AWS IoT 렌즈

AWS Well-Architected 프레임워크

2019 년 12 월

This paper has been archived.

The latest version is now available at:

https://docs.aws.amazon.com/ko_kr/wellarchitected/latest/iot-lens/welcome.html

고지 사항

고객에게는 본 문서에 포함된 정보를 독립적으로 평가할 책임이 있습니다. 본 문서는 (a) 정보 제공만을 위한 것이며, (b)사전 고지 없이 변경될 수 있는 현재의 AWS 제품 제공 서비스 및 사례를 보여 주며, (c) AWS 및 자회사, 공급업체 또는 라이선스 제공자로부터 어떠한 약정 또는 보증도 하지 않습니다. AWS 제품 또는 서비스는 명시적이든 묵시적이든 어떠한 종류의 보증, 진술 또는 조건 없이 "있는 그대로" 제공됩니다. 고객에 대한 AWS 의 책임과 법적 책임은 AWS 계약서에 준하며 본 문서는 AWS 와 고객 간의 어떠한 계약도 구성하거나 수정하지 않습니다.

목차

소개 1

정의 1

- 설계 및 제조 계층 2
- 엣지 계층 3
- 프로비저닝 계층 3
- 통신 계층 4
- 수집 계층 5
- 분석 계층 5
- 애플리케이션 계층 7

일반 설계 원칙 8

시나리오 10

- 디바이스 프로비저닝 10
- 디바이스 원격 측정 12
- 디바이스 명령 13
- 펌웨어 업데이트 16

Well-Architected 프레임워크의 기반 17

- 운영 우수성 기반 17
- 보안 기반 24

안정성 기반 39

성능 효율성 기반 47

비용 최적화 기반 57

결론 63

기고자 63

문서 개정 63

요약

이 백서에서는 클라우드 기반 아키텍처를 검토하여 개선하고 설계 의사 결정의 비즈니스 영향을 이해하는 데 도움이 되는 AWS Well-Architected 프레임워크의 **AWS IoT 렌즈**를 설명합니다. 이 문서에는 일반 설계 원칙과 더불어 Well-Architected 프레임워크의 5 가지 기반에 대한 구체적인 모범 사례 및 지침이 설명되어 있습니다.

소개

AWS Well-Architected 프레임워크는 AWS 에서 시스템을 구축할 때 내리는 의사 결정의 장점과 단점을 이해하는 데 도움이 됩니다. 이 프레임워크를 사용하면 클라우드에서 안정적이고 안전하며 효율적이고 경제적인 시스템을 설계하고 운영하기 위한 설계 모범 사례를 알아볼 수 있습니다. 이 프레임워크는 모범 사례를 기준으로 아키텍처를 일관적으로 측정하고 개선 영역을 식별하는 방법을 제공합니다. 제대로 설계된 시스템을 갖추면 비즈니스 성공 가능성이 높아집니다.

이 "렌즈"에서는 AWS 클라우드 기반 IoT(사물 인터넷) 워크로드의 설계, 배포 및 아키텍처 구성 방법을 집중적으로 살펴봅니다. 제대로 설계된 IoT 애플리케이션을 구현하려면 연결된 물리적 자산(사물)의 조달에서 시작하여 안전하고 안정적이며 자동화된 방식으로 해당 자산을 최종적으로 폐기할 때까지 Well-Architected 프레임워크의 원칙을 따라야 합니다. 이 문서에는 AWS 클라우드 모범 사례에 더해 물리적 자산을 인터넷에 연결할 때의 영향, 고려 사항 및 권장 사항도 설명되어 있습니다.

이 문서에서는 Well-Architected 프레임워크의 IoT 관련 워크로드 세부 정보만 다룹니다. AWS Well-Architected 프레임워크 백서를 읽고 다른 렌즈의 모범 사례 및 질문을 고려하시기 바랍니다.

이 문서는 CTO(최고 기술 책임자), 아키텍트, 개발자, 임베디드 엔지니어 및 운영 팀 팀원을 포함한 기술 업무 담당자를 위해 작성되었습니다. 이 문서를 읽으면 IoT 애플리케이션에 대한 AWS 모범 사례와 전략을 이해할 수 있습니다.

정의

AWS Well-Architected 프레임워크는 운영 우수성, 보안, 안정성, 성능 효율성, 비용 최적화라는 5 가지 핵심 요소를 기반으로 합니다. 기술 솔루션을 설계할 때는 비즈니스 상황에 따라 정보를 기반으로 이러한 핵심 요소를 절충해야 합니다. IoT 워크로드에 있어서 AWS 는 강력한 애플리케이션 아키텍처를 설계할 수 있는 여러 서비스를 제공합니다. IoT(사물 인터넷) 애플리케이션은 상호 보완적인 엣지 기반 및 클라우드 기반 구성 요소와 안전하게 연결하고 상호

작용하여 비즈니스 가치를 제공하는 다수의 디바이스(또는 사물)로 구성됩니다. IoT 애플리케이션은 연결된 디바이스에서 생성된 데이터를 수집, 처리 및 분석하고 데이터를 기반으로 작업을 수행합니다. 이 섹션에서는 IoT 워크로드의 아키텍처 설계와 관련하여 이 문서 전체에서 사용되는 AWS 구성 요소에 대한 개요를 제공합니다. IoT 워크로드를 구축할 때는 7 가지 논리적 계층을 고려해야 합니다.

- 설계 및 제조 계층
- 엣지 계층
- 프로비저닝 계층
- 통신 계층
- 수집 계층
- 분석 계층
- 애플리케이션 계층

설계 및 제조 계층

설계 및 제조 계층은 제품 개념화, 비즈니스 및 기술 요구 사항 수집, 프로토타입 생성, 모듈 및 제품 레이아웃 및 설계, 구성 요소 소싱과 제조로 구성됩니다. 각 단계에서 내려진 의사 결정은 아래에 설명된 IoT 워크로드의 다음 논리적 계층에 영향을 미칩니다. 예를 들어 일부 IoT 디바이스 제작자는 하청 제조업체를 통해 일반 펌웨어 이미지를 굽고 테스트하는 것을 선호합니다. 프로비저닝 계층 중에 필요한 단계는 부분적으로 이 의사 결정에 따라 결정됩니다.

한 단계 더 나아가서 제조 중에 각 디바이스에 고유한 인증서와 개인 정보 키를 구울 수 있습니다. 자격 증명 유형에 따라 이후 네트워크 프로토콜의 선택이 달라지므로 이 의사 결정은 통신 계층에 영향을 미칠 수 있습니다. 자격 증명에 만료 기간이 없는 경우 통신 및 프로비저닝 계층이 간소화될 수 있지만 발급 인증 기관의 침해로 인한 데이터 손실 위험은 증가할 수 있습니다.

엣지 계층

IoT 워크로드의 엣지 계층은 디바이스의 물리적 하드웨어, 디바이스의 프로세스를 관리하는 내장된 운영 체제 및 IoT 디바이스에 프로그래밍된 소프트웨어 및 명령인 디바이스 펌웨어로 구성됩니다. 엣지는 다른 주변 디바이스를 감지하고 필요한 작업을 수행하는 역할을 합니다. 일반적인 사용 사례로는 엣지 디바이스에 연결된 센서를 읽거나 사용자 작업에 따라 주변 장치의 상태를 변경(예: 동작 센서가 활성화될 때 조명 켜기)하는 것이 있습니다.

AWS IoT Device SDK 는 프로그래밍 언어 또는 플랫폼에 맞게 조정된 API 를 통해 디바이스 및 애플리케이션에서 AWS IoT Core 를 간편하게 사용할 수 있는 기능을 제공합니다.

Amazon FreeRTOS 는 메모리 효율적이고 안전한 임베디드 라이브러리를 활용하면서 소형 저출력 엣지 디바이스를 프로그래밍할 수 있는, 마이크로컨트롤러용 실시간 운영 체제입니다.

AWS IoT Greengrass 는 IoT 디바이스의 Linux 운영 체제를 확장하는 소프트웨어 구성 요소입니다. AWS IoT Greengrass 를 사용하면 디바이스, 데이터 캐싱, AWS IoT 섀도우 동기화, 로컬 AWS Lambda 함수 및 기계 학습 알고리즘 간의 MQTT 로컬 라우팅을 실행할 수 있습니다.

프로비저닝 계층

IoT 워크로드의 프로비저닝 계층은 고유한 디바이스 자격 증명을 생성하는 데 사용되는 PKI(퍼블릭 키 인프라)와 디바이스에 구성 데이터를 제공하는 애플리케이션 워크플로로 구성됩니다. 프로비저닝 계층은 시간이 지나면서 디바이스의 지속적인 유지 관리 및 최종 폐기에 관여합니다. IoT 애플리케이션을 통해 마찰 없이 디바이스를 추가하고 관리하려면 IoT 애플리케이션에 강력하고 자동화된 프로비저닝 계층이 필요합니다. IoT 디바이스를 프로비저닝할 때는 디바이스에 고유한 암호화 자격 증명을 설치해야 합니다.

X.509 인증서를 사용하면 디바이스의 신뢰할 수 있는 자격 증명을 안전하게 생성하는 프로비저닝 계층을 구현하여 통신 계층에 대한 인증 및 권한 부여에 이러한 자격 증명을 사용할 수 있습니다. X.509 인증서는 CA(인증 기관)라고 하는 신뢰할 수 있는 엔터티에서 발급합니다. X.509 인증서는 메모리 및 처리 요구 사항으로 인해 제약된 디바이스에서 리소스를 소비하지만, 운영 확장성과 표준 네트워크 프로토콜을 통한 광범위한 지원 덕에 이상적인 자격 증명 메커니즘입니다.

AWS Certificate Manager Private CA 를 사용하면 IoT 디바이스에 대한 프라이빗 인증서의 수명 주기 관리 프로세스를 API 를 사용하여 자동화할 수 있습니다. X.509 인증서와 같은 프라이빗 인증서가 있으면 프로비저닝 중에 장기 자격 증명을 생성하여 디바이스에 안전하게 제공하고 이러한 자격 증명을 사용하여 IoT 애플리케이션에 대한 디바이스 권한을 식별하고 부여할 수 있습니다.

AWS IoT JITR Just In Time Registration(JITR)을 사용하면 AWS IoT Core 와 같은 관리형 IoT 플랫폼에서 사용할 디바이스를 프로그래밍 방식으로 등록할 수 있습니다. 적시 등록을 사용하면 디바이스가 AWS IoT Core 엔드포인트에 처음 연결될 때 디바이스 인증서 자격 증명의 유효성을 확인하고 부여해야 할 권한을 결정하는 워크플로를 자동으로 트리거할 수 있습니다.

통신 계층

통신 계층은 연결, 원격 디바이스 간 메시지 라우팅 및 디바이스와 클라우드 간 라우팅을 처리합니다. 통신 계층에서는 디바이스의 IoT 메시지 송수신 방법과 디바이스의 물리적 상태를 클라우드에 표현하고 저장하는 방법을 설정할 수 있습니다.

AWS IoT Core 는 MQTT 프로토콜을 사용하여 디바이스 간에 IoT 메시지를 게시 및 구독할 수 있도록 지원하는 관리형 메시지 브로커를 제공하므로 IoT 애플리케이션을 구축하는 데 도움이 됩니다.

AWS IoT 디바이스 레지스트리는 사물을 관리하고 운영하는 데 도움이 됩니다. 사물은 클라우드의 특정 디바이스 또는 논리적 엔터티의 표현입니다. 사물에는 배포된 자산을 식별, 분류 및 검색하는 데 도움이 되는 사용자 지정 정의된 정적 속성이 포함될 수도 있습니다.

AWS IoT 디바이스 섀도우 서비스를 사용하면 특정 디바이스의 현재 상태가 포함되는 데이터 스토어를 생성할 수 있습니다. 디바이스 섀도우 서비스는 AWS IoT 에 연결하는 각 디바이스의 가상 표현을 고유한 디바이스 섀도우의 형태로 유지합니다. 각 디바이스 섀도우는 해당하는 사물의 이름으로 고유하게 식별됩니다.

Amazon API Gateway 를 사용하면 IoT 애플리케이션에서 HTTP 요청을 통해 IoT 디바이스를 제어할 수 있습니다. IoT 애플리케이션에는 원격 기술자를 위한 대시보드와 같은 내부 시스템용 API 인터페이스와 가정용 소비자 모바일 애플리케이션과 같은 외부 시스템용 API 인터페이스가

필요합니다. Amazon API Gateway 를 사용하면 기반 인프라의 프로비저닝 및 관리 없이 공통 API 인터페이스를 생성할 수 있습니다.

수집 계층

IoT 의 핵심 비즈니스 동인은 디바이스에서 생성된 모든 개별 데이터 스트림을 집계하고 데이터를 안전하고 안정적인 방식으로 IoT 애플리케이션으로 전송할 수 있는 기능입니다. 수집 계층은 데이터 흐름을 디바이스 간 통신과 분리하면서 디바이스 데이터를 수집하는 데 중요한 역할을 합니다.

