

[강연(Talk / 20-30분)]

UbuCon Korea 2026

“GPU 어디서 돌려요?”

Ubuntu + 컨테이너로 AI 학습부터 추론, 서빙까지 한방에



“GPU 어디서 돌려요?”

Ubuntu + 컨테이너로 AI 학습부터 추론, 서빙까지 한방에

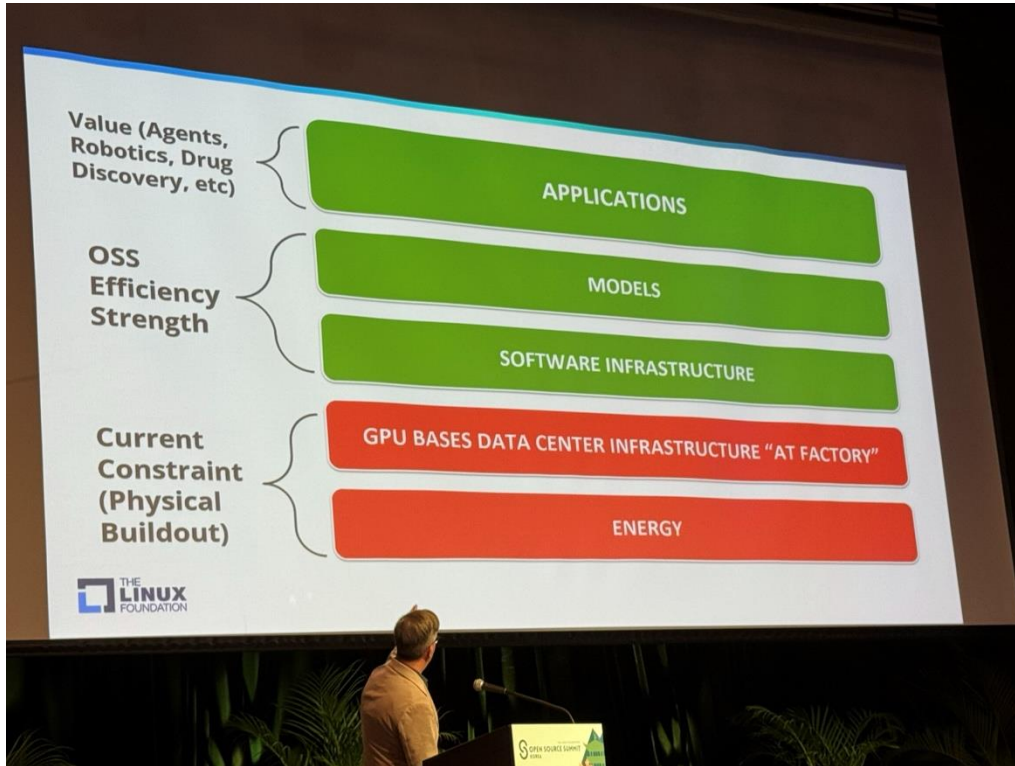
UbuCon Korea 2026 | 개발자와 인프라 엔지니어를 위한 실전 AI 인프라 구축기
2026. 8. 29.

최영락

Containers Specialist
AWS WWSO GTM AppMod



“Open Source in the Age of AI” vs. 두 가지 큰 장벽



Keynote by Jim Zemlin at “Open Source Summit Korea 2025” (Dec 2025)

1 GPU 인프라 구축

- 드라이버·CUDA·프레임워크 버전 조합의 미로
- 학습·추론·서빙마다 서로 다른 인프라 요구사항
- 팀·서버마다 제각각인 환경 — 재현이 안 되는 실험

