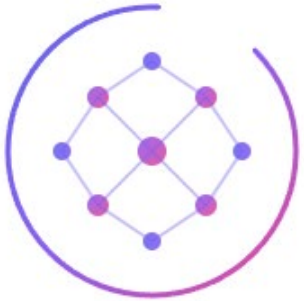


Open LLM with CPU

Open Source Language Models

DUTBOX__

소개

과정 및 강사

목차

쿠버네티스 101

목차

강사

강사

강사

이름: 최국현

메일: tang@linux.com, 회신은 다른 메일 주소로 드리고 있습니다. :)

사이트: tang.dustbox.kr

언제든지 질문 및 요청 환영 입니다.

LAB

랩 및 교육대상

DUSTBOX

LAB

강사 설명

AI 소개

앤서블 기능

AI 소개-비용

최근 인공지능(AI)은 대규모 연산 자원과 고가의 GPU를 기반으로 발전해 왔습니다. 그러나 **AI의 본질은 알고리즘의 개방성과 효율성**에 있으며, 반드시 GPU만이 답은 아닙니다.

오픈소스 AI(Open Source AI) 는 이러한 문제의식을 바탕으로, 누구나 접근 가능한 **개방형 모델과 경량화된 실행 구조**를 통해 인공지능의 민주화를 이끌고 있습니다.

AI 소개-오픈소스 엔진

기존에는 GPU만이 대규모 AI 모델을 실행할 수 있다고 여겨졌지만, 최근에는 **CPU 기반 최적화 기술(OpenVINO, llama.cpp, GGUF, quantization 기법 등)** 을 통해

비용 효율적이고 유지보수가 용이한 AI 환경을 구축하는 사례가 늘고 있습니다.

- **장점**: 저비용, 전력 효율, 유지보수 용이성, 서버 자원 재활용 가능
- **활용 기술**: quantized 모델(4bit/8bit), SIMD 명령어 최적화, NUMA-aware scheduling
- **대표 사례**: Intel OpenVINO, llama.cpp, MLX, RWKV-Runner 등

AI 소개 - GPU/CPU

GPU는 여전히 대규모 병렬 연산과 고속 학습(Training)에 있어 핵심적인 역할을 담당합니다. 반면 CPU는 **추론(Inference)** 중심 환경에서 효율적인 대안으로 부상하고 있습니다.

구분	GPU 기반	CPU 기반
주요 용도	대규모 학습(Training), 딥러닝 모델 훈련	경량 추론(Inference), 모델 배포
특징	높은 연산 병렬성, 고비용	범용성, 저비용, 유지보수 용이
장비 요구	고성능 GPU, 냉각/전력 인프라 필요	일반 서버 또는 VM 환경에서도 운영 가능
대표 기술	CUDA, cuDNN, TensorRT	OpenVINO, llama.cpp, oneDNN

AI 소개-민주주의

오픈소스 AI는 **폐쇄된 상용 모델(GPT, Claude 등)** 과 달리, 모델 구조·가중치·추론 코드가 공개되어 있으며, 커뮤니티 중심의 지속적 개선이 가능합니다.

