서버리스 아키텍처로 엔터프라이즈 경제 최적화

2017 년 *9* 월

고지 사항

이 문서는 정보 제공 목적으로만 제공됩니다. 본 문서의 발행일 당시 AWS 의 현재 제품 및 실행방법을 설명하며, 예고 없이 변경될 수 있습니다. 고객은 본 문서에 포함된 정보나 AWS 제품 또는 서비스의 사용을 독립적으로 평가할 책임이 있으며, 각 정보 및 제품은 명시적이든 묵시적이든 어떠한 종류의 보증 없이 "있는 그대로" 제공됩니다. 본 문서는 AWS 및 그 계열사, 공급업체 또는 라이선스 제공자로부터 어떠한 보증, 표현, 계약 약속, 조건 또는 보증을 구성하지 않습니다. 고객에 대한 AWS 의 책임 및 의무는 AWS 계약에 준거합니다. 본 문서는 AWS 와 고객 간의 어떠한 계약도 구성하지 않으며 이를 변경하지도 않습니다.

목차

요약

이 백서는 CIO(최고 정보 책임자), CTO(최고 기술 책임자) 및 수석 아키텍트가 서버리스 아키텍처와 서버리스 아키텍처가 출시 시간, 팀 민첩성 및 IT 경제에 미치는 영향에 대해 파악하는 데 도움을 주기 위해 작성되었습니다. 설계 수준에서 유휴 서버 및 제대로 활용되지 않는 서버가 사라지고 클라우드 기반 소프트웨어 설계가 급격히 간소화됨에 따라 서버리스 접근 방식으로 인해 IT 환경이 급속도로 변화하고 있습니다.

이 백서에서는 서버리스 접근 방식의 기본 사항과 AWS 서버리스 포트폴리오를 다루며, 기존 회사가 서버리스 접근 방식을 채택하여 이미 막대한 민첩성과 경제 이점을 얻은 방법을 설명하는 수많은 사례 연구도 함께 살펴봅니다. 이 백서에서는 모든 규모의 조직이 서버리스 아키텍처를 사용하여 대응적 이벤트 기반 시스템을 설계하고 기존 비용의 일부만으로 클라우드 네이티브 마이크로서비스를 빠르게 제공할 수 있는 방법을 설명합니다.

서론

많은 회사가 사용량에 따라 지불하는 결제 방식으로 비용을 절감하고 온디맨드 IT 리소스 사용을 통해 민첩성을 향상하는 등 퍼블릭 클라우드에서 애플리케이션을 실행하여 얻는 이점을 이미 누리고 있습니다. 애플리케이션 유형 및 산업 전반에 걸친 여러 연구를 통해 기존 애플리케이션 아키텍처를 클라우드에 마이그레이션하면 총 소유 비용(TCO)이 절감되고 출시 시간이 단축된다는 사실을 확인했습니다.[1]

온프레미스 및 프라이빗 클라우드 솔루션에 비해 퍼블릭 클라우드는 서버 집합과 서버에서 실행하는 애플리케이션을 구축, 배포 및 관리하기가 매우 간편합니다. 하지만 오늘날 회사들은 기존 서버 또는 VM 기반 아키텍처를 넘어서는 추가 옵션을 확보하여 퍼블릭 클라우드를 적극적으로 활용하고 있습니다. 클라우드를 통해 회사에서 자체 하드웨어를 구매하고 유지 관리할 필요가 없어졌지만 모든 서버 기반 아키텍처는 계속해서 확장성과 안정성을 구축해야 합니다. 또한 회사들은 애플리케이션이 진화함에 따라 해당 서버 집합을 패치하고 배포해야 하는 어려움에 직면해 있습니다. 그뿐만 아니라 이러한 서버 집합을 최고 로드가 처리되도록 확장한 후 비용을 절감할 수 있는 시기나 위치에서는 규모를 축소해야 합니다. 이러한 모든 작업은 최종 사용자 환경과 내부 시스템 무결성을 보호하면서 이루어져야 합니다. 유휴 서버 및 제대로 활용되지 않는 서버는 비용이 많이 들고 비경제적임이 입증되었습니다. 분석가는 실제로 85% 정도의 서버가 서버 용량을 제대로 활용하지 않는다고 추정합니다.[2]

AWS Lambda 와 같은 서버리스 컴퓨팅 서비스는 본질적으로 비용을 낮추고 출시 시간을 단축하는 애플리케이션 설계에 대한 다양한 접근 방식을 회사에 제공하여 이러한 과제를 해결하도록 설계되었습니다. *AWS Lambda* 는 모든 기술 스택 수준에서 서버를 다루는 복잡성을 해소하고 유휴 컴퓨팅 파워에 대해 더 이상 비용이 청구되지 않는 요청에 따른 비용 청구 방식을 도입합니다. 또한 Lambda 함수를 사용하면 조직이 손쉽게 마이크로서비스 아키텍처를 실행할 수 있습니다. 인프라를 없애고 Lambda 모델로 전환하면 다음과 같은 일석이조의 경제적 이점을 누릴 수 있습니다.

- 유휴 서버와 같은 문제를 간단하게 해소하여 경제적으로 절약할 수 있습니다. AWS Lambda 와 같은 서버리스 컴퓨팅 서비스는 밀리초 단위의 세부 청구 기준으로 유용한 작업이 수행될 때만 요금이 누적되므로 “무자비”하게 비용이 책정되는 일이 없습니다.

- 집합 관리(서버 보안 패치, 배포 및 모니터링 포함)가 더 이상 필요하지 않습니다. 이는 연중무휴 서버 집합을 가동하도록 지원하는 데 필요한 관련 도구, 프로세스 및 대기 중인 교대를 유지 관리할 필요가 없어졌음을 의미합니다. Lambda 를 사용하여 마이크로서비스를 구축하면 조직의 민첩성이 향상됩니다. 회사는 서버 관리에 대한 부담 없이 부족한 IT 리소스를 회사의 중요한 작업인 비즈니스에 투입할 수 있습니다.

이미 서버리스 접근 방식을 도입한 회사는 인프라 비용이 대폭 절감되고, 팀의 민첩성과 집중성이 높아지고, 출시 시간이 단축되어 경쟁업체보다 우위를 선점할 수 있는 핵심 이점을 누리고 있습니다.

서버리스 애플리케이션 이해

앞서 언급한 서버리스 접근 방식의 이점은 매우 매력적이지만 실제로 구현하려면 어떤 점을 고려해야 할까요? 기존 서버 기반 방식과 서버리스 앱의 차이점은 무엇일까요?