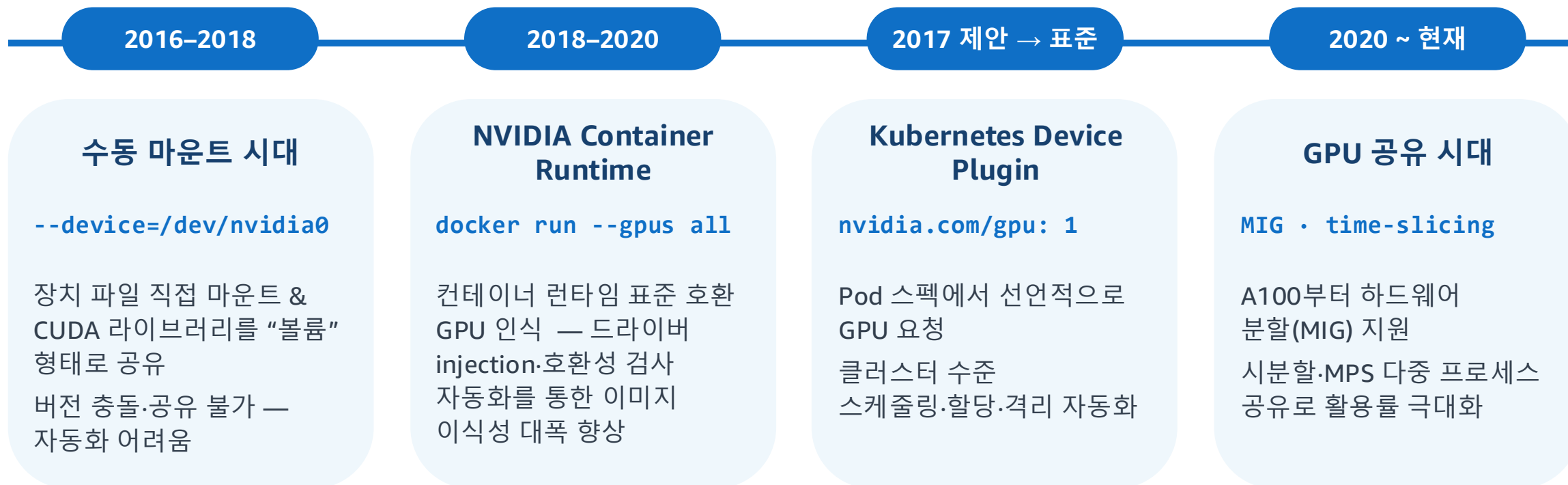
2 감당하기 힘든 비용

- GPU 인스턴스는 시간당 수 달러~수십 달러
- 유휴 상태의 GPU도 켜져 있는 동안 계속 과금
- 추론 서비스는 24시간 상시 구동 — 비용이 누적

이 세션의 접근

거창하고 복잡한 MLOps 플랫폼 이야기 대신,
AWS 클라우드 위에서 검증된 “컨테이너 기반의 실전 AI 인프라 구축기”를 공유

Context: 지난 10여년간 진화한 GPU 컨테이너 지원



이 진화의 전 과정을 Ubuntu 위에서 직접 겪어온 경험을 담아 — 오늘 Ubuntu + 컨테이너로 AI 학습부터 추론, 서빙까지 발표해보고자 합니다.

Agenda

1. AI 워크로드에서 Ubuntu를 선택하는 이유
2. GPU 인스턴스 유형별 선택 기준 — 학습 vs 추론
3. Docker 컨테이너로 CUDA 환경 구성하기
4. Amazon EKS + Karpenter로 GPU 워크로드 오케스트레이션
5. 학습된 모델을 컨테이너 기반으로 추론 서빙하는 패턴
6. Spot·MIG·GPU time-slicing을 활용한 비용 최적화 전략

1. AI 워크로드에서 Ubuntu를 선택하는 이유

AI 워크로드에서 Ubuntu를 선택하는 이유



CUDA·드라이버 1순위 지원

NVIDIA가 CUDA Toolkit과 GPU 드라이버를 Ubuntu LTS에 최우선으로 검증·배포합니다.

최신 GPU(H100, L4 등) 지원이 가장 빠르게 도착하는 배포판입니다.



AI 생태계의 사실상 표준

PyTorch·TensorFlow 공식 문서와 CUDA 베이스 컨테이너 이미지 대부분이 Ubuntu 기반입니다.

커뮤니티 트러블슈팅 자료가 가장 풍부해 문제 해결이 빠릅니다.



LTS 기반의 예측 가능한 운영

5년 지원 LTS 릴리스로 커널·드라이버 호환성을 장기간 고정할 수 있습니다.

Ubuntu Pro(ESM)로 보안 패치 기간을 10년까지 연장 가능합니다.



AWS와의 깊은 통합

공식 Ubuntu AMI, Deep Learning AMI, EKS 노드용 검증 이미지가 제공됩니다.

