AWS 결정 가이드

AWS Fargate 또는 AWS Lambda?

AWS Fargate 또는 AWS Lambda?: AWS 결정 가이드

Table of Contents

결정 가이드 1
소개 1
차이 4
사용 10
문서 기록 12
..... xiii

AWS Fargate 또는 AWS Lambda?

차이점을 이해하고 자신에게 적합한 것을 선택하세요.

용도	서버리스 컴퓨팅 서비스에 대한 요구 AWS Lambda 사항을 충족하는지 AWS Fargate 여부를 탐색합니다.
최종 업데이트 날짜	2024년 11월 15일
적용 대상 서비스	• AWS Fargate • AWS Lambda

소개

AWS Fargate 서버리스 컴퓨팅 서비스를 선택할지 AWS Lambda 여부를 탐색하기 전에 더 광범위한 AWS 컴퓨팅 서비스(AWS 컴퓨팅 서비스 선택 결정 가이드에서 설명)를 고려하고 다음 두 가지 선택 사항으로 범위를 좁혔을 것입니다.

- 운영 오버헤드 감소: Lambda와 Fargate 모두 서버 관리를 추상화하여 패치 적용, 유지 관리 및 용량 계획의 필요성을 줄입니다.
- Pay-per-use: 실제로 사용하는 컴퓨팅 리소스에 대해서만 비용을 지불하면 가변 워크로드 비용이 절감될 수 있습니다.
- 더 빠른 배포: 일반적으로 EC2 인스턴스 프로비저닝 및 구성에 비해 더 빠른 배포 시간을 제공합니다.
- 기본 제공 고가용성: 두 서비스 모두 인프라 중복을 자동으로 처리합니다.
- 간소화된 규정 준수: 공격 표면이 줄어들고 보안 기능이 내장되어 규정 준수 작업을 간소화할 수 있습니다.
- 코드에 집중: 개발자는 인프라를 관리하는 대신 애플리케이션 코드 작성에 더 집중할 수 있습니다.

Lambda와 Fargate는 모두 서버리스 옵션이지만 둘 사이에는 상당한 차이가 있습니다.

AWS Fargate는 컨테이너용 서버리스 컴퓨팅 엔진으로, 주로 Amazon ECS와 함께 사용됩니다. 인프라를 자동으로 관리하므로 컨테이너화된 애플리케이션을 배포하고 확장하는 데 집중할 수 있습니다.

Fargate는 리소스 할당(CPU, 메모리)을 세밀하게 제어해야 하고 기본 서버를 관리하지 않으려는 장기 실행 애플리케이션, 마이크로서비스 또는 배치 처리에 적합합니다.

AWS Lambda는 이벤트에 대한 응답으로 코드를 자동으로 실행하고 기본 컴퓨팅 리소스를 관리하는 서버리스 컴퓨팅 서비스입니다. Amazon S3에 업로드된 파일 처리, HTTP 요청에 응답 또는 예약된 작업 실행과 같은 이벤트 기반 애플리케이션에 가장 적합합니다. Lambda는 이벤트에 대응하여 자동으로 규모를 조정하고 대량의 데이터를 실시간으로 처리할 수 있기 때문에 스트림 처리 및 데이터 처리 애플리케이션에도 적합합니다. Lambda는 Amazon Kinesis 또는 Amazon DynamoDB와 같은 소스의 데이터 스트림을 처리할 수 있으므로 인프라를 관리하지 않고도 효율적인 서버리스 데이터 변환, 필터링 및 분석을 수행할 수 있습니다. Lambda는 수명이 짧은 작업(최대 15분)을 위해 설계되었으며 요청 수와 실행 시간에 따라 요금이 청구되므로 산발적인 워크로드에 비용 효율적입니다.

프로젝트에 이벤트 중심의 단기 작업 또는 예측할 수 없는 워크로드가 포함된 경우가 더 적합할 AWS Lambda 수 있습니다. 특정 리소스 요구 사항이 있는 컨테이너화된 애플리케이션을 실행해야 하는 경우(또는 영구 프로세스가 필요한 경우) AWS Fargate 가 더 적절할 것입니다.

