설계(조현민)

설계테마

거인의 어깨위에 올라타서 구현하자

Building on the shoulders of giants

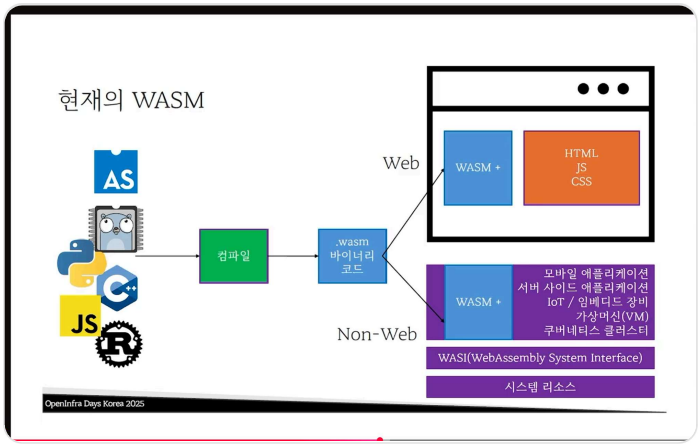
0. 아이디어 출처 및 배경

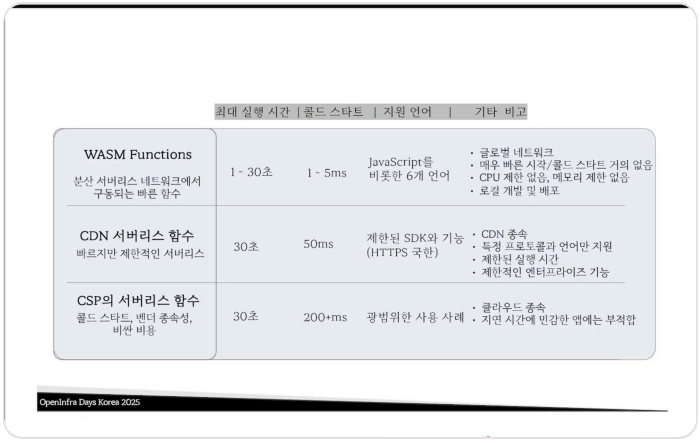
Open InfraDay2025 에 참여해서 마지막 세션이었고 WASM이 인상깊었음 Akamai에서 한 사례

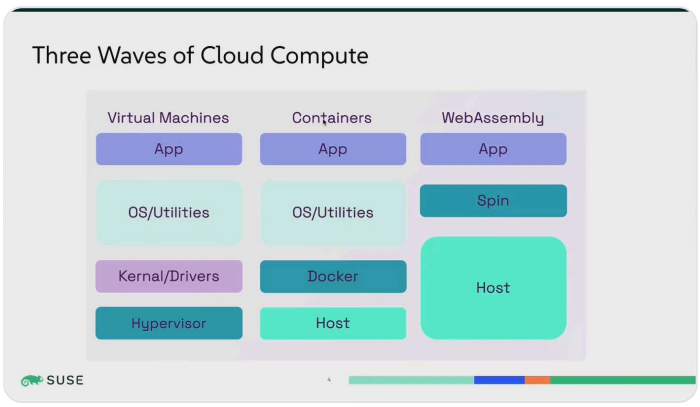
이미 AKamai에서 서비스로 사용 Wasm을 업로드 하면 금방 엣지 서버까지 배포

Introducing Fermyon Wasm Functions on Akamai

[Session] Inference Anywhere - GPU와 WebAssembly로 만든 쿠버네티스 엣지 AI 배포 사례 | OpenInfra Days Korea 2025





