정 결과가 빨리 확정되어 흐름이 단순하다

한계 그 노드에 Pod을 못 올리면 장치가 묶인 채 남는다

1.34 GA에서는 어떻게 바뀌었나

ResourceSlice
드라이버가 노드별 장치 재고를 API에 게시한다

스케줄러가 직접 선택
재고와 CEL 조건을 보고 노드 선정과 장치 배정을 한 번에 처리

PodSchedulingContext 제거
alpha 시절 스케줄 조율용 리소스는 더 쓰지 않는다

Source: KEP-3063 (sig-node) · bells17 발표자료 (2024) · Kubernetes 1.34 릴리스 노트

Device Plugin으로 안 되던 일곱 가지를 DRA가 푸는 방식

KEY INSIGHT **Device Plugin은 장치를 개수로만 세고, 준비·정리 단계가 없다 —**
DRA는 속성 게시(ResourceSlice) · 클레임 수명주기 · 노드 준비 단계(CDI) 세 가지로 일곱 가지를 모두 해결한다

	Device Plugin으로는 안 되는 일	DRA가 가능하게 하는 방식
1	여러 Pod·컨테이너가 같은 장치를 함께 쓴다	한 ResourceClaim을 여러 Pod가 함께 참조한다
2	장치를 워크로드에 맞춰 재구성한다 (FPGA 등)	노드 준비 단계에서 드라이버가 장치 설정을 수행한다
3	워크로드가 끝나면 장치 설정을 정리한다	NodeUnprepareResources → Deallocate 가 순서대로 실행된다
4	장치를 부분만 할당한다 (MIG 동적 분할)	ResourceSlice가 분할 장치를 게시하고 claim 조건으로 고른다
5	장치에 옵션을 붙여 할당한다	DeviceClass·claim의 config로 드라이버 파라미터를 넘긴다
6	네트워크·패브릭 경유 장치를 다룬다	장치 속성과 토폴로지 조건으로 매칭한다 (ComputeDomain 등)
7	장치 단위 초기화를 수행한다	준비 단계에서 장치별 초기화를 드라이버가 실행한다

일곱 가지를 가능하게 한 세 가지 구조

장치를 속성으로 게시
ResourceSlice · attributes / capacity
개수가 아니라 조건으로 장치를 고를 수 있다 → 1·4·6번

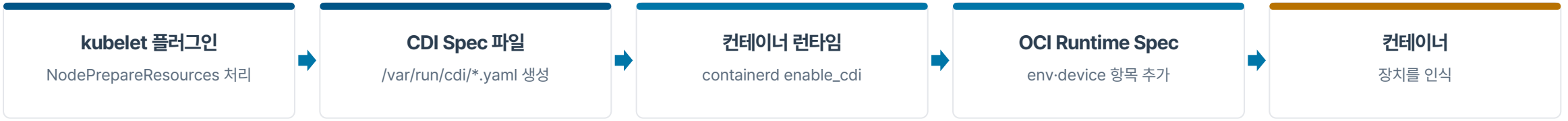
클레임 단위 수명주기
ResourceClaim · Allocate / Deallocate
배정과 반환이 API에 기록된다 → 3·5번

노드 준비 단계 분리
NodePrepareResources · CDI
초기화·재구성·주입을 드라이버가 수행한다 → 2·7번

Source: KEP-3063 (sig-node) README · bells17 발표자료 (2024)

CDI가 장치를 컨테이너 설정에 주입한다

KEY INSIGHT **배정이 끝나도 컨테이너가 장치를 보려면 마지막 한 칸이 필요하다 —**
kubelet 플러그인이 쓴 CDI 파일을 런타임이 읽어 컨테이너 기동 설정을 고친다



파일이 놓이는 곳

기본값은 /etc/cdi 또는 /var/run/cdi

CDI 규격에 맞는 설정 파일을 이 아래에 둔다

런타임에서 켜는 스위치

containerd는 enable_cdi 를 켜면 동작

CRI가 컨테이너 생성 시 CDI 장치를 반영한다

무엇이 바뀌는가

컨테이너 기동 설정(OCI Spec)을 고친다

수정 가능한 항목은 SPEC의 OCI Edits에 정리돼 있다

비유로 보면

예약과 배차가 끝나도 키를 차에 꽂아 주는 사람이 없으면 차는 움직이지 않는다 — CDI가 그 마지막 한 칸이다

확인 포인트

장치가 보이지 않으면 배정(ResourceClaim)이 아니라 노드의 CDI 파일과 런타임 설정을 먼저 본다

Source: CNCF Container Device Interface · containerd 소스(container_create_linux.go) · bells17 발표자료 (2024)