다음 표에서는 시작하기 위해 이러한 서비스 간의 몇 가지 차이점을 자세히 살펴봅니다.

Feature	AWS Fargate	AWS Lambda
실행 모델	컨테이너 기반 서버리스 컴퓨팅	이벤트 기반 서버리스 함수
지원되는 언어	컨테이너에서 실행할 수 있는 모든 언어	지원되는 언어: Node.js, Python, Java, C#, Go, Ruby 및 PowerShell. 또한 사용자 지정 런타임을 구축하여 원하는 언어로 AWS Lambda 함수를 구현할 수 있습니다.
사용 사례:	장기 실행 컨테이너화된 애플리케이션	단기 이벤트 기반 작업
스케일링	원하는 작업 수에 따른 자동 조정	요청당 자동 조정
콜드 스타트	35초~2분	100ms~2초
실행 시간 제한	하드 제한 없음	최대 15분

메모리 할당	최대 120GiB	최대 10GiB
CPU 할당	최대 16개의 vCPU	메모리에 비례, 최대 6개의 vCPU
네트워킹	VPC에서 실행, ENIs 사용 가능	AWS 관리형 VPC에서 실행하거나 Hyperplane을 사용하여 AWS 고객 관리형 VPC에 연결할 수 있습니다.
상태 관리	Fargate의 컨테이너는 컨테이너가 실행되는 동안 요청 전반에서 상태를 유지할 수 있으므로 외부 스토리지 없이 세션을 처리하거나, 데이터를 캐시하거나, 메모리 내 상태를 유지할 수 있습니다. 중요한 데이터에는 외부 스토리지를 사용하는 것이 좋습니다.	상태 비저장 설계(상태는 Amazon S3, Amazon DynamoDB, Amazon EFS 등 외부에서 관리해야 함)
컨테이너 지원	컨테이너 지원	제한된 컨테이너 지원(컨테이너 이미지 배포를 통해)
오케스트레이션	Amazon ECS와 통합	오케스트레이션 필요 없음
요금 모델	vCPU 및 사용된 메모리에 대한 초당 결제	호출당 및 지속 시간(GB-초)
동시성 제한	클러스터 용량 기준	기본적으로 동시 실행 1,000개(증가 가능)
이벤트 기반 호출	추가 설정 필요	다양한 AWS 이벤트 소스에 대한 기본 지원
콜드 스타트 완화	Seekable OCI를 사용하여 이미지를 지연 로드하면 Fargate 작업 시작 속도가 빨라질 수 있습니다.	프로비저닝된 동시성 사용 가능

패키지 크기 제한	특정 제한 없음(구성된 임시 스토리지에 의해 컨테이너 크기가 제한됨, 최대 200GiB)	계층을 포함한 250MB 압축 해제, 컨테이너 이미지 배포의 경우 10GB

Fargate와 Lambda의 차이점

여러 주요 영역에서 Fargate와 Lambda의 차이점을 살펴봅니다.

Languages supported

Fargate: AWS Fargate 는 컨테이너 오케스트레이션 서비스입니다. 즉, Docker 컨테이너에 패키징할 수 있는 모든 프로그래밍 언어 또는 런타임 환경을 지원합니다. 이러한 유연성을 통해 개발자는 애플리케이션 요구 사항에 맞는 거의 모든 언어, 프레임워크 또는 라이브러리를 사용할 수 있습니다. Python, Java, Node.js, Go, .NET, Ruby, PHP 또는 사용자 지정 언어와 환경을 사용하든 관계없이 Fargate는 컨테이너에 캡슐화되어 있는 한 이를 실행할 수 있습니다. 이러한 광범위한 언어 지원을 통해 Fargate는 레거시 시스템, 다국어 마이크로서비스, 최신 클라우드 네이티브 애플리케이션을 비롯한 다양한 애플리케이션을 실행하는 데 이상적입니다.