서버리스 앱은 개발자가 실제 비즈니스 로직을 작성하는 본래의 핵심 역량에 집중할 수 있도록 설계된 도구입니다. 웹 서버와 같은 다양한 앱의 표준 구성 요소 및 안정성과 확장성을 처리하는 소프트웨어와 같은 모든 획일적인 업무 부담을 개발자가 처리하지 않도록 대폭 축소합니다. 이러한 사항을 제거하면 모바일 사용자가 메시지를 보내고, 이미지가 클라우드에 업로드되고, 레코드가 스트림에 도달하는 등 필요할 때만 비즈니스 로직이 트리거되는 깔끔한 기능적 방식만 남게 됩니다. 애플리케이션 설계에 대한 비동기식 이벤트 기반 접근 방식(필수 아님)은 서버리스 애플리케이션에서 매우 일반적입니다. 수행된 작업이 있을 경우에만 실행되는(비용도 발생함) 코드 개념과 완벽하게 맞아떨어지기 때문입니다.

서버리스 애플리케이션은 퍼블릭 클라우드에서 이벤트 또는 클라이언트 호출을 처리한 후 코드를 호출하고 실행하는 AWS Lambda 와 같은 서비스를 통해 실행됩니다. 이 모델은 기존 서버 기반 애플리케이션 설계에 비해 다음과 같은 다양한 이점을 제공합니다.

- 서버를 프로비저닝, 배포, 업데이트, 모니터링 또는 관리할 필요가 없습니다. 모든 실제 하드웨어 및 서버 소프트웨어는 클라우드 공급업체가 처리합니다.
- 애플리케이션이 실제 사용을 기준으로 트리거되며 자동으로 확장됩니다. 이는 최고 로드로 확장하는 데 수신자 집합과 명시적 용량 관리가 필요한 기존 애플리케이션과는 본질적으로 다릅니다.

- 또한 확장성, 가용성 및 내결함성이 기본으로 제공됩니다. 이러한 기능의 이점을 활용하는 데 코딩, 구성 또는 관리가 필요하지 않습니다.
- 유휴 용량에 대한 비용이 청구되지 않습니다. 용량을 사전 프로비저닝하거나 오버 프로비저닝할 필요가 없습니다(실제로 기능 자체가 없음). 청구는 요청당 지불 방식으로 진행되며, 코드가 실행된 시간을 기준으로 합니다.

서버리스 애플리케이션 사용 사례

서버리스 애플리케이션 모델은 포괄적이며, 스타트업의 웹 앱부터 Fortune 100 대 기업의 주식 거래 분석 플랫폼에 이르기까지 거의 모든 종류의 애플리케이션에 적용됩니다. 다음은 몇 가지 예입니다.

- **웹 앱 및 웹사이트** – 서버를 제거하면 트래픽이 없을 때 비용이 거의 들지 않으며 동시에 예측하지 못한 경우에도 최고 로드를 처리할 수 있도록 동시에 확장되는 웹 앱을 제작할 수 있습니다.
- **모바일 백엔드** – 서버리스 모바일 백엔드는 개발자가 분산 시스템 설계의 전문가가 되지 않아도 안전하고 가용성이 뛰어나며 완벽하게 확장되는 백엔드를 손쉽게 제작하도록 클라이언트 개발에 집중할 수 있는 방법을 제시합니다.
- **미디어 및 로그 처리** – 서버리스 접근 방식은 복잡한 멀티스레드 시스템을 구축하거나 수동으로 컴퓨팅 집합을 확장하지 않아도 대량의 컴퓨팅 작업로드를 간단하게 처리할 수 있어 자연스럽게 병렬화됩니다.
- **IT 자동화** – 필요한 경우 사용자 지정 기능을 제공하기 위해 서버리스 기능을 경보 및 모니터에 연결할 수 있습니다. Cron 작업 및 다른 IT 인프라 요구 사항은 특히 이러한 작업 및 요구 사항이 드물거나 사실상 변수가 많은 경우, 사용할 서버를 보유하고 유지 관리할 필요가 없어져 구현이 매우 간단해집니다.
- **IoT 백엔드** – 네이티브 라이브러리를 포함한 모든 코드를 가져올 수 있는 기능으로 다비이스별 알고리즘을 구현할 수 있는 클라우드 기반 시스템을 생성하는 프로세스를 간소화합니다.
- **챗봇(음성 인식 지원 포함) 및 기타 웹훅 기반 시스템** – 서버리스 접근 방식은 챗봇과 같은 웹훅 기반 시스템에 매우 적합합니다. 필요한 경우에만(예: 사용자가 챗봇에서 정보를 요청하는 경우) 작업을 수행(예: 코드 실행)할 수 있는 기능 덕분에 이러한 아키텍처에 대한 접근 방식이 단순해지고, 대개 소요되는 비용이 절감됩니다. 예를 들어 Amazon Echo Alexa 기술 대부분은 AWS Lambda 를 사용해 구현되었습니다.

- **클릭스트림 및 거의 실시간으로 처리되는 다른 스트리밍 데이터 프로세스** – 서버리스 솔루션은 각 애플리케이션에 대해 확장 가능한 컴퓨팅 시스템을 복잡하게 구축하지 않고도 처리량 요구 사항에 부합하도록 데이터 흐름을 통해 규모를 확장 및 축소할 수 있는 유연성을 제공합니다. AWS Lambda 를 Amazon Kinesis 와 같은 기술과 함께 사용할 경우 클릭스트림 분석, NoSQL 데이터 트리거, 주식 거래 정보 등에 대한 기록을 고속으로 처리할 수 있습니다.

위에서 논의한 매우 최적화된 사용 사례 외에도 회사는 다음과 같은 분야에도 서버리스 접근 방식을 적용할 수 있습니다.

- 빅 데이터(예: 맵-리듀스 문제), 동영상 고속 트랜스코딩, 주식 거래 분석, 대출 신청에 대한 컴퓨팅 집약적 몬테카를로 시뮬레이션. 개발자는 특히 이벤트를 통해 트리거될 경우 3서버리스 접근 방식을 통해 병렬화하는 일이 훨씬 쉽기 때문에 인프라를 관리할 필요 없이 다양한 빅 데이터 문제에 서버리스 기술을 적용하는 경우가 점점 증가하고 있음을 확인했습니다.
- 콘텐츠 전송 네트워크를 통한 웹 애플리케이션 및 자산의 지연 시간이 짧은 사용자 지정 처리. 개발자는 서버리스 이벤트 처리를 인터넷의 엣지로 옮겨 짧은 지연 시간과 회수 및 콘텐츠 가져오기를 쉽게 사용자 지정할 수 있는 기능과 같은 이점을 활용할 수 있습니다. 이를 통해 클라이언트 위치를 기반으로 지연 시간이 최적화된 새로운 스펙트럼의 사용 사례를 구현할 수 있습니다.
- AWS Lambda 함수와 같은 서버리스 기능을 상업용, 주거용 및 핸드헬드 사물 인터넷(IoT) 디바이스에서 실행할 수 있는 연결된 디바이스. Lambda 함수와 같은 서버리스 솔루션은 기본 물리적 하드웨어(및 가상 하드웨어)에서 자연스러운 추상화를 제공하고 프로그래밍 모델을 방해하지 않고 데이터 센터에서 엣지로, 하드웨어 아키텍처에서 다른 하드웨어 아키텍처로 보다 손쉽게 전환되도록 합니다.
- AWS Snowball Edge 와 같은 온프레미스 어플라이언스에서 사용자 지정 로직 및 데이터 처리. 서버리스 애플리케이션은 실행 환경의 세부 정보에서 비즈니스 로직을 분리하기 때문에 어플라이언스 등 매우 다양한 환경에서 손쉽게 역할을 수행할 수 있습니다.

