

Inference Anywhere - GPU와 WebAssembly로 만든 쿠버네티스 엣지 AI 배포 사례

강상진

Principal Technical Solutions Architect
Akamai Technologies, Cloud Technology Group APJ
sangjinn@gmail.com



오늘의 주제

글로벌하게 배포된 플랫폼

+

빠른 성능의 애플리케이션

→

쿠버네티스 기반의
WebAssembly로 구동되는
서버리스 함수와
GPU의 활용

리전 vs 엣지 플랫폼



리전 - 전 세계 주요 도시의
대규모 데이터센터



엣지 - 리전이 없는 음영 지역 중,
사용자 트래픽이 발생하는 위치

리전 vs 엣지 플랫폼

리전 기반 (중앙 클라우드)

- 대도시 데이터센터 / 중앙 집중형 자원 관리
- 물리적 거리는 보통 수백 km 이상
- 50ms 이상 (대륙 간 이동 시 ~200ms)
- 강력한 컴퓨팅/스토리지 제공
- 사용자와 물리적 거리가 멀수록 응답 속도 느림
- 대규모 AI 학습, 빅데이터 처리, 중앙 집중 서비스

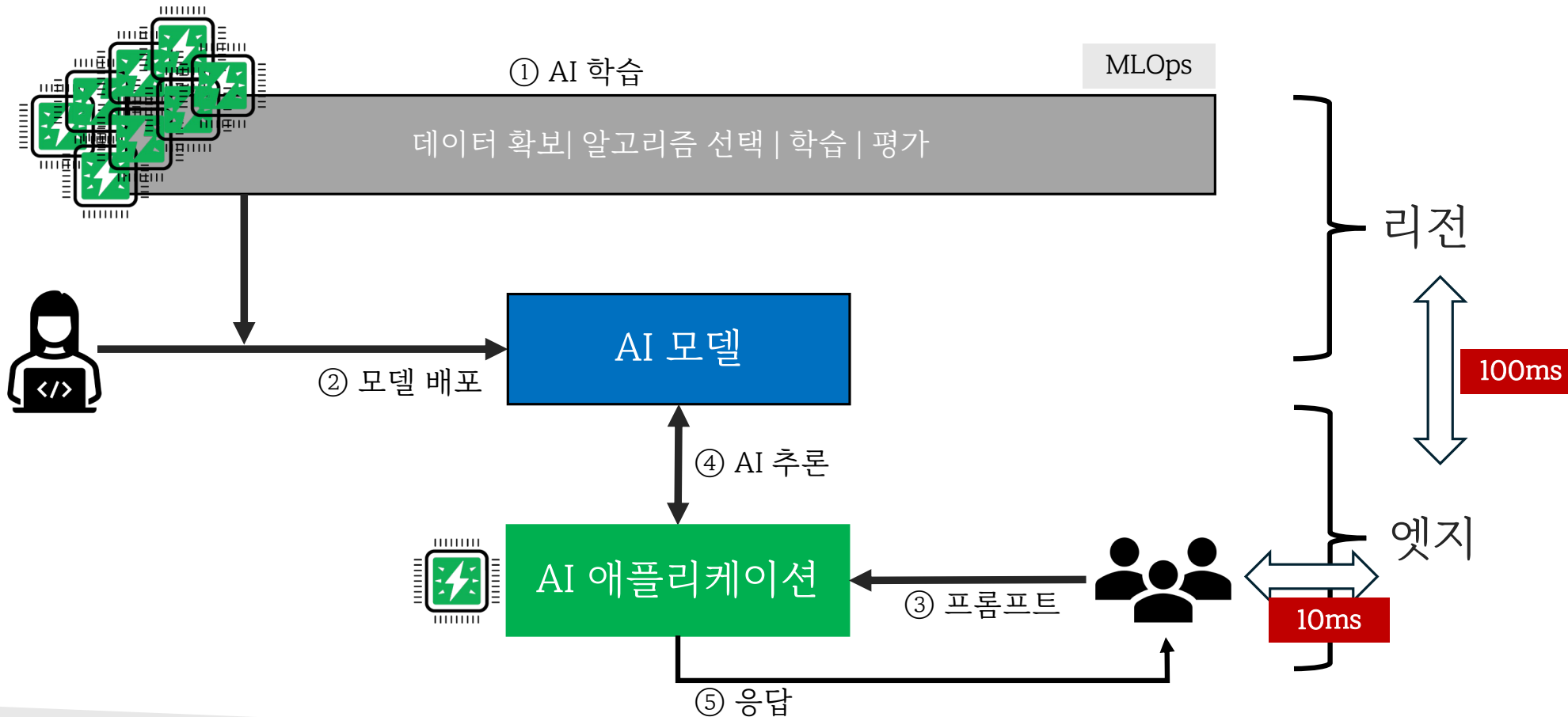
GPU 리소스를 활용한 AI 학습

엣지 기반 (분산 플랫폼)