온프레미스에서 검증한 환경을 클라우드에 그대로 가져갈 수 있습니다.

AWS 위의 Ubuntu — 검증된 시작점

공식 Ubuntu AMI

- Canonical이 직접 빌드·검증하는 공식 이미지
- LTS 기준 정기 업데이트 제공
- EC2 콘솔·SSM 파라미터로 바로 조회 가능

AWS Deep Learning AMI

- NVIDIA 드라이버·CUDA·프레임워크 사전 구성
- Ubuntu 기반 — 설치 삽질 없이 바로 학습 시작
- GPU 인스턴스 계열별 호환성 사전 검증

EKS 노드용 Ubuntu

- EKS용으로 검증된 Ubuntu 노드 이미지 제공
- kubelet·컨테이너 런타임 사전 구성
- Karpenter에서 커스텀 AMI로 지정 가능

포인트 — AWS는 Amazon Linux·RHEL 등 다양한 OS를 동일하게 지원합니다.

개발자에게 가장 익숙한 Ubuntu조차 이렇게 완벽하게 받쳐주니, OS 셋업이 아니라 워크로드에 집중할 수 있습니다.

2. GPU 인스턴스, 무엇을 고를까

학습 vs 추론 — 워크로드 특성이 다르다

학습 (Training)

- 대규모 병렬 연산 — GPU 여러 장을 묶어 장시간 배치 작업
- 메모리·인터커넥트가 관건 — HBM 용량, NVLink·EFA 대역폭
- 중단 허용 가능 — 체크포인트만 있으면 재시작 OK
- 피크성 수요 — 실험 몰릴 때만 대량 필요

추론 (Inference)

- 낮은 지연시간이 목표 — 요청당 밀리초 단위 응답
- 트래픽 변동 대응 — 낮과 밤, 이벤트에 따라 출렁임
- 중단에 민감 — 상시 서비스 — 항상 켜져 있어야 함
- GPU 활용률이 낮기 쉬움 — 한 GPU를 다 못 쓰는 경우 다수

결론 — 학습과 추론은 인스턴스 선택, 스케일링 전략, 비용 전략이 모두 달라야 합니다.

GPU 인스턴스 유형별 선택 기준

인스턴스 계열	GPU	적합한 워크로드	특징
P5	NVIDIA H100	대규모 분산 학습, 파운데이션 모델	최고 성능, EFA 고대역 인터커넥트
P4d	NVIDIA A100	분산 학습, 대형 모델 파인튜닝	MIG 지원, 검증된 스테디셀러
G6e	NVIDIA L40S	중대형 LLM 추론	48GB 메모리로 중형 모델 서빙
G6	NVIDIA L4	추론, 경량 파인튜닝	비용 효율적 추론의 기본 선택지
G5	NVIDIA A10G	추론, 소규모 학습	폭넓게 검증됨, 가용성 좋음

* 사양·세대는 리전과 시점에 따라 다를 수 있습니다. 비용 효율 대안으로 AWS 자체 칩 기반 Inf2(추론)·Trn 시리즈(학습)도 검토할 만합니다.

3. Docker로 CUDA 환경 정리하기

의존성 지옥 — 왜 매번 환경이 깨질까

컨테이너가 없을 때

- GPU 드라이버 · CUDA Toolkit · cuDNN · 프레임워크, 4단 버전 매트릭스가 서로 얽힘
- 프로젝트마다 요구하는 CUDA 버전이 달라 서버 하나에 공존 불가
- OS 업그레이드·드라이버 패치 한 번에 전체 환경이 깨짐
- “제 서버에서는 됐는데요...” — 재현 불가능한 실험 결과

컨테이너로 정리하면

- 호스트에는 GPU 드라이버 하나만 관리
- CUDA·cuDNN·프레임워크는 전부 컨테이너 이미지 안으로
- 이미지 태그로 버전을 고정 — 언제 어디서나 동일하게 실행
- 프로젝트별로 서로 다른 CUDA 버전을 같은 서버에서 공존

NVIDIA Container Toolkit 동작 구조

컨테이너 A — CUDA 12.4 +
PyTorch

컨테이너 B — CUDA 11.8 +
TensorFlow

Docker Engine + NVIDIA Container Toolkit

Ubuntu 호스트 — NVIDIA 드라이버만 설치

GPU 하드웨어 (EC2 GPU 인스턴스)

