기획

서비스명
Cold Start and Hot Passion

목표
서버리스 환경의 Cold Start 문제를 eBPF가 얼마나 개선하는가를 확인한다.

구현 기능
- 서버리스 환경 구축
- 함수에서 system call (sysexcec()) 될 때마다 로그 찍어주는 함수 구현
- eBPF 매핑 함수 구현 (⇒ eBPF 매핑 후 위의 출력 함수 활용해 성능 비교)

Role & Responsibility

멤버	직책	임무
김선희	Kernel Engineer	eBPF, kernel 핸들링
김준엽	DevOps Engineer	CI/CD 테스트 구축
오진우	Infra Engineer	인프라 세팅
이예은	Go Data Engineer	bpf2go, 프로토콜

시장 가능성 조사
- **SWOT** 분석

Strengths

- eBPF 기반이라 커널 수준의 높은 가시성 확보 가능
- LocalStack 기반이라 저비용 로컬 재현 가능
- 서버리스 Cold Start라는 명확한 문제 정의
- 기존 발표 자료에서 목표·기능·파이프라인이 이미 구조화되어 있음

Weaknesses

- LocalStack은 실제 AWS Lambda와 완전히 동일한 운영 환경은 아님
- eBPF는 러닝커브가 높아 초기 진입 장벽이 있음
- 초기 버전은 로컬 테스트 환경 중심이라 시장 범위가 좁음
- 실제 가치 증명은 정량적 성능 개선 데이터가 나와야 강해짐

Opportunities

- cloud native / Kubernetes adoption이 매우 높음
- 서버리스는 자동 확장과 운영 단순화 때문에 꾸준히 도입 가치가 있음
- 운영자들이 실제로 cold start와 시스템 레벨 메트릭을 관측하고 있음
- Cilium/Hubble, LoxiLB처럼 eBPF의 성공 사례가 있어 “eBPF 기반 인프라 도구” 자체에 대한 시장 신뢰가 존재함

Threats

- AWS CloudWatch Lambda Insights 같은 클라우드 제공자 기본 도구와 경쟁해야 함
- Cilium/Hubble 등 강력한 eBPF 생태계 프로젝트와 비교될 수 있음
- 서버리스 플랫폼 자체의 개선으로 Cold Start 문제가 점차 완화될 가능성
- 사용자 입장에서 eBPF 기반 툴은 설치/권한/운영 부담이 있을 수 있음

기능 명세서

1. 서버리스 환경 구축
    - 활용 오픈소스: localstack

Classification	Domain Function	Detailed Function	Method	API Endpoint	User API	System Call	Behavior
서버리스 환경 구축	인프라 시뮬레이션	Localstack 기반 Lambda 컨테이너 실행 환경 구축	-	http://localhost:(port)	aws_local_lambda_create()	-	로컬 환경에서 AWS Lambda와 동일한 생명주기(Init-Invoke-Shutdown)를 구현하여 테스트 준비
	Cold Start 유도	Lambda 함수 강제 재시작 및 첫 호출 시나리오 생성	POST	/env/functions/{name}/invocations	aws_local_lambda_invoke	-	컨테이너가 없는 상태에서 함수를 호출하여 'Cold Start'가 발생하는 시점을 식별함

2. 시스템 콜 로그 출력

Classification	Domain Function	Detailed Function	Method	API Endpoint	User API	System Call	Behavior
시스템 콜 로그 출력	모니터링/프로세스 추적	함수 실행 시 sys_execve() 호출 감지 및 로그 기록	-	-	print_log()	sys_execve()	Lambda 함수 인스턴스가 생성되며 새로운 프로세스가 실행될 때마다 커널 로그에 기록
	모니터링/실행 시간 분석	유저 수준에서의 함수 응답 시간 기록	-	-	get_time_of_exec()	-	시스템 콜 발생 시점과 함수 완료 시점 사이의 간격을 측정하여 Cold Start 지연 시간을 계산함