AWS IoT 규칙 엔진을 사용하면 디바이스와 AWS 서비스의 상호 작용에 사용되는 IoT 애플리케이션을 구축할 수 있습니다. AWS IoT 규칙이 분석된 후 메시지가 수신되는 MQTT 주제 스트림을 기반으로 작업이 수행됩니다.

Amazon Kinesis 는 스트리밍 데이터를 위한 관리형 서비스로서, 이 서비스를 사용하면 적시에 통찰력을 얻고 IoT 디바이스의 새로운 정보에 신속하게 대응할 수 있습니다. Amazon Kinesis 는 AWS IoT 규칙 엔진과 직접 통합되므로 디바이스의 경량 디바이스 프로토콜과 다른 프로토콜을 사용하는 내부 IoT 애플리케이션을 MQTT 를 사용하여 원활하게 연결할 수 있습니다.

Kinesis 와 마찬가지로, 통신 계층을 애플리케이션 계층에서 분리하려면 IoT 애플리케이션에서 **Amazon Simple Queue Service(Amazon SQS)**를 사용해야 합니다. 애플리케이션에서 메시지 순서가 필요하지 않은 IoT 애플리케이션을 처리해야 하는 경우 Amazon SQS 를 사용하여 이벤트 중심의 확장 가능한 수집 대기열을 지원할 수 있습니다.

분석 계층

IoT 솔루션 구현의 이점 중 하나는 로컬/엣지 환경에서 일어나는 일에 대한 심층적인 통찰력과 데이터를 확보할 수 있다는 것입니다. 상황에 맞는 통찰력을 얻는 기본적인 방법 중 하나는 IoT 데이터에 대한 분석을 처리하고 수행할 수 있는 솔루션을 구현하는 것입니다.

스토리지 서비스

IoT 워크로드는 대량의 데이터를 생성하도록 설계되는 경우가 많습니다. 이 개별 데이터는 안전하게 전송, 처리 및 소비되어야 하며 그 동안 내구력이 있는 위치에 저장되어야 합니다.

Amazon S3 는 인터넷을 통해 어디서나 원하는 양의 데이터를 저장하고 검색할 수 있도록 설계된 객체 기반 스토리지입니다. Amazon S3 를 사용하면 규제, 비즈니스 변화, 지표, 장기 연구, 분석 기계 학습 및 조직 지원 등 다양한 목적으로 대량의 데이터를 저장하는 IoT 애플리케이션을 구축할 수 있습니다. Amazon S3 는 비용 최적화 및 지연 시간은 물론, 액세스 제어 및 규정 준수를 위해 다양한 방법으로 데이터를 관리할 수 있는 유연성을 제공합니다.

분석 및 기계 학습 서비스

IoT 데이터가 중앙 스토리지 위치에 도달한 후에는 디바이스 동작에 대한 분석 및 기계 학습을 구현하여 IoT 의 모든 가치를 끌어낼 수 있습니다. 분석 시스템을 사용하면 분석을 기반으로 데이터 중심 의사 결정을 수행하여 디바이스 펌웨어, 엣지 및 클라우드 로직의 개선을 시작할 수 있습니다. 분석 및 기계 학습을 통해 IoT 시스템은 예측 유지 관리 또는 이상 탐지와 같은 사전 예방적 전략을 구현하여 시스템의 효율성을 개선할 수 있습니다.

AWS IoT Analytics 를 사용하면 IoT 데이터의 볼륨에 대한 정교한 분석을 손쉽게 실행할 수 있습니다. 사용자는 AWS IoT Analytics 로 기반 IoT 데이터 스토어를 관리하면서 자체 분석 쿼리 또는 Jupyter 노트북을 사용하여 구체화된 여러 데이터 보기를 구축할 수 있습니다.

Amazon Athena 는 표준 SQL 을 사용하여 Amazon S3 에 저장된 데이터를 간편하게 분석할 수 있는 대화형 쿼리 서비스입니다. Athena 는 서버리스 서비스이므로 관리할 인프라가 없으며 실행한 쿼리에 대해서만 비용을 지불하면 됩니다.

Amazon SageMaker 는 클라우드 및 엣지 계층에서 기계 학습 모델을 신속하게 구축, 학습 및 배포할 수 있는 완전관리형 플랫폼입니다. Amazon SageMaker 를 사용하면 IoT 아키텍처에서 디바이스 기록을 원격 측정하여 향후 동작을 추론하는 모델을 개발할 수 있습니다.

애플리케이션 계층

AWS IoT 는 IoT 디바이스로부터 생성된 데이터를 클라우드 네이티브 애플리케이션에서 손쉽게 소비할 수 있는 몇 가지 방법을 제공합니다. 이러한 연결된 기능에는 서버리스 컴퓨팅, IoT 데이터의 구체화된 보기를 생성하는 관계형 데이터베이스 및 관리 애플리케이션부터 IoT 작업의 운영, 검사, 보안 및 관리하는 기능이 포함됩니다.

관리 애플리케이션

관리 애플리케이션의 목적은 현장에 배포된 디바이스를 운영할 수 있는 확장 가능한 방법을 만드는 것입니다. 시스템에서 애플리케이션 문제를 해결하는 데 필요한 가시성을 확보하려면 디바이스의 연결 상태를 검사하고, 디바이스 자격 증명이 올바르게 구성되었는지 확인하고, 현재 상태를 기반으로 디바이스를 쿼리하는 등의 일반적인 운영 작업을 시작 전에 수행해야 합니다.

AWS IoT Device Defender 는 디바이스 플릿을 감사하고, 비정상적인 디바이스 동작을 감지하고, 보안 문제에 대해 경고하고, 일반적으로 발생하는 IoT 보안 문제를 조사 및 완화하는 데 도움이 되는 완전관리형 서비스입니다.

AWS IoT Device Management 를 사용하면 대규모의 IoT 디바이스를 손쉽게 구성, 모니터링 및 관리할 수 있습니다. 대규모 환경에서 고객은 여러 물리적 위치의 디바이스 플릿을 관리합니다. AWS IoT Device Management 를 사용하면 디바이스를 그룹화하여 관리를 간소화할 수 있습니다. Device Management 의 플릿 인덱싱을 통해 디바이스의 현재 상태에 대한 실시간 검색 인덱싱을 활성화할 수도 있습니다. 업데이트해야 하는 대상 디바이스를 결정할 때는 디바이스 그룹과 플릿 인덱싱을 OTA(Over the Air Updates)와 함께 사용할 수 있습니다.

사용자 애플리케이션

관리형 애플리케이션에 더해, 다른 내부 및 외부 시스템에서는 IoT 데이터의 서로 다른 세그먼트를 사용하여 다양한 애플리케이션을 구축해야 합니다. 최종 소비자 보기, 비즈니스 운영 대시보드 및 지속적으로 구축되는 기타 새로운 애플리케이션을 지원하려면 연결 및 수집 계층에서 필요한 정보를 수신한 후 다른 시스템에서 사용하도록 형식을 지정할 수 있는 몇 가지 다른 기술이 필요합니다.

데이터베이스 서비스 – NoSQL 및 SQL

데이터 레이크는 형식이 지정되지 않은 모든 IoT 생성 데이터의 랜딩 존으로 작동할 수 있지만, IoT 데이터 기반의 형식이 지정된 모든 보기를 지원하려면 구조적 및 반구조적 데이터 스토어로 데이터 레이크를 보완해야 합니다. 이러한 목적으로, NoSQL 데이터베이스와 SQL 데이터베이스를 모두 활용해야 합니다. 이러한 유형의 데이터베이스를 사용하면 애플리케이션의 개별 최종 사용자에 대해 서로 다른 IoT 데이터 보기를 생성할 수 있습니다.

Amazon DynamoDB 는 IoT 데이터를 위한 빠르고 유연한 NoSQL 데이터베이스 서비스입니다. IoT 애플리케이션을 사용하는 고객은 종종 안정적인 성능을 제공하고 처리 용량을 자동으로 조정하는 유연한 데이터 모델을 필요로 합니다.

Amazon Aurora 를 사용하면 IoT 아키텍처에서 성능 및 비용 효율적인 오픈 소스 데이터베이스에 구조적 데이터를 저장할 수 있습니다. 미리 정의된 SQL 쿼리를 위해 다른 IoT 애플리케이션에서 데이터에 액세스해야 하는 경우 관계형 데이터베이스를 사용하면 데이터의 개별 세그먼트에 따라 작업을 수행해야 하는 최종 비즈니스 애플리케이션에서 수집 계층의 디바이스 스트림을 분리할 수 있습니다.

컴퓨팅 서비스

IoT 워크로드에서는 데이터가 생성, 수집 또는 소비/실행될 때 애플리케이션 코드를 실행해야 하는 경우가 많습니다. 컴퓨팅 코드를 실행해야 하는 시기와 관계없이 서버리스 컴퓨팅은 매우 비용 효율적인 선택입니다. 서버리스 컴퓨팅은 엣지에서 코어까지, 코어에서 애플리케이션 및 분석까지 활용될 수 있습니다.

AWS Lambda 를 사용하면 서버를 프로비저닝하거나 관리하지 않고도 코드를 실행할 수 있습니다. IoT 워크로드는 수집 규모가 크기 때문에 AWS Lambda 를 사용하여 관리형 플랫폼에서 상태 비저장 이벤트 중심 IoT 애플리케이션을 실행하는 것이 좋습니다.

일반 설계 원칙

Well-Architected 프레임워크는 클라우드에서 IoT 의 올바른 설계를 촉진하는 다음과 같은 설계 원칙 세트를 식별합니다.

- **처리와 수집 분리**: IoT 애플리케이션에서 수집 계층은 높은 비율의 스트리밍 디바이스 데이터를 처리할 수 있는 고도로 확장 가능한 플랫폼이어야 합니다. 대기열, 버퍼 및 메시징 서비스를 사용하여 애플리케이션의 처리 부분에서 빠른 속도로 수집을 분리하면 IoT 애플리케이션에서 디바이스에 영향을 주지 않고 데이터 처리 빈도 또는 관심 데이터 유형과 같은 여러 의사 결정을 내릴 수 있습니다.
- **오프라인 동작에 적합한 설계**: 연결 문제 또는 잘못된 설정 등으로 인해 디바이스는 예상보다 훨씬 더 오랜 시간 동안 오프라인 상태가 될 수 있습니다. 장기간의 오프라인 연결을 처리할 임베디드 소프트웨어를 설계하고, 정기적으로 통신하지 않는 디바이스를 추적하는 지표를 클라우드에서 생성하십시오.
- **엣지에서 린 데이터를 설계하고 클라우드에서 보강:** IoT 디바이스에는 제약적인 특성이 있기 때문에 초기 디바이스 스키마는 물리적 디바이스의 스토리지와 디바이스에서 IoT 애플리케이션으로의 효율적인 데이터 전송에 맞춰 최적화됩니다. 이러한 이유로 형식이 지정되지 않은 디바이스 데이터는 클라우드에서 추론할 수 있는 정적 애플리케이션 정보로 보강되지 않는 경우가 많습니다. 따라서 데이터가 애플리케이션으로 수집될 때, 인간이 판독 가능한 속성으로 데이터를 먼저 보강하거나 디바이스가 직렬화한 모든 필드를 역직렬화하거나 확장한 후 애플리케이션 읽기 요구 사항을 지원하도록 조정된 데이터 스토어에서 데이터 형식을 지정하는 것이 좋습니다.
- **개인화 처리**: Wi-Fi 를 통해 엣지 또는 클라우드에 연결하는 디바이스는 액세스 포인트 이름과 네트워크 암호를 수신해야 합니다. 이 작업은 디바이스를 설정할 때 수행되는 첫 번째 단계 중 하나로 수행됩니다. 이 데이터는 일반적으로 디바이스 제조 중에 디바이스에 기록되지 않는데 데이터 자체가 중요하고 사이트별로 다르거나 클라우드에서는 아직 디바이스가 연결되지 않은 상태이기 때문입니다. 개인화 데이터는 이러한 요인으로 인해 개념적으로 업스트림인 디바이스 클라이언트 인증서 및 프라이빗 키와 개념적으로 다운스트림인 클라우드 제공 펌웨어 및 구성 업데이트와 구별되는 경우가 많습니다. 개인화를 지원하려면 디바이스 자체에 직접 데이터 입력을 위한 사용자 인터페이스를 구축해야 하거나 디바이스를 로컬 네트워크에 연결하는 스마트폰 애플리케이션을 제공해야 합니다. 따라서 개인화 지원은 설계 및 제조에 영향을 미칠 수 있습니다.

- **디바이스가 상태 확인을 주기적으로 전송하는지 확인**: 장시간 주기적으로 오프라인이 되는 디바이스의 경우에도 디바이스 상태 정보를 IoT 애플리케이션에 전송할 정기적 간격을 설정하는 애플리케이션 로직이 디바이스 펌웨어에 포함되어야 합니다. 애플리케이션에 올바른 수준의 가시성을 제공하기 위한 기능이 디바이스에 포함되어야 합니다. 정기적으로 발생하는 이 IoT 메시지를 전송하면 IoT 애플리케이션에서 디바이스의 전체 상태에 대한 업데이트된 정보를 확인하고 디바이스가 예상된 시간 내에 통신하지 않는 경우 프로세스를 생성할 수 있습니다.