Lambda:는 이벤트 기반 함수용으로 특별히 설계된 Fargate에 비해 더 제한된 언어 집합에 대한 기본 지원을 AWS Lambda 제공합니다. 현재 Lambda는 다음 언어를 공식적으로 지원합니다.

- Node.js
- Python
- Java
- Go
- Ruby
- C#
- PowerShell

Lambda는 자체 언어 또는 런타임 환경을 가져올 수 있는 사용자 지정 런타임도 지원하지만, 이를 위해서는 기본적으로 지원되는 옵션을 사용하는 것보다 더 많은 설정 및 관리가 필요합니다. 컨테이너 이미지에서 Lambda 함수를 배포하도록 선택한 경우 AWS OS 전용 기본 이미지를 사용하고 이미지에 Rust 런타임 클라이언트를 포함하여 Rust로 함수를 작성할 수 있습니다. AWS제공 런타임 인터페이스 클라이언트가 없는 언어를 사용하는 경우 직접 생성해야 합니다.

Event-driven invocation

Lambda는 본질적으로 이벤트 기반 컴퓨팅을 위해 설계되었습니다. Lambda 함수는 데이터, 사용자 작업 또는 예약된 작업의 변경 사항을 비롯한 다양한 이벤트에 대한 응답으로 트리거됩니다. Amazon S3(예: 파일 업로드 시 함수 호출), DynamoDB(예: 데이터 업데이트 시 트리거) 및 API Gateway(예: HTTP 요청 처리) AWS 서비스와 같은 많은와 원활하게 통합됩니다. Lambda 이벤트 기반 아키텍처는 영구 컴퓨팅 리소스 없이 이벤트에 즉시 응답해야 하는 애플리케이션에 적합합니다.

Fargate는 기본적으로 이벤트 기반이 아니지만 일부 추가 표준 문안 로직을 사용하면 Amazon SQS 및 Kinesis와 같은 이벤트 소스와 통합할 수 있습니다. Lambda는이 통합 로직의 대부분을 처리하지만 이러한 서비스의 APIs를 사용하여이 통합을 직접 구현해야 합니다.

Runtime/use cases

Fargate는 컨테이너화된 애플리케이션을 실행하도록 설계되어 컨테이너에 대한 CPU, 메모리 및 네트워킹 설정을 정의할 수 있는 유연한 런타임 환경을 제공합니다. Fargate는 컨테이너 기반 모델에서 작동하므로 장기 실행 프로세스, 영구 서비스 및 특정 런타임 요구 사항이 있는 애플리케이션을 지원합니다. 실행 시간에 대한 엄격한 제한이 없으므로 Fargate의 컨테이너는 무기한 실행될 수 있으므로 지속적으로 실행해야 하는 애플리케이션에 적합합니다.