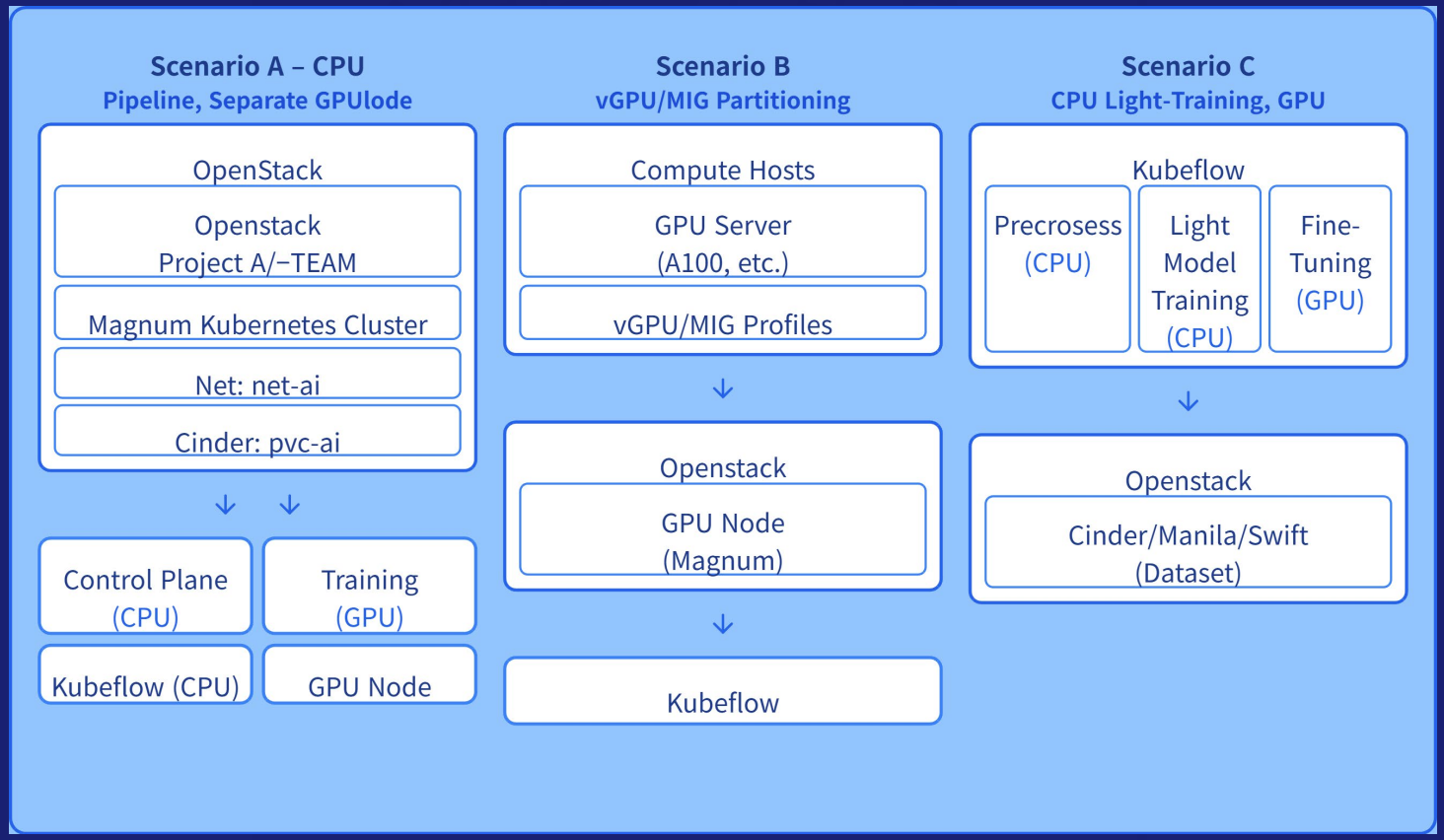
"AI의 진보는 데이터와 코드의 개방에서 시작된다."

오픈소스 생태계는 **AI의 민주화, 독립성, 지속가능성**을 가능하게 합니다.