핵심 아이디어

- Toolkit이 컨테이너 시작 시 GPU 디바이스와 드라이버 라이브러리를 자동 주입
- `docker run --gpus all` 플래그 하나로 GPU 노출
- 드라이버 호환 범위 안에서 컨테이너마다 다른 CUDA 버전 공존 가능
- Kubernetes에서는 NVIDIA device plugin이 같은 역할 수행

실전 — GPU 컨테이너 실행하기

1) NVIDIA Container Toolkit 설치 (Ubuntu)

```
sudo apt install -y nvidia-container-toolkit
sudo nvidia-ctk runtime configure --runtime=docker
sudo systemctl restart docker
```

2) GPU 컨테이너 실행

```
docker run --rm --gpus all \
  nvcr.io/nvidia/pytorch:24.05-py3 \
  python -c "import torch; print(torch.cuda.is_available())"
```

True # 컨테이너 안에서 GPU 인식 완료

포인트

- 호스트에 설치하는 것은 드라이버 + Toolkit 뿐
- CUDA·프레임워크는 NGC(nvcr.io) 베이스 이미지에서
- 팀 표준 베이스 이미지를 만들어 환경 통일
- ECR에 푸시하면 EC2·EKS 어디서든 동일하게 실행

4. EKS + Karpenter로 오케스트레이션

GPU 워크로드 오케스트레이션의 과제

GPU 노드는 비싸다

유휴 GPU 노드 한 대가 하루 수십만 원.
쓸 때만 띄우고, 안 쓰면 바로 회수해야 합니다.

수요가 들쭉날쭉하다

학습 실험이 몰리는 낮, 텅 비는 밤.
고정 크기 노드 그룹으로는 낭비 또는 대기가 발생합니다.

