클라우드 컴퓨팅 서비스 추천 시스템 - 졸업 프로젝트

FinOps 관점에서 EC2, Lambda, Fargate 선택을 위한 자동화된 추천 시스템

202211252 고승우

워크로드에 따른 컴퓨팅 서비스 선택의 어려움

컴퓨팅 서비스 선택의 어려움

EC2, Lambda, Fargate 중 어떤 서비스를 선택해야 할지 판단이 어려움

비용 낭비

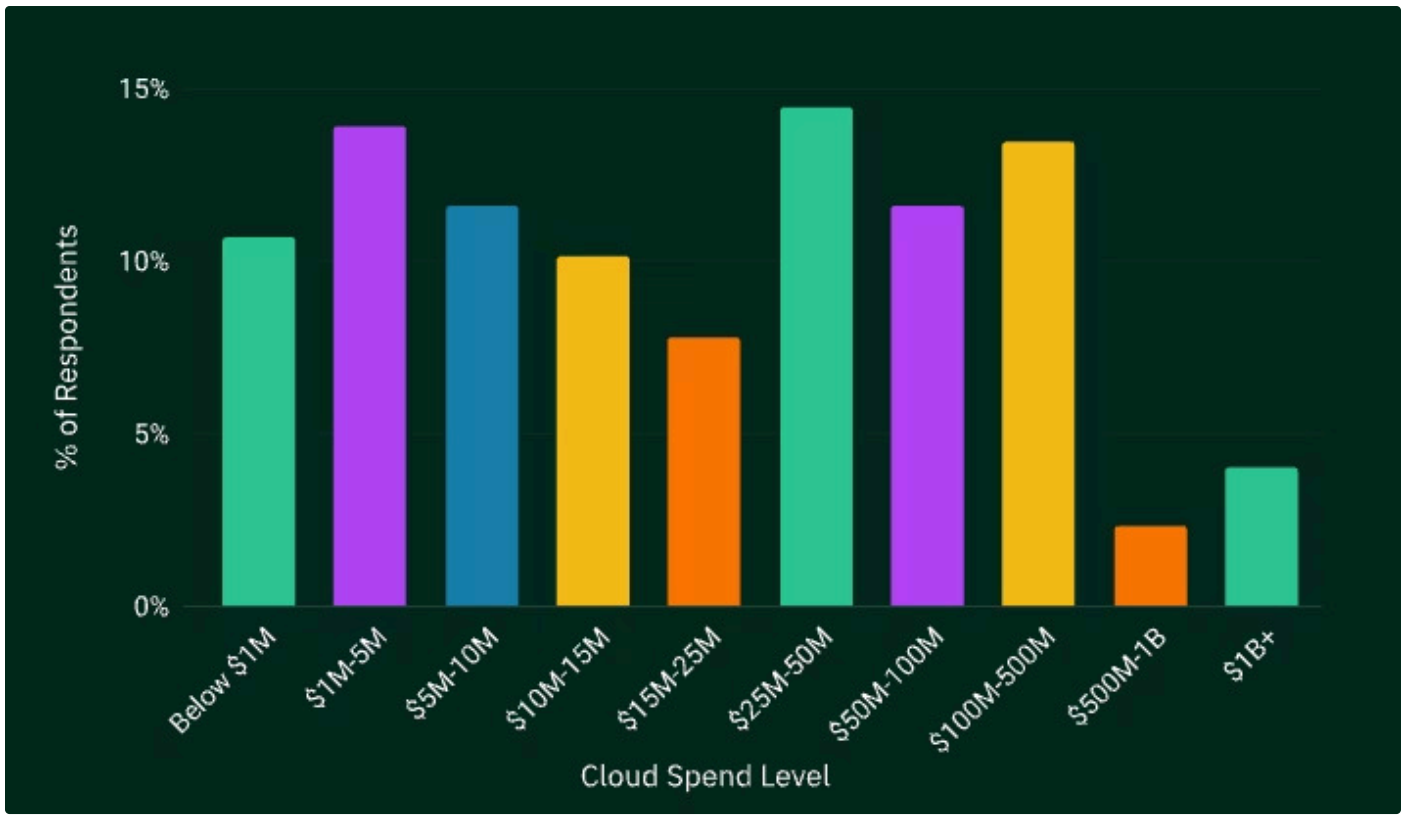
잘못된 선택으로 인한 예상치 못한 비용 증가와 리소스 낭비

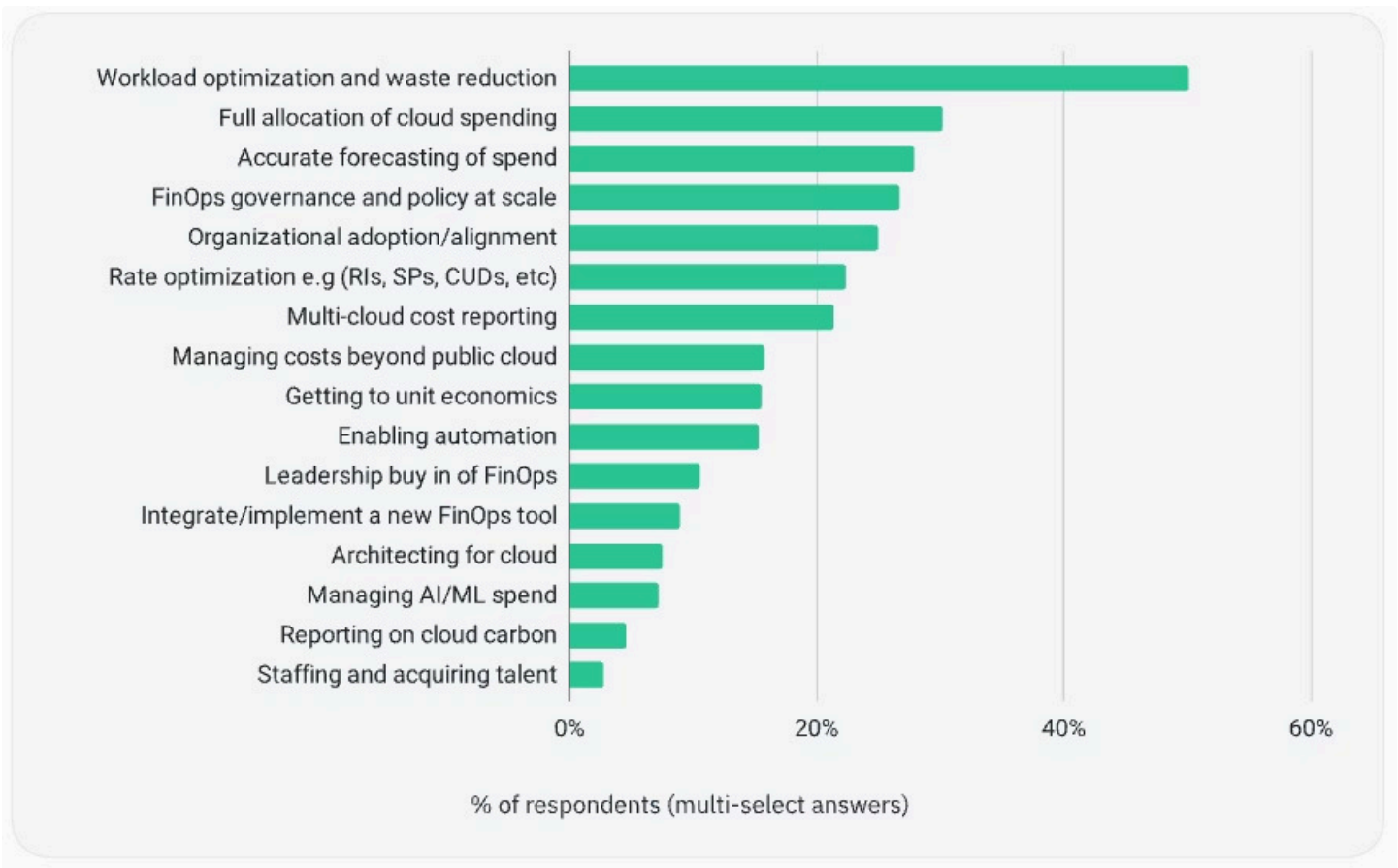
운영 복잡성

확장성 부족과 운영 관리의 복잡성 증가

결론: FinOps 관점에서 체계적인 선택 가이드가 필요합니다.