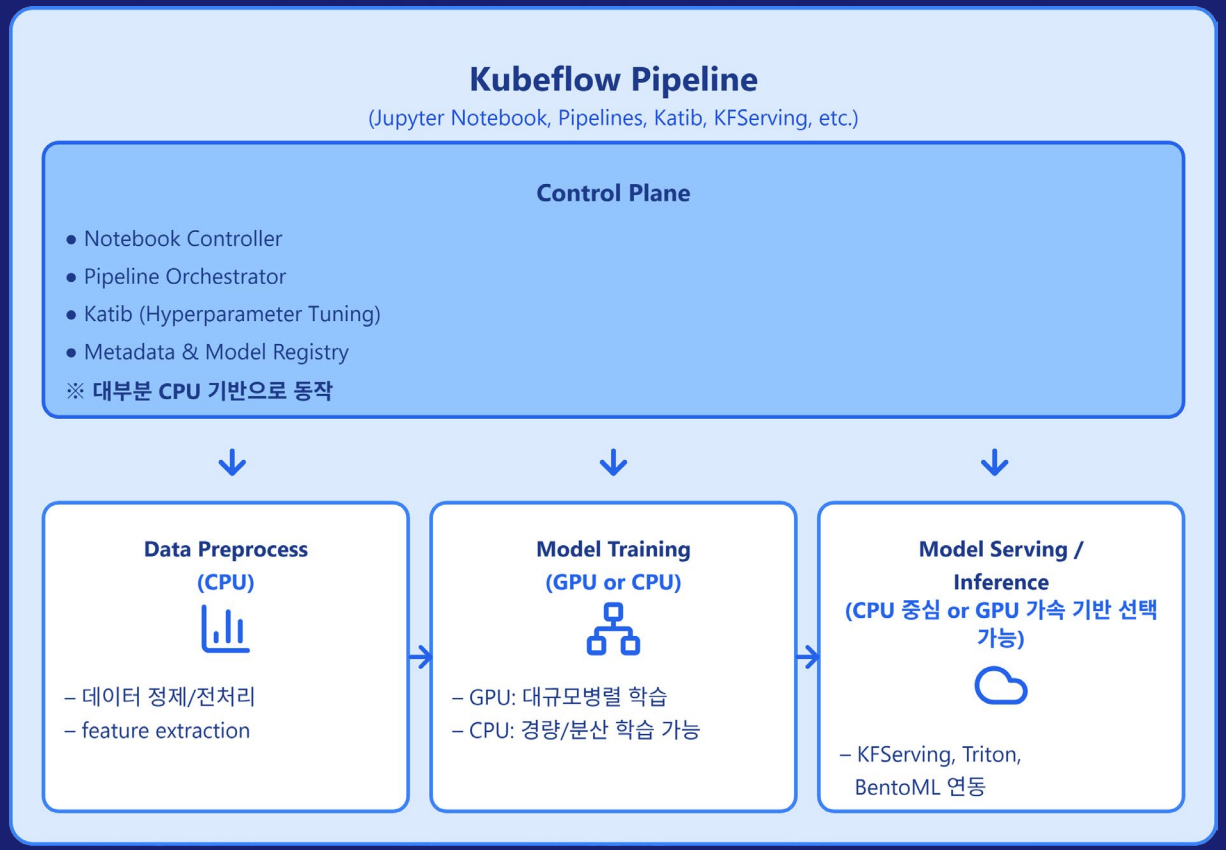
AI 기능 소개

구분	주요 역할	기본 리소스	GPU 활용 여부
Kubeflow Control Plane	파이프라인 관리, 튜닝, 메타데이터 관리	CPU	GPU 불필요
데이터 전처리	CSV, 이미지 등 정규화·클리닝	CPU	가끔 가능하지만 드묾
모델 학습(Training)	신경망 모델 학습	GPU (기본)/CPU (가능)	GPU 중심
모델 검증 및 배포(Serving)	Inference, KFServing	CPU (기본)/GPU (선택적)	옵션

오픈스택 AI


Scenario A – CPU
Pipeline, Separate GPUlode
OpenStack
Openstack
Project A/–TEAM
Magnum Kubernetes Cluster
Net: net-ai
Cinder: pvc-ai
Control Plane
(CPU)
Training
(GPU)
Kubeflow (CPU)
GPU Node
Scenario B
vGPU/MIG Partitioning
Compute Hosts
GPU Server
(A100, etc.)
vGPU/MIG Profiles
Openstack
GPU Node
(Magnum)
Kubeflow
Scenario C
CPU Light-Training, GPU
Kubeflow
Precrosess
(CPU)
Light
Model
Training
(CPU)
Fine-
Tuning
(GPU)
Openstack
Cinder/Manila/Swift
(Dataset)

쿠버네티스 AI


Kubeflow Pipeline
(Jupyter Notebook, Pipelines, Katib, KFServing, etc.)
Control Plane
• Notebook Controller
• Pipeline Orchestrator
• Katib (Hyperparameter Tuning)
• Metadata & Model Registry
※ 대부분 CPU 기반으로 동작
Data Preprocess
(CPU)
– 데이터 정제/전처리
– feature extraction
Model Training
(GPU or CPU)
– GPU: 대규모병렬 학습
– CPU: 경량/분산 학습 가능
Model Serving /
Inference
(CPU 중심 or GPU 가속 기반 선택 가능)
– KFServing, Triton,
BentoML 연동

결론

GPU 중심의 시대는 여전히 이어지고 있지만, **CPU 기반 오픈소스 AI는 '접근 가능한 인공지능'으로의 패러다임 전환**을 이끌고 있습니다.
이제 AI는 **'누가 더 큰 GPU를 가졌는가'**가 아니라, **'누가 더 효율적이고 개방적인 시스템을 설계하는가'**의 경쟁으로 바뀌고 있습니다.

