Amazon ECS Essentials Tips #1-2

신 정 섭

Solutions Architect

Agenda

- Fargate Dive Deep
- Service Connect
- 카오스 엔지니어링를 위한 AWS FIS
- Amazon ECS 비용 최적화
- 대규모 트래픽 준비하기

카오스 엔지니어링를 위한 AWS FIS 사용하기

카오스 엔지니어링이란?

- 2010년 Netflix 엔지니어가 제안하여 시스템 엔지니어링 방법으로 진화
  - 최초의 카오스 몽키라는 도구는 AWS 인프라의 인스턴스를 무작위로 마비시키도록 고안됨
  - https://netflix.github.io/chaosmonkey/
- 왜 카오스라는 용어를 사용했을까?
  - 분산 시스템에서 개별 서비스는 정상적으로 동작해도 상호 작용은 카오스 그 자체
  - 폭발적인 이벤트로 인해서 예측 불가능한 상황에 빠질 수 있음
  - 단위 테스트, 통합 기능 테스트의 한계 (모르는데 어떻게 테스트해요)

카오스 엔지니어링이란?

- 카오스 엔지니어링이란, 복잡하며 대규모의 분산 시스템에 대한 신뢰 확보하기 위해서 시스템에 대한 실험을 통해 생산 환경에서의 불안정한 조건을 처리할 능력에 대한 신뢰를 구축하는 것
- 환경을 무작위로 파괴하는 것이 아니라, 통제되는 환경에서 계획된 실험을 통해서 시스템의 견고함을 확인하는 것

카오스 엔지니어링 도입은 어려움

1

다양한 도구와
기존 스크립트
결합

에이전트 또는
라이브러리 설치
필요

2

3

실험 안정성
확보가 어려움

"실제"이벤트를
재현하기 어려움
(한 번에 다중 장애)

4

카오스 엔지니어링는 <u>실험도구</u>를 사용하여 <u>제어</u>를 수행하는 것이 필수

카오스 엔지니어링 과정

정상 상태

비지니스에 영향을 주는 통계치 사용

Amazon: 100 ms of extra load time caused a 1% drop in sales (Greg Linden).

가설 수립

What if?

- 404 혹은 503 상태인가?
- 300ms까지 지연중인가?
- 서버에 접속할 수 있는가?

실험 디자인

통제된 환경에서 실험 구성

- 실험할 가설 선택
- 실험 범위 측정 지표 정하기
- 실험 계획 알리기 및 실행

결과 확인

실험 평가 및 오류 확인

- 문제 인지 및 알림 시간?
- 자동 롤백 시작 시점?
- 전체 복구 및 안정화 시간?

문제점 수정

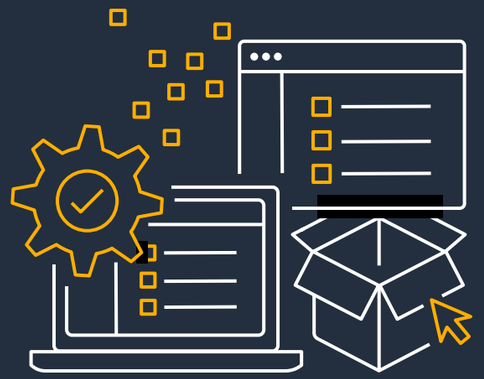
재발 방지 대책 수립

- 문제점 해부(PostMortems)
- 오류 해결(Correction of Errors)

참고 – 카오스 엔지니어링 원칙 Principlas of Chaos Engineering
http://channy.creation.net/blog/netflix-principles-of-chaos-engineering

AWS FIS (Fault Injection Simulator)

완전 관리형 카오스 엔지니어링 서비스

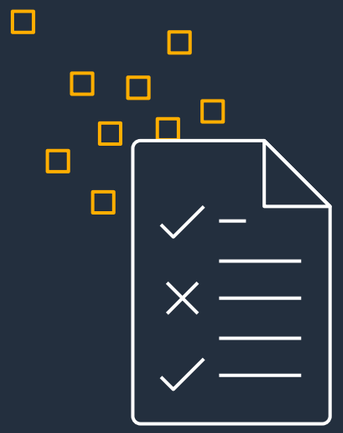


손쉽게 장애 주입

실제 장애 사례 기반

안전하게 실험 가능

AWS FIS 구성요소



Actions
(작업)

Targets
(대상)

Experiment templates
(실험 템플릿)

Experiments
(실험)

AWS FIS 구성요소 (1) Actions

Actions (작업)

Actions – 실험 중에 실행되는 오류 주입 내용

aws:⟨service-name⟩:⟨action-type⟩

작업 내용:

- 장애 유형
- 지속 시간
- 대상 자원
- 다른 작업과 관련된 타이밍
- 롤백 동작 같은 오류 별 매개 변수

AWS FIS 구성요소 (2) Targets

Targets (대상)

Targets (대상) – 작업을 수행 할 하나 이상의 AWS 리소스

대상 내용:

- 리소스 유형
- 리소스 IDs, 태그 및 필터
- 선택 모드 (e.g., ALL, RANDOM)

AWS FIS 구성요소 (3) Experiment templates

Experiment templates
(실험 템플릿)

Experiment templates 실험 시작을 위한 구성

템플릿 내용:

- Actions (작업)
- Targets (대상)
- Stop condition alarms (알람 조건)
- IAM 역할
- 설명
- 태그

AWS FIS 구성요소 (4) Experiments

Experiments

Experiments (실험)은 실험 템플릿이 실행된 목록 및 내역

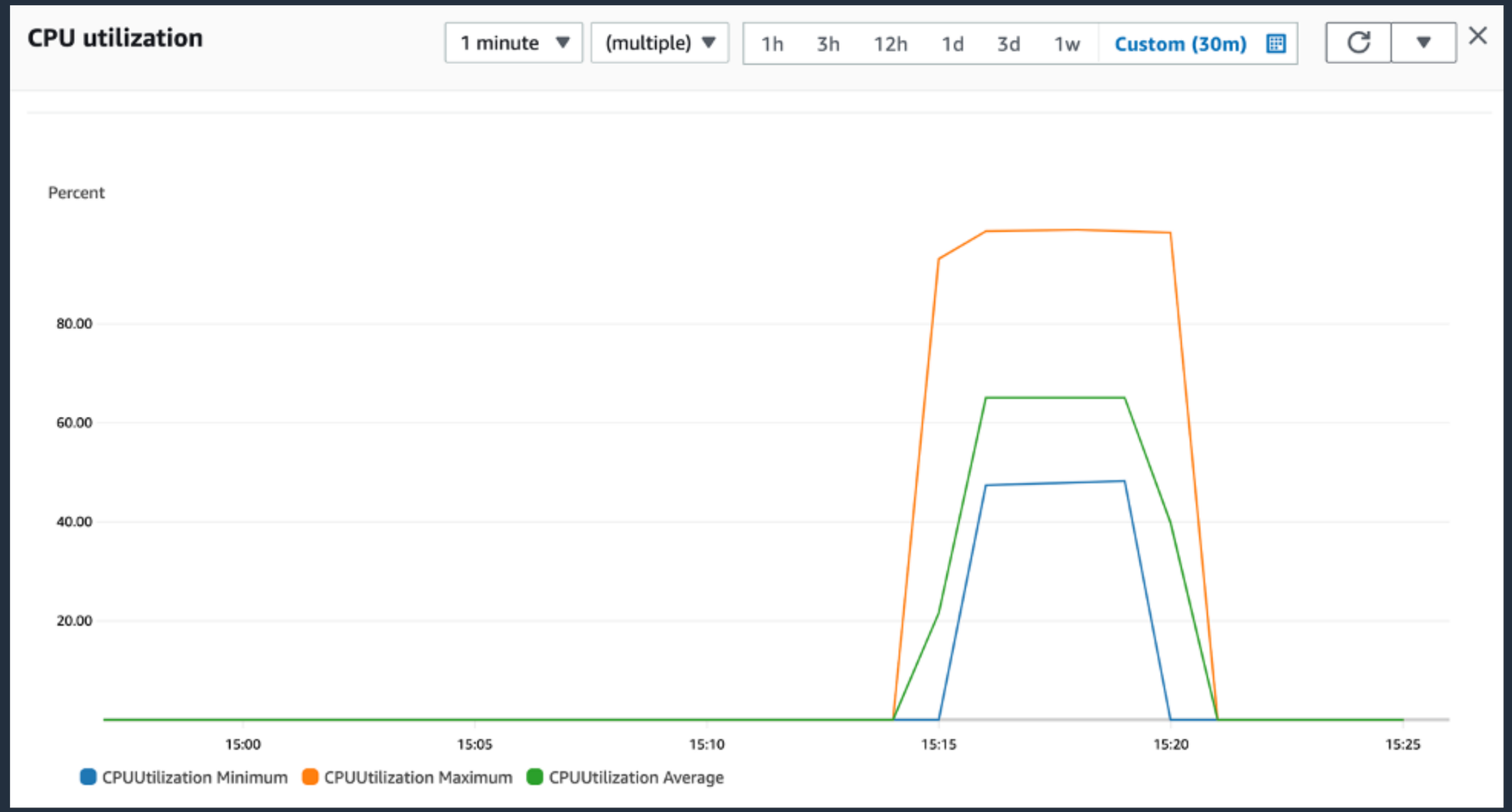
실험 결과에 있는 항목:

- 실험 스냅 샷
- 생성 및 시작 시간
- 상태
- 실행 ID
- 실험 템플릿 ID
- IAM 역할 ARN

Amazon ECS 태스크 Actions

Action Identifier	Description	Applicable Compute Engine
aws:ecs:task-cpu-stress	CPU Stress 시뮬레이션	Amazon EC2 and AWS Fargate
aws:ecs:task-io-stress	I/O Stress 시뮬레이션	Amazon EC2 and AWS Fargate
aws:ecs:task-kill-process	특정 프로세스 종료 시뮬레이션	Amazon EC2 only
aws:ecs:task-network-blackhole-port	버려지는 네트워크 트래픽 시뮬레이션	Amazon EC2 only
aws:ecs:task-network-latency	네트워크 레이턴시 시뮬레이션	Amazon EC2 only
aws:ecs:task-network-packet-loss	네트워크 패킷 손실 시뮬레이션	Amazon EC2 only