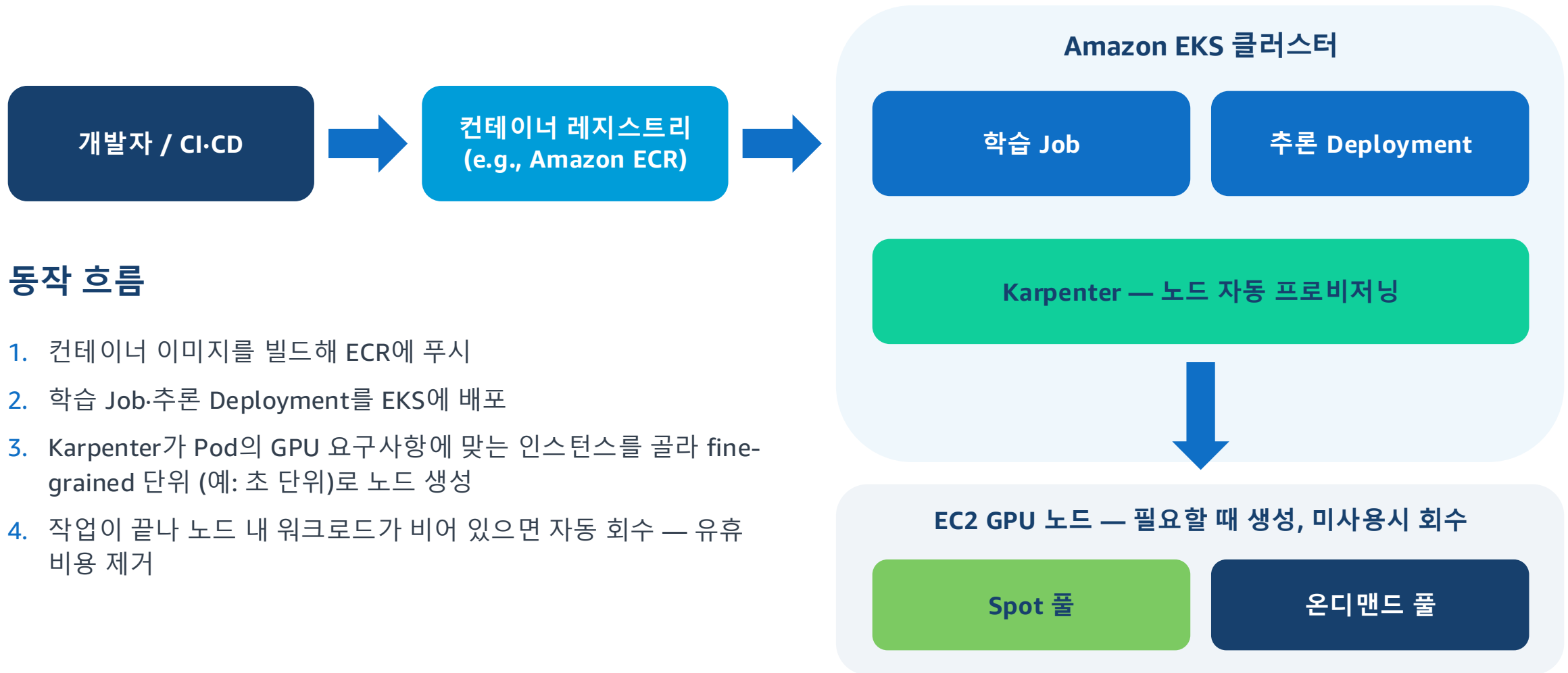
인스턴스가 다양하다

P 계열·G 계열, Spot·온디맨드, 리전·AZ별 가용성.
조합을 사람이 일일이 고르기 어렵습니다.

스케줄링이 복잡하다

GPU 리소스 요청, taint/toleration, 드라이버 준비까지.
일반 워크로드와 GPU 워크로드의 분리가 필요합니다.

전체 아키텍처 — EKS + Karpenter



Karpenter NodePool로 GPU 노드 자동 프로비저닝

```
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: gpu-pool
spec:
  template:
    spec:
      requirements:
        - key: karpenter.k8s.aws/instance-category
          operator: In
          values: ["g", "p"] # GPU 계열만
        - key: karpenter.sh/capacity-type
          operator: In
          values: ["spot", "on-demand"] # Spot 우선
      taints:
        - key: nvidia.com/gpu
          effect: NoSchedule
      disruption:
        consolidationPolicy: WhenEmptyOrUnderutilized
```

이 설정 하나로

- **필요할 때만 생성** — GPU Pod가 Pending되면 맞는 인스턴스를 골라 노드 기동
- **Spot 우선 + 풀백** — Spot 가용성이 없으면 온디맨드로 자동 전환
- **GPU 노드 보호** — taint로 일반 Pod가 비싼 GPU 노드에 끼어들지 못하게
- **자동 회수** — 노드가 비거나 저활용이면 통합·종료로 비용 절감

GPU 스케줄링 — Pod를 GPU에 올리기

```
# 학습 Job Pod 스펙 (발체)
spec:
  containers:
    - name: train
      image: <ECR>/train:v1.2
      resources:
        limits:
          nvidia.com/gpu: 1    # GPU 1장 요청
  tolerations:
    - key: nvidia.com/gpu
      operator: Exists
      effect: NoSchedule
  nodeSelector:
    karpenter.k8s.aws/instance-family: g6
```

동작 원리

- **device plugin** — NVIDIA device plugin이 GPU를 nvidia.com/gpu 리소스로 노출
- **toleration** — GPU 노드의 taint를 통과할 수 있는 Pod만 스케줄
- **nodeSelector** — 학습은 P 계열, 추론은 G 계열로 계열 지정 가능
- **드라이버 준비** — EKS GPU 최적화 이미지에 드라이버 포함 — 별도 설치 불필요