시나리오

이 섹션에서는 IoT 애플리케이션과 관련된 일반적인 시나리오를 살펴보면서 각 시나리오가 IoT 워크로드의 아키텍처에 미치는 영향을 중점적으로 다룹니다. 이러한 예제는 완전한 것은 아니지만 IoT 의 일반적인 패턴을 포함합니다. 각 시나리오에 대한 배경, 시스템 설계에 대한 일반적인 고려 사항 및 시나리오 구현 방법에 대한 참조 아키텍처가 나와 있습니다.

디바이스 프로비저닝

IoT 에서 디바이스 프로비저닝은 몇 가지 순차적 단계로 구성됩니다. 가장 중요한 측면은 각 디바이스에 고유한 자격 증명을 부여한 후 IoT 애플리케이션에서 이 자격 증명을 사용하여 디바이스를 인증하는 것입니다.

따라서 디바이스를 프로비저닝하는 첫 번째 단계는 자격 증명을 설치하는 것입니다. 고객이 디바이스를 받을 때 디바이스에 프로덕션용 펌웨어 이미지 및/또는 고유한 클라이언트 자격 증명이 포함되는지 여부는 디바이스를 설계하고 제조할 당시의 의사 결정에 따라 결정됩니다. 의사 결정에 따라 프로덕션 디바이스 자격 증명을 설치하기 위해 프로비저닝 시점에서 추가 단계를 수행해야 할 수 있습니다.

X.509 클라이언트 인증서는 대규모에서 정적 암호보다 안전하고 관리가 쉽습니다. 따라서 IoT 애플리케이션에는 이 인증서를 사용하는 것이 좋습니다. AWS IoT Core 에서 디바이스는 디바이스 인증서와 고유한 사물 식별자를 사용하여 등록됩니다. 등록된 디바이스는 IoT 정책에 연결됩니다. IoT 정책을 사용하면 디바이스별로 세분화된 권한을 생성할 수 있습니다. 세분화된 권한을

사용하면 디바이스의 MQTT 주제 및 메시지와 상호 작용할 수 있는 권한을 해당 디바이스에만 부여할 수 있습니다.

이 등록 프로세스를 통해 디바이스는 IoT 자산으로 인식되고 디바이스에서 생성되는 데이터는 AWS IoT 를 통해 나머지 AWS 에코시스템에 제공됩니다. 디바이스를 프로비저닝하려면 자동 등록을 활성화하고 프로비저닝 템플릿 또는 AWS Lambda 함수를 초기 디바이스 프로비저닝 이벤트에 연결해야 합니다.

이 등록 메커니즘에서 디바이스는 프로비저닝 중(제조 중 또는 제조 후)에 디바이스에 제공된 고유한 인증서를 사용하여 IoT 애플리케이션(이 경우 AWS IoT)에 인증합니다. 이 접근 방식의 한 가지 장점은 디바이스를 다른 엔터티로 전송하고 다시 프로비저닝하여 새 소유자의 AWS IoT 계정 세부 정보로 등록 프로세스를 반복할 수 있다는 것입니다.

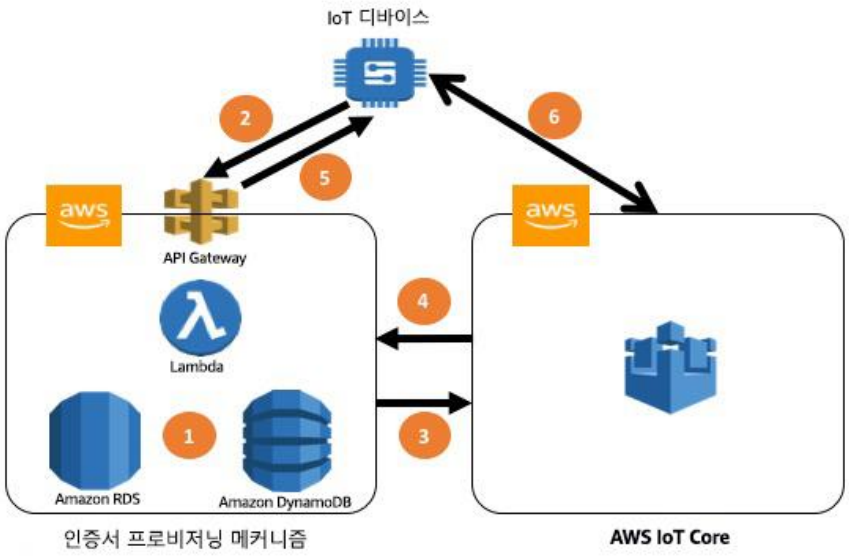

그림 1: 등록 흐름

1. 제조 디바이스 식별자를 데이터베이스에 설정합니다.
2. 디바이스가 API Gateway 에 연결되고 CPM 에서 등록을 요청합니다. 요청이 검증됩니다.
3. Lambda 가 프라이빗 CA(인증 기관)의 X.509 인증서를 요청합니다.

4. 프로비저닝 시스템이 CA 를 AWS IoT Core 에 등록했습니다.
5. API Gateway 가 디바이스 자격 증명을 디바이스로 전달합니다.
6. 디바이스가 AWS IoT Core 를 사용하여 등록 워크플로를 시작합니다.

디바이스 원격 측정

머신의 성능에 대한 원격 측정을 수집하여 IoT 의 가치를 실현할 수 있는 많은 사용 사례(예: 산업용 IoT)가 있습니다. 예를 들어 이 데이터를 사용하여 예측 유지 관리를 활성화하면 예상치 못한 장비 오류로 인한 비용을 방지할 수 있습니다. 원격 측정은 머신에서 수집되어 IoT 애플리케이션에 업로드되어야 합니다. 원격 측정을 전송하면 클라우드 애플리케이션에서 이 데이터를 분석에 사용하고 최적화를 해석하여 펌웨어에 이러한 최적화를 지속적으로 적용할 수 있다는 이점도 있습니다.

원격 측정 데이터는 수집된 후 IoT 애플리케이션으로 전송되는 읽기 전용 데이터입니다. 원격 측정 데이터는 수동 데이터이기 때문에 원격 측정 메시지에 대한 MQTT 주제는 IoT 명령과 관련된 다른 주제와 겹치지 않아야 합니다. 예를 들어 원격 측정 주제가 data/device/sensortype 인 경우 여기서 "data"로 시작하는 모든 MQTT 주제는 원격 측정 주제로 간주됩니다.

디바이스 데이터 원격 측정의 캡처 및 상호 작용에 대한 몇 가지 시나리오를 논리적 관점에서 정의해 보았습니다.

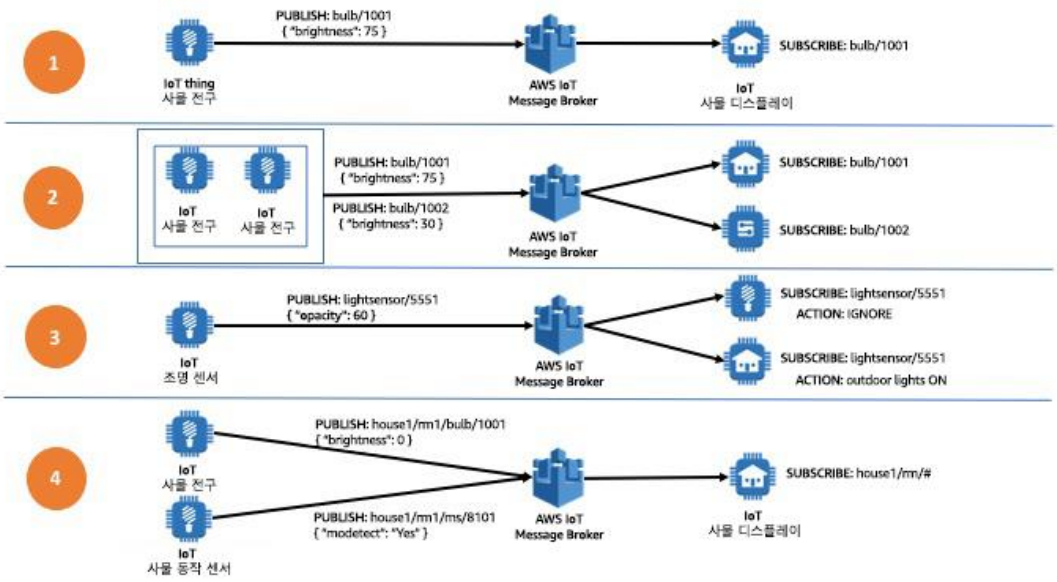

그림 2: 원격 측정 캡처 옵션

1. 게시 주제 하나와 구독자 하나. 예: 단일 애플리케이션만 구독할 수 있는 단일 주제에 조도를 게시하는 스마트 전구 1 개.
2. 변수가 포함된 게시 주제 하나와 구독자 하나. 예: 유사하지만 고유한 주제에 조도를 게시하는 스마트 전구의 모음. 각 구독자는 고유한 게시 메시지를 수신할 수 있습니다.
3. 단일의 게시 주제와 다수의 구독자. 예: 집안의 모든 전구가 구독하는 단일 주제에 값을 게시하는 조명 센서 1 개.
4. 다수의 게시 주제와 단일 구독자. 예: 동작 센서가 있는 전구의 모음. 스마트 홈 시스템은 모든 전구 주제(동작 센서 포함)를 구독하고 조도 및 동작 센서 데이터에 대한 종합적인 보기를 생성합니다.

디바이스 명령

IoT 애플리케이션을 구축할 때는 명령을 통해 원격으로 디바이스와 상호 작용할 수 있는 기능이 필요합니다. 산업 분야에서 한 가지 예는 원격 명령을 사용하여 장비의 특정 데이터를 요청하는 것입니다. 스마트 홈 분야의 사용 사례 중 하나는 원격 명령을 사용하여 경보 시스템을 원격으로 예약하는 것입니다.

AWS IoT Core 를 사용하면 MQTT 주제 또는 AWS IoT 디바이스 섀도우로 명령을 구현하여 디바이스에 명령을 보내고 디바이스가 명령을 실행한 후 승인을 받을 수 있습니다. 명령을 구현할 때는 MQTT 주제 대신 디바이스 섀도우를 사용하십시오. 디바이스 섀도우는 표준 MQTT 주제(예: clientToken)와 달리 요청의 오리진 및 충돌 해결 관리를 위한 버전 번호를 추적하는 기능과 오프라인 상태의 디바이스가 실행된 명령을 수신할 수 없는 경우 클라우드에 명령을 저장하는 기능을 제공한다는 이점이 있습니다. 디바이스의 섀도우는 디바이스가 현재 온라인이 아닌 경우 클라우드에 명령을 유지해야 하는 경우 주로 사용됩니다. 온라인 상태가 된 디바이스는 최신 섀도우 정보를 요청하고 명령을 실행합니다.

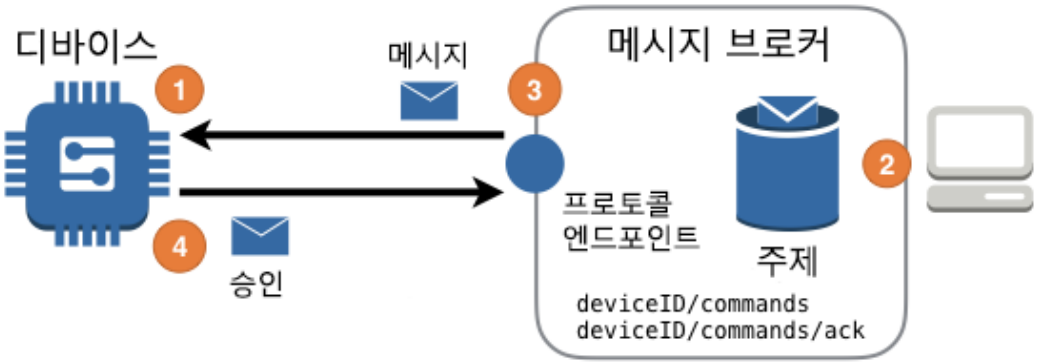

그림 3: 메시지 브로커를 사용하여 디바이스에 명령 전송

AWS IoT 디바이스 섀도우 서비스

AWS IoT Core 의 디바이스 섀도우 서비스를 사용하는 IoT 솔루션에서는 안정적이고 확장 가능하며 간단한 방법으로 명령 요청을 관리할 수 있습니다. 디바이스 섀도우 서비스는 처방적 접근 방식에 따라 디바이스 관련 상태와 상태 변경이 전달되는 방법을 관리합니다. 디바이스 섀도우 서비스는 이 접근 방식에 설명된 대로 JSON 문서를 사용하여 디바이스의 현재 상태, 원하는 향후 상태 및 현재 상태와 원하는 상태의 차이점을 저장합니다.