3. **eBPF** 매핑 적용
    - 활용 오픈소스: eBPF

Classification	**Domain Function**	**Detailed Function**	**Method**	**API Endpoint**	**User API**	**System Call**	**Behavior**
eBPF 매핑	고성능 데이터 추출	bpf2go를 이용한 eBPF 프로그램 컴파일 및 로드	GET	bpf2go/tracepoint	-	eBPF Hooking	C로 작성된 eBPF 코드를 Go로 바인딩하여 커널 이벤트에 직접 후킹(Hooking)함
	정밀 측정/타임스탬프 로깅	bpf_ktime_get_ns() 기반 나노초 단위 실행 시간 추출	GET	kprobe/tracepoint	bpf_ktime_get_ns()	sys_execve() (Hooked)	시스템 콜 호출 시점에 커널 내부에서 직접 나노초 단위의 시간을 측정하여 오차를 최소화함

활용 오픈소스

1. **eBPF**
   https://github.com/cilium/ebpf
   - 타임 스탬프 출력: bpf_ktime_get_ns()
   - 빌드: bpf2go
2. **localstack**
   https://github.com/localstack/localstack

   - Lambda를 로컬에서 생성·배포·테스트할 수 있음.

의존성 트리

```
$ toy-project@1.0.0
├── localstack@latest
│   └── aws-lambda-simulation (로컬 배포 및 테스트)
│       └── cold-start-trigger (성능 측정 대상)
├── kernel-monitor-logic (시스템 콜 분석)
│   └── sys_execve-logger (프로세스 실행 추적)
│       └── timestamp-collector (기존 방식 로그 출력)
└── ebpf-optimization-suite (eBPF 매핑 적용)
    └── cilium-ebpf-library (오픈소스 프레임워크)
        └── bpf2go-compiler (Go 바인딩 및 빌드 도구)
```

SBOM

1. **Primary**

Component	Type	Scope	Purpose / Role	Supplier / Source	Evidence
Cold Start and Hot Passion	Application / Project	Primary	프로젝트 전체 서비스/실험 대상	Project team	서비스명과 목표가 명시됨.

2. Third Party

Component	Type	Scope	Purpose / Role	Supplier / Source	Evidence
LocalStack	Open-source software	Third-party	로컬에서 AWS Lambda 생명주기(Init-Invoke-Shutdown)를 시뮬레이션하고 Cold Start 테스트 환경을 구축	github.com/localstack/localstack	활용 오픈소스와 Lambda 로컬 생성·배포·테스트 용도가 명시됨.
AWS Lambda emulation layer (via LocalStack)	Service capability	Third-party	Lambda 컨테이너 실행 환경 구축, 함수 호출, Cold Start 유도	Provided through LocalStack	LocalStack 기반 Lambda 실행 환경 구축, invoke 시나리오가 명시됨.
eBPF	Kernel technology / open-source ecosystem	Third-party	커널 이벤트 직접 후킹, 정밀 계측	eBPF ecosystem	활용 오픈소스로 명시되고, 성능 비교용 매핑 적용 대상임.
Cilium eBPF Go library	Open-source library	Third-party	eBPF 프로그램 로드/바인딩에 사용되는 라이브러리 계열	github.com/cilium/ebpf	활용 오픈소스 URL로 명시됨.
bpf2go	Build tool / code generation tool	Third-party	eBPF 프로그램 컴파일 및 Go 바인딩 생성	Included in Cilium eBPF toolchain	문서에서 빌드 도구로 명시됨.