- 사용자 가까운 PoP/마이크로 데이터센터
- 물리적 거리는 보통 수 km ~ 수십 km 이내
- 실시간 처리, 초저지연 (1~20 ms)
- 네트워크 트래픽 절감 및 개인정보 보호 강화
- 자원 규모는 제한적이나 빠른 응답
- 실시간 게임, IoT, 자율주행, 영상 스트리밍

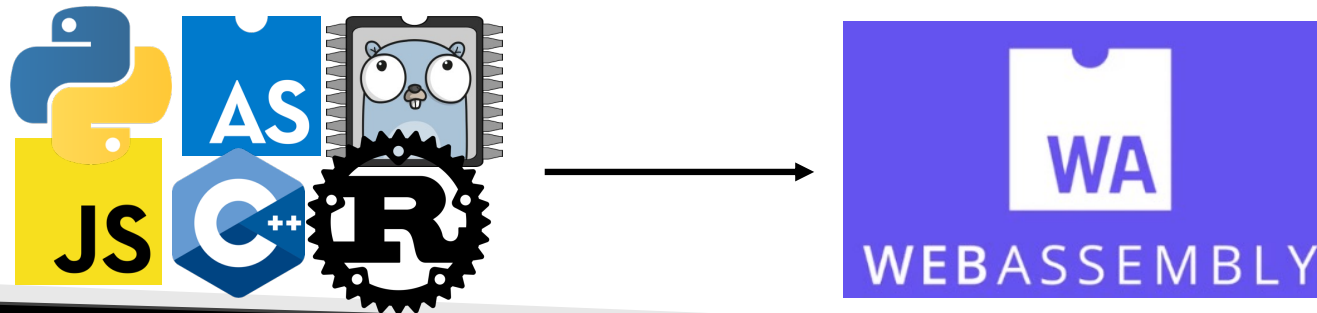
빠른 연산을 활용한 AI 추론

리전 vs 엣지 플랫폼

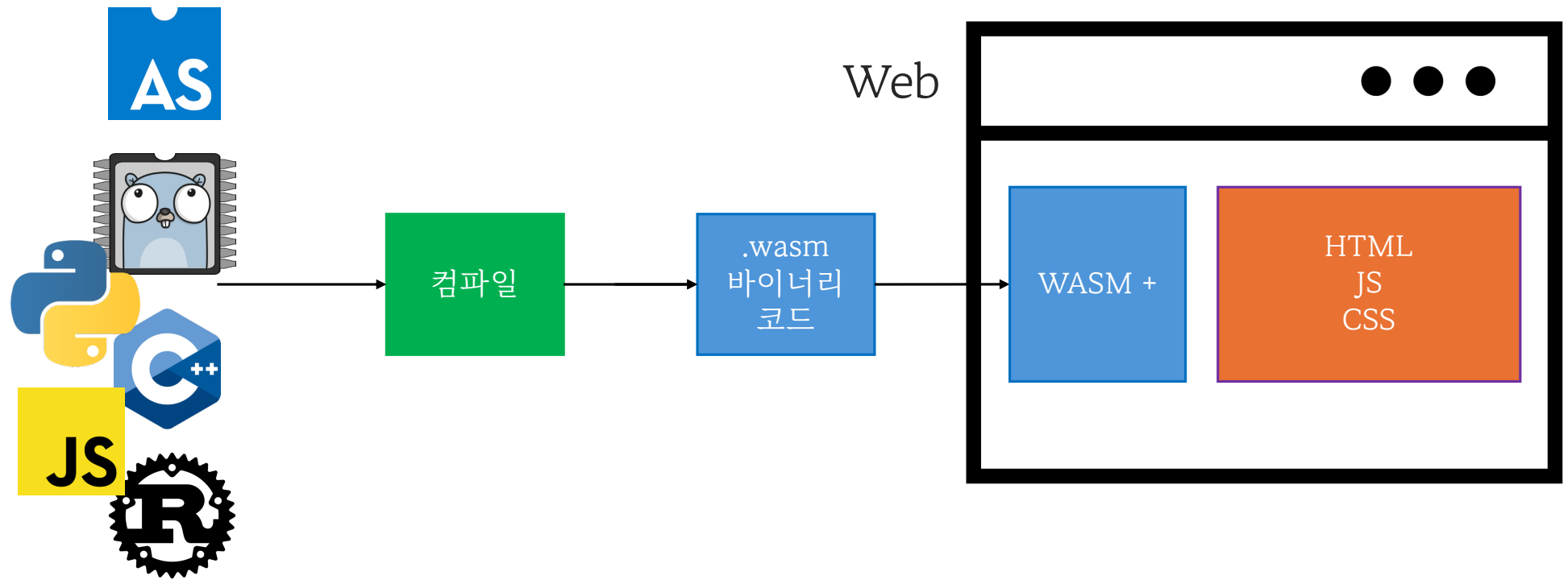


웹어셈블리(WebAssembly a.k.a. WASM)