대개 서버리스 애플리케이션은 애플리케이션이 개별 작업을 수행하는 독립 구성 요소로 분리되는 마이크로서비스 아키텍처를 사용하여 구축됩니다. API, 메시지 대기열, 데이터베이스 및 기타 구성 요소를 포함한 개별 Lambda 함수로 생성된 이러한 구성 요소는 개별적으로 배포되고, 테스트되고, 확장될 수 있습니다. 실제로 서버리스 애플리케이션은 함수 기반 모델을 갖추고 있어 마이크로서비스에 매우

적합합니다. 모놀리식 설계 및 아키텍처를 방지하면 필요한 경우 개발자가 데이터베이스 티어와 같은 개별 구성 요소를 교체하거나 업그레이드하고 점점 더 많이 배포할 수 있기 때문에 조직은 보다 높은 민첩성을 확보할 수 있습니다.

서버리스 앱으로 전환하는 데 애플리케이션의 비즈니스 로직을 분리하기만 하면 되는 경우가 많습니다. AWS Lambda 와 같은 서비스는 많이 사용되는 프로그래밍 언어를 지원하고 사용자 지정 라이브러리 사용을 활성화합니다. 오랜 시간 지속되는 작업은 합리적인 기간 내에 실행되는 개별 함수로 구성된 워크플로라고 명시되며, 이를 통해 해당 시스템이 필요 시 개별 컴퓨팅 장치를 재시작하거나 병렬화할 수 있습니다.

서버리스는 모든 분야에 적합한가요?

서버리스 플랫폼에서 거의 모든 최신 애플리케이션을 성공적으로 실행되도록 수정할 수 있으며, 대부분의 경우 보다 경제적이고 확장 가능한 방식으로 수정합니다. 하지만 서버리스가 최선의 선택이 아닌 경우도 몇 가지 있습니다.

- 어떤 경우에도 애플리케이션을 명시적으로 변경하지 않는 것이 목표인 경우.
- 환경에 대한 세분화된 제어가 필요한 경우(예: 코드가 제대로 실행되도록 특정 운영 체제 패치를 지정하거나 낮은 수준의 네트워킹 작업에 액세스).
- 온프레미스 애플리케이션이 퍼블릭 클라우드에 마이그레이션되지 않은 경우.

클라우드 공급업체의 서버리스 플랫폼 평가

서버리스 애플리케이션을 설계할 때 회사 및 조직은 앱의 코드를 실행하는 서버리스 컴퓨팅 기능 이외에 더 많은 요소를 고려해야 합니다. 완벽한 서버리스 앱에는 광범위한 서비스와 도구 및 스토리지, 메시지, 진단 등을 포괄하는 기능이 있어야 합니다. 클라우드 공급업체의 불완전하거나 조각난 서버리스 포트폴리오는 서버리스 개발자에게 문제가 될 수 있으며, 지속적인 추상화 단계에서 코딩할 수 없을 경우 서버 기반 아키텍처로 돌아갈 수도 있습니다.

서버리스 플랫폼은 컴퓨팅 및 스토리지 구성 요소와 같은 서버리스 앱은 물론, 서버리스 앱을 작성, 구축, 배포 및 진단하는 데 필요한 도구가 포함된 서비스 세트로 구성되어 있습니다. 프로덕션 환경에서 서버리스 애플리케이션을 운영하려면 소규모 스타트업부터 세계적인 글로벌 기업의 수요를 처리할 수 있는 안정적이고 유연하며 신뢰할 수 있는 플랫폼이 필요합니다. 플랫폼은 모든

애플리케이션 요소를 확장하고 엔드 투 엔드 안정성을 제공해야 합니다. 기존 앱과 마찬가지로 개발자가 서버리스 솔루션을 성공적으로 생성 및 제공하도록 하는 것이 다차원적인 과제입니다. 다양한 산업에 걸친 대규모 기업의 요구 사항을 충족하려면 서버리스 플랫폼은 다음 기능을 제공해야 합니다.

그림 1: 서버리스 플랫폼의 기능

- 안정적인 고성능 확장 가능 클라우드 로직 계층.
- 반응형 자사 이벤트 및 데이터 소스, 간단한 타사 시스템과의 연결성.
- 개발자가 손쉽게 시작하고 기존 솔루션에 새로운 패턴을 빠르고 안전하게 추가할 수 있는 통합 라이브러리.
- 개발자가 다양한 도메인에서 광범위한 타사 시스템 및 사용 사례에 대한 솔루션을 발견하고 적용할 수 있는 강력한 개발자 에코시스템.
- 목적에 부합하는 애플리케이션 모델링 프레임워크 모음.
- 상태와 워크플로 관리 기능을 갖춘 오케스트레이션.
- 보장 프로그램 인증을 포함한 글로벌 규모 및 광범위한 도달 범위.
- 용량을 특정 규모로 프로비저닝할 필요 없는 기본 안정성 및 규모에 따른 성능.
- 자사 및 타사 리소스 및 서비스 모두에 유연한 액세스 제어와 기본적인 보안.

서버리스 플랫폼의 핵심은 비즈니스 로직을 나타내는 함수 실행을 담당하는 클라우드 로직 계층입니다. 이러한 함수는 이벤트에 대한 응답으로 실행되는 경우가 많기 때문에 자사 및 타사 이벤트 소스 모두와의 간단한 통합은 솔루션을 간단하게 명시하고 다양한 워크로드에 대한 응답으로 자동으로 규모를 확장하는 데 필수 요소입니다. 예를 들어 객체 스토어에서 객체가 생성되거나 서버리스 NoSQL 데이터베이스에서 발생한 각 업데이트에 대해 객체가 생성될 때마다 서버리스 함수를 실행해야 할 수 있습니다. 서버리스 아키텍처는 해당 시스템을 통합하는 데 주로 필요한 확장 및 관리 코드를 모두 제거합니다. 운영 부담은 클라우드 공급업체가 지게 됩니다.