예제 드라이버는 배정 결과를 환경 변수로 보여 준다

KEY INSIGHT **예제 드라이버(dra-example-driver)는 배정된 장치를 환경 변수로 넣어 준다 —**
컨테이너 안의 값과 노드의 CDI 파일이 같은지 보면 어디까지 진행됐는지 알 수 있다

컨테이너 안에서

```
$ kubectl exec -it -n gpu-test4 pod0 -- env | grep GPU
GPU_NODE_NAME=dra-example-driver-cluster-worker
GPU_DEVICE_0=GPU-f11773a1-5bfb-e48b-3d98-...
GPU_DEVICE_1=GPU-93d37703-997c-c46f-a531-...
GPU_DEVICE_2=GPU-e7b42cb1-4fd8-91b2-bc77-...
GPU_DEVICE_3=GPU-ee3e4b55-fcda-44b8-0605-...
```

노드의 CDI 파일에서

```
# /var/run/cdi/k8s.gpu.resource.example.com-gpu_....yaml
cdiVersion: 0.3.0
devices:
- containerEdits:
    env:
    - GPU_DEVICE_0=GPU-f11773a1-5bfb-e48b-3d98-...
```

읽는 법

예제 드라이버는 실제 GPU를 쪼개지 않고 UUID를 환경 변수로 넣어 "어떤 장치가 배정됐는지"만 보여준다 — 실제 드라이버는 같은 자리에 장치 노드와 마운트를 넣는다

① 컨테이너 env
배정 결과가 주입됐는지

② 노드 CDI 파일
플러그인이 파일을 썼는지

③ 런타임 설정
enable_cdi 가 켜져 있는지

Source: kubernetes-sigs/dra-example-driver 데모 · bells17 발표자료 (2024)

KEY INSIGHT **이미 아는 볼륨 개념에 하나씩 대응시키면 DRA 오브젝트는 새로 외울 것이 거의 없다 —**
달라진 것은 노드별 장치 재고를 API에 올리는 ResourceSlice 한 가지다

볼륨에서 익숙한 것	DRA에서 대응하는 것	하는 일	요청 예 (YAML · 명령)
StorageClass	**DeviceClass (alpha: ResourceClass)**	드라이버와 기본 설정을 정의	`deviceClassName: gpu.nvidia.com`
PersistentVolumeClaim	**ResourceClaim**	장치 요구 한 건	`kind: ResourceClaim`
PVC 템플릿 (StatefulSet)	**ResourceClaimTemplate**	Pod마다 클레임 자동 생성	`resourceClaimTemplateName: tpl`
CSI Driver	**DRA Driver (컨트롤러 + 플러그인)**	배정 기록과 노드 준비·정리	`driver: gpu.nvidia.com`
대응 없음	**ResourceSlice**	노드별 장치 재고 게시	`kubectl get resourceslices`
kubelet Volume Manager	**kubelet의 준비 호출**	기동 전 준비 완료까지 대기	`NodePrepareResources (gRPC)`
대응 없음	**PodSchedulingContext (alpha)**	스케줄 조율 — GA에서 제거	— (GA에서 제거)

한 줄 요약
DRA의 구조는 볼륨 플러그인 + CSI Driver와 거의 같다 — 클래스로 종류를 고르고, 클레임으로 요청하고, 드라이버가 노드에서 준비한다

Source: KEP-3063 (sig-node) · Kubernetes 1.34 resource.k8s.io 문서 · bells17 발표자료 (2024)

KEY INSIGHT **요청에는 "몇 개"만 아니라 "어떤 장치를 어떻게 준비할지"까지 적는다 —**
적는 문법은 alpha와 1.34 GA에서 달라졌지만, 클래스와 클레임 두 곳에 붙인다는 원리는 같다

alpha (v1.26) 예시

지금은 쓰지 않는 문법

```
apiVersion: resource.k8s.io/v1alpha2
kind: ResourceClaim
spec:
  resourceClassName: gpu.example.com
  parametersRef:
    kind: GpuClaimParameters
    name: multiple-gpus
---
kind: GpuClaimParameters
spec:
  count: 4
```