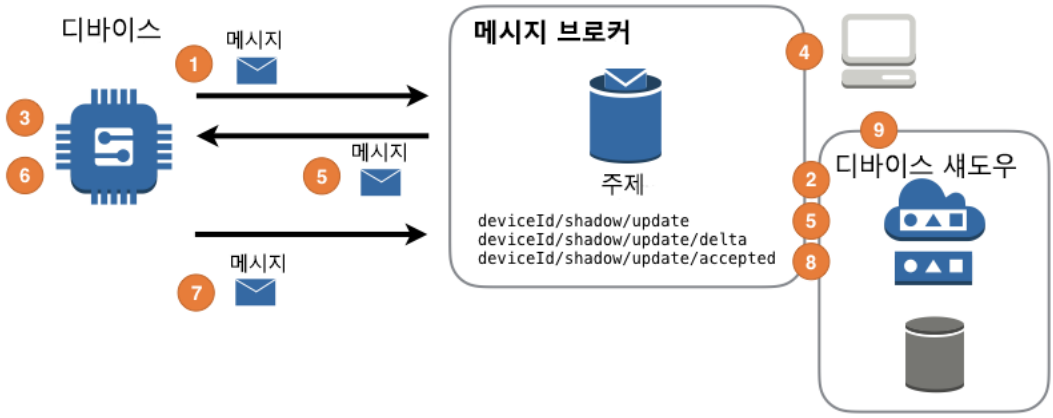

그림 4: 디바이스에서 디바이스 섀도우 사용

1. 디바이스가 디바이스의 초기 상태를 메시지 형태로 업데이트 주제 `deviceID/shadow/update` 에 게시하여 해당 상태를 보고합니다.
2. 디바이스 섀도우는 이 주제에서 메시지를 읽고 디바이스 상태를 영구 데이터 스토어에 기록합니다.
3. 디바이스가 델타 메시징 주제 `deviceId/shadow/update/delta` 를 구독합니다. 이 주제는 디바이스 관련 상태 변경 메시지를 수신합니다.
4. 솔루션 구성 요소가 `deviceID/shadow/update` 주제에 원하는 상태 메시지를 게시하고 이 디바이스를 추적하는 디바이스 섀도우가 원하는 디바이스 상태를 영구 데이터 스토어에 기록합니다.
5. 디바이스 섀도우가 델타 메시지를 `deviceId/shadow/update/delta` 에 게시하면 메시지 브로커가 이 메시지를 디바이스로 전송합니다.
6. 디바이스가 델타 메시지를 수신하고 원하는 상태 변경을 수행합니다.
7. 디바이스가 업데이트 주제 `deviceID/shadow/update` 에 새 상태를 반영한 승인 메시지를 게시하고 이 디바이스를 추적하는 디바이스 섀도우가 영구 데이터 스토어에 새 상태를 기록합니다.
8. 디바이스 섀도우가 `deviceId/shadow/update/accepted` 주제에 메시지를 게시합니다.
9. 이제 솔루션 구성 요소는 디바이스 섀도우의 업데이트된 상태를 요청할 수 있습니다.

펌웨어 업데이트

모든 IoT 솔루션은 디바이스 펌웨어 업데이트를 허용해야 합니다. 인적 개입이 없는 펌웨어 업그레이드 지원은 보안, 확장성 및 새로운 기능의 제공에 있어서 중요한 역할을 합니다.

AWS IoT Device Management 는 펌웨어 업데이트를 실행하고 상태를 추적하는 등 IoT 배포를 안전하고 쉽게 관리할 수 있는 방법을 제공합니다. AWS IoT Device Management 는 MQTT 프로토콜을 AWS IoT 메시지 브로커 및 AWS IoT 작업과 함께 사용하여 펌웨어 업데이트 명령을 디바이스로 전송하고 펌웨어 업데이트의 상태를 지속적으로 수신합니다.

IoT 솔루션에서 이 기능을 제공하려면 다음 다이어그램에 표시된 AWS IoT Jobs 를 사용하여 펌웨어 업데이트를 구현해야 합니다.

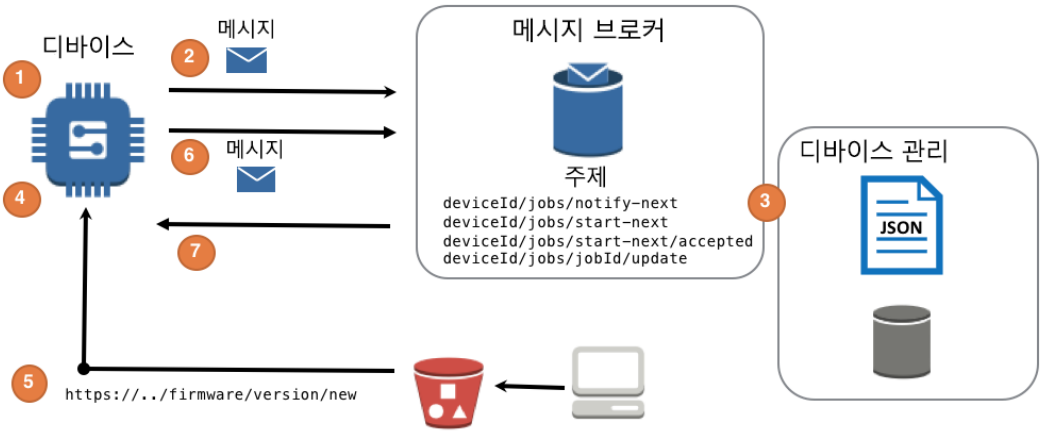

그림 5: 디바이스의 펌웨어 업데이트

1. 디바이스가 IoT 작업 알림 주제 `deviceId/jobs/notify-next` 를 구독합니다. 이 주제는 IoT 작업 알림 메시지를 수신합니다.
2. 디바이스가 `deviceId/jobs/start-next` 에 메시지를 게시하여 다음 작업을 시작하고 다음 작업, 작업 문서 및 statusDetails 에 저장된 모든 상태를 포함한 기타 세부 정보를 가져옵니다.
3. AWS IoT Jobs 서비스는 특정 디바이스에 대한 다음 작업 문서를 검색하고 구독 주제 `deviceId/jobs/start-next/accepted` 에서 이 문서를 전송합니다.

4. 디바이스가 `deviceId/jobs/jobId/update MQTT` 주제를 사용하여 작업 문서에 지정된 작업을 수행하고 작업의 진행률을 보고합니다.
5. 업그레이드 프로세스 중에 디바이스는 Amazon S3 의 미리 서명된 URL 을 사용하여 펌웨어를 다운로드합니다. Amazon S3 에 펌웨어를 업로드하는 경우 코드 서명을 사용하여 펌웨어를 서명하십시오. 펌웨어를 코드로 서명하면 최종 디바이스에서 펌웨어를 설치하기 전에 해당 펌웨어의 신뢰성을 확인할 수 있습니다. Amazon FreeRTOS 디바이스는 MQTT 를 통해 직접 펌웨어 이미지를 다운로드할 수 있으므로 별도의 HTTPS 연결이 필요하지 않습니다.
6. 디바이스가 작업 주제 `deviceId/jobs/jobId/update` 에 업데이트 상태 메시지를 게시하여 성공 또는 실패를 보고합니다.
7. 이 작업의 실행 상태가 최종 상태로 변경되었으므로 실행할 수 있는 다음 IoT 작업(있는 경우)이 변경됩니다.

Well-Architected 프레임워크의 기반

이 섹션에는 각 기반에 대한 설명과 AWS IoT 용 솔루션의 설계와 관련된 정의, 모범 사례, 질문, 고려 사항 및 필수 AWS 서비스가 포함되어 있습니다.

운영 우수성 기반

운영 우수성 기반에는 프로덕션 워크로드의 관리에 사용되는 운영 사례 및 절차가 포함됩니다. 운영 우수성은 계획된 변경을 실행하는 방법과 예기치 않은 운영 이벤트에 대응하는 방법으로 형성됩니다. 변경 실행 및 대응은 자동화되어야 합니다. 운영 우수성을 위한 모든 프로세스 및 절차는 문서화되고 테스트되어야 하며 정기적으로 검토되어야 합니다.

설계 원칙

전반적인 Well-Architected 프레임워크 운영 우수성 설계 원칙에 더해, 클라우드에서 IoT 의 운영 우수성을 달성하기 위한 5 가지 설계 원칙은 다음과 같습니다.

- **디바이스 프로비저닝 계획**: 안전한 위치에서 초기 디바이스 자격 증명을 생성하도록 디바이스 프로비저닝 프로세스를 설계합니다. 고유한 인증서를 IoT 디바이스에 배포하는 작업을 담당하는 PKI(퍼블릭 키 인프라)를 구현합니다. 앞서 설명한 것과 같이 미리 생성된 프라이빗 키 및 인증서가 있는 암호화 하드웨어를 선택하면 PKI 를 실행하는 데 드는 운영 비용이 제거됩니다. 그렇지 않은 경우 제조 프로세스 또는 디바이스 부트스트래핑 중에 HSM(하드웨어 보안 모듈)을 사용하여 오프라인으로 PKI 를 구현할 수 있습니다. 클라우드에서 CA(인증 기관) 및 HSM 을 관리할 수 있는 기술을 사용하십시오.
- **디바이스 부트스트래핑 구현**: 기술자(산업 분야) 또는 사용자(소비자 분야)에 의한 개인화를 지원하는 디바이스를 프로비저닝할 수도 있습니다. Bluetooth LE 를 통해 디바이스와 상호 작용하고 Wi-Fi 를 통해 클라우드와 상호 작용하는 스마트폰 애플리케이션을 예로 들 수 있습니다. 디바이스에서 전역에 분산된 부트스트랩 API 를 사용하여 디바이스 구성 정보를 프로그래밍 방식으로 업데이트할 수 있는 기능을 설계해야 합니다. 부트스트래핑 설계에서는 디바이스의 새 구성 설정을 클라우드를 통해 프로그래밍 방식으로 전송할 수 있습니다. 이러한 변경 사항에는 통신할 IoT 엔드포인트, 디바이스의 전체 상태를 전송할 빈도, 업데이트된 보안 설정(예: 서버 인증서)과 같은 설정이 포함되어야 합니다. 부트스트래핑 프로세스는 초기 프로비저닝을 넘어 디바이스 운영에 있어서도 중요한 역할을 합니다. 클라우드를 통해 프로그래밍 방식으로 디바이스 구성을 업데이트할 수 있기 때문입니다.
- **디바이스 통신 패턴 문서화**: IoT 애플리케이션에서 디바이스 동작은 하드웨어 수준에서 수동으로 문서화됩니다. 클라우드에서는 운영 팀이 디바이스 플릿에 배포된 디바이스의 동작을 확장하는 방법을 공식화해야 합니다. 클라우드 엔지니어는 디바이스 통신 패턴을 검토하고 디바이스 데이터의 총 예상 인바운드 및 아웃바운드 트래픽을 추정하여 클라우드에서 전체 디바이스 플릿을 지원하는 데 필요한 예상 인프라를 결정해야 합니다. 그런 다음 운영 계획 중에 디바이스 및 클라우드 측 지표로 이러한 패턴을 측정하여 시스템에서 예상되는 사용 패턴이 충족되는지 확인해야 합니다.
- **OTA(Over The Air) 업데이트 구현**: 하드웨어의 장기 투자에서 이익을 실현하려면 디바이스의 펌웨어를 새 기능으로 지속적으로 업데이트할 수 있어야 합니다. 클라우드에서는 펌웨어 업데이트의 대상 디바이스를 지정하고, 변경 사항을 지속적으로 롤아웃하고, 업데이트의 성공 및 실패를 추적하며, KPI 에 따라 펌웨어 변경을 롤백하거나 중지할 수 있는 기능이 있는 강력한 펌웨어 업데이트 프로세스를 적용할 수 있습니다.

- **물리적 자산에 대한 기능 테스트 구현**: IoT 디바이스 하드웨어 및 펌웨어는 현장 배포 전에 철저한 테스트를 거쳐야 합니다. 수용 및 기능 테스트는 프로덕션으로 전환하는 과정에서 매우 중요합니다. 기능 테스트의 목표는 하드웨어 구성 요소, 임베디드 펌웨어 및 디바이스 애플리케이션 소프트웨어를 엄격한 테스트 시나리오(예: 간헐적 연결 또는 연결 속도 저하 또는 주변 센서 오류)에서 실행하면서 하드웨어의 성능을 프로파일링하는 것입니다. 이 테스트를 수행하면 배포 후에 IoT 디바이스가 예상 성능을 제공할지 여부를 확인할 수 있습니다.

정의

클라우드의 운영 우수성에는 3 가지 모범 사례 영역이 있습니다.

1. 준비
2. 운영
3. 개선

프로세스, 런북 및 게임데이와 관련하여 Well-Architected 프레임워크에서 다룬 내용에 더해 IoT 애플리케이션의 운영 우수성을 촉진하기 위해 검토해야 할 특정 영역이 몇 가지 더 있습니다.