5. 학습부터 추론 서빙까지

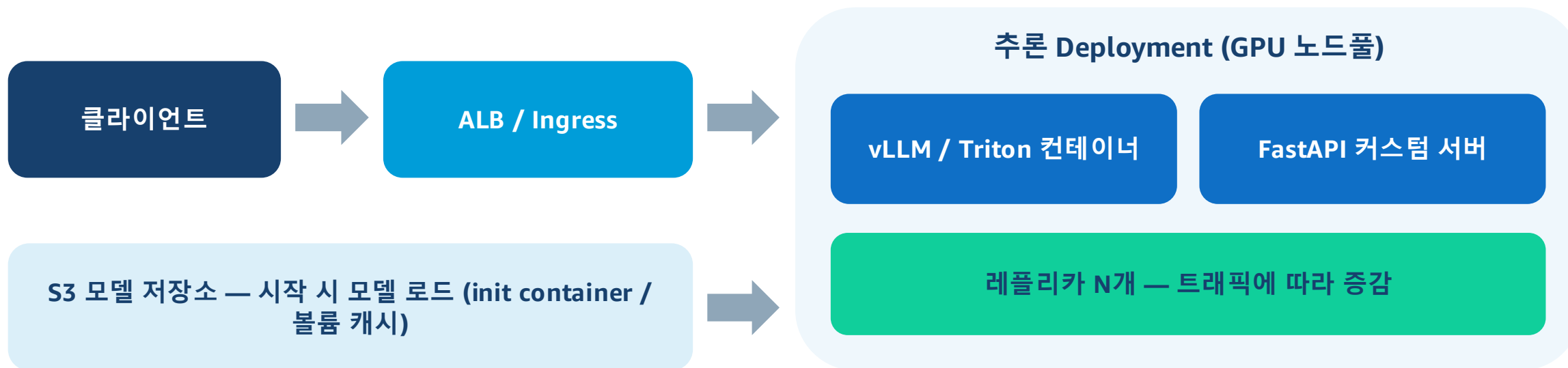
컨테이너 기반 학습 파이프라인



실무 포인트

- 체크포인트는 항상 S3에 — Spot 중단·장애 시 마지막 체크포인트에서 재개 — 학습 유실 방지
- Job이 끝나면 노드도 끝 — Karpenter가 빈 GPU 노드를 자동 회수해 유휴 과금 차단
- 파이프라인 도구는 선택 — cron·Argo Workflows·Kubeflow 무엇이든 — 핵심은 '컨테이너 + Job' 패턴
- 학습 이미지 그대로 서빙까지 — 학습에 쓰던 컨테이너 이미지를 코드 수정 없이 추론·프로덕션 서빙까지 그대로 사용

추론 서빙 아키텍처 패턴



패턴 정리

- 서빙 프레임워크 — LLM은 vLLM, 멀티 프레임워크는 Triton, 가벼운 모델은 FastAPI로 충분
- 모델과 이미지 분리 — 이미지는 서빙 엔진만, 모델 가중치는 S3에서 로드 — 배포 없이 모델 교체

추론 오토스케일링 전략

Pod 수준 — HPA

GPU 활용률.QPS.지연시간 같은 커스텀 메트릭 기반으로 레플리카 증감.

DCGM exporter + Prometheus 조합이 표준.

Pod 수준 — KEDA

큐 길이.이벤트 기반 스케일링, 트래픽이 없으면 0으로(scale-to-zero).

배치성 추론.비동기 처리에 특히 효과적.

노드 수준 — Karpenter

Pod가 늘면 GPU 노드를 자동 증설, 줄면 통합.회수.

Pod 스케일링과 노드 스케일링의 역할 분담이 핵심.

콜드 스타트 줄이기

이미지 슬리밍, 모델 볼륨 캐시, 워م 풀 유지로 첫 응답 시간 단축.

대형 모델일수록 준비 시간 설계가 중요.

프로덕션에서 챙길 것 — 보안·네트워킹·스토리지

보안 (IAM·이미지)

- IRSA로 Pod 단위 최소 권한 — 노드가 아닌 워크로드에 IAM 역할
- ECR 이미지 스캔으로 취약점 차단
- 자격 증명은 Secrets Manager로 관리

네트워킹 (VPC)

- GPU 노드는 프라이빗 서브넷에, S3·ECR은 VPC 엔드포인트로
- 분산 학습은 EFA로 노드 간 고대역 통신
- 추론 엔드포인트는 ALB + WAF로 보호

스토리지 (데이터)

- 학습 데이터는 S3 + FSx for Lustre로 GPU에 고속 공급
- 노드 로컬 캐시는 EBS gp3
- 모델·체크포인트 공유는 EFS 또는 S3

“컨테이너로 띄우면 끝”이 아닙니다 — VPC·IAM·데이터 파이프라인까지 묶여야 프로덕션입니다.