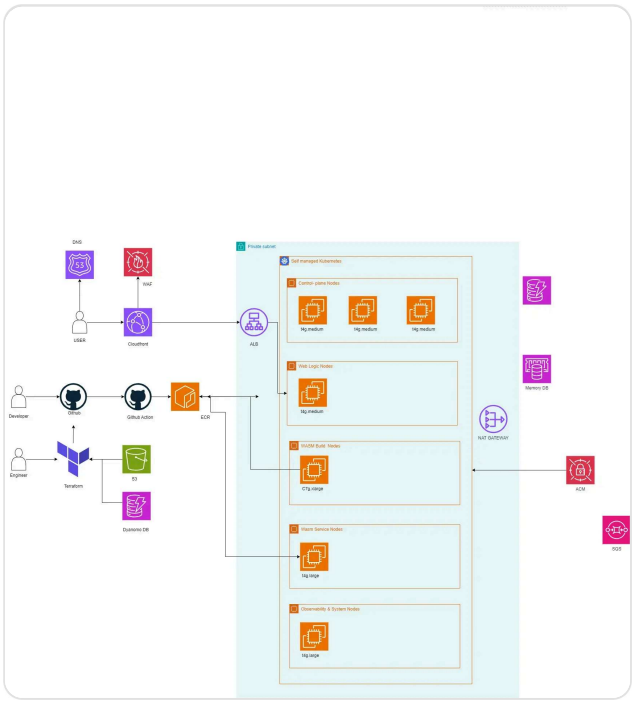


1. 핵심 목표 및 제약 사항 해결

- **미션:** "사용자의 코드를 받아서 실행하는 FaaS(Function as a Service)를 구축하라." HTTP ENDPOINT 필요
- **제약 사항:** AWS Lambda, EKS 등 관리형(Managed) 서비스 사용 금지. **반드시 VM(EC2) 위에서 실행 환경을 직접 구축**

- **해결책:** AWS EC2 인스턴스 위에 경량 쿠버네티스(**k3s**)를 직접 설치하여 제약 사항을 준수하면서도 오케스트레이션 환경을 확보함

2. 클라우드 아키텍쳐(Architecture)



3. 기술 스택 (Tech Stack)

Kubernetes vs VM[단순 Docker]

단순 VM은 스케일링시 속도 문제 및 관리성이 어려움, VM을 늘리고 그 VM과 ALB나 백엔드 와 같은 API서비스 랑 어떻게 매핑? 사실상 스케일링 불가, 개최자가 보고싶은 주제랑 안맞음, 구현난이도는 쉽겠지만

기존의 무거운 컨테이너 방식을 버리고, **속도**와 **보안**을 위해 WebAssembly를 채택

- 대안 : Knative (FaaS) 을 제공하며 Scale to Zero가 특징 ← 컨테이너 이미지 빌드 필요
- 문제점 컨테이너를 구동하는데 시간

구분	선택 기술	대안	선정 이유
인프라	**AWS EC2 (on-demand, spot 혼용)**	On-demand 요금 최적화 불가	master node 일시 스팟으로 꺼질 시 모든 노드가 다운되는 불상사 발생 Spot으로 요금 절약
OS/플랫폼	**k3s (Lightweight K8s)**	RKE2 / mini-kube	쉽게 설치및 운용 가능한 경량 쿠버네티스이며 팀원들이 Upstream K8s 보다 K3s의 익숙해서 선정
런타임	**Fermyon Spin (Wasm)**	X	Docker보다 수십 배 빠른 부팅 속도(Cold Start Zero) 및 완벽한 샌드박싱
CNI	Calico [k3s 기본]	Cilium	eBPF를 지원으로 패킷을 커널레벨에서 처리해 가장 빠름
인프라 관리	Terraform	Cloud formation	Cloudformation이나 CDK의 경우 인프라를 수정할 경우 뭐가 수정되는지 제대로 파악하기 어려움 초기 구축은 좋은편
LB	AWS LB	Metal LB	AWS LB로 자동으로 ELB를 생성해 타겟 그룹으로 선정하는게 관리하기 편하고 MetalLB는 실제 On-premise에서만 사용가능해서 선택지가 AWS LB로 제한

3. 핵심 경쟁력 (Why Wasm?)

A. 속도: "컨테이너 빌드는 너무 느리다"

- **기존(Container):** `코드 수신 -> OS/라이브러리 설치 -> 이미지 빌드(수십 초) -> 배포` (실시간성 부족)
- **우리(Wasm):** `코드 수신 -> 바이너리 컴파일(수 ms) -> 즉시 실행`
- **어필:** 무거운 컨테이너 빌드 과정 없이, AWS Lambda급의 **즉각적인 실행 속도**를 구현함.

B. 보안: "커널 공유의 위험 차단" (RCE 방어)

- **문제:** Docker는 호스트 OS 커널을 공유하므로, **Container Escape** 취약점 발생 시 서버 전체가 탈취될 위험이 있음. (특히 RCE가 허용된 플랫폼에서는 치명적)
- **해결:** Wasm은 **메모리 샌드박스** 기술을 사용하여 시스템 콜 접근을 원천 차단.
- **비교:** AWS는 Firecracker(MicroVM)로 막지만, 우리는 EC2 위에서 중첩 가상화 성능 저하를 피하기 위해 Wasm을 선택함.