모범 사례

준비

IoT 애플리케이션의 경우 다양한 환경에서 하드웨어를 조달하고 프로비저닝하고 테스트하고 배포해야 합니다. 따라서 운영 우수성을 준비할 때는 그 범위를 확장하여 클라우드에서 실행되지 않고 물리적 디바이스에서 주로 실행될 배포의 측면까지 다루어야 합니다. 비즈니스 결과를 측정하고 개선하는 데 사용할 운영 지표를 정의한 다음 디바이스에서 이러한 지표를 생성하여 IoT 애플리케이션으로 전송할지 여부를 결정하십시오. 또한 다양한 환경에서 디바이스의 동작을 시뮬레이션할 수 있는 간소화된 기능 테스트 프로세스를 만들어 운영 우수성을 계획해야 합니다.

오류 발생 시 IoT 워크로드의 복원력을 유지하는 방법, 문제가 있을 때 인적 개입 없이 디바이스를 자가 복구하는 방법 및 연결된 하드웨어의 로드 증가 요구 사항을 충족하기 위해 클라우드 기반 IoT 애플리케이션을 확장하는 방법에 대한 답을 찾는 것이 필수적입니다.

IoT 플랫폼을 사용하는 경우 추가 구성 요소/도구를 사용하여 IoT 작업을 처리할 수 있습니다. 이러한 도구에는 디바이스 동작의 모니터링 및 검사, 연결 지표 캡처, 고유한 자격 증명을 사용한 디바이스 프로비저닝 및 디바이스 데이터 기반 장기 분석 기능을 제공하는 서비스가 포함됩니다.

IOTOPS 1. 무엇을 기준으로 운영 우선 순위를 결정합니까?

IOTOPS 2. 디바이스에서 IoT 워크로드의 작업을 지원할 준비가 되었는지 어떻게 확인합니까?

IOTOPS 3. 새로 프로비저닝된 디바이스가 필수적인 운영 사전 조건을 충족하는지 확인하려면 어떻게 합니까?

IoT 와 데이터 센터의 논리적 보안은 주로 머신 간 인증과 관련된다는 점에서 유사합니다. 그러나 이 둘은 IoT 디바이스가 물리적으로 안전하다고 가정할 수 없는 환경에 자주 배포된다는 점에서 다릅니다. 또한 일반적으로 IoT 애플리케이션은 중요한 데이터를 인터넷을 통해 전송해야 합니다. 이러한 점을 고려하면, 디바이스 자격 증명을 안전하게 받고 디바이스 자격 증명을 지속적으로 입증하며 적절한 수준의 메타데이터를 디바이스에 제공하고 모니터링을 위해 디바이스를 구성 및 분류하며 올바른 권한 세트로 디바이스를 활성화할 방법을 결정하는 아키텍처를 마련하는 것이 중요합니다.

IoT 애플리케이션의 성공 및 확장성을 보장하려면 이전, 현재 및 예상 디바이스 동작에 대한 데이터 기반의 자동화된 관리 프로세스가 필요합니다. IoT 애플리케이션은 증분 롤아웃 및 롤백 전략을 지원해야 합니다. 이러한 점을 고려하여 운영 효율성을 계획하면 내결함성이 우수하고 효율성이 뛰어난 IoT 애플리케이션을 시작할 수 있습니다.

AWS IoT 에서는 다수의 기능을 사용하여 CA 가 서명한 개별 디바이스 자격 증명을 클라우드에 프로비저닝할 수 있습니다. 이 경로를 사용하는 경우 디바이스에 자격 증명을 프로비저닝한 다음 JITP(적시 프로비저닝), JITR(적시 등록) 또는 BYOC(자체 인증서 사용)를 사용하여 디바이스 인증서를 클라우드에 안전하게 등록할 수 있습니다. Route 53, Amazon API Gateway, Lambda 및 DynamoDB 같은 AWS 서비스를 사용하면 단순한 API 인터페이스에서 디바이스 부트스트래핑을 통해 프로비저닝 프로세스를 확장할 수 있습니다.

운영

IoT 에서 운영 상태는 클라우드 애플리케이션의 운영 상태를 넘어 애플리케이션에 포함되지만 로컬로 문제 해결을 수행하기가 어렵거나 불가능할 수 있는 위치에 원격으로 배포되는 디바이스를 측정, 모니터링, 문제 해결 및 해결하는 기능으로 확대됩니다. 이러한 원격 디바이스에서 전송된 지표를 검사, 분석 및 실행할 수 있으려면 설계 및 구현 시점에 이 원격 작업에 대한 요구 사항을 고려해야 합니다.

IoT 에서는 디바이스 동작의 올바른 기준 지표를 설정해야 하고, 여러 디바이스에서 발생하는 문제를 집계 및 추론할 수 있어야 하며, 디바이스 펌웨어의 일부로 클라우드에서 실행되는 강력한 수정 계획을 마련해야 합니다. 프로덕션 시스템을 기준으로 일반적인 디바이스 상호 작용을 직접 테스트하는 다양한 디바이스 시뮬레이션 카나리아를 구현해야 합니다. 디바이스 카나리아는 운영 지표가 충족되지 않는 경우 조사해야 하는 잠재적 영역을 좁히는 데 도움이 됩니다. 디바이스 카나리아를 사용하면 카나리아 지표가 예상 SLA 아래로 떨어질 때 예방적 경보를 생성할 수 있습니다.

AWS 에서는 AWS IoT Core 의 디바이스 레지스트리에 있는 각 물리적 디바이스에 대해 AWS IoT 사물을 생성할 수 있습니다. 레지스트리에 사물을 생성하면 메타데이터를 디바이스에 연결하고, 디바이스를 그룹화하고, 디바이스에 대한 보안 권한을 구성할 수 있습니다. 사물 레지스트리의 정적 데이터를 저장할 때는 AWS IoT 사물을 사용해야 하지만 동적 디바이스 데이터는 사물의 연결된 디바이스 섀도우에 저장됩니다. 디바이스 섀도우는 디바이스의 상태 정보를 저장 및 검색하는 데 사용되는 JSON 문서입니다.

디바이스 레지스트리에 디바이스의 가상 표현을 생성하는 것에 더불어 운영 프로세스의 일부로 IoT 디바이스를 정의하는 유사한 정적 속성을 캡슐화하는 사물 유형을 생성해야 합니다. 사물 유형은 디바이스의 제품 분류와 유사합니다. IoT 작업에 사용할 중요 메타데이터의 저장을 위한 첫 시작점으로 사물, 사물 유형 및 디바이스 섀도우를 조합하여 사용할 수 있습니다.

AWS IoT 에서는 사물 그룹을 사용하여 디바이스를 범주별로 관리할 수 있습니다. 그룹에 다른 그룹을 포함하여 계층 구조를 구축할 수도 있습니다. IoT 애플리케이션의 조직 구조를 사용하면 디바이스 그룹별로 관련 디바이스를 신속하게 식별하고 작업을 수행할 수 있습니다. 클라우드를 활용하면 비즈니스 로직과 디바이스의 수명 주기를 기반으로 그룹에서 디바이스를 추가하거나 제거하는 작업을 자동화할 수 있습니다.

IoT 에서 디바이스가 생성하는 원격 측정 또는 진단 메시지는 레지스트리 또는 디바이스 섀도우에 저장되지 않습니다. 대신 이러한 메시지는 여러 MQTT 주제를 사용하여 AWS IoT 로 전송됩니다. 이 데이터를 실행 가능한 데이터로 만들려면 AWS IoT 규칙 엔진을 사용하여 오류 메시지를 자동화된 수정 프로세스로 라우팅하고 진단 정보를 IoT 메시지에 추가하십시오. 다음은 오류 상태 코드가 포함된 메시지를 사용자 지정 워크플로로 라우팅할 때 사용할 수 있는 방법의 예입니다. 규칙 엔진은 메시지의 상태를 검사하고 오류인 경우 오류 메시지 세부 정보 페이로드를 기반으로 디바이스를 수정하는 Step Function 워크플로를 시작합니다.

```
{
    "sql": "SELECT * FROM 'command/iot/response WHERE code =
'eror'",
    "ruleDisabled": false,
    "description": "Error Handling Workflow",
    "awsIotSqlVersion": "2016-03-23",
    "actions": [{
        "stepFunctions": {
            "executionNamePrefix": "errorExecution",
            "stateMachineName": "errorStateMachine",
            "roleArn":
"arn:aws:iam::123456789012:role/aws_iot_step_functions"
        }
    }]
}
```

클라우드 애플리케이션에 대한 운영 통찰력을 지원하려면 AWS IoT Core 의 디바이스 브로커에서 수집된 모든 지표에 대한 대시보드를 생성합니다. 이러한 지표는 CloudWatch Metrics 를 통해 제공됩니다. 또한 CloudWatch Logs 에는 성공한 인바운드 메시지 합계, 아웃바운드 메시지, 연결 성공, 오류 등의 정보가 포함됩니다.

프로덕션 디바이스 배포를 보강하려면 여러 AWS 리전에 있는 디바이스 카나리아 형태의 IoT 시뮬레이션을 Amazon Elastic Compute Cloud(Amazon EC2)에 구현하십시오. 이러한 디바이스 카나리아는 장기 실행 트랜잭션과 같은 오류 조건 시뮬레이션, 원격 측정 전송, 제어 작업 구현 등 여러 비즈니스 사용 사례를 미러링합니다. 디바이스 시뮬레이션 프레임워크는 성공, 오류, 지연 시간 및 디바이스 순서를 포함하되 이에 국한되지 않는 광범위한 지표를 출력한 다음 모든 지표를 운영 체제로 전송해야 합니다.

AWS IoT 에서는 사용자 지정 대시보드에 더해 AWS IoT 플릿 인덱싱 같은 검색 기능을 통해 사물 레지스트리 및 디바이스 섀도우 서비스가 제공하는 플릿 수준 및 디바이스 수준 정보를 활용할 수도 있습니다. 플릿 전체를 검색할 수 있으므로 IoT 문제가 디바이스 수준에서 발생하든 플릿 전체 수준에서 발생하든 문제를 진단하는 운영 오버헤드가 완화됩니다.

개선

IOTOPS 4. 다운스트림 IoT 디바이스에 미치는 영향을 최소화하면서 IoT 애플리케이션을 개선하는 방법은 무엇입니까?

IoT 솔루션에는 저출력 디바이스, 원격 위치, 낮은 대역폭 및 간헐적 네트워크 연결의 조합이 수반되는 경우가 많습니다. 이러한 각 요인은 펌웨어 업그레이드를 비롯한 통신 문제의 원인이 됩니다. 따라서 다운스트림 디바이스 및 작업에 미치는 영향을 최소화하는 IoT 업데이트 프로세스를 포함하고 구현하는 것이 중요합니다. 디바이스는 다운스트림 영향을 줄이는 기능에 더해 간헐적인 네트워크 연결 및 정전 등 로컬 환경에 존재하는 일반적인 문제에 대한 복원력을 갖추어야 합니다. 따라서 IoT 디바이스를 그룹화하여 배포하고 일정 기간에 걸쳐 펌웨어 업그레이드를 수행하는 것이 좋습니다. 현장에서 디바이스가 업데이트될 때 디바이스의 동작을 모니터링하고 일정 비율의 디바이스가 성공적으로 업그레이드된 후에만 계속해서 업데이트를 진행하십시오.

AWS IoT Device Management 를 사용하여 디바이스 배포 그룹을 생성하고 OTA(Over The Air)를 통해 특정 디바이스 그룹으로 전송합니다. 업그레이드 중에 모든 CloudWatch Logs, 원격 측정 및 IoT 디바이스 작업 메시지의 수집을 계속하고 이 정보를 전체 애플리케이션 상태 및 장기 실행 카나리아 성능을 측정하는 데 사용되는 KPI 와 결합합니다.

펌웨어 업데이트 전후에 비즈니스 참가자와 함께 운영 지표에 대한 소급 분석을 수행하여 개선 기회와 방법을 결정합니다. AWS IoT Analytics 및 AWS IoT Device Defender 같은 서비스를 사용하여 전체 디바이스 동작의 이상을 추적하고 업데이트된 펌웨어의 문제를 나타낼 수 있는 성능 편차를 측정할 수 있습니다.

주요 AWS 서비스

여러 서비스를 사용하여 IoT 애플리케이션의 운영 우수성을 달성할 수 있습니다. AWS Device Qualification Program 은 AWS IoT 상호 운용성을 위해 설계되고 테스트된 하드웨어 구성 요소를 선택하는 데 도움이 됩니다. 적격 하드웨어를 사용하면 제품 출시를 앞당기고 운영 마찰을 줄일 수 있습니다. AWS IoT Core 는 디바이스의 초기 온보딩을 관리하는 데 사용되는 기능을 제공합니다. AWS IoT Device Management 는 디바이스 그룹화 및 검색 등 플릿 전체에서 수행되는 작업의 운영 오버헤드를 줄여줍니다. 또한 Amazon CloudWatch 는 IoT 지표 모니터링, 로그 수집, 알림 생성 및 응답 트리거에 사용됩니다. 운영 우수성의 3 가지 영역을 지원하는 기타 서비스 및 기능은 다음과 같습니다.