6. GPU 비용, 현실적으로 줄이기

GPU 비용의 현실

p5.48xlarge

H100 × 8

~\$98/시간

한 달 상시 구동 시 약 \$70,000

p4d.24xlarge

A100 × 8

~\$33/시간

한 달 상시 구동 시 약 \$24,000

g6.xlarge

L4 × 1

~\$0.8/시간

한 달 상시 구동 시 약 \$580

학습 클러스터 하나가 월 수천만 원 — GPU 비용 최적화는 선택이 아니라 필수입니다.

* 온디맨드, us-east-1 기준 대략치 — 리전·시점에 따라 다릅니다.



Spot 인스턴스 — 최대 90% 절감

무엇이고 얼마나 아끼나

- EC2의 미사용 용량을 온디맨드 대비 최대 90% 할인된 가격에 사용
- 대신 AWS가 2분 전 통지 후 회수할 수 있음
- GPU 학습·배치 작업과 궁합이 가장 좋은 절감 수단

최대 90% ↓

온디맨드 대비 Spot 할인율

중단 대응 플레이북

- 체크포인트를 주기적으로 S3에 저장 — 중단 후 재개
- 2분 중단 통지를 받아 정리 작업 수행 (node termination handler)
- Karpenter에 다양한 인스턴스 계열·AZ를 열어두어 대체 용량 확보
- 학습·배치는 Spot, 상시 추론은 온디맨드+Spot 혼합으로 안정성 확보
- 재시도 로직이 있는 Job 컨트롤러 사용 (backoffLimit)

MIG — 하드웨어 수준의 GPU 분할



최대 7개의 독립 GPU 인스턴스로 분할

- 파티션마다 메모리·연산 유닛이 하드웨어 수준으로 완전 격리
- 이웃 워크로드의 간섭 없음 — 예측 가능한 성능

Kubernetes에서 쓰는 법

- device plugin이 MIG 파티션을 개별 리소스로 노출 — nvidia.com/mig-1g.10gb: 1 로 요청
- 작은 추론 서비스 여러 개를 GPU 한 장에 안전하게 배치
- P4d(A100)·P5(H100) 등 MIG 지원 인스턴스에서 사용 가능

적합 — 격리가 필요한 멀티테넌트 추론, 소형 모델 다수 서빙

부적합 — GPU 전체가 필요한 대규모 학습

GPU time-slicing — 소프트웨어 수준의 공유

어떻게 동작하나

- GPU 한 장을 여러 Pod가 시간 단위로 번갈아 사용
- device plugin 설정만으로 활성화 — 하드웨어 요구사항 없음
- 모든 NVIDIA GPU에서 사용 가능 (T4, A10G, L4 포함)

주의할 점

- 메모리 격리 없음 — 한 Pod의 OOM이 이웃에 영향
- 성능 보장 없음 — 지연시간에 민감한 프로덕션에는 부적합
- 개발.테스트, 저활용 내부 추론에 적합

```
# device plugin ConfigMap
apiVersion: v1
kind: ConfigMap
metadata:
  name: time-slicing-config
data:
  any: |-
    version: v1
    sharing:
      timeSlicing:
        resources:
          - name: nvidia.com/gpu
            replicas: 4 # 1장을 4개로
```

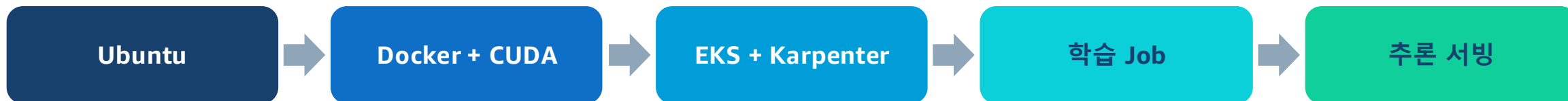
GPU 1장이 스케줄러에는 4개로 보임 — 활용률 4배

세 가지 전략, 언제 무엇을 쓸까

전략	절감 방식	격리 수준	적합한 워크로드	주의점
Spot	미사용 용량 할인 (최대 90%)	노드 전체 사용 (격리 무관)	학습, 배치, 중단 허용 작업	2분 통지 후 회수 체크포인트 필수
MIG	GPU 1장을 최대 7분할 (활용률 ↑)	하드웨어 수준 완전 격리	멀티테넌트 추론, 소형 모델 다수	A100·H100 등 지원 GPU 한정
Time-slicing	GPU 1장을 N개 Pod가 시분할 공유	격리 없음	개발·테스트, 저활용 내부 추론	메모리·성능 보장 없음

실전 조합: 학습은 Spot으로, 추론은 MIG로 밀도를 높이고, 개발 환경은 time-slicing으로 — 세 전략은 배타적이지 않습니다.