Amazon ECS aws:ecs:task-cpu-stress 액션 실험



Amazon ECS aws:ecs:task-cpu-stress 액션 실험

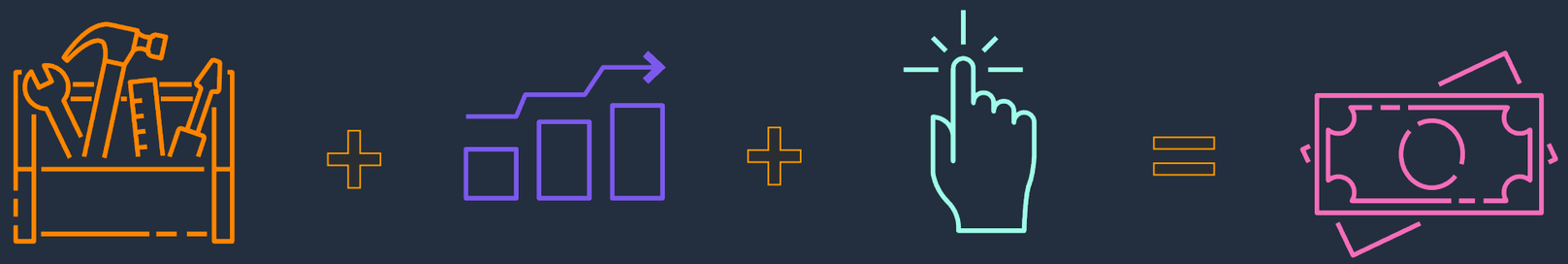
Amazon ECS 비용 최적화

Amazon ECS 비용 최적화 3가지 핵심원리

운영 효율화

리소스 활용도 높이기

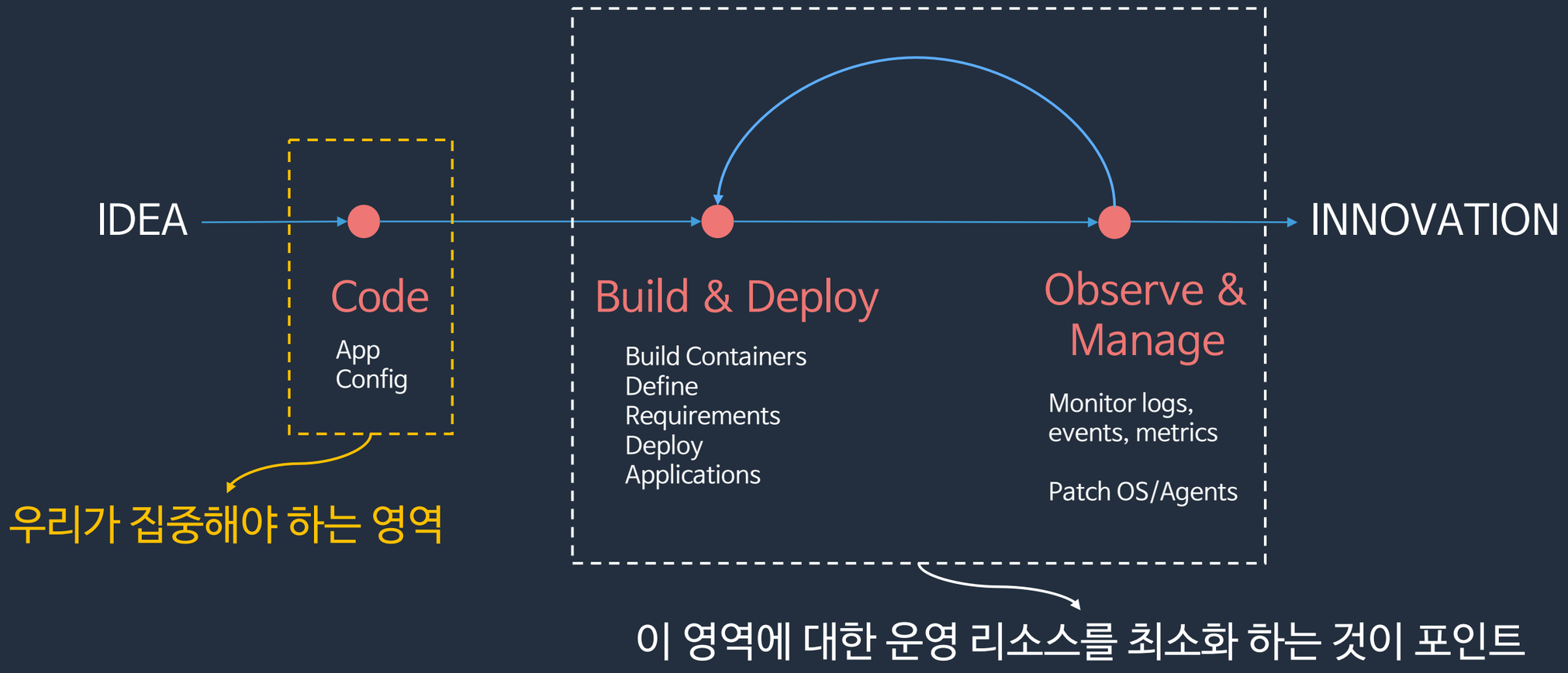
리소스 다양하게 활용하기



Operate Less + Utilize More + Choose Wisely =

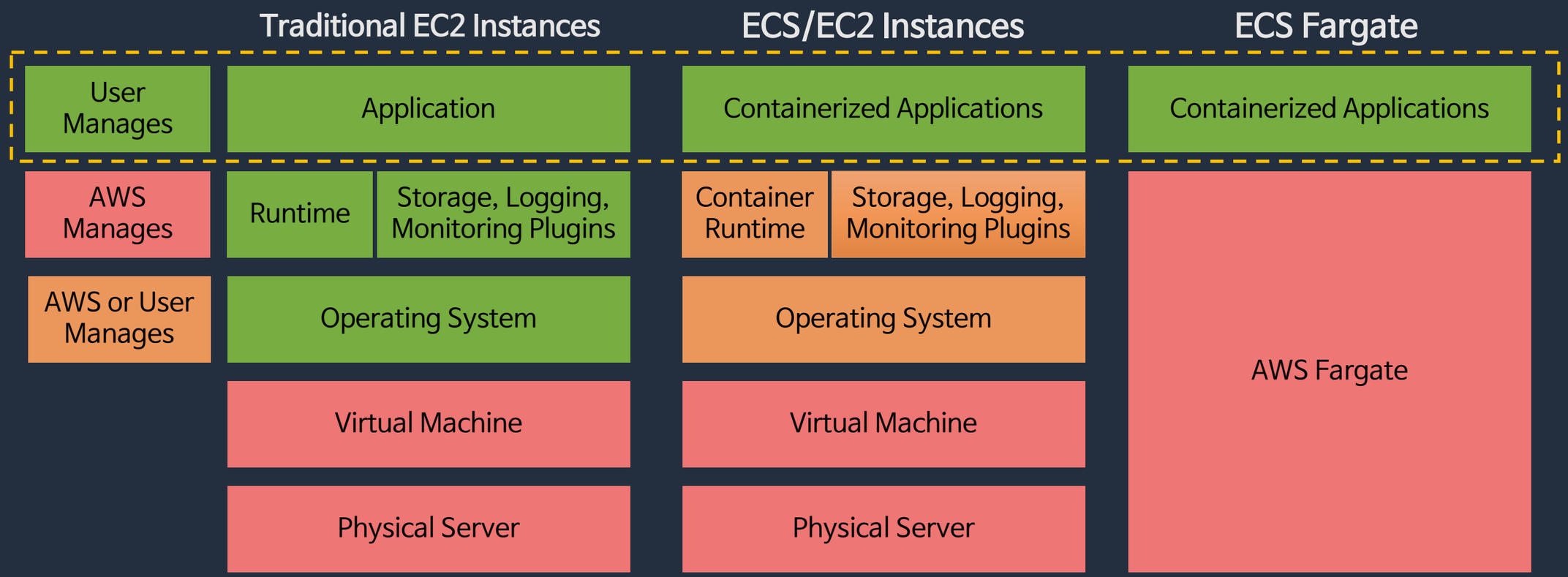
운영 효율화

1. Narrow our focusing area



운영 효율화