- **준비**: **AWS IoT Core** 는 적시 프로비저닝, 적시 등록 또는 자체 인증서 사용을 통해 디바이스 자격 증명을 등록하는 것을 포함하여 현장에서의 디바이스 프로비저닝 및 온보딩을 지원합니다. 그런 다음에는 디바이스 레지스트리 및 디바이스 섀도우를 사용하여 디바이스를 메타데이터 및 디바이스 상태에 연결할 수 있습니다.
- **작업**: **AWS IoT 사물 그룹 및 플릿 인덱싱**을 사용하여 디바이스의 조직 구조를 신속하게 개발하고 디바이스의 현재 메타데이터 전체를 검색하여 반복되는 디바이스 작업을 수행할 수 있습니다. Amazon CloudWatch 를 사용하면 디바이스 및 애플리케이션의 운영 상태를 모니터링할 수 있습니다.
- **응답**: **AWS IoT Jobs** 를 사용하여 펌웨어 업데이트 또는 디바이스 구성과 같은 업데이트를 하나 이상의 디바이스에 사전에 푸시할 수 있습니다. AWS IoT 규칙 엔진을 사용하면 AWS IoT Core 에 수신되는 IoT 메시지를 검사하고 가장 세분화된 수준에서 즉시 데이터에 응답할 수 있습니다. AWS IoT Analytics 및 AWS IoT Device Defender 를 사용하면 AWS IoT Analytics 를 사용한 실시간 분석과 Device Defender 를 사용한 실시간 보안 및 데이터 임계값을 기반으로 알림 또는 수정을 사전 예방적으로 트리거할 수 있습니다.

보안 기반

보안 기반에는 정보, 시스템, 자산을 보호하는 동시에 비즈니스 가치를 제공하는 기능이 포함됩니다.

설계 원칙

전반적인 Well-Architected 프레임워크의 보안 설계 원칙에 더해 IoT 보안과 관련된 특정 설계 원칙이 있습니다.

- **디바이스 보안 수명 주기를 전체적으로 관리**: 데이터 보안은 설계 단계에서 시작되어 하드웨어 및 데이터의 사용 중단 및 폐기에서 끝납니다. 경쟁 우위를 유지하고 고객 신뢰를 지속하려면 전체적인 방식으로 IoT 솔루션의 보안 수명 주기에 접근하는 것이 중요합니다.
- **최소 권한 사용**: 디바이스에는 디바이스 통신에 사용할 수 있는 주제를 제한하는 세분화된 액세스 권한이 적용되어야 합니다. 액세스를 제한하면 손상된 디바이스 하나가 다른 디바이스에 영향을 미칠 가능성이 줄어듭니다.
- **저장된 디바이스 자격 증명 보호**: 디바이스는 전용 암호화 요소 또는 보안 플래시와 같은 메커니즘을 사용하여 저장된 자격 증명 정보를 안전하게 저장해야 합니다.
- **디바이스 자격 증명 수명 주기 관리 구현**: 디바이스 자격 증명은 생성된 시기부터 수명이 종료될 때까지 디바이스에 유지됩니다. 제대로 설계된 자격 증명 시스템은 디바이스 자격 증명을 추적하고 자격 증명의 유효성을 추적하며 시간 경과에 따라 IoT 권한을 사전에 연장하거나 해지합니다.
- **데이터 보안을 전체적으로 파악**: IoT 배포에는 많은 수의 원격 배포 디바이스가 포함되기 때문에 데이터 절도 및 개인 정보 침해에 대한 공격 표면이 매우 넓습니다. Open Trusted Technology Provider 표준과 같은 모델을 사용하여 공급망 및 솔루션 설계의 위험 요소를 체계적으로 검토한 다음 적절한 완화 조치를 적용하십시오.

정의

클라우드의 보안에는 5 가지 모범 사례 영역이 있습니다.

1. IAM(자격 증명 및 액세스 관리)
2. 탐지 제어
3. 인프라 보호

4. 데이터 보호
5. 인시던트 대응

인프라 및 데이터 보호에는 IoT 디바이스 하드웨어와 엔드투엔드 솔루션이 포함됩니다. IoT 구현에서는 기존의 보안 모델을 확장하여 디바이스에 하드웨어 보안 모범 사례를 구현하고 IoT 애플리케이션에서 적절한 범위가 지정된 디바이스 권한 및 탐지 제어 같은 요인에 대한 보안 모범 사례를 준수하도록 해야 합니다.

보안 기반은 정보 및 시스템을 보호하는 데 중점을 둡니다. 주요 주제로는 데이터의 기밀성 및 무결성, 권한 관리를 통한 사용자 권한 식별 및 관리, 시스템 보호, 보안 이벤트 탐지를 위한 제어 기능 설정이 있습니다.

모범 사례

IAM(자격 증명 및 액세스 관리)
IoT 디바이스는 신뢰할 수 있는 자격 증명으로 프로비저닝되고, 전략적 고객 또는 비즈니스 데이터(예: 펌웨어 자체)를 저장하거나 액세스할 수 있으며, 인터넷을 통해 원격으로 액세스할 수 있고, 직접적인 물리적 변조에 취약할 수 있기 때문에 표적이 되는 경우가 많습니다. 무단 액세스를 방지하려면 디바이스 수준에서 보안을 구현하는 것부터 시작해야 합니다. 하드웨어 측면에서는 다음과 같은 여러 메커니즘을 구현하여 디바이스의 중요한 정보를 임의 변경하는 공격 표면을 줄일 수 있습니다.

- 하드웨어 암호화 모듈
- 소프트웨어 지원 솔루션(예: 보안 플래시)
- 복제할 수 없는 물리적 함수 모듈
- 최신 암호화 라이브러리 및 표준(예: PKCS #11 및 TLS 1.2)

디바이스 하드웨어를 보호하려면 프라이빗 키와 중요한 자격 증명을 디바이스의 안전한 하드웨어 위치에만 저장하고 각 프라이빗 키 및 중요한 자격 증명에 고유한 솔루션을 구현해야 합니다. AWS IoT 와의 통신에 사용되는 프라이빗 키에 대한 액세스를 안전하게 저장하고 관리하는 하드웨어 또는 소프트웨어 기반 모듈을 구현합니다. 하드웨어 보안에 더해 IoT

애플리케이션의 인증 및 권한 부여에 사용할 유효한 자격 증명을 IoT 디바이스에 부여해야 합니다.

디바이스의 수명 주기 동안 인증서 갱신 및 해지를 관리할 수 있어야 합니다. 디바이스의 인증서 정보 변경을 처리하려면 먼저 현장에서 디바이스를 업데이트할 수 있어야 합니다. 따라서 제대로 설계된 IoT 애플리케이션에는 하드웨어의 펌웨어 업데이트를 수행하는 기능이 필수적입니다. OTA 업데이트를 통해 인증 기관을 포함하여 디바이스 인증서가 만료되기 전에 해당 인증서를 안전하게 교체하십시오.

IOTSEC 1. 디바이스의 디바이스 인증서와 프라이빗 키를 안전하게 저장할 수 있습니까?

IOTSEC 2. AWS IoT 자격 증명을 어떻게 디바이스에 연결합니까?

예를 들어 AWS IoT 를 사용하는 경우 먼저 X.509 인증서를 프로비저닝한 다음 IoT 연결, 메시지 게시 및 구독, 업데이트 수신을 위한 IoT 권한을 별도로 생성합니다. 이러한 자격 증명과 권한을 분리하면 디바이스 보안을 유연하게 관리할 수 있습니다. 권한을 구성하는 동안 각 디바이스의 MQTT 작업에 대한 액세스를 제한하는 IoT 정책을 생성하여 모든 디바이스에 올바른 수준의 자격 증명을 제공하고 올바른 수준의 액세스 제어가 시행될 수 있도록 합니다.

각 디바이스에는 AWS IoT 에 고유한 X.509 인증서가 있어야 하며 디바이스는 인증서를 공유해서는 안 됩니다(디바이스 1 개당 인증서 1 개 규칙). AWS IoT 를 사용할 때는 디바이스당 1 개의 인증서를 사용하는 것에 더해 각 디바이스의 고유한 사물이 IoT 레지스트리에 있어야 합니다. 사물 이름은 MQTT 연결을 위한 MQTT ClientID 의 기준으로 사용됩니다.

단일 인증서가 AWS IoT Core 의 자체 사물과 페어링되는 이 연결을 생성하면 손상된 인증서가 의도치 않게 다른 디바이스의 자격 증명을 가장하는 상황을 방지할 수 있습니다. 또한 MQTT ClientID 와 사물 이름이 일치하는 경우 모든 ClientID 로그 메시지를 특정 통신에 연결된 사물과 관련지을 수 있으므로 문제 해결 및 수정이 간소화됩니다.

디바이스 자격 증명 업데이트를 지원하려면 디바이스에 OTA 통신 및 바이너리를 배포하는 관리형 플랫폼인 AWS IoT Jobs 를 사용하십시오. AWS IoT Jobs 는 AWS IoT 에 연결된 하나 이상의 디바이스로 전송되어 실행되는 원격 작업 세트를 정의하는 데 사용됩니다. AWS IoT

Jobs 는 기본적으로 상호 인증 및 권한 부여, 업데이트 진행 상황에 대한 디바이스 추적, 지정된 업데이트에 대한 플릿 전체 지표 등 여러 모범 사례를 통합합니다.

AWS IoT Device Defender 감사를 활성화하여 디바이스 구성, 디바이스 정책 및 만료 인증서 확인을 자동화된 방식으로 추적하십시오. 예를 들어 Device Defender 를 사용하여 예약된 날짜에 감사를 시행하고 인증서 만료에 대한 알림을 트리거할 수 있습니다. 취소된 인증서 또는 보류 중인 만료 인증서에 대한 알림을 함께 수신하면 인증서를 사전에 교체할 수 있는 OTA 를 자동으로 예약할 수 있습니다.

IOTSEC 3. IoT 애플리케이션에 대한 사용자 액세스 권한을 어떻게 부여하고 인증합니까?

IoT 의 사물 측면에 초점을 두는 애플리케이션이 많지만 IoT 의 거의 모든 산업 분야에는 디바이스와 통신하고 디바이스의 알림을 수신해야 하는 인적 구성 요소도 있습니다. 예를 들어 소비자 IoT 의 경우 일반적으로 사용자가 디바이스를 온라인 계정에 연결하여 온보딩해야 합니다. 산업용 IoT 에는 일반적으로 하드웨어 원격 측정을 거의 실시간으로 분석할 수 있는 기능이 수반됩니다. 두 경우 모두 특정 디바이스와 상호 작용해야 하는 사용자를 식별하고 인증하고 필요한 권한을 부여할 때 애플리케이션에서 사용할 방법을 결정하는 것이 필수적입니다.

IoT 자산에 대한 사용자 액세스를 제어하는 작업은 자격 증명에서 시작됩니다. IoT 애플리케이션에는 사용자의 자격 증명을 추적하고 사용자가 이 자격 증명을 사용하여 인증하는 방법을 추적하는 스토어(일반적으로 데이터베이스)가 있어야 합니다. 자격 증명 스토어에는 권한 부여 시점에 사용될 수 있는 추가 사용자 속성(예: 사용자 그룹 멤버십)이 포함될 수 있습니다.

IoT 디바이스 원격 측정 데이터는 보안을 적용할 수 있는 자산의 한 예입니다. 이러한 방법을 사용하면 각 사용자의 액세스 권한을 제어하고 개별 사용자 상호 작용에 대한 감사를 시행할 수 있습니다.

AWS 를 사용하여 IoT 애플리케이션 사용자에 대한 인증 및 권한 부여를 수행하는 경우 다수의 옵션을 사용하여 자격 증명 저장소를 구현하고 이 저장소에 사용자 속성을 유지하는 방법을 구성할 수 있습니다. 자체 애플리케이션의 경우 Amazon Cognito 를 자격 증명 스토어로 사용하십시오. Amazon Cognito 는 앱 및 기타 AWS 서비스에서 권한 부여를 결정할 때 직접 사용할 수 있는 방법으로 자격 증명을 표현하고 사용자를 인증하는 표준 메커니즘을 제공합니다.

AWS IoT 를 사용하는 경우 Amazon Cognito Identity Pools, AWS IoT 정책 및 AWS IoT 사용자 지정 권한 부여자를 비롯한 여러 자격 증명 및 권한 부여 서비스 중에서 선택할 수 있습니다.

사용자를 위한 분리된 원격 측정 보기를 구현하려면 AWS AppSync 또는 Amazon API Gateway 같은 모바일 서비스를 사용하십시오. 이 두 AWS 서비스를 모두 사용하면 IoT 데이터 스트림을 사용자의 디바이스 데이터 알림 스트림에서 분리하는 추상화 계층을 생성할 수 있습니다. 예를 들어 Amazon DynamoDB 또는 Amazon ElasticSearch Service 같은 중간 데이터 스토어에 외부 사용자를 위한 별도의 데이터 보기를 생성하면 AWS AppSync 를 사용하여 중간 스토어에서 허용된 데이터를 기준으로 사용자별 알림을 수신할 수 있습니다. AWS AppSync 를 통해 외부 데이터 스토어를 사용하는 것에 더해 사용자별 알림 주제를 정의하고 이 주제를 사용하여 IoT 데이터의 특정 보기를 외부 사용자에게 푸시할 수 있습니다.