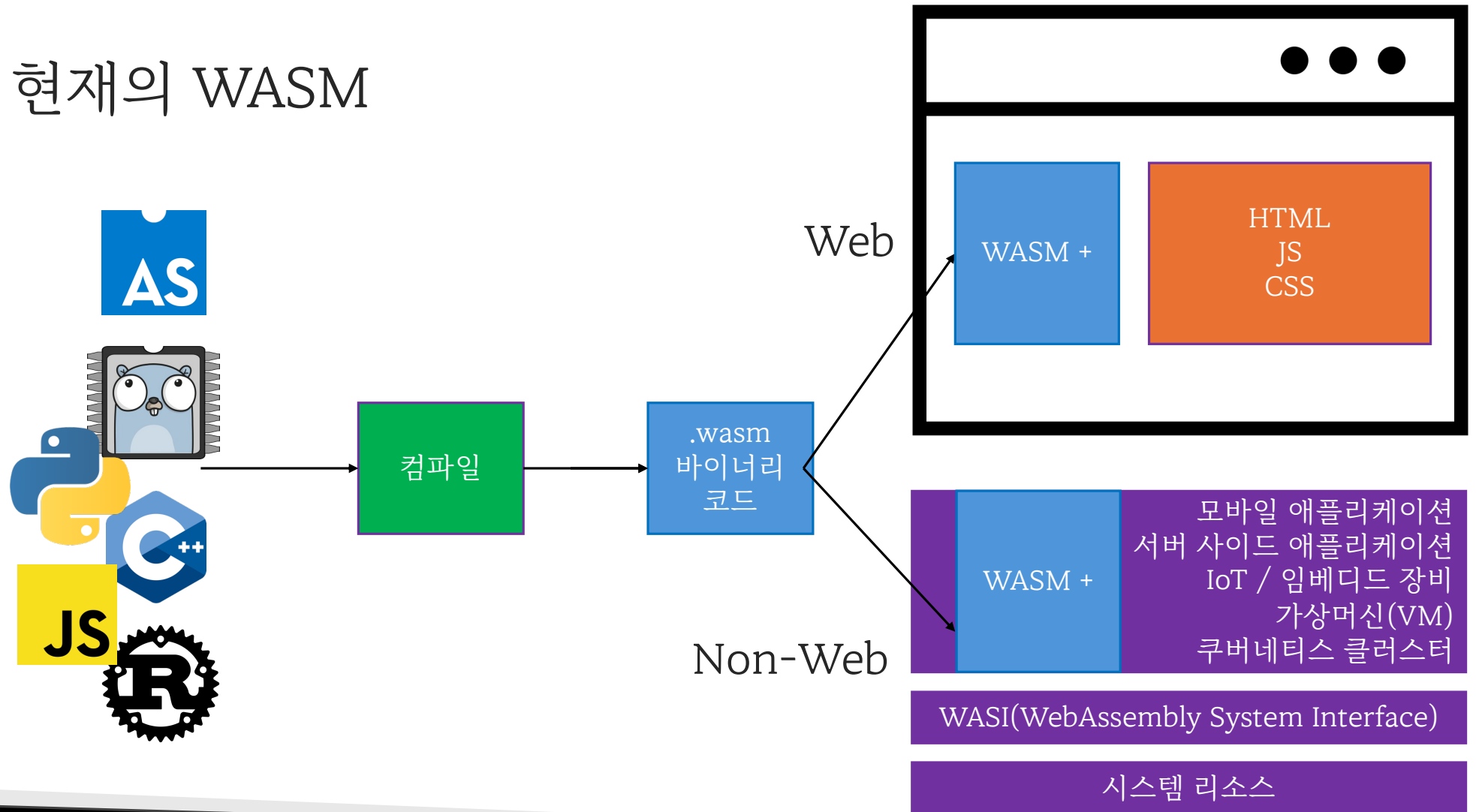
- 고속 실행을 위한 컴파일된 바이너리 포맷
- 격리된 워크로드에 적합한 샌드박스 런타임
- Envoy 필터, 서버리스, 엣지 런타임 등에서 광범위하게 활용
- Rust, Go, C++, AssemblyScript 등 여러 언어 지원
- 운영체제/하드웨어와 무관한 클라우드 네이티브 & 이식성
- 컨테이너보다 가벼운 실행과 빠른 시작 속도
- 안전한 격리와 메모리 보호 등, 보안성 제공
- 모듈 형태로 손쉽게 기능을 추가할 수 있는 확장성