FinOps Foundation





FinOps Foundation에 따르면, 잘못된 리소스 매핑으로 **최대 50%의 비용 손실**이 발생할 수 있습니다.

한편, **AWS**와 여러 보고서에서 **EC2, Lambda, Fargate**의 비교표는 제공하고 있습니다.
하지만 실제로는 **사용자가 직접 해석해야 하고**, 이로 인해 잘못된 선택으로 인한 **비용 낭비**와 **운영 비효율**이 자주 발생합니다.

프로젝트 솔루션 개요

클라우드 컴퓨팅 서비스 추천 시스템

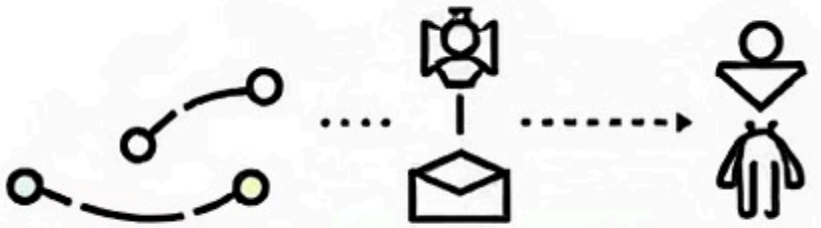
워크로드 정보를 입력하면 최적의 클라우드 실행 환경을 추천하는 서비스

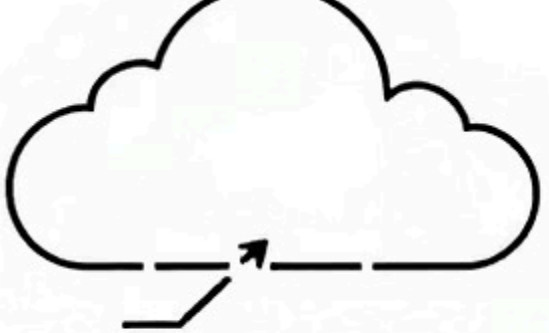
- 워크로드 특성 분석
- 실시간 추천 제공
- 근거 기반 설명

개발자의 고민: "EC2? Lambda? Fargate?" → 즉시 해답 제공!

서비스 비교

구분	EC2	Lambda	Fargate
비용 구조	시간/인스턴스 고정 요금	호출/실행시간 기반 과금	CPU/메모리 시간 기반 과금
확장성	Auto Scaling	이벤트 기반 자동	ECS 관리형
운영 방식	직접 관리	완전 서버리스	컨테이너 기반 매니지드
적합한 용도	지속적 고정 부하	짧은 이벤트 처리	중간 규모 컨테이너





기존 솔루션과의 차별점

기존 방식의 문제

- 개발자 개인의 경험과 감에 의존
- 산발적인 문서와 가이드만 존재
- 체계적인 비교 기준 부재

우리 프로젝트의 장점

- 워크로드 데이터 기반 분석
- 자동화된 추천 시스템 구현
- 명확한 선택 근거 제시

프로젝트 목표와 기대효과

비용 최적화

워크로드 특성에 맞는 최적 서비스 선택으로 클라우드 비용을 절감

효율적 리소스 사용

리소스 사용량과 성능 요구사항을 고려한 정확한 서비스 매칭

운영 단순화

복잡한 의사결정 과정을 자동화하여 개발팀의 생산성 향상

핵심 추천 로직

1. 입력 수집 (Workload Snapshot)

요청 수(평균/피크/월간), 실행시간, 메모리, CPU, 연결 특성 등 입력 DTO를 수집합니다.

2. 특징 추출 (Feature Engineering)

평균/피크 동시성, burstiness, duty(Always-on 여부), diurnalScore, memGB, coldStartRisk 등 핵심 특징(출력 DTO)을 계산합니다.