서버리스 플랫폼에서 성공적으로 개발이 이루어지려면 자사 및 타사 서비스와 관련된 일반적인 사용 사례에 맞춰 바로 사용할 수 있는 템플릿을 찾는 작업을 포함하여 회사 측에서 손쉽게 작업을 시작할 수 있어야 합니다. 이러한 통합 라이브러리는 특히 개발자가 서버 기반에서 서버리스 아키텍처로 마이그레이션하는 기간 동안 레코드 스트림을 처리하거나 웹훅을 구현하는 등 성공적인 패턴을 전달하는 데 필수 요소입니다. 밀접하게 관련된 요구 사항은 핵심 플랫폼을 둘러싼 광범위하고 다양한 에코시스템입니다. 강력한 대규모 에코시스템을 통해 개발자는 커뮤니티에서 솔루션을 쉽게 찾아 사용할 수 있으며, 새로운 아이디어와 접근 방식을 쉽게 고안할 수 있습니다. 애플리케이션 수명 주기 관리 사용에서의 다양한 도구 체인을 고려할 때 모든 언어, IDE 및 엔터프라이즈 구축 기술에는 기존 접근 방식에 서버리스 앱 구축 및 배포를 통합하는 데 필요한 실행 시간, 플러그인 및 오픈 소스 솔루션이 포함되어 있음을 보장하는 데 건강한 에코시스템도 필요합니다. 또한 서버리스 앱이 Express 및 Flask 와 같은 프레임워크와 많이 사용되는 프로그래밍 언어에 대한 개발자의 지식 등 기존 투자 요소를 활용하는 것도 중요합니다. 광범위한 에코시스템은 도메인 전반에 걸쳐 중요한 가속화를 제공하고 개발자가 서버리스 아키텍처에서 더욱 쉽게 기존 코드의 용도를 변경할 수 있도록 합니다.

개발자는 오픈 사양 AWS 서버리스 애플리케이션 모델(AWS SAM)과 같은 애플리케이션 모델링 프레임워크를 사용하여 서버리스 앱을 구성하는 구성 요소를 명시하고 이러한 애플리케이션을 구축, 배포 및 모니터링하는 데 필요한 도구와 워크플로를 활성화할 수 있습니다. 서버리스 플랫폼 성공 여부의 핵심인 다른 프레임워크는 오케스트레이션 및 상태 관리입니다. 서버리스 컴퓨팅의 전반적인 상태 비저장 특성에는 오랜 시간 지속되는 워크플로를 활성화하기 위한 상호 보완적인 메커니즘이 필요합니다. 개발자는 오케스트레이션 솔루션을 통해 서버리스 애플리케이션에서 일반적인 여러 관련 애플리케이션 구성 요소를 조정하는 동시에 이러한 애플리케이션이 소규모 및 수명이 짧은 함수로 구성되도록 할 수 있습니다. 또한 오케스트레이션 서비스를 통해 오류 처리를 간소화하고 일반적으로 자체 허용되는 서버리스 기능보다 더 오랜 시간 동안 실행되는 항목을 포함한 기존 시스템 및 워크플로와 통합할 수 있습니다.

서버리스 플랫폼은 글로벌 다국적 기업 등 전 세계 고객을 지원하기 위해 전 세계에 위치한 엣지 로케이션과 데이터 센터를 포함한 글로벌 규모를 제공해야 합니다. 엣지 로케이션은 최종 사용자와 가까운 위치에 지연 시간이 짧은 서버리스 컴퓨팅을 제공하는 데 핵심 요소입니다. 애플리케이션 개발자보다 플랫폼이 서버리스 앱의 확장성과 고가용성 제공을 담당하기 때문에 내장 안정성은 매우 중요합니다. 처리되지 않은 이벤트에 대한 전송 실패 대기열 및 기본적인 재시도 등의 기능을 통해 개발자는 서버리스 접근 방식을 사용하여 엔드 투 엔드 안정성을 갖춘 견고한 시스템을 구축할 수 있습니다. 언어 실행 시간 및 고객 코드가 서버리스 앱에서 온디맨드로 인스턴스화되는 점을 고려하면 성능, 특히 짧은 지연 시간(오버헤드)도 똑같이 중요합니다.

마지막으로 플랫폼에는 가상 프라이빗 네트워크, 역할 기반 및 액세스 기반 권한, API 기반 인증 및 액세스 제어 메커니즘(타사 및 기존 시스템 포함)과의 견고한 통합에 대한 지원, 그리고 환경 변수 설정과 같은 애플리케이션 요소 암호화에 대한 지원을 포함한 광범위한 보안 및 액세스 제어가 있어야 합니다. 서버 시스템은 설계상 다음과 같은 이유로 높은 수준의 보안 및 제어를 기본 제공합니다.

- **보안 패칭을 포함한 최고의 집합 관리** – AWS Lambda 와 같은 시스템에서 요청을 실행하는 서버는 지속적으로 모니터링 및 재시동되고 보안 검사를 받습니다. 이는 패칭 및 업데이트에 대해 SLA 가 훨씬 느슨할 수 있는 여러 기업 컴퓨팅 집합과는 반대로 주요 보안 업데이트가 제공되는 몇 시간 이내에 패치될 수 있습니다.
- **제한된 서버 수명** – AWS Lambda 에서 고객 코드를 실행하는 모든 시스템은 하루에 여러 번에 걸쳐 재시동되며 공격에 대한 노출을 제한하고 계속해서 최신 운영 체제 및 보안 패치 상태를 유지합니다.
- **요청별 인증, 액세스 제어 및 감사** – 소스와 관계없이 AWS Lambda 에서 실행되는 모든 컴퓨팅 요청은 특정 리소스에 액세스하기 위해 개별적으로 인증 및 공인되며, 철저한 감사를 받습니다. AWS 데이터 센터 외에서 Amazon API Gateway 를 통해 도착하는 요청은 DoS 공격 방어를 포함한 추가 인터넷 연결 방어 시스템을 제공합니다. 서버리스 아키텍처로 마이그레이션하는 회사는 AWS CloudTrail 을 사용하여 어떤 사용자가 무슨 권한으로 어떤 시스템에 액세스하는지 상세하게 파악할 수 있으며, AWS Lambda 를 사용하여 감사 레코드를 프로그래밍 방식으로 처리할 수 있습니다.

AWS 서버리스 플랫폼

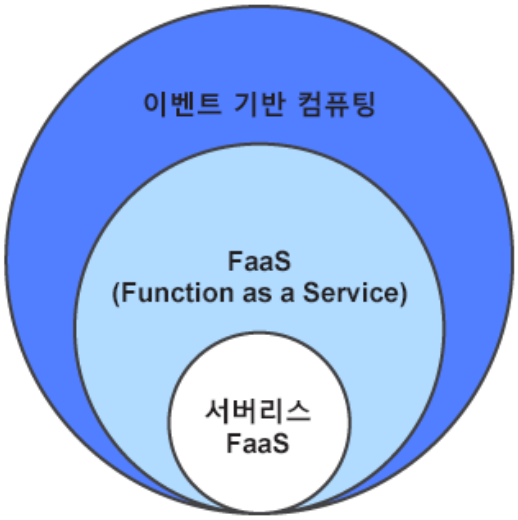
2014 년 Lambda 출시 이후 AWS 는 완벽한 서버리스 플랫폼을 제작했습니다. 여기에는 조직이 AWS 서비스 및 타사 서비스와 원활하게 통합할 수 있는 서버리스 앱을 생성할 수 있는 다양한 완전관리형 서비스 모음이 포함되어 있습니다. 그림 는 AWS 서버리스 플랫폼 내 구성 요소의 하위 집합과 그 관계를 보여 줍니다.