초기의 WASM



현재의 WASM



최대 실행 시간 | 콜드 스타트 | 지원 언어 | 기타 비교

WASM Functions 분산 서버리스 네트워크에서 구동되는 빠른 함수	1 ~ 30초	1 ~ 5ms	JavaScript를 비롯한 6개 언어	• 글로벌 네트워크 • 매우 빠른 시작/콜드 스타트 거의 없음 • CPU 제한 없음, 메모리 제한 없음 • 로컬 개발 및 배포
CDN 서버리스 함수 빠르지만 제한적인 서버리스	30초	50ms	제한된 SDK와 기능 (HTTPS 국한)	• CDN 종속 • 특정 프로토콜과 언어만 지원 • 제한된 실행 시간 • 제한적인 엔터프라이즈 기능
CSP의 서버리스 함수 콜드 스타트, 벤더 종속성, 비싼 비용	30초	200+ms	광범위한 사용 사례	• 클라우드 종속 • 지연 시간에 민감한 앱에는 부적합

WASM 성능

테스트 항목	네이티브 (C/C++)	자바스크립트(JS)	WebAssembly (WASM)
로드/파싱 속도	즉시 실행	텍스트 파싱 필요 → 느림	바이너리 포맷이라 매우 빠름
메모리 접근	직접 접근, 최고 속도	동적 타입 체크 때문에 상당히 느림	샌드박스된 메모리 접근 → 약간의 오버헤드
실행 안정성	빠르지만 보안 취약 가능	안전하지만 성능 낮음	거의 네이티브 성능 + 안전한 샌드박스
이식성(Portability)	OS/CPU 종속	브라우저만 있으면 실행	어디서나 동일한 실행 가능

WASM 성능

테스트 항목	네이티브 (C)	자바스크립트 (Node.js)	WASM (Wasmtime/V8)
루프 (1억 회)	1.0초	5 ~ 10초 (최적화 전), 2 ~ 4초 (JIT)	1.1 ~ 1.2초
피보나치 (재귀)	0.9초	3 ~ 6초	1.0 ~ 1.1초
JSON 파싱	0.05초 (50ms)	0.07 ~ 0.08초 (70~80ms)	0.06초 (55~60ms)
이미지 처리 (픽셀 변환)	0.2초	0.45 ~ 0.6초	0.22 ~ 0.23초

- 네이티브가 항상 가장 빠름
- JS는 JIT 덕분에 최적화되더라도 WASM보다 대부분 느림
- WASM은 네이티브보다 10~20% 느리지만, 보안성과 이식성을 보장
- 연산은 살짝 느리더라도, 웹 환경으로의 이식성과 보안성이 훨씬 우수하다면?