3. 라벨링 & 제약 확인

Always-on, Bursty, Cron-like, Long-conn 태그를 부여하고, Lambda 제한 조건(15분 실행 초과, 메모리 10GB 초과, 장기 연결 등) 플래그를 설정합니다.

4. 가중치 부여 (서비스별 적합도)

각 특징이 EC2 / Fargate / Lambda에 유리하거나 불리한 정도를 점수화하여 적합도를 평가합니다.

5. 최종 추천 (AnalysisResult)

설정된 제약 조건(flags 기준)을 기반으로 불가능한 서비스를 후보군에서 제외하여 최종 추천 서비스를 선별합니다.

6. 결과 출력

남은 서비스 중 점수가 가장 높은 옵션을 추천하고, 해당 추천의 상세 근거를 요약하여 제공합니다.

DTO 설계

입력 DTO

필드	의미	EC2 영향	Fargate 영향	Lambda 영향
region	실행 리전	요금표 반영	요금표 반영	요금표 반영
avgRps	평균 요청 수	최소 인스턴스 가늠	최소 태스크 가늠	동시성/GB-초 계산
peakRps	피크 요청 수	스케일 업 기준	스케일 아웃 기준	콜드스타트 민감도
monthlyRequests	월간 요청 수	고정비 대비 효율	고정비 대비 효율	요청 단가 직결
hourlyRps	시간대별 분포	예약 인스턴스 고려	스케줄 워크로드 유리	Cron-like 시 가점
avgExecMs	평균 실행시간	인스턴스 효율	태스크 효율	GB-초 과금 직결
p95ExecMs	95% 실행시간	스케일 여유 필요	안정적 스케일링	15분 제한 고려
memoryMB	메모리 요구량	대메모리 선택	메모리-CPU 조합	GB-초 과금, 4GB↑ 불리
requiredVcpu	CPU 요구량	CPU 인스턴스 활용	vCPU 조합 유연	CPU 많으면 비효율
tempStorageMB	임시 디스크	EBS 활용 자유	Volume 붙이기 자유	512MB↑ 불리
ioWaitRatio	I/O 비중	캐시/풀링 최적	I/O 부하 강점	CPU-intensive 아니면 비효율
coldStartAllowed	콜드스타트 허용	무관	Warm start 유리	허용 시 가점, 불허 시 감점
longLived / websocket	장기 연결	적합	적합	부적합
alwaysOn	상시 서비스	가점	가능하나 비용↑	비효율

출력 DTO

필드	의미	서비스별 영향
concurrencyAvg / Peak	평균/최대 동시성	EC2/Fargate 최소·최대 인스턴스 추정, Lambda 동시성 관리
duty	바쁨 비율	높으면 EC2 가점, Lambda 비효율
burstiness	피크/평균 비	Bursty+콜드스타트 허용 → Lambda 가점, 불허 → Fargate 가점
diurnalScore	시간대 집중도	Cron-like 패턴 → 컨테이너/스케줄 워크로드 유리
memGB	메모리 크기(GB)	4GB↑ → EC2/Fargate 선호, 0.5GB↓ → Lambda 선호
ioWaitRatio	I/O 비중	I/O↑ → EC2/Fargate 유리
coldStartRisk	콜드스타트 민감도	Lambda 적합성 판단
labels	워크로드 유형 태그	Always-on → EC2, Long-conn → Lambda 제외, Bursty → Lambda/Fargate
flags	제약 조건	Lambda 실행시간/메모리 초과 시 제외
notes	요약 설명	사용자에게 설명

특징 추출

입력 DTO를 기반으로 파생 지표를 계산해 워크로드의 **형태를 수치화**

- **동시성 (Concurrency)**
  - `concurrencyAvg` = 평균 요청 * 평균 실행시간
  - `concurrencyPeak` = 피크 요청 * p95 실행시간
- **듀티(Duty)**
  - 항상 바쁜 비율 (0~1), 0.4 이상이면 상시 서비스로 간주
- **버스티(Burstiness)**
  - `peakRps / avgRps` → 5 이상이면 고버스팅
- **메모리/CPU 지표**
  - `memGB = memoryMB / 1024`
  - `requiredVcpu`로 CPU 집중도 파악
- **일중 패턴 점수 (diurnalScore)**
  - 시간대별 요청 변동성 (0~1)
- **콜드스타트 민감도 (coldStartRisk)**
  - Cold Start 허용 여부 + burstiness/동시성 조합으로 LOW/MEDIUM/HIGH 판정