1. Narrow our focusing area



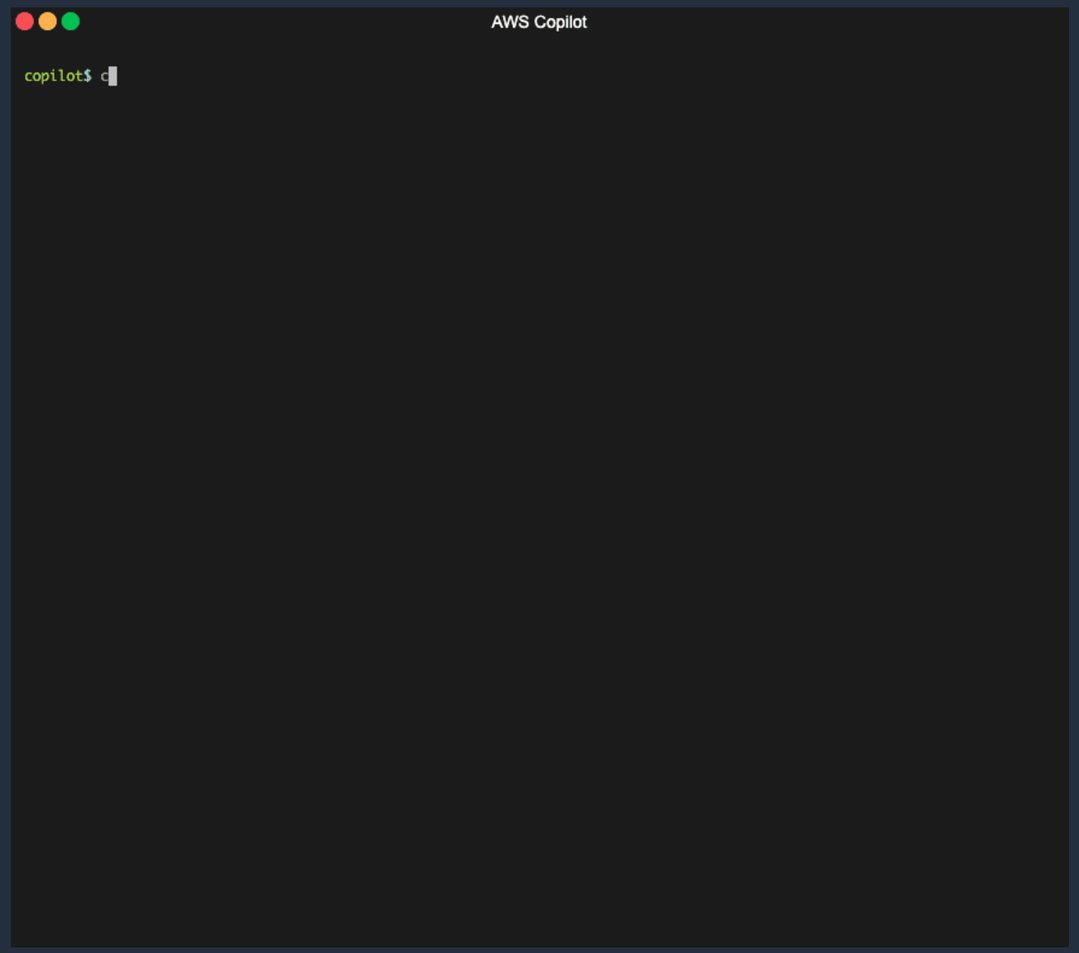
운영 효율화

2. Infrastructure as a Code

- 1/ Amazon ECS Blueprints
  - IaC 도구를 처음 사용하는 사용자를 위한 모범 사례 / well-designed 패턴의 IaC 솔루션
  - Terraform / CDK 지원
  - https://github.com/aws-ia/ecs-blueprints
- 2/ Amazon ECS CDK extensions
- 3/ AWS Copilot CLI
  - Amazon ECS 기반의 클라우드 아키텍처 구성을 지원해주는 도구
  - Auto Scaling, CD pipeline, metrics, alarms and logs 구성