WASM 성능

← → ↻ takahirox.github.io/WebAssembly-benchmark/

JS vs WASM

- [collisionDetection](#)
- [Fibonacci](#)
- [ImageConvolute](#)
- [ImageGrayscale](#)
- [ImageThreshold](#)
- [MultiplyInt](#)
- [MultiplyDouble](#)
- [MultiplyIntVec](#)
- [MultiplyDoubleVec](#)
- [QuicksortInt](#)
- [QuicksortDouble](#)
- [SumInt](#)
- [SumDouble](#)
- [VideoConvolute](#)
- [VideoGrayscale](#)
- [VideoMarkerDetection](#)
- [VideoThreshold](#)

[source code](#)

run Done

Result (average [ms])

JavaScript: 443.9500

WebAssembly: 135.1500

JavaScript/WebAssembly: 3.2849

+ Test code

- JavaScript code

```
function jsFib(n) {  
  if (n === 1) return 1;  
  if (n === 2) return 1;  
  return jsFib(n-1) + jsFib(n-2);  
}
```

- WebAssembly C code

```
int fib(int n) {  
  if (n == 1) return 1;  
  if (n == 2) return 1;  
  return fib(n-1) + fib(n-2);  
}
```

+ WebAssembly Compile shell script

+ WebAssembly Instantiation code

엣지 플랫폼에 WASM을 배포한다면?

클라우드 리전에서 구동되는 관리형 서버리스 함수



WebAssembly 도입 이후,



엣지 플랫폼 적용 이후,



엣지 플랫폼에서 WASM 구동으로 인해 Cold Start 개선 및 RTT 단축

WASM의 GPU 활용

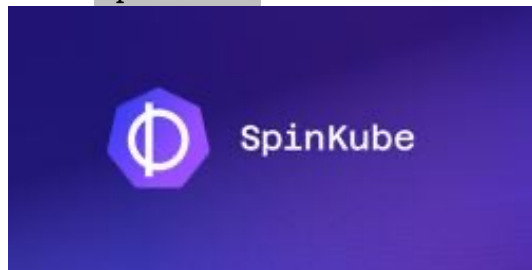
- 기본적으로 WASM은 GPU 직접 접근 불가
 - WebAssembly는 격리된 샌드박스 환경에서 실행
 - 운영체제나 하드웨어에 접근 불가능
 - GPU 서비스의 API를 접근하는 방식으로 사용 가능
- GPU 접근을 가능하게 하는 표준/확장
 - WASI-NN (WebAssembly System Interface for Neural Networks)
 - WASM에서 외부의 하드웨어 가속기(GPU, TPU, VPU 등)에 추론 요청을 보낼 수 있도록 설계된 표준 API
- WebGPU API
 - 브라우저 및 서버 환경에서 GPU 연산을 제어할 수 있게 만든 차세대 그래픽/연산 API
 - WebAssembly와 결합하여 GPU 연산 활용 가능
- 런타임 확장 (예: Wasmtime, WasmEdge)
 - WasmEdge: TensorFlow Lite, ONNX Runtime과 연동해 GPU 가속 가능
 - Wasmtime: 플러그인/호스트 기능을 통해 GPU 연동 가능

WASM의 GPU 활용

- 스마트 카메라 사례
 - 스마트카메라에서 영상 추론을 WASM으로 실행, GPU 호출은 WASI-NN을 통해 처리
 - → 기존 대비 응답속도 30% 감소
- WASI 표준화 진행 상황
 - CNCF 내에서도 WasmEdge, Krustlet 같은 프로젝트가 활발히 발전 중
- WebGPU API
 - 단순 그래픽 API가 아니라 브라우저/서버 환경에서 ML 추론까지 확장
- 보안과 성능
 - 멀티 테넌트 환경에서 컨테이너 대신 WASM 사용 시, 메모리/시스템 콜 격리로 인한 공격면 감소
 - 쿠버네티스 클러스터에서 컨테이너와 WASM을 하이브리드로 배포했을 때, 리소스 절감 효과

WASM 관련 CNCF 프로젝트

SpinKube



K R U S T L E T

Krustlet

Envoy WASM

envoyproxy/**envoy-wasm**



*ATTENTION!: The content of this repo is merged into <https://github.com/envoyproxy/envoy> and future development is happening there.