4. 아키텍처 흐름도 (Workflow)

1. **User:** 코드(Python/Go 등) 업로드
2. **Backend:** `spin build`로 Wasm 바이너리 생성 및 레지스트리 Push
3. **K8s(EC2):** `SpinApp` 리소스 배포 (SpinKube Operator 감지)
4. **Ingress:** `user1.ip.nip.io` 라우팅 규칙 자동 생성
5. **Execution:** 사용자가 URL 접속 시 Wasm 런타임이 0.1초 만에 응답

5. 쿠버네티스 디스트리뷰션 선정: K3s (vs RKE2 vs MicroK8s)

A. K3s vs. RKE2 (경량화 vs 표준 준수)

비교 항목	K3s (Lightweight K8s)	RKE2 (Rancher Government K8s)
정체성	**Edge & IoT 최적화 (다이어트 버전)**	**데이터센터 & 보안 최적화 (표준 버전)**
구조	Legacy, Alpha 기능, Cloud Provider 플러그인 등 불필요한 코드를 삭제하여 바이너리 하나로 통합	Upstream(표준) 쿠버네티스를 수정 없이 그대로 패키징 (Docker 대신 Containerd 기본 사용)
표준 준수	CNCF 인증은 받았으나, 내부 구조가 **비표준(Modified)**에 가까움 (SQLite 지원 등)	**완벽한 업스트림 준수** 및 FIPS 140-2(미국 보안 표준) 준수
리소스	**매우 적음** (512MB RAM으로도 동작 가능 sqllite나 외부 dbms를 사용시)	K3s보다는 무거움 (표준 K8s와 유사)

B. K3s vs. MicroK8s (왜 MicroK8s가 아닌가?)

둘 다 '가벼운 쿠버네티스'를 표방하지만, 생태계와 운영 측면에서 K3s가 압도적입니다.

1. 커뮤니티 활성화 및 성숙도 (Maturity)

- **K3s (Rancher/SUSE):**
  - Rancher는 쿠버네티스 관리 플랫폼의 **업계 표준**에 가까운 성숙도를 가짐.
  - GitHub Star, Issue 처리 속도, 써드파티 도구 호환성 면에서 커뮤니티가 훨씬 거대하고 활발함.
- **MicroK8s (Canonical):**
  - Ubuntu 생태계에 종속적임. 커뮤니티 규모가 K3s에 비해 상대적으로 작음.

2. 아키텍처 및 패키징

- **K3s:** 단일 바이너리(`curl ... | sh`)로 설치 끝. 의존성이 거의 없고 RHEL, Debian등 다양한 배포판 호환
- **MicroK8s: Snap(스냅)** 패키지 관리자를 강제함.

- *문제점:* Snap 자체의 오버헤드가 있고, 자동 업데이트로 인해 원치 않게 클러스터 버전이 올라가는 등 제어하기 까다로운 경우가 많음.

노드구성

컨트롤 플레인 3대로 구성한 이유 : 컨트롤 플레인이 하나일 경우 Control-plane이 한대가 마비되면 전체 클러스터가 마비
쿠버네티스는 4달마다 Minior 버전을 업데이트를 하기 때문에 실제 운영환경에서 3대 ~ 5대 사이를 권장 [etcd 포함시]
ETCD가 많으면 key value 복제하는데 오래걸림 , 아니면 외부 etcd사용
etcd : 쿠버네티스의 클러스터를 저장하는 key value 스토어

역할	노드 그룹명	구성 방식	인스턴스 타입	수량	주요 탑재 컴포넌트	7일 요금
Contorol-plane	**Control-Plane**	**On-Demand**	t4g.medium	3대	k3s Server, Embedded etcd (HA)	$20.9664
Observability	**Mgmt-Obs**	**On-Demand**	t4g.large	2대	Prometheus, Grafana, **Karpenter/CA**, CoreDNS(권장), LB Controller	$27
Worker (FaaS)	**Worker-Wasm**	**Spot**	t4g.xlarge	0 → N	SpinKube Apps (Wasm), Traefik Ingress	$13 스팟할인 받을시 (최소 50%)
Worker(Build)	**Worker-Build**	**Spot**	c7g.xlarge 개발시 낮은 인스턴스로 시연 때	1 → N	Build Jobs (Kaniko/Spin build)	$14 예상 $27.4176
Worker(Backend)	Worker-backend	Spot	T4g.medium	1 → N	Web Framework	$ 4 예상

고려사항

- 스팟을 사용할경우 Karpenter나 NodeGroup이 선행 필요
- 카펜터의 경우 scaling 속도가 더빠름
- AWS에서 쿠버네티스를 쓸 경우 되도록이면 Dockerhub가 아닌 `gallery.ecr.aws` 에서 당겨오거나 우리 ECR에 저장해서 당겨오는 형태 스케일링시 nodelocaldns와 같은 `Daemonset` Pod를 당길때 rate-limit에 의해 불가능할 수 있음