운영 효율화

2. Infrastructure as a Code

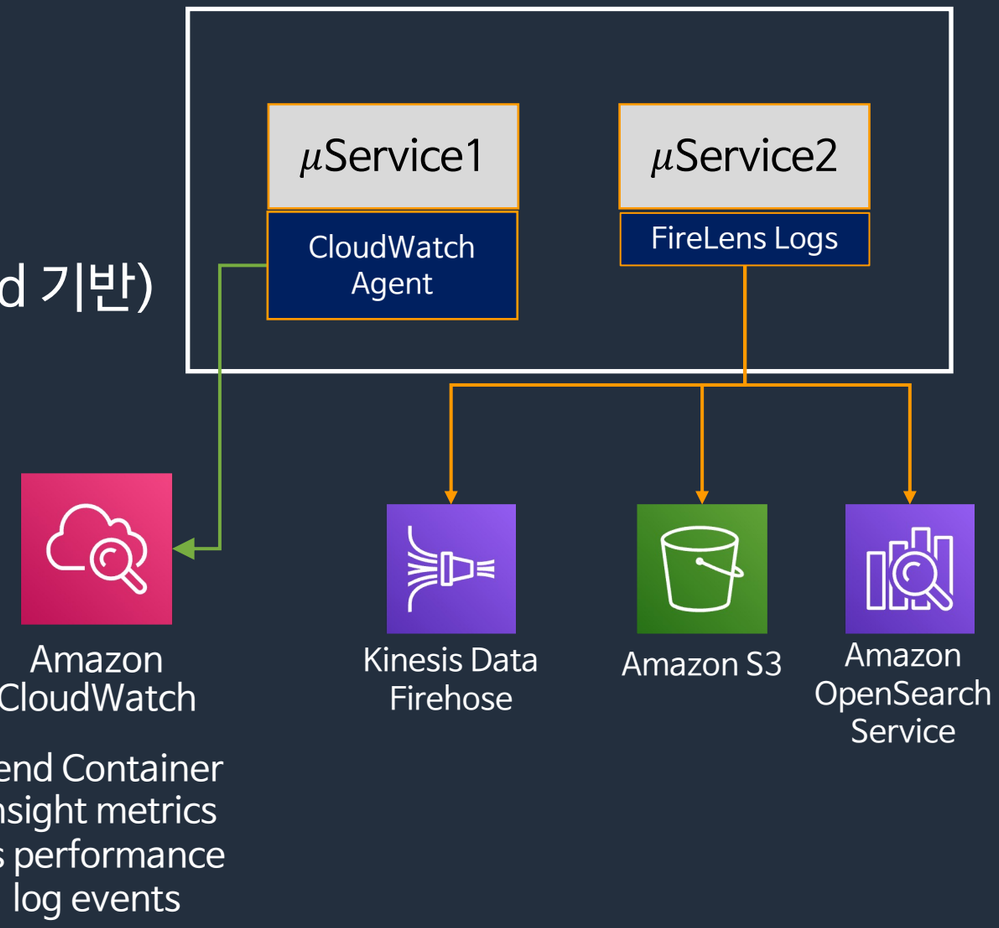


- Well-architected 기반의 인프라 구성
- Continuous deployment pipelines
- 효율적인 트러블슈팅과 운영
- 애플리케이션 개발에 집중

운영 효율화

3. Use Observability Tools

- CloudWatch Logs
  - awslogs 로그 드라이버 제공
  - AWS FireLens 로그 라우터 제공 (Fluent Bit, Fluentd 기반)
- CloudWatch Container Insights
  - 컨테이너 지표, 성능 로그를 수집하고 집계하여 요약
  - 기본 제공되는 대시보드