👤 592 🗨️ 0 ⭐ 205 🍴 86
Contributors Issues Stars Forks



deislabs/**containerd-wasm-shims**

containerd shims for running WebAssembly workloads in Kubernetes



👤 23 🗨️ 21 💬 2 ⭐ 326 🍴 50
Contributors Issues Discussions Stars Forks



containerd-shim-wasm



WasmEdgeRuntime

WasmEdge

WASM의 난해한 빌드 과정

```
demo-filter > envoy-wasm-filter > src > lib.rs
34 impl RootContext for MyRoot {
35     fn on_vm_start(&mut self, _: usize) -> bool {
36
37     }
38
39     fn on_configure(&mut self, _: usize) -> bool {
40         proxy_wasm::hostcalls::log(LogLevel::Info, "🔥 Rust filter configured").ok();
41         true
42     }
43 }
```

```
myuser@localhost:~$ cd demo-filter/
myuser@localhost:~/demo-filter$ cd envoy-wasm-filter/src
myuser@localhost:~/demo-filter/envoy-wasm-filter/src$ ls
cargo.toml  lib.rs
myuser@localhost:~/demo-filter/envoy-wasm-filter/src$ cargo build --target=wasm32-unknown-unknown --release
    Finished `release` profile [optimized] target(s) in 0.10s
myuser@localhost:~/demo-filter/envoy-wasm-filter/src$ cd ../target
myuser@localhost:~/demo-filter/envoy-wasm-filter/target$ ls
CACHEDIR.TAG  release  wasm32-unknown-unknown
myuser@localhost:~/demo-filter/envoy-wasm-filter/target$ cd wasm32-unknown-unknown/
myuser@localhost:~/demo-filter/envoy-wasm-filter/target/wasm32-unknown-unknown$ ls
CACHEDIR.TAG  release
myuser@localhost:~/demo-filter/envoy-wasm-filter/target/wasm32-unknown-unknown$ cd release
myuser@localhost:~/demo-filter/envoy-wasm-filter/target/wasm32-unknown-unknown/release$ ls
app.py          echo-configmap.yaml  envoy_wasm_filter.wasm  otel-collector-config.old  v1-yaml
build          echo-server.yaml     envoy_wasm_filter.wasm.b64  otel-collector-config.yaml  wasm-filter-configmap.yaml
deps           envoy-config.yaml    envoy.yaml               otel-collector.yaml
Dockerfile     envoy-deploy.yaml    examples                  otel.log
echo-api.yaml  envoy_wasm_filter.d  incremental               rust-wasm-filter.yaml
```