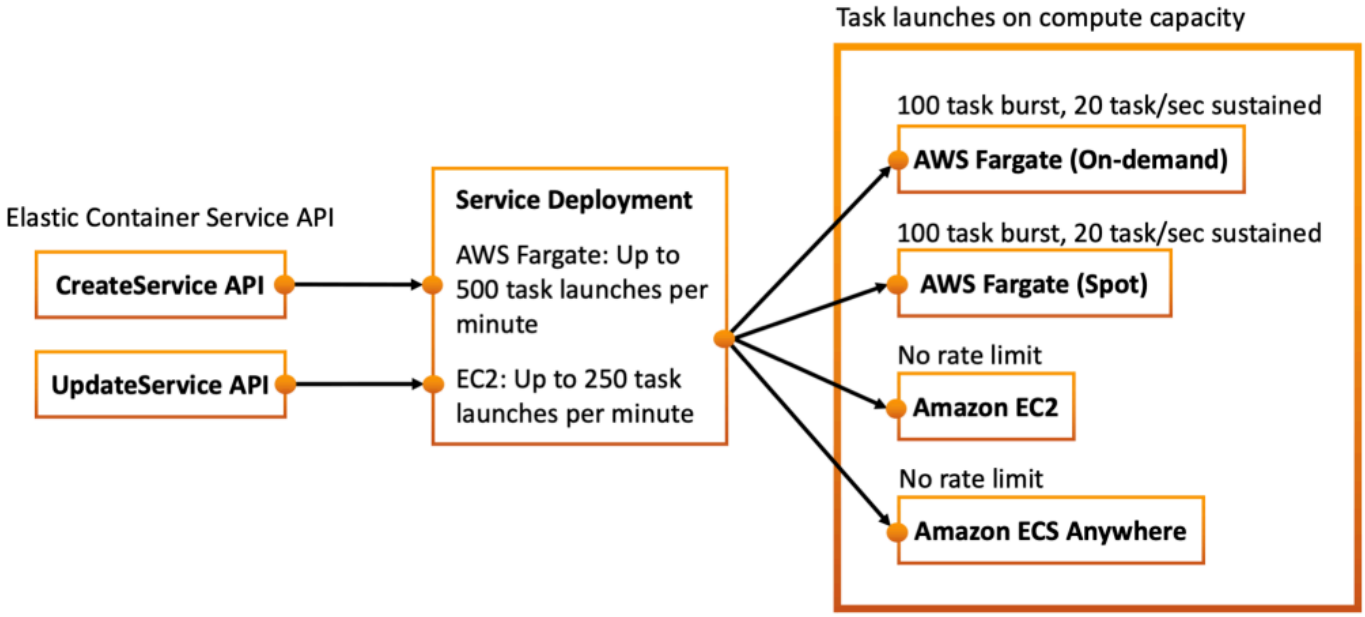
반면 Lambda는 수명이 짧은 이벤트 기반 작업에 최적화되어 있습니다. Lambda 함수는 최대 실행 시간이 15분으로 제한되는 상태 비저장 환경에서 실행됩니다. 따라서 Lambda는 작업이 간단하고 장기 실행 프로세스가 필요하지 않은 파일 처리, 실시간 데이터 스트리밍 및 HTTP 요청 처리와 같은 시나리오에 적합합니다.

Lambda에서는 런타임 환경이 더 추상화되고 기본 인프라에 대한 제어가 줄어듭니다. Lambda의 상태 비저장 속성은 각 함수 호출이 독립적이며 호출 사이에 지속되어야 하는 모든 상태 또는 데이터는 외부(예: 데이터베이스 또는 스토리지 서비스)에서 관리되어야 함을 의미합니다.

Scaling

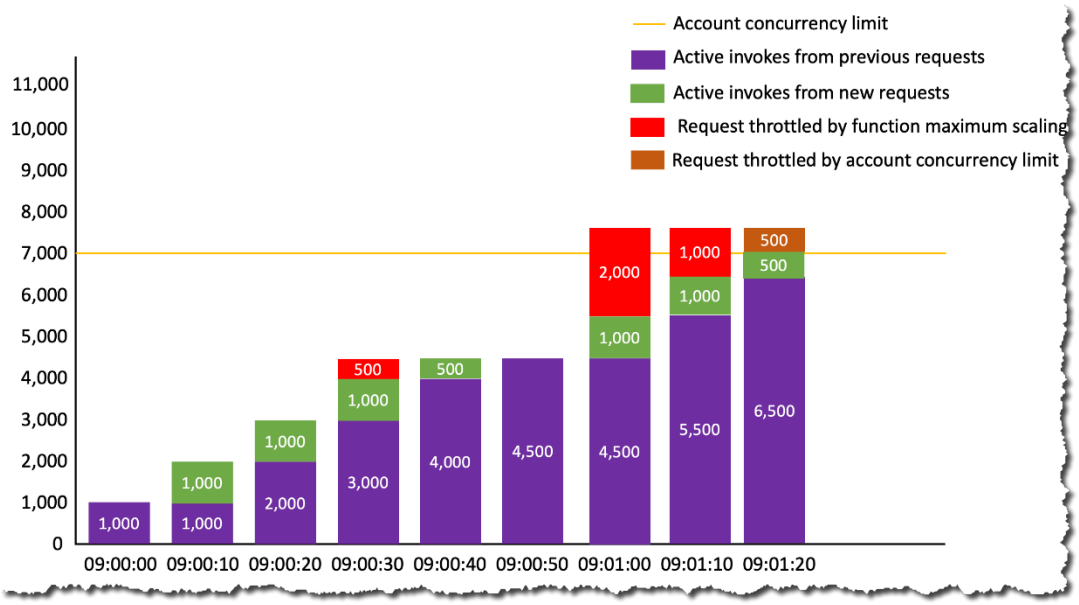
Fargate는 컨테이너 오케스트레이션 서비스(Amazon ECS)에 정의된 원하는 상태에 따라 실행 중인 컨테이너 수를 조정하여 규모를 조정합니다. 이 조정은 Amazon EC2 Auto Scaling을 통해 수동 또는 자동으로 수행할 수 있습니다. 이 블로그 게시물은 방법에 대한 추가 세부 정보를 제공합니다.