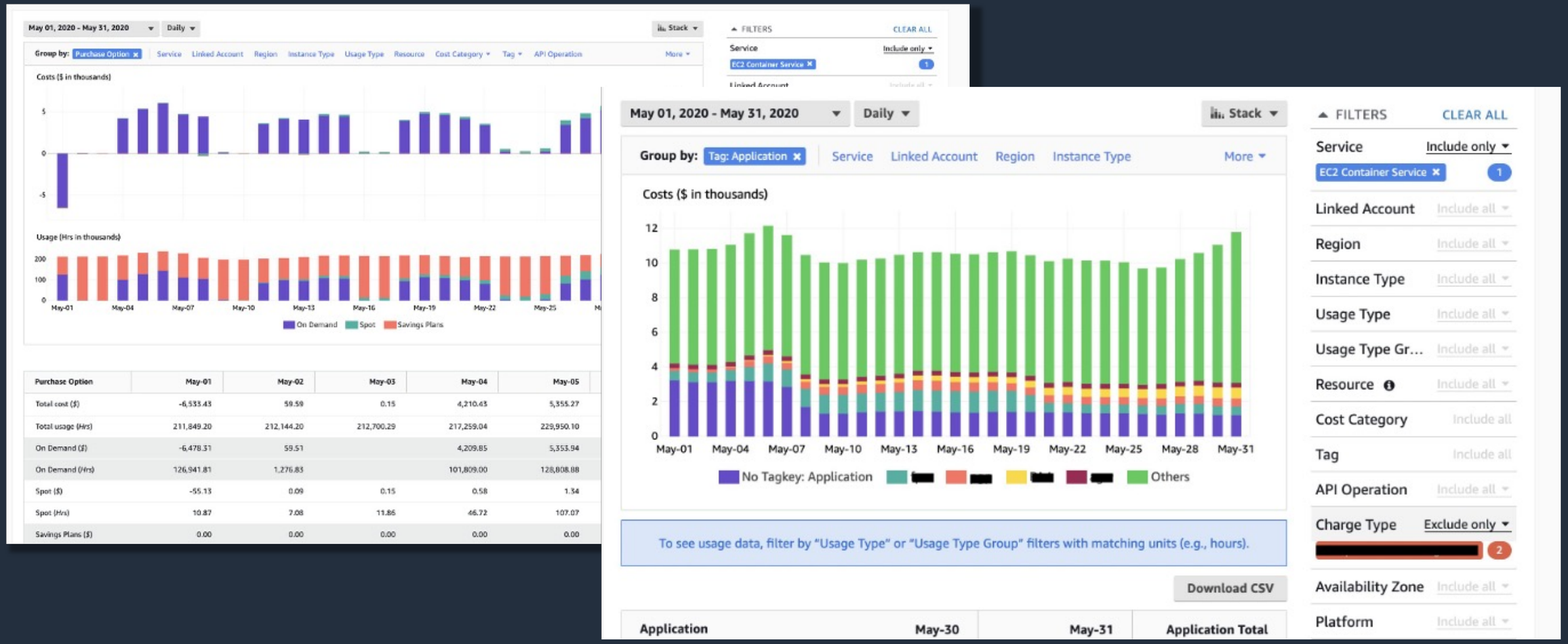
운영 효율화

4. Cost Visibility Tools

- 비용 최적화를 위한 첫 번째 단계는 비용 가시성 확보
- Amazon ECS on Fargate의 경우 작업 단위로 비용 가시성을 확보할 수 있음
- AWS Cost Explorer를 활용하여 비용 가시성 확보하기
  - 비용이 고정적이라면 사용량을 확인해보고 리소스 효율화하기
  - 비용이 많이 지출되는 Task를 최적화

운영 효율화

4. Cost Visibility Tools



리소스 활용도 높이기

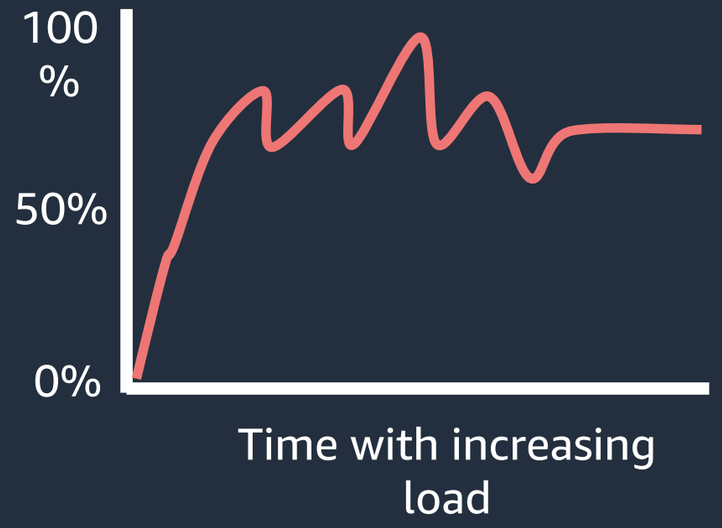
1. 적절한 Task 사이즈 찾아가기

- 처음부터 적절한 Task 사이즈를 찾는 것은 쉽지 않음
- Observability 도구의 Metric 정보를 활용해서 리소스 사용량을 파악
  - CPU / Memory utilization 을 시작으로 워크로드에 맞는 기준을 찾기
  - 평균의 함정에 빠지지 않기(기간 별 평균 사용량은 기간 조건에 따라서 결과가 매우 달라짐)
  - 일정 기간(일, 주, 월)에 대한 측정 값을 파악해서 조정해나가기

리소스 활용도 높이기

2. 적절한 Task 개수 찾아가기

- Task 개수는 유연하게 운영하는 전략이 유리함. 결국 오토 스케일링이 핵심
- 오토 스케일링 기준 설정하기
  - ECS 제공 Metric (CPU, Memory utilization, Service Connect Metric)
  - 미리 정의된 Metric (ELB 평균 latency, SQS queue depth)
  - 사용자 정의 Metric (SQS backlog per task)
- 하지만, Task 개수로 해결하기 어려운 상황도 있음
  - Cache 사용, Gateway를 통한 Throttling, Event Eriven 구조 등