Spin – WASM 함수 사용을 위한 도구

개발자 친화적인 도구

CLI · SDK · 이벤트 트리거 제공

빠르고 쉬운 개발

적은 코드 작성, 로컬 테스트, 몇 초 만에 배포

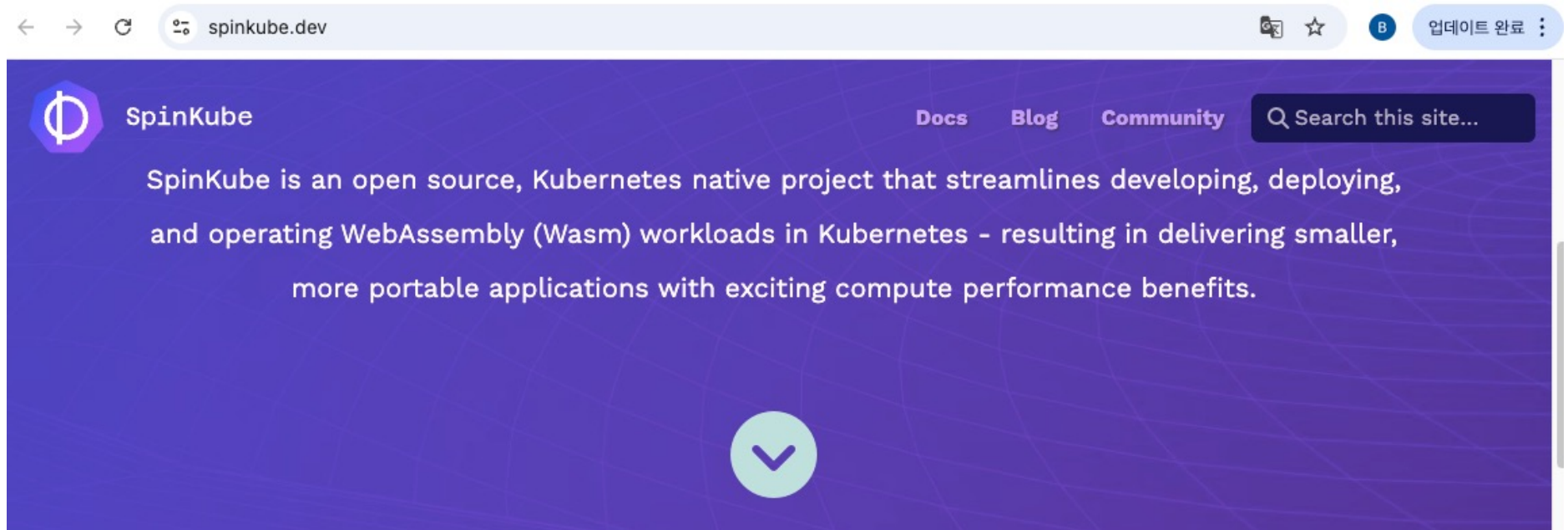
이식성이 좋은 빌드

Spin 앱은 한 번 컴파일하면 어디서든 실행 가능

효율적이고 안전한 앱 개발을 위한
풍부한 API



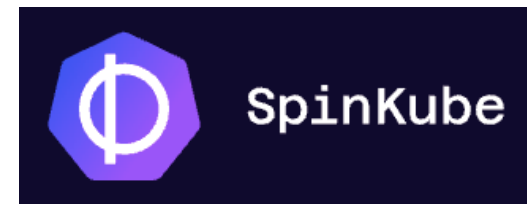
SpinKube – 쿠버네티스에서 WASM 사용하기



<https://www.spinkube.dev/>

SpinKube

- Fermyon에서 개발한 Spin 프레임워크와 쿠버네티스를 통합하는 오픈소스
- 쿠버네티스에서 WASM 앱을 쉽게 빌드, 배포 및 운영
- SpinKube Controller: Spin 앱을 쿠버네티스 리소스로 관리
- CRD(Custom Resource Definition): Spin 애플리케이션을 네이티브하게 정의
- 런타임 연동: WASI 런타임(예: Wasmtime)을 기반으로 동작
- 빠른 시작 속도(초 단위보다 빠른 기동)
- 경량화된 실행(컨테이너보다 리소스 소모 적음)
- 보안 강화(샌드박싱된 실행 환경)
- 쿠버네티스에서 서버리스(Serverless) 함수 실행
- 쿠버네티스에서 엣지 컴퓨팅(저자원 환경에서의 앱 실행)
- 빠른 scale-out이 필요한 마이크로서비스 프로덕션 운영



- DEMO -

<https://youtu.be/QMe-5kApH3I>

~/openinfra

>

SpinKube를 활용한 AI 애플리케이션

The screenshot shows the GitHub repository page for 'fermyon/ai-examples'. The repository is public and has 12 watchers, 17 forks, and 50 stars. The main branch is 'main'. The repository contains several example applications, each with a commit history table.

Commit	Message	Time
4af35e4	Merge pull request #51 from mikkelhegn/update-sentiment-examples	8 months ago
113	Commits	

File	Message	Time
.github/workflows	add ability to push example apps as oci artifact	2 years ago
blog-recommendation-ts	Update blog-recommendation-ts/README.md	2 years ago
code-generator-rs	Bump h2 from 0.3.24 to 0.3.26 in /code-generator-rs	last year
haiku-generator-assets	added in typescript haiku generator demo	2 years ago
haiku-generator-rs	Update readmes to consolidate prerequisites.	2 years ago
haiku-generator-ts	Update readmes to consolidate prerequisites.	2 years ago
helloworld-py	added hello world serverless AI code samples for Py, Ts a...	2 years ago
helloworld-rs	git ignore runtimeconfig files	last year
helloworld-ts	added hello world serverless AI code samples for Py, Ts a...	2 years ago
newsfeeder-ts	Fix typo in prompt	10 months ago
openapi-rs	Changed openapi exmaple to use relative URLs	2 years ago

About

A collection of serverless apps that show how Fermyon's Serverless AI (currently in private beta) works. Reference: <https://developer.fermyon.com/spin/serverless-ai-tutorial>

developer.fermyon.com/cloud/serverless-ai-tutorial

ai serverless llama spin fermyon

Readme Security policy Activity Custom properties 50 stars 12 watching 17 forks Report repository