Fargate에서 각 컨테이너는 격리된 환경에서 실행되며, 규모 조정에는 추가 컨테이너를 시작하거나 부하에 따라 컨테이너를 중지하는 작업이 포함됩니다. Amazon ECS 서비스 스케줄러는 웹 및 기타 장기 실행 서비스에 대해 서비스당 1분 이내에 최대 500개의 작업을 시작할 수 있습니다.



Lambda의 경우 동시성은 AWS Lambda 함수가 동시에 처리하는 진행 중인 요청 수입니다. 이는 사용 가능한 컴퓨팅 및 네트워크 리소스가 있는 한 각 Fargate 태스크가 동시 요청을 처리할 수 있는 Fargate의 동시성과 다릅니다. 각 동시 요청마다 Lambda는 실행 환경의 개별 인스턴스를 프로비저닝합니다. 함수가 요청을 더 많이 수신하면 Lambda는 사용자가 계정의 동시성 한도에 도달할 때까지 실행 환경 수를 자동으로 조정합니다. 기본적으로 Lambda는의 모든 함수에서 1,000개의 동시 실행이라는 총 동시성 제한을 계정에 제공하며 AWS 리전, 필요한 경우 할당량 증가를 요청할 수 있습니다.

리전의 각 Lambda 함수에 대해 동시성 조정 속도는 최대 계정 동시성까지 10초마다 1,000개의 실행 인스턴스입니다. 이 블로그에 설명된 대로 10초 기간의 요청 수가 1,000개를 초과하면 추가 요청이 제한됩니다. 다음 그래프는 Lambda 조정이 계정 동시성을 7000으로 가정하여 작동하는 방식을 보여줍니다.



Cold start and cold-start mitigation

Lambda는 콜드 스타트를 경험할 수 있으며, 이는 일정 시간 동안 유휴 상태인 후 함수가 호출될 때 발생합니다. 콜드 스타트 중에 Lambda 서비스는 런타임, 종속성 및 함수 코드 로드를 포함하여 새 실행 환경을 초기화해야 합니다. 이 프로세스는 특히 초기화 시간이 더 긴 언어(예: Java 또는 C#)의 경우 지연 시간을 초래할 수 있습니다. 콜드 스타트는 애플리케이션, 특히 지연 시간이 짧은 응답이 필요한 애플리케이션의 성능에 영향을 미칠 수 있습니다.

Lambda에서 콜드 스타트를 완화하기 위해 몇 가지 전략을 사용할 수 있습니다.