활용코드

앤서블

오픈스택

에이전트 기능

AI기반으로 시스템 운영을 위해서 LLM 에이전트가 필요하다. LLM 에이전트는 다음과 같은 역할을 지원한다.

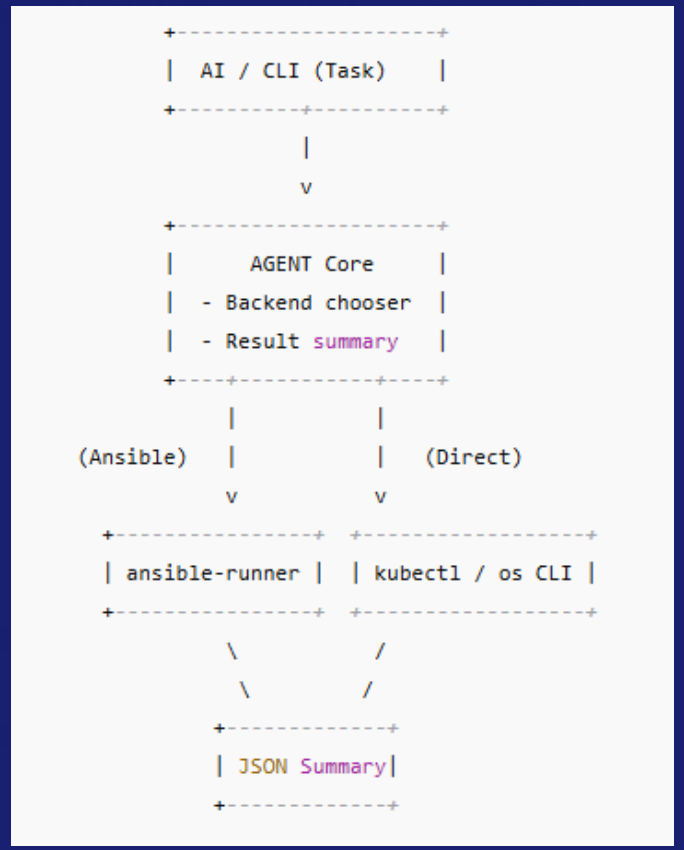
1. 앤서블 플레이북 생성 및 실행
2. **오픈스택/쿠버네티스** 명령어 실행
3. GUFF와 같은 학습된 데이터를 통한 자연어 인식
4. 플레이북 혹은 **오픈스택/쿠버네티스**에 직접적으로 API 호출 및 실행
5. 자연어를 통한 YAML코드 생성

동작환경

현재 에이전트는 vCPU 기반으로 동작. 동작하는 인스턴스의 사양은 다음과 같이 구성이 되어있음.

- **CPU:** 8 vcpu
- **MEM:** 32 GiB
- DISK: 20GiB+

구조


AI / CLI (Task)
AGENT Core
- Backend chooser
- Result summary
(Ansible)
(Direct)
ansible-runner
kubectl / os CLI
JSON Summary

LLM 에이전트 AI

현재 사용하는 에이전트 AI는 두 가지를 지원하고 있다.

1. llama.cpp
2. OpenVINO

현재는 손쉽게 사용하기 위해서 llama.cpp를 사용하고 있지만, 라이선스 및 CPU기반에서 고성능을 위해서 추후에 OpenVINO로 변경할 예정이다.