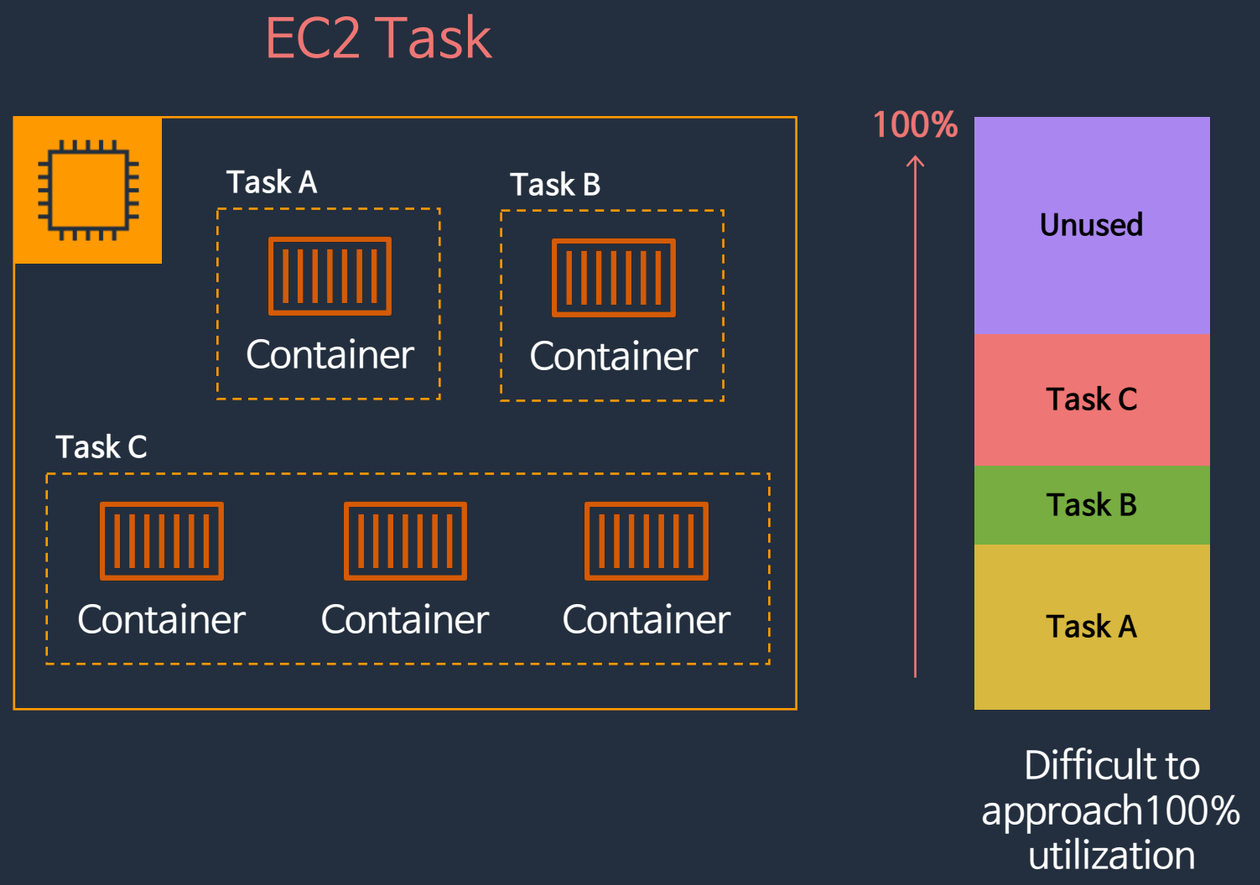
리소스 활용도 높이기

3. 적절한 Task 컴퓨팅 용량 찾아가기

- 적절한 리소스 여유를 유지하면서 Task 개수와 사이즈를 운영하는 것은 더 어렵다!
- ECS on EC2의 경우, 용량 공급자를 사용하면 자동 계산되면서 컴퓨팅 용량을 조정
  - 하지만 용량 공급자의 목적은 스케일링을 위한 도구로 컴퓨팅 용량을 위한 도구는 아님
  - EC2 Scale-out은 비용이 크기 때문에 여유 리소스가 너무 작아도 스케일링에 불리함
- ECS on Fargate의 경우, 1 Task 1 VM 구조라서 여유 리소스를 신경쓸 필요 없음
  - 사전에 Task에 알맞은 컴퓨팅 리소스를 결정했다는 가정