- 함수 크기 최소화: 함수 패키지 및 해당 종속성의 크기를 줄이면 초기화에 필요한 시간이 줄어들 수 있습니다.
- 메모리 할당 증가: 메모리 할당이 높을수록 CPU 용량이 증가하여 초기화 시간이 줄어들 수 있습니다.
- 함수를 웜으로 유지: Lambda 함수를 주기적으로 호출(예: CloudWatch Events 사용)하면 함수를 활성 상태로 유지하고 콜드 스타트 가능성을 줄일 수 있습니다.
- Lambda SnapStart: Lambda SnapStart for Java 함수를 사용하여 시작 시간을 줄입니다.
- 프로비저닝된 동시성:이 기능은 지정된 수의 함수 인스턴스를 웜 상태로 유지하고 요청을 처리할 준비를 하여 콜드 스타트 지연 시간을 줄입니다. 그러나 요청을 적극적으로 처리하지 않더라도 프로비저닝된 인스턴스에 대한 비용을 지불하면 비용이 증가합니다.

Fargate는 일반적으로 Lambda와 동일한 방식으로 콜드 스타트의 영향을 받지 않습니다. Fargate 작업을 시작하는 데 걸리는 시간은 이미지 레지스트리에서 작업에 정의된 컨테이너 이미지를 가져

오는 데 걸리는 시간과 직접적인 상관 관계가 있습니다. 또한 Fargate는 Seekable OCI(SOCI)로 인덱싱된 컨테이너 이미지의 지연 로딩을 지원합니다. SOCI를 사용하여 컨테이너 이미지를 지연 로드하면 Fargate에서 Amazon ECS 작업을 시작하는 데 걸리는 시간이 줄어듭니다. Fargate는 필요한 기간 동안 활성 상태로 유지되는 컨테이너를 실행합니다. 즉, 항상 요청을 처리할 준비가 되어 있습니다. 그러나 조정 이벤트에 대한 응답으로 새 컨테이너를 시작해야 하는 경우 컨테이너가 초기화되는 동안 약간의 지연이 발생할 수 있지만 일반적으로 Lambda 콜드 스타트에 비해 덜 중요합니다.

Memory and CPU options

Fargate는 컨테이너화된 애플리케이션의 메모리 및 CPU 리소스를 세밀하게 제어합니다. Fargate에서 작업을 시작할 때 애플리케이션의 요구 사항에 따라 정확한 CPU 및 메모리 요구 사항을 지정할 수 있습니다. CPU 및 메모리 할당은 독립적이므로 워크로드에 가장 적합한 조합을 선택할 수 있습니다. 예를 들어 구성에 따라 0.25 vCPUs~16 vCPUs 범위의 CPU 값과 컨테이너당 0.5GB~120GB의 메모리를 선택할 수 있습니다.

이러한 유연성은 메모리 집약적인 데이터베이스 또는 CPU 바운드 계산 작업과 같은 특정 성능 특성이 필요한 애플리케이션을 실행하는 데 적합합니다. Fargate를 사용하면 리소스 할당을 최적화하여 비용과 성능의 균형을 효과적으로 맞출 수 있습니다.

Lambda에서는 메모리와 CPU가 연결되며 CPU는 선택한 메모리 양에 비례하여 자동으로 할당됩니다. 1MB 단위로 128MB에서 10GB 사이의 메모리 할당을 선택할 수 있습니다. CPU는 최대 6개의 vCPU까지 메모리에 따라 확장됩니다. 즉, 메모리 설정이 높을수록 CPU 성능이 향상되지만 CPU 할당 자체를 직접 제어할 수는 없습니다.

이 모델은 간소화를 위해 설계되었으므로 개발자는 CPU 구성을 관리할 필요 없이 메모리 설정을 빠르게 조정할 수 있습니다. 그러나 CPU와 메모리 리소스 간의 특정 균형이 필요한 워크로드의 경우 유연성이 떨어질 수 있습니다. Lambda의 모델은 메모리 요구 사항에 따라 간단한 조정을 원하는 작업에 적합하지만 복잡하거나 매우 구체적인 리소스 요구 사항이 있는 애플리케이션에는 적합하지 않을 수 있습니다.

Networking

Fargate에서 작업을 배포하면 Amazon VPC(Amazon Virtual Private Cloud)에서 실행되므로 네트워킹 환경을 완벽하게 제어할 수 있습니다. 여기에는 보안 그룹, 네트워크 액세스 제어 목록(ACLs) 및 라우팅 테이블 구성이 포함됩니다. 각 Fargate 태스크는 전용 프라이빗 IP 주소를 사용하여 자체 네트워크 인터페이스를 가져오며, 필요한 경우 퍼블릭 IP 주소를 할당할 수 있습니다.