그림 2: AWS 서버리스 플랫폼 구성 요소

AWS 서버리스 플랫폼의 기능

AWS 는 완벽한 서버리스 플랫폼에 대한 요구 사항으로 이전 섹션에서 확인된 핵심 기능을 모두 제공합니다. AWS Lambda 에서 제공하는 클라우드 로직 계층은 함수를 기반으로 하는 프로비저닝이 필요 없는 대규모 서버리스 컴퓨팅 솔루션입니다. AWS Lambda 는 엣지 최적화 라우팅을 사용하여 지연 시간이 극도로 짧은 Lambda 함수 실행에 대한 유사한 지원을 제공하는 AWS Lambda@Edge 와 AWS Snowball 과 같은 어플라이언스를 포함한 연결된 디바이스에서 Lambda 함수가 실행할 수 있는 AWS Greengrass 에서 보완됩니다.

Lambda 함수는 여러 자사 및 타사 이벤트를 통해 손쉽게 트리거될 수 있으며, 개발자는 인프라를 설정하고 관리해야 하는 기존의 번거로운 작업 없이 대응형 이벤트 기반 시스템(그림 3 참조)을 구축할 수 있습니다. 여러 개의 동시 이벤트가 발생할 경우 Lambda 는 단순히 각 개별 트리거에 대응하여 병렬 방식으로 함수의 사본을 더 많이 실행합니다. Lambda 함수는 워크로드 규모에 정확하게 맞춰 개별 요청으로 규모를 축소합니다. 결과적으로 유휴 서버 또는 컨테이너가 발생할 가능성이 없습니다. Lambda 함수를 사용하는 아키텍처에서는 계획적으로 낭비되는 인프라 지출 문제가 해소됩니다.



FaaS(*Function as a Service*)는 함수를 배포 및 실행 도구로서 신뢰하는 이벤트 기반 컴퓨팅 시스템을 구축하는 하나의 접근 방식입니다. 서버리스 *FaaS* 는 FaaS 의 한 유형으로 프로그래밍 모델에 가상 머신이나 컨테이너가 없으며, 공급업체가 프로비저닝이 필요 없는 확장성과 기본 안정성을 제공합니다.

그림 3: 이벤트 기반 컴퓨팅, FaaS 및 서버리스 간의 관계.

Lambda 에서 이용할 수 있는 AWS 서버리스 컴퓨팅 기능은 모두 서로 원활하게 통합되는 다음과 같은 AWS 기반 관리형 서비스에 중요한 요소입니다.

- Amazon API Gateway – API 프록시 및 API 관리 기능 전체를 포함하는 Lambda 함수에 대한 HTTP 엔드포인트.
- Amazon S3 – 객체가 생성, 복사 또는 삭제될 때 자동 이벤트 트리거로 사용할 수 있는 Lambda 함수.
- Amazon DynamoDB – 데이터베이스 테이블에서 변경된 모든 사항을 처리하는 데 사용할 수 있는 Lambda 함수.
- Amazon SNS – 게시된 콘텐츠에 동적으로 대응하기 위해 기능을 추가하고 처리하는 Lambda 함수에 라우팅될 수 있는 메시지.
- Amazon SQS – Lambda 함수를 통해 손쉽게 처리할 수 있는 대기열의 메시지.
- Amazon Kinesis Streams – Lambda 함수를 통해 제공되는 스트리밍 데이터의 순차적인 레코드 처리로, 거의 실시간으로 분석 엔진을 손쉽게 구축할 수 있음.
- Amazon Kinesis Firehose – Firehose 에서 수집된 레코드에 자동으로 적용되는 Lambda 함수로, 데이터 스트림에 변환, 필터링 및 분석 기능을 손쉽게 추가할 수 있음.
- Amazon Athena – 쿼리 결과 집합의 각 객체에 대해 자동으로 트리거할 수 있는 Lambda 함수.
- AWS Step Functions – 사람 중심 및 자동화된 프로세스 모두에 대해 오랜 시간 지속되는 워크플로를 생성하는 데 오케스트레이션할 수 있는 여러 Lambda 함수.
- Amazon CloudWatch Events – 타사 이벤트를 포함한 이벤트에 자동으로 응답하는 데 사용할 수 있는 Lambda 함수.
- Amazon Aurora – Lambda 함수로 작성될 수 있는 데이터베이스 트리거.

Lambda 는 Slack, Algorithmia, Twilio, Loggly, Splunk, SumoLogic, Box 등 다양한 타사 서비스의 청사진이 포함된 통합 라이브러리를 제공하며, 개발자는 이를 이용해 코드 단 몇 줄로 분석, 고급 알고리즘, 통신 등을 포함한 반응형 애플리케이션을 구축할 수 있습니다. Express(NodeJS 애플리케이션용) 및 Flask(Python 애플리케이션용)를 포함한 다양한 웹 애플리케이션 프레임워크는 Lambda 함수와 원활하게 작동하도록 강화되었습니다. 오픈 소스 웹 애플리케이션 프로젝트는 Serverless Framework, Sparta, Chalice 등 아주 다양합니다.

서버리스 앱은 대개 하나 이상의 함수, Amazon DynamoDB 등의 서버리스 데이터베이스 및 클라이언트가 호출하는 API 또는 앱을 트리거하는 이벤트 소스와 같은 여러 부분으로 구성되어 있습니다. AWS 는 이러한 부분을 조직화된 상태로 유지하기 위해 오픈 사양 서버리스 애플리케이션 모델인 SAM 을 사용합니다. 개발자는 SAM 이 사용해 함수, API, 이벤트 소스, 데이터베이스 테이블 및 서버리스 앱의 다른 부분을 손쉽게 설명할 수 있습니다. 또한 SAM 을 활용하면 개발자는 소프트웨어 개발 수명 주기의 전 단계를 관리할 수 있으며, AWS 는 다양한 지원 도구 및 서비스를 제공합니다. 여기에는 SAM Local 을 통한 선택권 IDE(또는 명령줄을 통한)에서의 로컬 테스트 및 디버깅, 그리고 AWS CloudFormation 을 사용한 SAM 앱 배포에 대한 기본 지원과 AWS CodeBuild 에서의 SAM 앱 구축에 대한 지원, AWS CodePipeline 에 구축된 SAM 앱에 대한 GitHub 기반 CI/CD 에 대한 지원이 포함됩니다. 또한 자사 지원 외에도 수많은 오픈 소스 프레임워크, CI/CD 공급업체 및 성능 관리 공급업체는 Serverless Framework, Claudia, CloudBees, Datadog 등 SAM 및 Lambda 함수에 대한 지원을 제공합니다. 추가 사례는 서버리스 애플리케이션 개발자 도구를 참조하십시오.[4]

Lambda 함수(또는 SAM 앱의 경우 잠재적으로 함께 작동될 수 있는 몇 가지 Lambda 함수)가 생성된 후 개발자는 Amazon CloudWatch 및 CloudWatch Logs 에서 사용할 수 있는 자동 생성된 지표 및 로그를 사용하여 손쉽게 함수를 모니터링할 수 있습니다. 또한 AWS 는 크로스 서비스 요청 추적 및 성능 분석 솔루션인 AWS X-Ray 를 제공하며, 이를 통해 개발자는 자신이 처리하는 개별 함수 및 이벤트에 대한 작업 및 동작을 추적할 수 있습니다.