라벨링

워크로드 성격을 한눈에 표시 → 서비스별 가중치 부여에 활용

- **Always-on**: `duty ≥ 0.4` 또는 `alwaysOn=true` → **EC2** 가점
- **Bursty**: `burstiness ≥ 5` → **Lambda/Fargate**에 유리
- **Cron-like**: 특정 시간대만 트래픽 집중 → **Fargate**에 유리
- **Long-conn**: 장기 연결(WebSocket 등) 필요 → **Lambda** 제외
- **Steady**: 변동 적고 안정적 트래픽 → 세 서비스 모두 적합, 비용 고려

제약 조건

물리적 한계/제외 조건 → 불가능한 서비스 미리 필터링

- **EXCEED_LAMBDA_MAX_DURATION**
  - 실행시간 > 15분 → Lambda 불가
- **HIGH_MEMORY_PRESSURE**
  - 메모리 ≥ 10GB → Lambda 불가
- **LAMBDA_TMP_EXCEEDED**
  - 임시 디스크 > 512MB → Lambda 불리
- **LONG_CONNECTION_UNFAVORABLE_FOR_LAMBDA**
  - 장기 연결/웹소켓 필요 → Lambda 사실상 제외

서비스별 점수화 기준

Lambda

강점 (가점 요인)

- Bursty 트래픽 (급등락 대응)
- Cold Start 허용 시 유리
- 작은 메모리/짧은 실행시간 워크로드

약점 (감점 요인)

- 상시 트래픽 (Duty↑) → 비용 비효율
- 메모리 ≥ 4GB → 비용 증가, 10GB 이상이면 사실상 불가
- 임시 디스크 512MB 초과 → 추가 비용/비효율
- I/O 비중↑ → CPU 기반 과금이라 불리
- VPC 리소스 접근 → 초기 지연 발생

Fargate

강점 (가점 요인)

- CPU/메모리 요구량 큰 워크로드 → vCPU/메모리 조합 자유
- Cold Start 불허 환경 → 상시 Warm Container 유지 가능
- 스케줄형(Cron-like) 워크로드 → 태스크 단위로 실행 최적
- 중간 수준의 Bursty도 안정적으로 대응 가능

약점 (감점 요인)

- 소규모 트래픽 → 컨테이너 관리 오버헤드로 비효율
- 임시 디스크 과다 사용 → 추가 비용/성능 저하 가능

EC2

강점 (가점 요인)

- 상시 서비스 (Always-on, Duty↑)
- 장기 연결(WebSocket, SSE 등) 필요
- 고성능/대자원 워크로드 (메모리↑, CPU↑, I/O↑)
- 커스텀 최적화 가능 (인스턴스 패밀리 다양)

약점 (감점 요인)

- Bursty 트래픽 → 순간 스케일 아웃에 불리
- 소규모 워크로드 → 고정비 비효율

데모 시나리오

Case 1: 짧은 이벤트 기반 **API**

> **"매일 수천 건의 짧은 요청이 몰리는 이벤트 알림 서비스"**
>
> **추천 결과: Lambda**
>
> **근거:** 짧은 실행시간과 이벤트 기반 특성으로 서버리스 아키텍처가 최적

Case 2: 장기 연결 웹서비스

> **"실시간 알림/채팅이 있는 상시 웹 서비스"**
>
> **추천 결과: EC2**
>
> **근거:** 장기 연결 및 상태관리에 안정적인 EC2가 비용 효율적

향후 발전 계획

1단계: 로직 개선

좀 더 다양한 입력 항목 추가 및 점수화 로직 개선

1

2단계: 비용 계산 기능 추가

실시간 비용 계산 API 연동 및 AWS Pricing API 활용으로 정확한 비용 예측

2

3단계: 사용자 UI/UX 개선

사용자가 서비스를 사용하는데 문제 없도록 UI/UX 개선

3