Fargate는 로드 밸런싱(AWS Elastic Load Balancing 사용), VPC 피어링, VPC AWS 서비스 내 다른에 대한 직접 액세스와 같은 고급 네트워킹 기능을 지원합니다. 인터넷을 통과 AWS 서비스하지 않고도 지원되는 AWS PrivateLink 에 대한 안전한 프라이빗 연결을 위해를 사용할 수도 있습니다.

기본적으로 Lambda 함수는 네트워크 인터페이스 또는 IP 주소를 직접 제어하지 않고 관리형 네트워크 환경에서 실행됩니다. 그러나 Lambda는 AWS Hyperplane을 사용하여 고객 관리형 VPC에 연결할 수 있으므로 VPC 내의 리소스에 대한 액세스를 제어할 수 있습니다.

Lambda 함수가 고객 관리형 VPC에 연결되면 VPC의 보안 그룹 및 서브넷 구성을 상속하여 동일한 VPC 내의 다른 AWS 서비스 (예: RDS 데이터베이스)와 안전하게 상호 작용할 수 있습니다.

Lambda 서비스는 네트워크 함수 가상화 플랫폼을 사용하여 Lambda VPC에서 고객 VPC로 NAT 기능을 제공합니다. 그러면 Lambda 함수가 생성되거나 업데이트되는 시점에 필요한 탄력적 네트워크 인터페이스(ENI)가 구성됩니다. 또한 계정의 ENI를 여러 실행 환경에서 공유할 수 있으므로 함수 규모를 조정할 때 Lambda에서 제한된 네트워크 리소스를 더 효율적으로 사용할 수 있습니다.

ENI는 소진 가능한 리소스이며 리전당 250개의 ENI와 같은 소프트 제한이 있으므로 VPC 액세스를 위해 Lambda 함수를 구성하는 경우 탄력적 네트워크 인터페이스 사용을 모니터링해야 합니다. 동일한 AZ 및 동일한 보안 그룹의 Lambda 함수는 ENIs 공유할 수 있습니다. 일반적으로 Lambda에서 동시성 제한을 늘리는 경우 탄력적 네트워크 인터페이스 증가가 필요한지를 평가해야 합니다. 제한에 도달하면 VPC 지원 Lambda 함수의 간접 호출이 스로틀링됩니다.

Pricing model

Fargate 요금은 컨테이너에 할당된 리소스, 특히 각 작업에 대해 선택한 vCPU 및 메모리를 기반으로 합니다. 컨테이너가 사용하는 CPU 및 메모리에 대해 최소 1분 동안 초당 요금이 청구됩니다. 비용은 애플리케이션이 요청을 적극적으로 처리하는지 여부에 관계없이 애플리케이션이 소비하는 리소스와 직접 연결됩니다. 즉, 프로비저닝 비용을 지불합니다. Fargate는 특정 리소스 구성이 필요한 예측 가능한 워크로드에 적합하며 할당된 리소스를 조정하여 비용을 최적화할 수 있습니다. 또한 데이터 전송, 스토리지 및 네트워킹(예: VPC, Elastic Load Balancing)과 같은 관련 서비스에 대한 추가 요금이 발생할 수 있습니다.

Lambda는 이벤트 기반 및 pay-per-execution 방식인 요금 구조가 다릅니다. 함수가 수신하는 요청 수와 밀리초 단위로 측정되는 각 실행 기간을 기준으로 요금이 청구됩니다. 또한 Lambda는 사용된 메모리와 실행 시간에 따라 비용 조정을 통해 함수에 할당하는 메모리의 양을 고려합니다. 요금 모델에는 1백만 개의 무료 요청과 매월 400,000GB-초의 컴퓨팅 시간을 제공하는 프리 티어가 포함되어 있으므로 Lambda는 소량, 산발적 워크로드에 특히 비용 효율적입니다.