AWS 서버리스 플랫폼은 사실상 전 세계 모든 AWS 지역에서 AWS Lambda 및 Amazon API Gateway 에 대한 지원과 더불어 글로벌 연결을 제공합니다. Lambda@Edge 는 모든 엣지 로케이션에서 사용할 수 있습니다.[5] Lambda 는 애플리케이션에서 성공적으로 처리되지 않은 이벤트를 캡처하기 위한 전송 실패 대기열 및 명령을 받은 비동기식 이벤트에 대한 자동 재시도를 포함해 고객이 애플리케이션의 안정성을 향상하도록 지원하는 다양한 기능을 제공합니다. Amazon Virtual Private Cloud(Amazon VPC)와의 심층적인 통합과 AWS Lambda 에서 제공된 유연한 범위의 인증 및 액세스 제어 기능을 통해 조직은 최소 권한의 원칙 등의 모범 사례를 준수하는 안전한 애플리케이션을 생성할 수 있습니다. 최종 사용자 보안 및 관리도 간단합니다. Amazon Cognito 는 Amazon API Gateway 및 AWS Lambda 과 손쉽게 결합할 수 있는 인증 및 공인 기능을 제공하며, 이를 통해 Facebook 및 기업 디렉터리와 같은 소셜 공급업체와의 통합을 포함한 서버리스 사용자 등록 및 로그인 기능을 활용할 수 있습니다.

사례 연구

기업들은 주식 거래 확인부터 전자 상거래 웹사이트 구조, 자연어 처리 등의 사용 사례에 서버리스 아키텍처를 적용했습니다. AWS Lambda 및 나머지 AWS 서버리스 포트폴리오는 PCI 또는 HIPAA 규정 준수 등의 보장 프로그램이 필요한 애플리케이션 등 다양한 애플리케이션을 생성하기 위한 유연성을 제공합니다. 다음 섹션에서는 가장 일반적인 사용 사례의 일부를 설명합니다. 이는 전체 목록이 아닙니다. 고객 레퍼런스 및 사용 사례 문서의 전체 목록은 서버리스 컴퓨팅을 참조하십시오.[6]

서버리스 웹사이트, 웹 앱 및 모바일 백엔드

서버리스 접근 방식은 로드가 매우 다양할 수 있는 애플리케이션에 적합합니다. 서버리스 접근 방식을 사용하면 최종 사용자 트래픽이 없는 경우 컴퓨팅 비용이 발생하지 않지만 갑작스러운 트래픽 유입이 발생하는 전자 상거래 사이트의 반짝 세일 또는 소셜 미디어 멘션과 같은 높은 수요를 맞추기 위해 즉각적으로 규모가 확장됩니다. 기존 인프라 접근 방식과 비교하면 서버리스 방식으로 설계했을 때 웹 또는 모바일 백엔드를 개발, 제공 및 운영하는 비용도 크게 절감된 경우가 많습니다.

AWS 는 개발자가 이러한 애플리케이션을 빠르게 구성하는 데 필요한 서비스를 제공합니다.

- Amazon S3 는 정적 콘텐츠에 대한 간단한 호스팅 솔루션을 제공합니다.
- Amazon API Gateway 와 결합된 AWS Lambda 는 함수를 사용하는 동적 API 요청에 대한 지원을 제공합니다.
- Amazon DynamoDB 는 세션 및 사용자별 상태에 대한 간단한 스토리지 솔루션을 제공합니다.
- Amazon Cognito 는 최종 사용자 등록, 인증 및 리소스에 대한 액세스 제어를 다루기 위한 손쉬운 방법을 제공합니다.
- AWS SAM 은 개발자가 애플리케이션의 다양한 요소를 설명하는 데 사용할 수 있습니다.
- AWS CodeStar 에서는 몇 번의 클릭만으로 CI/CD 도구 체인을 설정할 수 있습니다.

자세한 내용은 서버리스 웹 애플리케이션을 구축하기 위한 자세한 패턴 검사 내용을 알 수 있는 AWS Serverless Multi-Tier Architectures 백서를 참조하십시오.[7] 전체 참조 아키텍처는 GitHub 의 Serverless Reference Architecture for creating a Web Application[8] 및 Serverless Reference Architecture for creating a Mobile Backend[9]를 참조하십시오.

고객 사례 – Bustle.com

Bustle.com 은 여성을 위한 뉴스, 엔터테인먼트, 라이프스타일 및 패션 웹사이트입니다. Bustle.com 은 AWS Lambda 및 Amazon API Gateway 를 기반으로 서버리스 아키텍처로 전환하여 약 84%의 비용을 절감했습니다. Bustle 의 엔지니어는 추가로 민첩성도 확보하였으며, 이를 통해 인프라 관리 및 규모 확장을 처리하는 대신 새로운 제품 기능을 구축하는 데 집중할 수 있게 되었습니다. Bustle 의 팀은 이제 보통 Bustle 규모의 사이트를 구축하고 운영하는 데 필요했던 인원의 절반을 활용하여 효율성이 보다 높아졌습니다. 또한 Bustle 의 서버리스 백엔드는 Bustle 의 두 가지 웹 속성(Bustle 및 Romper)에 대해 iOS 앱도 지원합니다. 자세한 정보는 Bustle case study 를 참조하십시오.[10]

IoT 백엔드

서버리스 아키텍처가 웹 및 모바일 앱에 제공하는 이점을 활용하면 디바이스 수에 맞춰 원활하게 확장되는 디바이스 기반 분석 처리 시스템과 IoT 백엔드도 손쉽게 구축할 수 있습니다. 사례 참조 아키텍처는 GitHub 의 Serverless Reference Architecture for creating an IoT Backend 를 참조하십시오.[11]

고객 사례 – iRobot

Roomba 청소 로봇 등의 로봇을 제작하는 iRobot 은 AWS IoT 서비스와 함께 AWS Lambda 를 사용하여 IoT 플랫폼용 서버리스 백엔드를 생성합니다. iRobot 의 엔지니어링 팀은 서버리스 아키텍처를 사용하여 가용성 및 확장성을 처리하기 위해 직접 코드를 작성하거나 인프라를 관리하는 작업에 대해 걱정할 필요가 없게 되었습니다. 이를 통해 보다 빠르게 혁신을 이루고 고객에게 계속해서 집중할 수 있습니다. 자세한 정보는 iRobot 의 AWS re:Invent 2016 프레젠테이션 Serverless IoT Back Ends (IOT401)[12]의 슬라이드를 참조하거나 동영상을 시청하십시오.[13]