외부 사용자가 AWS IoT 엔드포인트와 직접 통신해야 하는 경우 사용자 자격 증명에 권한이 있는 Amazon Cognito 역할 및 세분화된 IoT 정책에 연결된 승인된 Amazon Cognito 연동 자격 증명을 사용하거나 자체 권한 부여 서비스로 권한 부여를 관리하는 AWS IoT 사용자 지정 권한 부여자를 사용해야 합니다. 어느 접근 방식을 사용하든 각 사용자의 연결, 게시, 구독 및 관심 MQTT 통신 메시지 수신을 제한하는 세분화된 정책을 사용자에게 연결해야 합니다.

IOTSEC 4. IoT 애플리케이션과 통신하는 보안 주체에 최소 권한을 적용하기 위해 어떤 방법을 사용합니까?

디바이스를 등록하고 디바이스 자격 증명을 설정한 후에는 모니터링, 지표, 원격 측정 또는 명령 및 제어에 필요한 추가 디바이스 정보를 제공해야 할 수 있습니다. 각 리소스에는 자체 액세스 제어 규칙이 할당되어야 합니다. 디바이스 또는 사용자가 애플리케이션에 수행할 수 있는 작업을 줄이고 각 리소스에 개별적으로 보안을 적용하면 단일 자격 증명 또는 리소스가 의도치 않게 사용될 때 발생할 수 있는 영향이 제한됩니다.

AWS IoT 에서 일관된 명명 규칙 세트를 사용하여 IoT 레지스트리에 세분화된 권한을 생성하십시오. 첫 번째 규칙은 MQTT ClientID 및 AWS IoT 사물 이름을 디바이스의 동일한 고유 식별자로 사용하는 것입니다. 이러한 모든 위치에서 동일한 고유 식별자를 사용하면 AWS IoT 사물 정책 변수를 사용하여 모든 디바이스에 적용할 수 있는 초기 IoT 권한 세트를 손쉽게 생성할 수 있습니다. 두 번째 명명 규칙은 디바이스의 고유 식별자를 디바이스 인증서에 포함하는

것입니다. 이 접근 방식을 사용하는 경우 인증서 주체 이름의 CommonName 으로 고유 식별자를 저장하면 인증서 정책 변수를 사용하여 IoT 권한을 각각의 고유 디바이스 자격 증명에 바인딩할 수 있습니다.

정책 변수를 사용하면 최소 권한을 유지하면서 모든 디바이스 인증서에 적용할 수 있는 몇 가지 IoT 정책을 생성할 수 있습니다. 예를 들어 아래의 IoT 정책은 인증서가 디바이스에 연결된 경우에만 디바이스의 고유 식별자(공통 이름에 저장됨)를 MQTT ClientID 로 사용하여 연결하도록 디바이스를 제한합니다. 또한 이 정책에서 디바이스는 디바이스의 개별 섀도우에만 게시 작업을 수행할 수 있습니다.

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Action": ["iot:Connect"],
        "Resource": ["arn:aws:iot:us-east-
1:123456789012:client/${iot:Certificate.Subject.CommonName}"],
        "Condition":{
            "Bool":{
                "iot:Connection.Thing.IsAttached":["true"]
            }
        }
    },
    {
        "Effect":"Allow",
        "Action":["iot:Publish"],
        "Resource":["arn:aws:iot:us-east-
1:123456789012:topic/$aws/things/${iot:Connection.Thing.ThingName}/
shadow/update"]
    }
    ]
}
```

AttachThingPrincipal 을 사용하여 디바이스 자격 증명(인증서 또는 Amazon Cognito 연동 자격 증명)을 AWS IoT 레지스트리의 사물에 연결하십시오.

이러한 시나리오는 자체 주제 및 디바이스 섀도우 세트와 통신하는 단일 디바이스에 적용되지만 단일 디바이스에서 다른 디바이스의 상태 또는 주제에 대해 작업을 수행해야 하는 시나리오도 있습니다. 예를 들어 산업 환경에서 엣지 어플라이언스를 운영하거나, 홈 게이트웨이를 생성하여 가정의 자동화 조정을 관리하거나, 사용자의 특정 역할에 따라 서로 다른 디바이스 세트에 액세스하는 것을 허용할 수 있습니다. 이러한 사용 사례에서는 엣지 게이트웨이의 그룹 식별자 또는 자격 증명 같은 알려진 엔터티를 게이트웨이와 통신하는 모든 디바이스의 접두사로 활용하십시오. 모든 엔드포인트 디바이스에 동일한 접두사를 사용하면 IoT 정책에서 "*" 와일드카드를 사용할 수 있습니다. 이 접근 방식은 MQTT 주제 보안과 관리 용이성 간의 균형을 유지합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
    {
        "Effect":"Allow",
        "Action":["iot:Publish"],
        "Resource":["arn:aws:iot:us-east-
1:123456789012:topic/$aws/things/edgegateway123-*/shadow/update"]
    }
    ]
}
```

앞의 예제에서 IoT 작업자는 식별자인 edgegateway123 을 사용하여 엣지 게이트웨이에 정책을 연결할 수 있습니다. 그러면 엣지 어플라이언스에서는 이 정책의 권한을 사용하여 엣지 게이트웨이로 관리되는 다른 디바이스 섀도우에 게시 작업을 수행할 수 있습니다. 이렇게 하려면 게이트웨이에 연결된 모든 디바이스에 게이트웨이의 식별자가 접두사로 추가된 사물 이름을 지정하면 됩니다. 예를 들어 다운스트림 동작 센서의 식별자를 edgegateway123-motionsensor1 로 지정하면 엣지 게이트웨이로 센서를 관리하면서 권한을 제한할 수 있습니다.

탐지 제어

IoT 애플리케이션의 데이터, 지표 및 로그는 규모가 크기 때문에 IoT 애플리케이션을 제대로 설계하려면 집계 및 모니터링 기능이 필수적입니다. 권한이 없는 사용자는 IoT 애플리케이션에서 버그를 찾아내고 개별 디바이스를 이용하여 다른 디바이스, 애플리케이션 및 클라우드 리소스에 대한 추가 액세스를 확보하려고 할 것입니다. 전체 IoT 솔루션을 운영하려면 개별 디바이스는

물론이고 애플리케이션의 전체 디바이스 플릿에 대한 탐지 제어를 관리해야 합니다. 디바이스 수준과 플릿 전체 수준에서 문제를 탐지하려면 여러 수준의 로깅, 모니터링 및 경보를 활성화해야 합니다.

제대로 설계된 IoT 애플리케이션에서는 IoT 애플리케이션의 각 계층에서 지표 및 로그가 생성됩니다. 최소한, 물리적 디바이스, 디바이스의 연결 동작, 디바이스당 메시지 입력 및 출력 속도, 프로비저닝 활동, 권한 부여 시도, 디바이스 데이터의 애플리케이션 간 내부 라우팅 이벤트와 관련된 지표 및 로그를 아키텍처에 포함해야 합니다.

IOTSEC 5: 클라우드와 디바이스 전체의 애플리케이션 로그 및 지표를 어떻게 분석하고 있습니까?

AWS IoT 에서는 AWS IoT Device Defender, CloudWatch Logs 및 CloudWatch Metrics 를 사용하여 탐지 제어를 구현할 수 있습니다. AWS IoT Device Defender 는 디바이스 동작 및 연결 동작과 관련된 로그 및 지표를 처리합니다. 또한 AWS IoT Device Defender 를 사용하면 디바이스 및 AWS IoT Core 의 보안 지표를 지속적으로 모니터링하여 각 디바이스에 정의된 적절한 동작과의 편차를 찾을 수 있습니다.

디바이스 동작 또는 연결 동작이 정상 활동과 다른 경우에 대한 기본 임계값 세트를 설정하십시오.

AWS IoT Core 에서 생성된 Amazon CloudWatch 지표, Amazon CloudWatch Logs 와 Amazon GuardDuty 를 통해 Device Defender 지표를 보강합니다. 이러한 서비스 수준 로그는 AWS IoT Platform 서비스 및 AWS IoT Core 프로토콜 사용과 관련된 활동은 물론이고 엔드투엔드 IoT 애플리케이션의 중요한 구성 요소인 다운스트림 애플리케이션(AWS 에서 실행되는 애플리케이션)에 대한 통찰력을 제공합니다. 모든 Amazon CloudWatch Logs 를 중앙에서 분석하여 모든 소스의 로그 정보를 연관지어야 합니다.

IOTSEC 6: IoT 애플리케이션에서 유효하지 않은 자격 증명은 어떻게 관리됩니까?

보안 자격 증명은 IoT 애플리케이션에서 디바이스 신뢰 및 권한 부여를 보장하기 위한 핵심 요소입니다. 유효하지 않은 자격 증명(예: 인증서)을 중앙에서 관리할 수 있어야 합니다. 유효하지 않은 인증서는 해지하거나 만료하거나 비활성화하면 됩니다. 애플리케이션을 제대로 설계하려면

유효하지 않은 모든 인증서를 캡처하는 프로세스와 인증서 상태에 따라 응답을 자동으로 트리거하는 프로세스가 있어야 합니다. 디바이스에는 유효하지 않은 인증서 이벤트를 캡처하는 기능에 더해 IoT 플랫폼에 대한 보안 통신을 설정하는 보조 수단도 있어야 합니다. 앞서 설명한 것과 같이 부트스트래핑 패턴을 활성화하여 디바이스에 2 가지 형태의 자격 증명을 사용하면 유효하지 않은 인증서를 탐지한 후 디바이스 또는 관리자에게 수정을 위한 신뢰할 수 있는 보안 통신을 설정할 수 있는 방법을 제공하는 안정적인 대체 메커니즘을 만들 수 있습니다.

제대로 설계된 IoT 애플리케이션은 CRL(인증서 해지 목록)을 설정합니다. 이 목록은 해지된 모든 디바이스 인증서 또는 CA(인증 기관)를 추적합니다. 디바이스를 온보딩할 때 신뢰할 수 있는 자체 CA 를 사용하고 CRL 을 IoT 애플리케이션과 정기적으로 동기화하십시오. IoT 애플리케이션은 더 이상 유효하지 않은 자격 증명을 사용한 연결을 거부해야 합니다.

AWS 에서는 전체 PKI 를 온프레미스로 관리할 필요가 없습니다. AWS Certificate Manager(ACM) 프라이빗 인증 기관을 사용하여 클라우드에서 CA 를 호스팅할 수 있습니다. 또는 APN 파트너와 협력하여 IoT 디바이스 하드웨어 사양에 사전 구성된 보안 요소를 추가할 수 있습니다. ACM 을 사용하는 경우 해지된 인증서를 S3 버킷의 파일로 내보낼 수 있습니다. 그런 다음 이 파일을 사용하여 AWS IoT Core 에 대한 인증서를 프로그래밍 방식으로 해지할 수 있습니다.

만료 날짜에 근접하지만 아직 유효한 상태의 인증서도 있습니다. 클라이언트 인증서는 최소한 디바이스의 서비스 수명 동안 유효해야 합니다. 따라서 IoT 애플리케이션은 만료 날짜에 가까운 디바이스를 추적하고 OTA 프로세스를 수행하여 나중에 만료되는 새 인증서로 인증서를 업데이트하고, 인증서 교체가 필요한 이유에 대한 로깅 정보를 기록하여 감사 목적으로 사용될 수 있도록 해야 합니다.

인증서 및 CA 만료와 관련된 AWS IoT Device Defender 감사를 활성화하십시오. Device Defender 는 30 일 이내에 만료될 예정인 인증서에 대한 감사 로그를 생성합니다. 이 목록을 사용하여 인증서가 무효화되기 전에 프로그래밍 방식으로 디바이스를 업데이트해야 합니다. 자체 만료 스토어를 구축하여 인증서 만료 날짜를 관리하고 디바이스 인증서 교체 또는 갱신을 위한 OTA 를 프로그래밍 방식으로 쿼리하고 식별하고 트리거할 수도 있습니다.

인프라 보호

설계 당시에 디바이스 및 솔루션의 전체 수명 주기에 걸친 인프라 보호에 대한 보안 요구 사항을 고려하는 것이 좋습니다. 디바이스를 인프라의 연장으로 간주하면 전체 디바이스 수명 주기가

인프라 보호를 위한 설계에 미치는 영향을 고려할 수 있습니다. 비용 관점에서 볼 때 설계 단계에서 변경을 수행하는 것이 나중에 변경을 수행하는 것보다 저렴합니다. 효과의 관점에서는 설계 시 구현된 데이터 손실 완화 기능이 이후에 조정된 완화 기능보다 포괄적일 수 있습니다. 따라서 설계 시 디바이스 및 솔루션 보안 수명 주기를 계획하면 비즈니스 위험이 감소하고 출시 전에 사전 인프라 보안 분석을 수행할 수 있습니다.