Lambda 요금 모델은 유휴 용량을 프로비저닝하거나 비용을 지불할 필요 없이 실제 함수 호출 및 실행 시간만 지불하므로 예측할 수 없거나 트래픽 패턴이 급증하는 애플리케이션에 적합합니다.

사용

이제 AWS Fargate 와 중에서 선택하는 기준에 대해 읽었으므로 필요에 맞는 서비스를 선택하고 다음 정보를 사용하여 각 서비스 사용을 시작할 AWS Lambda수 있습니다.

AWS Fargate

- Fargate 시작 유형에 대한 Amazon ECS Linux 작업을 생성하는 방법을 알아봅니다.

  Linux 작업에 Fargate 시작 유형을 AWS Fargate 사용하여에서 Amazon ECS를 시작합니다.

  가이드 살펴보기
- Fargate 시작 유형에 대한 Amazon ECS Windows 작업을 생성하는 방법을 알아봅니다.

  Windows 작업에 Fargate 시작 유형을 AWS Fargate 사용하여에서 Amazon ECS를 시작합니다.

  가이드 살펴보기
- Fargate 및 Amazon EKS 시작하기

  이 가이드에서는 Amazon EKS 클러스터를 사용하여에서 포드 실행 AWS Fargate 을 시작하는 방법을 설명합니다.

  가이드 살펴보기
- AWS Fargate 요금

  이 가이드를 사용하여 vCPU, 메모리, 스토리지 및 운영 체제 구성이 AWS Fargate 요금에 미치는 영향을 이해합니다.

  가이드 살펴보기
- AWS Fargate 자주 묻는 질문

  AWS Fargate 기능에 대한 일반적인 질문과 구현 모범 사례에 대한 답변을 얻습니다.

  가이드 살펴보기

AWS Lambda

- 서버리스 파일 처리 앱 생성

Amazon SNS 설정 및 사용에 대한 step-by-step 연습입니다. 주제 생성, 주제에 대한 엔드포인트 구독, 메시지 게시, 액세스 권한 구성과 같은 주제를 다룹니다.

가이드 살펴보기

- 서버리스 개발자 가이드

  이 가이드는 서버리스 애플리케이션 개발과 다양한 AWS 서비스 가에 어떻게 통합되어 클라우드 애플리케이션의 핵심을 형성하는 애플리케이션 패턴을 생성하는지에 대한 개념적 이해를 높이는 데 도움이 됩니다.

  가이드 살펴보기

- 서버리스 랜드

  이 사이트는 AWS Serverless에 대한 최신 정보, 블로그, 비디오, 코드 및 학습 리소스를 통합합니다. 저비용 완전관리형 서버리스 아키텍처에서 자동으로 확장되는 앱을 사용하고 구축하는 방법을 알아봅니다.

  사이트 탐색

- AWS Lambda 요금

  이 가이드를 사용하여 비용을 추정하고 함수 사용량 및 구성을 기반으로 비용을 최적화합니다. 여기에는 단일 견적으로 AWS Lambda 및 아키텍처 비용을 계산하는 요금 계산기가 포함되어 있습니다.

  가이드 살펴보기

- AWS Lambda 자주 묻는 질문

  AWS Lambda 기능에 대한 일반적인 질문과 구현 모범 사례에 대한 답변을 얻습니다.

  가이드 살펴보기

AWS Fargate 또는에 대한 문서 기록 AWS Lambda

다음 표에서는이 결정 가이드의 중요한 변경 사항에 대해 설명합니다. 이 가이드 업데이트에 대한 알림을 받으려면 RSS 피드를 구독하면 됩니다.

변경 사항	설명	날짜
최초 릴리스	결정 가이드의 최초 릴리스입니다.	2024년 11월 15일

기계 번역으로 제공되는 번역입니다. 제공된 번역과 원본 영어의 내용이 상충하는 경우에는 영어 버전이 우선합니다.