채널톡 사례

보안

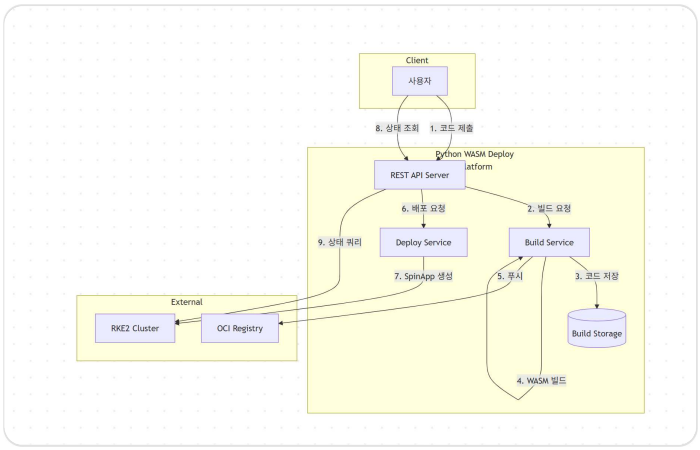
IRSA 사용 x 제한 시간내로 불가 → 설정 복잡, 버그 가능성 높음
Node IAM 사용
테라폼에서 `http_put_response_hop_limit` 을 2로 필요

```
resource "aws_instance" "k8s_node" {
  # ... 기타 설정 ...

  metadata_options {
    http_endpoint               = "enabled"
    http_tokens                 = "optional" # 혹은 "required"
    http_put_response_hop_limit = 2          # <--- 여기를 반드시 2로 설정!
  }
}
```

빌드앱 구성

사용자가 Spin형태로 된 프로젝트를 받아서 실행



POC

가능함을 확인 이 CRDS yaml을 배포하면 자동으로 SVC 생성 쉽게 자동화 가능 → 쉽게 http 연동가능

```
apiVersion: core.spinkube.dev/v1alpha1
kind: SpinApp
metadata:
  name: spin-rust-hello
  namespace: default
spec:
  enableAutoscaling: false
  executor: containerd-shim-spin
  image: ghcr.io/spinkube/containerd-shim-spin/examples/spin-rust-hello:v0.15.1
  replicas: 1
  resources: {}
  runtimeConfig: {}
```

```
$ kubectl get all -n default
NAME                                                    READY   STATUS             RESTARTS       AGE
pod/70c8b35f-3cde-4fa2-a275-eb92e1ff8364-67b48fd448-wq9rl   0/1     CrashLoopBackOff   16 (65s ago)   42m
pod/ba54a28d-8a68-448e-b160-8fcf80bc52c0-58d9dc95d4-zlcvt   0/1     CrashLoopBackOff   14 (72s ago)   30m
pod/simple-spinapp-656579dd9b-b414r                     1/1     Running            0              98s
pod/spin-rust-hello-7457fd74cc-zngsx                    0/1     CrashLoopBackOff   4 (73s ago)    52m
pod/test-python-574c9fddb8-5gkdn                        1/1     Running            0              28s
pod/test-python-574c9fddb8-g777g                        1/1     Running            0              34s

NAME                                                    TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE
service/ba54a28d-8a68-448e-b160-8fcf80bc52c0            ClusterIP   10.43.64.188    <none>        80/TCP    30m
service/kubernetes                                      ClusterIP   10.43.0.1       <none>        443/TCP   47h
service/simple-spinapp                                  ClusterIP   10.43.50.92     <none>        80/TCP    22h
service/spin-rust-hello                                 ClusterIP   10.43.35.100    <none>        80/TCP    52m
service/test-python                                     ClusterIP   10.43.114.182   <none>        80/TCP    9m10s
```

Wasm 빌드 배포 테스트
빌드 및 배포 : 백엔드서버에 wasm 빌드 묘듈 설치후 빌드

Rust는 되는데 파이썬이 실패 : 원인 containerd-shim-spin의 버전이 낮아서 호환 x
테스트 성공

Observability

자체적으로 OTEL 지원확인 , Grafana, Loki, Tempo 스택으로 하는게 쉽고 합리적 → 현업에서도 많이 쓰고

참고자료

- SpinKube 관련

Running Serverless Wasm Functions on the Edge with k3s and SpinKube

- EC2에서 Kubernetes 관련자료

AWS에서 직접 관리하는 Kubernetes 클러스터를 수동으로 만드는 방법 | 나빌 압디 작성 | 보통 → AWS IAM 설정 참고

EC2 기반 Kubernetes 사용

kubernetes/cloud/aws-k3s at main · morrismusumi/kubernetes

Self Managed Kubernetes in AWS

IRSA

AWS IRSA in self-hosted Kubernetes Clusters | by Security Guy | Level Up Coding → 엄청복잡

- Spin 관련

Building Spin Components in Python | Spin Docs

- Scale to Zero

Scaling Spin Apps With KEDA - DEV Community