핵심 메시지



① 아키텍처 레퍼런스

Ubuntu → Docker(CUDA) → EKS + Karpenter → 컨테이너 서빙으로 이어지는 GPU AI 파이프라인 전체 그림

② 오케스트레이션 모범 사례

NodePool 요구사항, taint/toleration, 자동 통합·회수까지 — GPU 워크로드를 위한 EKS + Karpenter 패턴

③ 바로 쓰는 비용 절감

학습은 Spot, 서빙 밀도는 MIG, 개발 환경은 time-slicing — 오늘 바로 적용할 수 있는 절감 전략

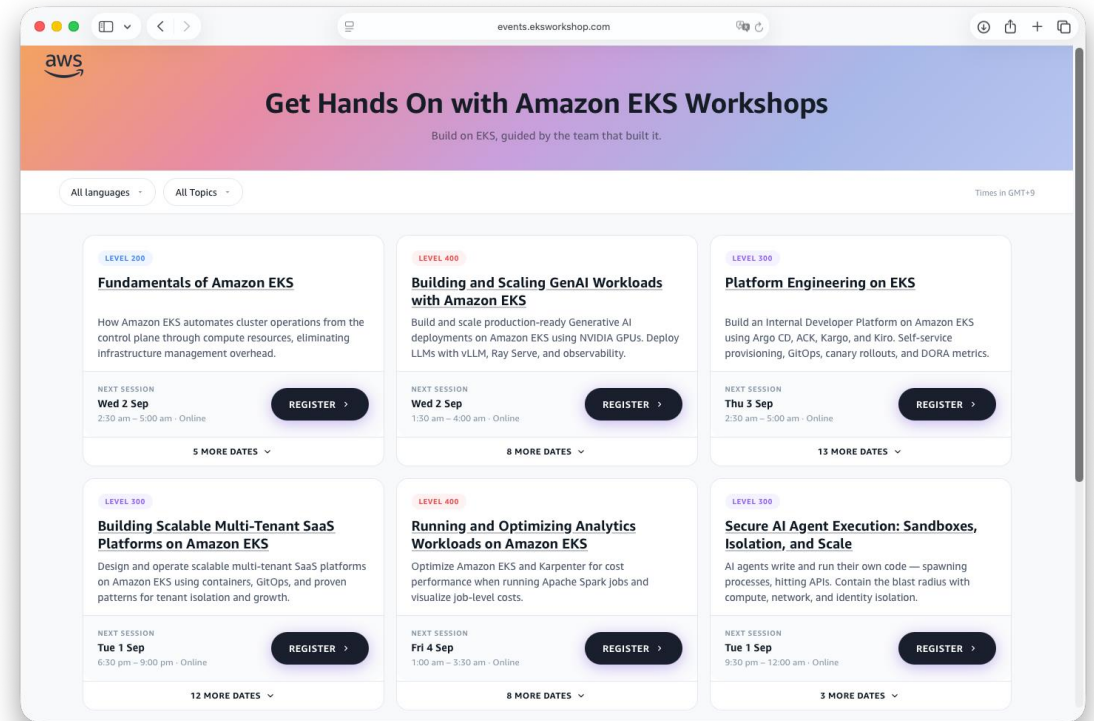
GPU 인프라, 거창할 필요 없습니다 — Ubuntu와 컨테이너, 그리고 AWS라면 오늘 바로 시작할 수 있습니다.

오늘 배운 것, 직접 핸드폰을 통해 확인해보세요



EKS 워크숍으로 직접 확인

- events.eksworkshop.com — AWS 공식 핸드온 워크숍 포털
- 오늘 다룬 내용을 포함해 다양한 AI/EKS 워크숍을 골라 실습할 수 있습니다
- 필요한 것은 AWS 계정과 터미널뿐



슬라이드는 잊혀도, 직접 해본 것은 남습니다.

Thank you!