3. Runtime Dependency

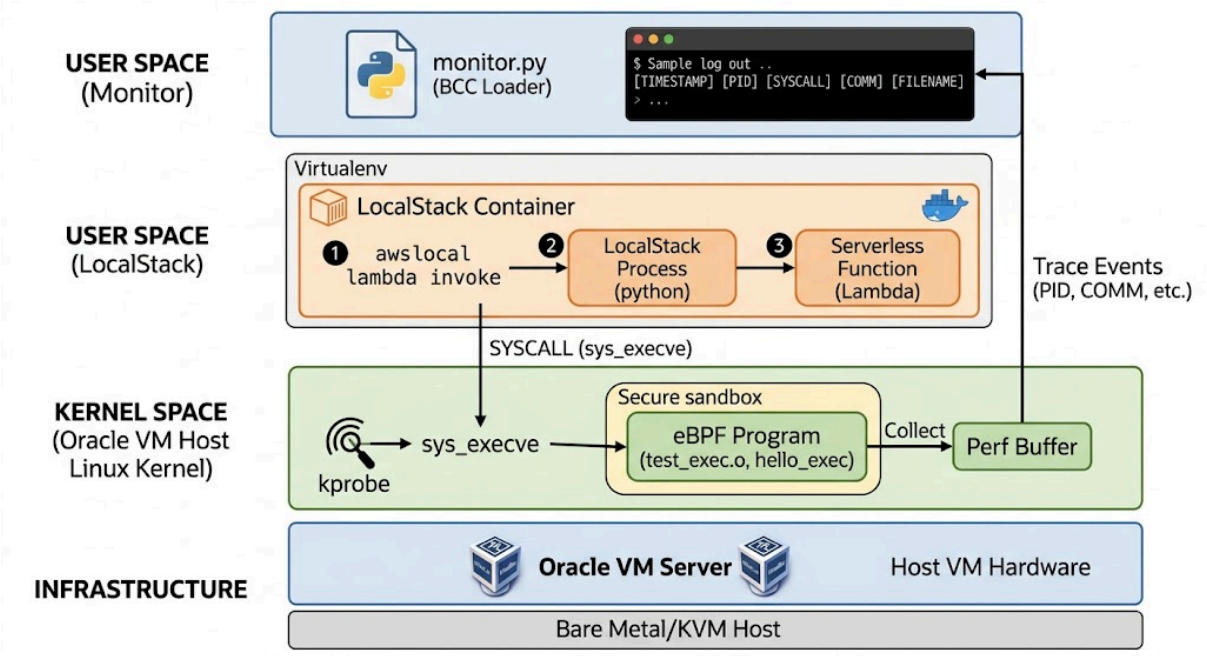
Component	Type	Scope	Purpose / Role	Supplier / Source	Evidence
`bpf_ktime_get_ns()`	Kernel helper API	Runtime dependency	나노초 단위 타임스탬프 획득	Linux eBPF helper	타임스탬프 출력 및 정밀 측정 수단으로 명시됨.
`kprobe` / `tracepoint`	Kernel tracing hook mechanism	Runtime dependency	eBPF 후킹 지점	Linux kernel tracing subsystem	API/Endpoint 열에 명시됨.
Linux system call `sys_execve()`	OS / kernel interface	Runtime dependency	프로세스 생성 시점 감지, Cold Start 관련 이벤트 관찰 지점	Linux kernel	로그 출력/후킹 대상 시스템 콜로 명시됨.

4. **local runtime**

Component	**Type**	**Scope**	**Purpose / Role**	**Supplier / Source**	**Evidence**
Local HTTP endpoint `http://localhost:(port)`	Interface / endpoint	Local runtime	LocalStack 기반 로컬 호출 인터페이스	Local environment	서버리스 환경 구축 항목에 명시됨.
Invocation endpoint `POST /env/functions/{name}/invocations`	Interface / endpoint	Local runtime	Cold Start 유도용 함수 호출 엔드포인트	Local environment / LocalStack integration	Cold Start 유도 항목에 명시됨.

구현

[서버리스 환경 구축]



구현 요약:

1. Ubuntu 위에서 LocalStack (: AWS 서비스를 로컬 컨테이너 안에서 에뮬레이션하는 서비스)을 이용해 AWS Lambda 같은 서버리스 환경을 로컬에서 mock
2. 구축한 lambda mocking 서버리스 환경에서, 임의의 테스트 파일을 실행시켜 system call 모킹

코드 구성:

- 인프라: docker-compose.yaml, localstackenv/, bash.sh
- Target: index.py
- Ebpf 모니터링 엔진: v1/

```
$ ebpf-server-implement@1.0.0
├── docker-compose.yaml (Oracle VM 위에서 S3, Lambda 등의 AWS 모의 환경을 컨테이너로 띄우는 역할)
│
├── index.py (test code. 이 코드가 실행될 때 발생하는 system call을 ebpf가 hooking)
│
├── bash.sh (test code. index.py를 압축하고 lambda를 배포하는 등 테스트 루틴 반복)
│
├── localstackenv/ (localstack의 설정 파일, awslocal 실행을 위한 가상 환경이 담긴 폴더)
│
└── v1/
    ├── kprobe.c (커널 코드. 리눅스 커널에 이식되어 프로세스 정보 수집)
    │
    └── main.go (유저 코드. 커널 코드를 컴파일 및 로드하고, 수집한 데이터를 가공 및 출력)
```

검증

[서버 테스트 원리 및 결과**]**

테스트 목적

1. 서버리스 환경 (Lamba mock)이 제대로 돌아가는지

2. 시스템 콜 호출이 제대로 hooking되는지 모니터링

우선 서버리스 환경이 제대로 돌아가는지, 그리고 시스템 콜 호출이 제대로 hooking되어 모니터링하는 과정이 원활한지 검증하기 위해서 간단한 실험을 진행하였습니다.

먼저 팀원간 원활한 협업을 위해 Oracle VM을 사용, 공용 Ubuntu 인스턴스를 구축하여 팀원 모두가 동시에 작업할 수 있는 환경을 조성하였습니다.

매우 간단한 테스트용 Python 함수를 만들었고, 이 파일을 압축해서 Lambda에 올릴 수 있는 형태로 준비했습니다. 즉, 로컬에서 만든 함수 코드를 서버리스 환경에서 배포 가능한 형태로 바꾸는 과정입니다.

이후 `awslocal` 명령어를 사용해 LocalStack 안에 Lambda 함수를 생성했고, Lambda를 직접 호출해서 테스트용 함수의 실행 결과를 확인했습니다. 이를 통해 실제로 AWS Lambda를 사용하지 않고도, 저희가 구축할 프로그램이 실제 Lambda 환경에서 어떻게 동작하는지 로컬 Ubuntu 환경에서 시험해볼 수 있습니다.

이를 통해 Ubuntu에서도 서버리스 환경을 비슷하게 구성이 가능하다는 것을 증명하였고, 시스템 콜 호출이 제대로 hooking되는지 모니터링이 가능함을 성공적으로 검증하였습니다.

테스트 결과

1. 서버리스 환경이 성공적으로 구동되었습니다.

```
Status: Downloaded newer image for localstack/localstack:latest
Creating ubuntu_localstack_1 ... done
```

2. 시스템 콜 (sys_execve())이 ebpf에 의해 hooking되었고, 이 정보를 user가 모니터링하도록 하는 출력 함수가 정상 실행되었습니다.

```
TIMESTAMP       PID         SYSCALL      COMM            FILENAME
03:23:33.038    16293       sys_execve   python3
03:23:33.241    16294       sys_execve   python3.12
03:23:38.551    16295       sys_execve   python3.12
03:23:38.551    16295       sys_execve   python3.12
03:23:38.551    16295       sys_execve   python3.12
03:23:38.551    16295       sys_execve   python3.12
03:23:38.551    16295       sys_execve   python3.12
03:23:38.638    16310       sys_execve   python3.12
03:23:38.638    16310       sys_execve   python3.12
03:23:38.638    16310       sys_execve   python3.12
03:23:38.638    16310       sys_execve   python3.12
03:23:38.638    16310       sys_execve   python3.12
```

[해결해야 하는 과제**]**

- 제한된 프로그래밍 환경: eBPF는 커널 안전성을 위해 복잡한 로직 구현이 어렵고 코드 크기에 제한이 있습니다. 일반 개발자가 짠 Python이나 Java 코드를 어떻게 eBPF가 이해할 수 있는 중간 단계로 변환하느냐가 승부처입니다.
- 보안 검증: 커널에서 직접 실행되므로 보안 루프홀(Loophole)이 없음을 증명해야 합니다.