설치

설치는 다음과 같이 진행한다.

```
# dnf install python3.11
# python3.11 -m venv .venv
# source .venv/bin/activate
# python -m pip install -U pip
# python -m pip install -r requirements.txt
```

실행

설치가 완료가 되면 다음과 같이 실행이 가능하다.

```
# python3 agent_openai.py --inventory localhost, --task "Kubernetes nginx 배포
" --verify
# python3 agent_openai.py \
  --backend direct \
  --k8s-file k8s/deploy.yaml \
  --task "apply k8s" \
  --verify
# python3 agent_openai.py \
  --backend direct \
  --openstack-op ensure-network \
  --openstack-args name=net-infra cidr=192.168.10.0/24 \
  --task "ensure os net"
```

인자설명

`--backend {auto|ansible|direct}`: 백엔드 선택(기본 auto)

`--inventory`: Ansible 인벤토리(기본 localhost,)

`--task`: Ansible일 땐 플레이북 경로, Direct일 땐 설명 문자열

`--verify`: 멱등성 체크(2회 실행)

Direct 전용

```
--k8s-file <file.yaml>
--openstack-op <op> (예: ensure-network)
--openstack-args key=value ...
```

핵심 함수(1): Ansible 경로

`self._runner(playbook_name):`

- ansible-playbook -i <inventory> <playbook> 실행 → rc/stdout 수집

`self._collect_summary_errors(r):`

- stdout의 **마지막 줄**이 JSON이면 summary로 파싱(없으면 None)

`run_playbook(playbook_name) → (ok, summary, errors):`

- summary가 없을 때도 **가드 처리**하고 rc로 성공 판정
- stats가 있으면 failed/unreachable==0 확인

`run_idempotency_check(playbook_name) → (idem_ok, {"first":s1,"second":s2}):`

- 2회 실행 후 **second**에서 changed==0 && failed==0 && unreachable==0 기대

핵심 함수(2): Direct 경로

`run_cmd_with_summary(cmd) → (rc, summary, stdout, stderr)`

- 외부 스크립트가 **마지막 줄**에 JSON 요약을 출력하도록 요구

`direct_k8s_apply(manifest) → (ok, summary)`

- adapters/k8s_apply.sh 호출
- 내부에서 kubectl diff/apply 후 요약 JSON 출력

`direct_openstack_op(op,args) → (ok, summary)`

- adapters/os_ensure_network.sh 등 호출
- 존재 확인→생성/갱신→요약 출력 패턴

`choose_backend(args) → "ansible"|"direct"`

- --k8s-file or --openstack-op 있으면 direct, 아니면 ansible

`run_direct_task(args) → (ok, summary):`

- k8s 또는 openstack 어댑터로 라우팅

어댑터 스크립트(Direct)

adapters/k8s_apply.sh

- kubectl diff -f → 변경 감지(changed=1/0)
- kubectl apply --server-side → 상태 적용
- 필요 시 kubectl wait deploy/...(옵션) → 안정화
- 마지막 줄에 {"stats":{"localhost":{"changed":X,"failed":Y,"unreachable":0}}} 출력

adapters/os_ensure_network.sh

- openstack network/subnet show로 존재 확인
- 없으면 create 실행 → changed=1
- 마지막 줄에 동일한 요약 JSON 출력

두 경로 모두 **동일 포맷의 요약 JSON**을 출력하므로 상위 로직을 재사용 가능. 아직 구현 예정…

제어 흐름(Sequence)

```
[CLI] --task/--backend/--verify
   |
   v
[agent.run_task]
   |
   +--> choose_backend(args)
         |
         +--"ansible"--> run_playbook()
         |                  |
         |                  +--(verify)--> run_idempotency_check()
         |
         +--"direct"----> run_direct_task()
                            |
                            +--(verify)--> run_direct_task() 2nd
```

오류/예외 처리 원칙

요약 미존재(summary=None)

- 크래시 금지 → rc로만 **보수적 판정**

통계(stats) 구조 변경/누락

- isinstance(stats, dict)로 확인 후만 접근

멱등성 실패

- 2nd 실행의 changed>0 또는 failed/unreachable>0이면 NOT OK

확장/마이그레이션 (Go 전환)

1. **Go 리팩토링** 시 동일 구조 유지(Strategy 패턴: ansible vs direct)
2. **client-go/gophercloud**로 Direct 네이티브화 가능 (초기에는 기존 쉘 어댑터 재사용)
3. 공통 요약 JSON 유지 → 상위 로직/리포팅 변경 불필요
4. 단일 바이너리/동시성/배포 이점 확보