SpinKube를 활용한 AI 애플리케이션

```
...  
  
/// Spin HTTP component: Ollama 호출  
#[http_component]  
fn handle_hello_openinfra2025(_req: Request) -> anyhow::Result<impl IntoResponse> {  
    // Ollama API URL  
    let url = "http://ollama-gpu.ai.svc.cluster.local:11434/api/generate";  
  
    // 요청 JSON 생성  
    let body = serde_json::to_vec(&OllamaRequest {  
        model: "llama3.1",  
        prompt: "쿠버네티스 안에서 동작하는 LLM의 장점을 3가지 알려줘",  
        stream: false,  
    })?;  
  
    // HTTP POST 요청 구성  
    let request = http::Request::post(url)  
        .header("content-type", "application/json")  
        .body(body.into())?;  
  
    // Spin outbound_http 로 전송  
    let resp = outbound_http::send(request)?;  
  
    // 응답 바디 파싱  
    let bytes = hyper::body::to_bytes(resp.into_body()).unwrap(); // Spin SDK에선 Vec<u8> 반환됨  
    let ollama_resp: OllamaResponse = serde_json::from_slice(&bytes)?;  
  
    // 최종 Spin 응답 반환  
    Ok(Response::builder()  
        .status(200)  
        .header("content-type", "text/plain")  
        .body(format!("Ollama 응답: {}", ollama_resp.response))  
        .build())  
}
```

GPU를 사용하는 AI 추론 서비스

- Spin HTTP 컴포넌트가 들어오는 요청을 받음
- Ollama API(ollama-gpu.ai.svc.cluster.local:11434/api/generate)에 프롬프트 전달
- outbound_http::send 로 Ollama에 POST 요청을 보내고, JSON 응답에서 response 필드를 파싱
- 최종적으로 Spin이 Ollama의 답변을 text/plain 형태로 클라이언트에 반환
- HTTP 요청 수신 → 외부 API 호출 → 응답 반환 과정을 Rust 몇 줄로 처리
- Spin의 Wasm 샌드박스 환경에서 운영, 작고 빠르게 실행되면서도, 호스트 리소스 접근은 제한

- DEMO -

<https://youtu.be/pGkl6lrsk-c>

Open WebUI

새 탭

← → ↻ 주의 요함 gpt-oss.brandonkang.org:8080

≡

이 새 채팅

✎

🔍 검색

📁 노트

🌐 워크스페이스

▼ 채팅

오늘

새 채팅

이전 7일

새 채팅

이전 30일

🌟 Hangul's Superior Design

새 채팅

Image Interpretation Challenge

New Chat

📌 Sticky Header Demo

📌 Sticky Header Example

📌 Sticky Header Code Snippet

🚀 AI Response Speed

New Chat

🌐 Meet ChatGPT

New Chat

📌 Sticky Header Tutorial

gpt-oss:20b

기본값으로 설정

⚙️

AK

이 gpt-oss:20b

무엇을 도와드릴까요?

+

📄 코드 인터프리터

🎤

🔄

🔗 제안

Give me ideas

for what to do with my kids' art

Overcome procrastination

give me tips

Tell me a fun fact

about the Roman Empire

세션 요약

- 리전 vs 엣지
 - 대규모 자원, 중앙집중형 처리 vs 초저지연, 분산형 실시간 처리
- WebAssembly(WASM)
 - 경량·보안·이식성 / 쿠버네티스 + GPU와 결합 시 강력한 엣지 AI 실행 환경
- GPU + WASM 활용
 - WASI-NN, WebGPU 등으로 GPU 접근 확장
 - 엣지에서 고성능 AI 추론 가능 / WASM에서 AI 서비스로 API 호출 방식으로 가능
- SpinKube 사례
 - 쿠버네티스에서 컨테이너와 WASM 앱을 함께 운영
 - 서버리스 & 엣지 컴퓨팅 최적화

“Inference Anywhere: GPU와 WASM으로
엣지에서 AI를 빠르고 안전하게 배포”

참고 자료

- <https://webassembly.org/>
- <https://www.spinkube.dev/>
- <https://github.com/spinframework/spin>
- <https://github.com/fermyon/ai-examples/>
- <https://takahirox.github.io/WebAssembly-benchmark/>
- <https://www.fermyon.com/blog/introducing-spinkube-fermyon-platform-for-k8s>

감사합니다.