1.34 GA에서는 이렇게 적는다

DeviceClass
드라이버와 기본 선택 조건(CEL)을 클래스에 정의한다

ResourceClaim · Template
requests에 deviceClassName과 조건·개수를 적는다

config
드라이버에 넘길 설정을 클래스와 클레임 양쪽에 붙인다

selectors (CEL)
"메모리 80GB 이상" 같은 조건으로 장치를 고른다

읽을 때 주의
alpha 자료의 parametersRef + 별도 CRD 방식은 GA에 없다 — 같은 목적을 DeviceClass·claim의 config와 CEL 조건이 대신한다

Source: bells17 발표자료 예시 (resource.k8s.io/v1alpha2) · Kubernetes 1.34 resource.k8s.io 문서

KEY INSIGHT **GPU가 없어도 kind와 예제 드라이버만으로 DRA의 전체 흐름을 손으로 확인할 수 있다 —**
개념을 확인한 뒤 실제 GPU용 드라이버로 옮겨 가는 순서가 가장 빠르다

dra-example-driver

kubernetes-sigs · 샘플

- GPU가 없어도 kind만 있으면 동작한다
- "GPU 1장을 여러 Pod가 공유"하는 예를 보여 준다
- 실제로 GPU를 쪼개지는 않는 학습용 구현

추천 · 개념과 흐름을 직접 확인할 때

NVIDIA k8s-dra-driver

NVIDIA · 실 환경

- NVIDIA GPU와 MIG 장치를 실제로 배정한다
- 리포지토리에 동작 설명 문서와 데모가 있다
- gpu-operator(노드 GPU 프로비저닝)와는 별개

추천 · 실 클러스터에 도입할 때

intel resource drivers

Intel · 실 환경

- Intel GPU 대상 DRA 드라이버
- 같은 조직의 device plugin 계열은 FPGA·QAT·SGX까지 지원
- 가속기 종류에 따라 지원 범위를 먼저 확인

추천 · Intel 가속기를 쓰는 환경

손으로 익히는 순서

01 kind 클러스터에 예제 드라이버 설치 → 02 데모 Pod을 띄워 클레임 상태 확인 → 03 컨테이너 env와 노드 CDI 파일 대조 → 04 실 GPU 노드에서 NVIDIA 드라이버로 전환

Source: kubernetes-sigs/dra-example-driver · NVIDIA/k8s-dra-driver · intel/intel-resource-drivers-for-kubernetes

KEY INSIGHT **DRA 자료는 대부분 alpha 시절(v1.26~) 기준이라 이름과 구조가 지금과 다르다 —**
아래 대응만 알고 보면 옛 자료의 설명도 그대로 쓸 수 있다

alpha 자료의 표현 (v1.26~)	1.34 GA에서는	읽을 때 주의
resource.k8s.io/v1alpha2	**resource.k8s.io/v1**	버전만이 아니라 필드 구조가 다르다
ResourceClass	**DeviceClass**	클래스가 CEL 선택 조건을 담는다
parametersRef + 별도 CRD	**config (클래스·클레임)**	CRD 없이 설정을 인라인으로 적는다
AllocationMode 두 가지	**스케줄러 기반 배정**	모드 선택 대신 재고를 보고 처리
PodSchedulingContext	**제거됨**	alpha 전용 조율 리소스였다
대응 없음	**ResourceSlice**	노드별 장치 재고 게시가 GA의 핵심
컨트롤러의 Allocate	**구조화 파라미터 기반 배정**	드라이버 컨트롤러 역할이 줄었다
NodePrepareResources	**그대로 유지**	노드 준비·정리 단계는 바뀌지 않았다

참고 자료
KEP-3063 README (sig-node) · kubernetes.io Blog 2022.12 · CNCF container-device-interface · kubernetes-sigs/dra-example-driver · bells17 발표자료 speakerdeck

Source: KEP-3063 (sig-node) · Kubernetes 1.34 릴리스 노트 · bells17 발표자료 (2024)