리소스 활용도 높이기

3. 적절한 Task 컴퓨팅 용량 찾아가기



리소스 활용도 높이기

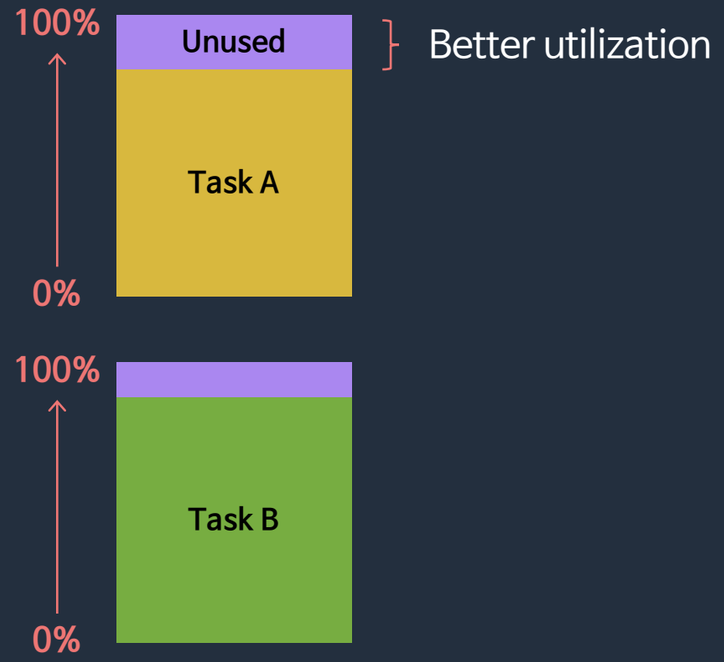
3. 적절한 Task 컴퓨팅 용량 찾아가기

Fargate Task

필요한 메모리 및 vCPU 리소스를 기반으로
각 작업의 크기를 알아서 조정

Resource utilization

Fargate로 리소스를 더욱 세밀하게 제어하여
낭비되는 용량을 최소화할 수 있음



리소스 다양하게 활용하기

1. Graviton 프로세서 활용하기

- Amazon Linux 2(arm64) AMI for EC2 제공
- Fargate 도 AWS Graviton2 프로세서를 지원
- Multi-arch images 완전 지원
- 고성능, 20% 저렴한 가격, 가격대비 성능비 40% 높음
- 다양한 AWS 서비스에서 활용 중이며 많은 고객 사례가 쌓이고 있음
- 애플리케이션 테스트를 통해서 바로 적용 가능함

리소스 다양하게 활용하기

2. Spot 인스턴스 활용하기

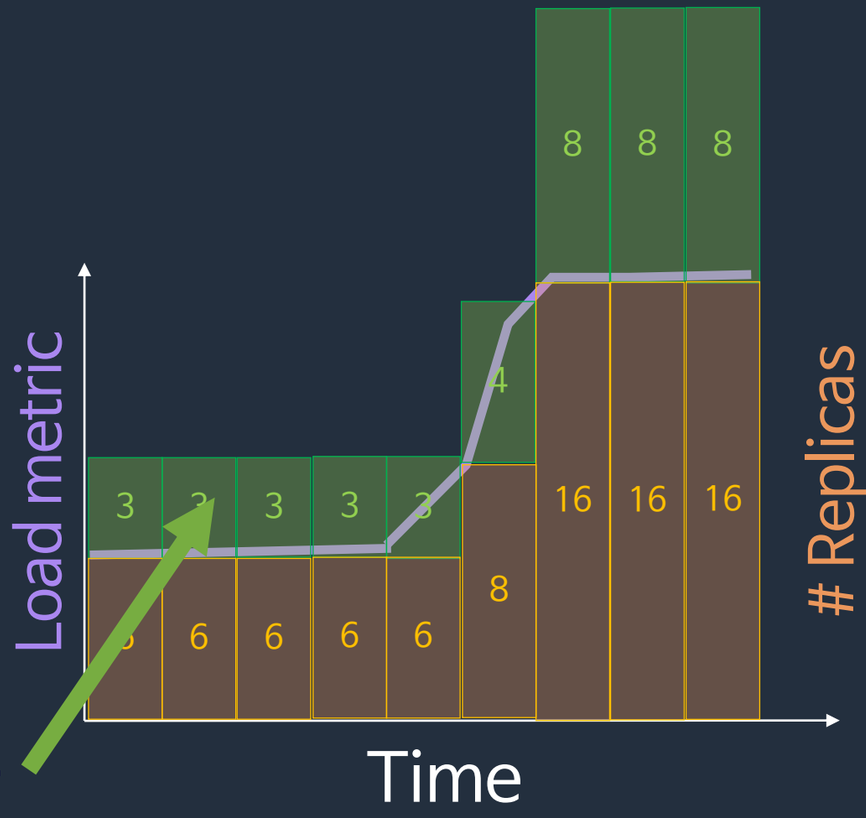
- EC2 Spot 인스턴스는 최대 90% 저렴 (instance pool 선택해야 함)
- Fargate Spot 인스턴스는 최대 70% 저렴 (자동으로 다양하게 선택됨)
- 회수 가능성이 있음 (경고 후 최대 2분)
  - 애플리케이션은 내결함성이 있도록, Stateless 하도록 구성
  - Amazon EventBridge를 통해 Spot 인스턴스 회수 신호(Task 상태 변경)를 받을 수 있음
- 단, Amazon ECS의 경우
  - Task 중단 및 종료 처리를 자동으로 진행
  - ELB 연결 종료를 자동으로 Graceful하게 Drain 처리함

리소스 다양하게 활용하기

3. On-demand & Spot 혼합하여 사용하기

Overprovision by 50%:
Reduce metric target value by 1/3

Run 2/3 On-Demand, 1/3 on Spot



+50% capacity for
+5 - 10% cost

리소스 다양하게 활용하기

4. Savings Plans 사용하기

Instance Saving Plans

- **Instance Savings Plans** apply savings to selected instance usage across any tenancy or OS

Compute Savings Plans

- **Compute Savings Plans** discounts apply to **any** compute usage automatically
- **Compute Savings Plans** apply to **Fargate**, **AWS Lambda**, and **Amazon EC2**

Savings Plans 은 예약 인스턴스와 동일한 비용 절감 효과를 제공하지만
더 **넓은 범위**와 획기적으로 **향상된 유연성**을 제공하며 **오버헤드 관리를 최소화**합니다.

Thank you!

신 정 섭