데이터 처리

최대 규모의 서버리스 애플리케이션은 막대한 양의 데이터를 처리하며, 그중 대부분을 실시간으로 처리합니다. 일반적인 서버리스 데이터 처리 아키텍처는 Amazon Kinesis 및 AWS Lambda 를 결합하여 사용하여 스트리밍 데이터를 처리하거나 Amazon S3 및 AWS Lambda 를 결합하여 객체 생성 또는 업데이트 이벤트에 대한 응답으로 컴퓨팅을 트리거합니다. 워크로드에서 단순한 트리거보다 복잡한 오케스트레이션을 요구할 경우 개발자는 AWS Step Functions 를 사용하여 진행 중인 Lambda 함수를 하나 이상 호출하는 상태 저장 워크플로 또는 오래 지속되는 워크플로를 생성합니다. 서버리스 데이터 처리 아키텍처에 대해 자세히 알아보려면 GitHub 에서 다음 항목을 참조하십시오.

- Serverless Reference Architecture for Real-time Stream Processing[14]
- Serverless Reference Architecture for Real-time File Processing[15]
- Image Recognition and Processing Backend reference architecture[16]

고객 사례 – FINRA

FINRA(Financial Industry Regulatory Authority)는 AWS Lambda 를 사용하여 서버리스 데이터 처리 솔루션을 구축했습니다. 이를 통해 매일 370 억 개의 주식 시장 이벤트에 대해 5,000 억 건의 데이터 검증을 수행할 수 있게 되었습니다. FINRA 의 수석 이사 Tim Griesbach 는 The State of Serverless Computing (SVR311)이라는 제목의 AWS re:Invent 2016 발표에서 이렇게 말했습니다.[17] “저희는 Lambda 를 통해 이 서버리스 클라우드 솔루션을 위한 최고의 해결책을 얻었습니다. Lambda 만 있으면 시스템의 속도는 빨라지고 비용이 절감되고 더 크게 확장됩니다. 따라서 일일 마감 시 50%가 넘는 비용을 절감했고 … 이를 매일, 또는 시간 단위로도 추적할 수 있습니다.”.

고객 사례 – Thomson Reuters

Thomson Reuters 는 미디어 및 정보 회사로 서버리스 비즈니스 분석 솔루션을 구축하였고, 이를 통해 제품 팀은 손쉽게 제품 사용 데이터를 분석했습니다. 이 솔루션은 AWS Lambda, Amazon Kinesis Streams 및 Amazon Kinesis Firehose 가 결합한 솔루션으로, 분석을 위해 스트리밍 이벤트 데이터를 수집하고 처리합니다. Product Insight 라는 결과가 예정보다 두 달 앞서 시작되었고 기술 면에서 기대 수준을 넘어섰습니다.

Thomson Reuters 의 제품 혁신 수석 관리자 Anders Fritz 는 이렇게 말했습니다. "저희의 첫 목표는 초당 2,000 개의 이벤트를 수용하는 것이었습니다. 테스트 결과 AWS 기반 Product Insight 가 최대 초당 4,000 개의 이벤트를 처리할 수 있음을 확인했고, 1 년 이내에 해당 수치가 증가해 초당 10,000 개 이상의 이벤트를 처리할 수 있을 것으로 예상됩니다." 이는 월별 이벤트 수로 계산하면 250 억 개가 넘는 수치입니다. 이렇게 높은 처리량에도 시스템이 시작 이후 어떤 데이터도 손실하지 않습니다. Fritz 는 이렇게 덧붙였습니다. "AWS 의 강력한 장애 조치 아키텍처와 기술적 기능 덕분에 데이터 수집 이후로 단 하나의 이벤트도 손실하지 않았습니다." 자세한 정보는 Thomson Reuters Case Study[18]를 참조하거나 AWS re:Invent 2016 프레젠테이션인 Real-time Data Processing Using AWS Lambda (SVR301)를 시청하십시오.[19]

빅 데이터

AWS Lambda 는 여러 대량 병렬 처리 워크로드에 매우 적합한 솔루션입니다. MapReduce 를 사용하는 참조 아키텍처 사례는 Reference architecture for running serverless MapReduce jobs 를 참조하십시오.[20]

고객 사례 – Fannie Mae

주택담보대출기관의 자금을 대주는 주요 기업인 Fannie Mae 는 AWS Lambda 를 사용하여 재무 모델링을 위한 "당황스러울 정도의 병렬" 워크로드를 실행합니다. Fannie Mae 는 몬테카를로 시뮬레이션 프로세스를 사용하여 대출 위험을 관리하는 데 도움이 되는 향후 대출 현금 흐름을 예측합니다. Fannie Mae 는 기존 HPC 그리드가 성장하는 비즈니스 요구 사항을 더이상 맞추지 못한다는 사실을 확인했습니다. Fannie Mae 는 Lambda 에서 새로운 플랫폼을 구축했고, 이 시스템은 테스트하는 동안 최대 15,000 건의 동시 함수 실행으로 성공적으로 규모를 확장했습니다. 새로운 시스템은 2,000 만 건의 대출에 대한 시뮬레이션 하나를 2 시간 만에 실행했습니다. 이는 기존 시스템보다 3 배 빠른 속도입니다. 서버리스 아키텍처를 사용하는 Fannie Mae 는 이제 유휴 컴퓨팅 리소스 비용을 지불하지 않기 때문에 비용 효율적인 방식으로 대규모 몬테카를로 시뮬레이션을 실행할 수 있습니다. 또한 여러 Lambda 함수를 동시에 실행하여 컴퓨팅 속도를 높일 수도 있습니다. 또한 Fannie Mae 는 이 시스템을 통해 이전에 애플리케이션 확장성 및 안정성을 관리하는 데 필요했던 많은 복잡한 코드를 제거하는 기능과 더불어 서버 관리 및 모니터링을 없앨 수 있었기 때문에 출시 시간도 이전과 비교해 더욱 짧게 단축할 수 있었습니다. 자세한 정보는 Fannie Mae AWS Summit 2017 프레젠테이션 SMC303: Real-time Data Processing Using AWS Lambda 를 참조하십시오.[21]

IT 자동화

서버리스 접근 방식을 통해 서버 관리 오버헤드를 해소하면서 프로비저닝, 구성, 관리, 경보/모니터 및 시간이 지정된 cron 작업을 포함한 대부분의 인프라 작업을 보다 쉽게 생성하고 관리할 수 있습니다.

고객 사례 – Autodesk

3D 디자인 및 엔지니어링 소프트웨어를 제작하는 Autodesk 는 AWS Lambda 를 사용하여 엔지니어링 조직 전반에 걸쳐 AWS 계정 생성 및 관리 프로세스를 자동화합니다. Autodesk 는 AWS Lambda 를 사용해 98%의 비용을 절감했다고 추정합니다(계정 프로비저닝에 소요되는 노동 시간의 예상 절감 시간 감안). 이전의 인프라 기반 프로세스를 통해 프로비저닝하는 데는 10 시간이 소요되었지만 이제는 그 대신 단 10 분 만에 계정을 프로비저닝할 수 있습니다. Autodesk 는 서버리스 솔루션을 통해 자동화는 증가하고 수동 터치포인트는 감소한 계정 자동 프로비저닝, 표준 구성 및 적용, 감사 실행 작업을 수행할 수 있습니다. 자세한 정보는 Autodesk AWS Summit 2017 프레젠테이션 SMC301: The State of Serverless Computing 을 참조하십시오.[22] Autodesk Tailor 서비스는 GitHub 에서 확인하십시오.

추가 사용 사례

이전 섹션에서 설명된 사용 사례는 Lambda 와 다른 AWS 서비리스 솔루션으로 얻을 수 있는 이점의 아주 표면적인 부분만 살펴본 것입니다. 기타 사용 사례에서는 Amazon Lex 및 AWS Lambda 를 사용하여 구축한 챗봇을 통한 강력한 인간 언어 이해, Amazon CloudFront 와 Lambda@Edge 를 사용한 짧은 지연 시간 글로벌 엣지 컴퓨팅, 그리고 AWS Snowball 내 Lambda 함수를 통한 강력한 온프레미스 파일 처리를 다룹니다. 이는 다양하게 활용할 수 있는 접근 방식의 흥미로운 일부 기능일 뿐입니다. 자세히 알아보려면 AWS Lambda 를 참조하십시오.[23]

결론

서버리스 접근 방식은 가치를 제공하지 않고 회사의 비용을 갉아 먹는 유휴 서버와 비즈니스가 차별화된 고객 가치를 창출하는 데 방해가 되는 서버 집합 및 서버 소프트웨어를 구축하고 운영하는 데 드는 비용, 즉 기존 IT 관리의 두 가지 문제점을 해결하도록 설계되었습니다. AWS Lambda 및 기타 AWS 서버리스 솔루션은 서버, 컨테이너, 디스크 및 다른 인프라 수준의 리소스를 프로그래밍 및 비용 청구

모델에서 없애 이러한 고질적인 문제를 해결해 줍니다. 따라서 개발자는 제공 속도를 단축하고 조직이 유용한 작업에 대해서만 결제하도록 하는 군더더기 없는 애플리케이션 모델로 작업할 수 있습니다. 대응형 이벤트 기반 시스템을 설계하고 클라우드 네이티브 마이크로서비스를 제공하는 가장 쉽고 빠른 방법은 서버리스 아키텍처를 사용하는 것입니다. 자세한 내용을 알아보고 관련 주제의 백서를 읽어보려면 서버리스 컴퓨팅 및 애플리케이션을 참조하십시오.[24]

기고자

다음은 본 문서 작성에 도움을 준 개인 및 조직입니다.

- Tim Wagner, AWS Serverless Applications, Amazon Web Services 총책임자

참고 문헌

추가 정보는 다음을 참조하십시오.

- Serverless Reference Architectures with AWS Lambda(Amazon.com, CTO Werner Vogels 작성)
- AWS re:Invent 2016: The State of Serverless Computing[프레젠테이션] (AWS Serverless Applications 총책임자 Tim Wagner 작성)
- The economics of serverless cloud computing(451 Research 연구 책임자 Owen Rogers 작성)

참조 아키텍처

- 웹 애플리케이션
- 모바일 백엔드
- IoT 백엔드
- 파일 처리
- 스트림 처리
- 이미지 인식 처리
- MapReduce

문서 수정

날짜	설명
2017 년 9 월	첫 게시

Notes

[1] https://www.forbes.com/sites/moorinsights/2016/04/11/tco-analysis-demonstrates-how-moving-to-the-cloud-can-save-your-company-money/#537e2bd07c4e
http://www.cloudstrategymag.com/articles/86033-understanding-tco-cloud-economics

[2] 2012 년에 Gartner 는 데이터 센터 사용률을 7~12%로 측정했습니다 (http://www.nytimes.com/2012/09/23/technology/data-centers-waste-vast-amounts-of-energy-belying-industry-image.html). 2008 McKinsey 연구에서는 6%(https://www.sallan.org/pdf-docs/McKinsey_Data_Center_Efficiency.pdf)로 예측했습니다. EC2 기반 애플리케이션 세트를 분석하는 Accenture 문서에서는 사용률을 약 7%로 보고 있습니다(http://ieeexplore.ieee.org/document/6118751/). NRDC 및 Anthesis 의 2014 년 연구에서는 2013 년에 30%가 넘는 서버가 완전히 "코마 상태"(전원이 꽂혀 있지만 가치 있는 작업을 전혀 수행하지 않음) (http://anthesisgroup.com/wp-content/uploads/2015/06/Case-Study_DataSupports30PercentComatoseEstimate-FINAL_06032015.pdf) 라고 밝혔습니다.

[3] Occupy the Cloud: Eric Jonas et al., *Distributed Computing for the 99%*, https://arxiv.org/abs/1702.04024.

[4] https://aws.amazon.com/serverless/developer-tools

[5] https://aws.amazon.com/lambda/edge/

[6] https://aws.amazon.com/serverless/

[7] https://d0.awsstatic.com/whitepapers/AWS_Serverless_Multi-Tier_Architectures.pdf

[8] https://github.com/awslabs/lambda-refarch-webapp

[9] https://github.com/awslabs/lambda-refarch-mobilebackend

[10] https://aws.amazon.com/solutions/case-studies/bustle/

[11] https://github.com/awslabs/lambda-refarch-iotbackend

[12] https://www.slideshare.net/AmazonWebServices/aws-reinvent-2016-serverless-iot-back-ends-iot401

[13] https://www.youtube.com/watch?v=gKMaf5E-z7Q

[14] https://github.com/awslabs/lambda-refarch-streamprocessing

[15] https://github.com/awslabs/lambda-refarch-fileprocessing

[16] https://github.com/awslabs/lambda-refarch-imagerecognition

[17] https://www.youtube.com/watch?v=AcGv3qUrRC4&feature=youtu.be&t=1153

[18] https://aws.amazon.com/solutions/case-studies/thomson-reuters/

[19] https://www.youtube.com/watch?v=VFLKOy4GKXQ&feature=youtu.be&t=1449

[20] https://github.com/awslabs/lambda-refarch-mapreduce

[21] https://www.slideshare.net/AmazonWebServices/smc303-realtime-data-processing-using-aws-lambda/28

[22] https://www.slideshare.net/AmazonWebServices/smc301-the-state-of-serverless-computing-75290821/22

[23] https://aws.amazon.com/lambda/

[24] https://aws.amazon.com/serverless/