디바이스 보안 수명 주기에 접근하는 한 가지 방법은 공급망 분석을 사용하는 것입니다. 예를 들어 규모가 그리 크지 않은 IoT 디바이스 제조업체 또는 솔루션 통합자의 경우에도 많은 수의 공급업체가 직접 또는 간접적으로 공급망을 형성합니다. 솔루션의 수명과 안정성을 극대화하려면 정품 구성 요소를 받고 있는지 확인하십시오.

소프트웨어도 공급망의 일부입니다. 디바이스의 프로덕션 펌웨어 이미지에는 반도체 파트너, 오픈 소스 집계 사이트(예: GitHub 및 SourceForge), 이전 타사 제품, 내부 엔지니어링에서 개발한 새로운 코드를 비롯한 다양한 소스의 드라이버와 라이브러리가 포함됩니다.

타사 펌웨어 및 소프트웨어에 대한 다운스트림 유지 관리 및 지원을 이해하려면 공급망의 각 소프트웨어 공급자를 분석하여 지원 여부와 패치 제공 방법을 확인해야 합니다. 이 분석은 연결된 디바이스에 특히 중요합니다. 소프트웨어 버그는 불가피하며, 취약한 디바이스가 원격으로 악용되어 고객에게 위험을 초래할 수 있기 때문입니다. IoT 디바이스 제조업체 또는 솔루션 엔지니어링 팀은 적시에 버그에 대해 알아보고 패치를 적용하여 이러한 위험을 줄여야 합니다.

IOTSEC 7. 공급업체, 하청 제조업체 및 기타 아웃소싱 관계를 어떻게 검증하고 있습니까?

IOTSEC 8. IoT 디바이스의 보안 수명 주기를 어떻게 계획하고 있습니까?

IOTSEC 9. 타사 펌웨어 및 소프트웨어 구성 요소의 보안 버그를 적시에 알리기 위해 어떤 작업을 수행하고 있습니까?

AWS IoT 서비스를 사용하는 경우 클라우드 인프라 관리가 필요하지 않지만 AWS IoT Core 는 통합 지점을 사용하여 사용자 대신 AWS 서비스와 상호 작용합니다. 예를 들어 AWS IoT 규칙 엔진은 MQTT 주제 스트림을 기반으로 다른 AWS 서비스에 대한 다운스트림 작업을 트리거할 수 있는 분석된 규칙으로 구성됩니다. AWS IoT 가 다른 AWS 리소스와 통신하므로 사용자는 애플리케이션에 대해 올바른 서비스 역할 권한이 구성되어 있는지 확인해야 합니다.

데이터 보호

IoT 애플리케이션을 설계하기 전에 데이터 분류, 거버넌스 및 제어를 설계하고 문서화하여 데이터를 클라우드에서 유지하는 방법과 디바이스 데이터 또는 디바이스 및 클라우드 간 데이터를 암호화하는 방법을 반영해야 합니다. IoT 애플리케이션의 경우 기존의 클라우드 애플리케이션과 달리 데이터 민감도 및 거버넌스가 네트워크 경계 외부의 원격 위치에 배포되는 IoT 디바이스까지 확장됩니다. 이러한 기술은 디바이스에서 전송된 개인 식별 데이터를 보호하고 규제 의무를 준수하도록 지원한다는 점에서 중요합니다.

설계 프로세스 중에 수명이 종료된 디바이스의 하드웨어, 펌웨어 및 데이터를 처리하는 방법을 결정하십시오. 장기 기록 데이터는 클라우드에 저장합니다. 현재 센서 판독값에 해당하는 부분은 디바이스에 로컬로 저장합니다. 즉, 로컬 작업을 수행하는 데 필요한 데이터만 저장합니다. 디바이스에 필요한 최소 데이터만 저장하면 의도하지 않은 액세스의 위험이 제한됩니다.

로컬 데이터 스토리지를 줄이는 것에 더해 디바이스 수명이 종료될 때 구현되어야 하는 다른 완화 기능도 있습니다. 첫째, 디바이스는 하드웨어 및 펌웨어를 기본 공장 버전으로 재설정할 수 있는 재설정 옵션을 제공해야 합니다. 둘째, IoT 애플리케이션은 모든 디바이스의 마지막 로그인 시간을 주기적으로 스캔할 수 있습니다. 너무 오랫동안 오프라인 상태이거나 비활성 고객 계정과 연결된 디바이스는 해지될 수 있습니다. 셋째, 특정 디바이스에 고유한 키를 사용하여 디바이스에서 유지해야 하는 민감한 데이터를 암호화합니다.

IOTSEC 10: 전송 중인 데이터와 저장된 데이터를 어떻게 분류, 관리 및 보호합니까?

AWS IoT 에서 송수신되는 모든 트래픽은 TLS(전송 계층 보안)를 사용하여 암호화되어야 합니다. AWS IoT 의 보안 메커니즘은 AWS IoT 와 다른 디바이스 또는 AWS 서비스 간에 이동하는 데이터를 보호합니다. AWS IoT 에 더해 디바이스의 프라이빗 키와 디바이스에서 수집 및 처리된 데이터까지 보호하는 디바이스 수준 보안을 구현해야 합니다.

임베디드 개발의 경우 AWS 의 여러 서비스를 사용하여 기본적으로 엣지에 AWS 보안 모범 사례를 통합하면서 애플리케이션 계층의 구성 요소를 추상화할 수 있습니다. 마이크로컨트롤러의 경우 Amazon FreeRTOS 를 사용하는 것이 좋습니다. Amazon FreeRTOS 는 Bluetooth LE, TCP/IP 및 기타 프로토콜용 라이브러리를 통해 FreeRTOS 커널을 확장합니다.

또한 Amazon FreeRTOS 에는 AWS IoT 와 안전하게 통신하는 임베디드 애플리케이션을 생성할 수 있는 보안 API 세트가 포함되어 있습니다.

Linux 기반 엣지 게이트웨이의 경우 AWS IoT Greengrass 를 사용하여 클라우드 기능을 네트워크의 엣지까지 확장할 수 있습니다. AWS IoT Greengrass 를 사용하면 연결된 디바이스와의 상호 X.509 인증서 기반 인증, AWS IoT Greengrass 와 클라우드 애플리케이션 간의 통신 권한을 관리하는 AWS IAM 정책 및 역할, 연결된 디바이스와 Greengrass 코어 간 데이터 라우팅 여부 및 방법을 결정하는 데 사용되는 구독을 포함한 여러 보안 기능을 구현할 수 있습니다.

인시던트 대응

IoT 에서 인시던트 대응을 준비하려면 IoT 워크로드의 2 가지 인시던트 유형을 처리하는 방법을 계획해야 합니다. 첫 번째 인시던트는 성능을 방해하거나 디바이스의 동작에 영향을 미치려는 시도로 개별 IoT 디바이스를 공격하는 것입니다. 두 번째 인시던트는 네트워크 중단 및 DDoS 공격과 같은 대규모 IoT 이벤트입니다. 두 시나리오에서 모두 신속하게 인시던트를 진단하고, 인시던트 전체에서 데이터의 상관 관계를 파악한 다음, 자동화되고 안정적인 방식으로 영향을 받는 디바이스에 런북을 적용하려면 IoT 애플리케이션의 아키텍처가 중요한 역할을 합니다.

IoT 애플리케이션의 경우 인시던트 대응에 대한 다음 모범 사례를 따르십시오.

- IoT 디바이스는 위치 및 하드웨어 버전과 같은 디바이스 속성에 따라 서로 다른 그룹으로 구성됩니다.
- 연결 상태, 펌웨어 버전, 애플리케이션 상태 및 디바이스 상태와 같은 동적 속성을 사용하여 IoT 디바이스를 검색할 수 있습니다.
- 디바이스에 대한 OTA 업데이트를 단계적으로 시행하고 일정 기간에 걸쳐 배포할 수 있습니다. 배포 롤아웃을 모니터링하고 디바이스가 적절한 KPI 를 유지하지 못할 경우 자동으로 중단할 수 있습니다.
- 모든 업데이트 프로세스는 오류에 대한 복원력이 있으며 실패한 소프트웨어 업데이트에서 디바이스를 복구하고 롤백할 수 있습니다.
- 디바이스의 현재 성능 및 기간별 성능에 대한 컨텍스트 정보가 포함된 세부 로깅, 지표 및 디바이스 원격 측정을 사용할 수 있습니다.

- 플릿 전체 지표는 플릿의 전반적인 상태를 모니터링하고 운영 KPI 가 일정 기간 동안 충족되지 않는 경우 알림을 보냅니다.
- 예상 동작과 다른 동작을 보이는 개별 디바이스를 격리한 후 펌웨어 및 애플리케이션의 잠재적인 손상 여부를 검사하고 분석할 수 있습니다.

IOTSEC 11: 단일 디바이스 또는 디바이스 플릿에 영향을 미치는 인시던트 대응을 어떻게 준비합니까?

InfoSec 팀이 수정이 필요한 디바이스를 신속하게 식별할 수 있는 전략을 구현합니다. InfoSec 팀에 펌웨어 버전 관리 및 디바이스 업데이트에 대한 패치 적용을 다루는 런북이 있는지 확인합니다. 취약한 디바이스가 온라인 상태가 될 때 보안 패치를 사전에 적용하는 자동화된 프로세스를 생성합니다.

최소한 보안 팀에는 디바이스 로그와 현재 디바이스 동작을 기반으로 특정 디바이스에서 인시던트를 탐지할 수 있는 기능이 있어야 합니다. 인시던트가 식별된 후 다음 단계는 애플리케이션을 격리하는 것입니다. AWS IoT 서비스를 사용하여 이를 구현하려면 AWS IoT 사물 그룹을 보다 제한적인 IoT 정책과 함께 사용하고 이러한 디바이스에 대해 사용자 지정 그룹 로깅을 활성화하면 됩니다. 이렇게 하면 문제 해결과 관련된 기능만 활성화하고 더 많은 데이터를 수집하여 근본 원인과 해결 방법을 찾을 수 있습니다. 마지막으로 인시던트가 해결된 후에는 디바이스에 펌웨어 업데이트를 배포하여 알려진 상태로 되돌릴 수 있어야 합니다.

주요 AWS 서비스

IoT 에서 필수적으로 사용해야 하는 AWS 보안 서비스는 AWS IoT 레지스트리, AWS IoT Device Defender, AWS Identity and Access Management(IAM) 및 Amazon Cognito 입니다. 이러한 서비스를 함께 사용하면 IoT 디바이스, AWS 서비스 및 리소스에 대한 액세스를 안전하게 제어할 수 있습니다. 다음 서비스 및 기능은 보안의 5 가지 영역을 지원합니다.

설계: AWS Device Qualification Program 은 AWS IoT 와의 상호 운용성에 대한 사전 테스트를 마친 IoT 엔드포인트 및 엣지 하드웨어를 제공합니다. 테스트에는 원격 패치 적용을 위한 상호 인증 및 OTA 지원이 포함됩니다.

AWS Identity and Access Management(IAM): 디바이스 자격 증명(X.509 인증서, IAM, Amazon Cognito 자격 증명 풀 및 Amazon Cognito 사용자 풀 또는 사용자 지정 권한 부여 토큰)을 사용하여 AWS 리소스에 대한 디바이스 및 외부 사용자 액세스를 안전하게 제어할 수 있습니다. 여기에 더해 AWS IoT 정책을 사용하면 IoT 디바이스에 대한 세분화된 액세스를 구현할 수 있습니다. ACM 프라이빗 CA 를 사용하면 클라우드 기반 접근 방식으로 디바이스 인증서를 생성하고 관리할 수 있습니다. AWS IoT 사물 그룹을 사용하면 IoT 권한을 개별적으로 관리하는 대신 그룹 수준에서 관리할 수 있습니다.

탐지 제어: AWS IoT Device Defender 는 AWS IoT Core 의 디바이스 통신 및 클라우드 측 지표를 기록합니다. AWS IoT Device Defender 를 사용하면 Amazon Simple Notification Service(Amazon SNS)를 통해 내부 시스템 또는 관리자에게 알림을 전송하여 보안 대응을 자동화할 수 있습니다. AWS CloudTrail 은 IoT 애플리케이션의 관리 작업을 기록합니다. Amazon CloudWatch 는 AWS IoT Core 와 통합되는 모니터링 서비스이며 CloudWatch Events 를 트리거하여 보안 대응을 자동화할 수 있습니다. CloudWatch 는 IoT 엣지 구성 요소와 클라우드 서비스 간의 연결 및 보안 이벤트와 관련된 세부 로그를 캡처합니다.

인프라 보호: AWS IoT Core 는 연결된 디바이스와 클라우드 애플리케이션 및 다른 디바이스의 간편하고 안전한 상호 작용을 지원하는 관리형 서비스입니다. AWS IoT Core 의 AWS IoT 규칙 엔진은 IAM 권한을 사용하여 다른 다운스트림 AWS 서비스와 통신합니다.

데이터 보호: AWS IoT 에는 TLS 를 통한 디바이스 암호화를 통해 전송 중인 데이터를 보호하는 암호화 기능이 포함되어 있습니다. AWS IoT 는 저장된 데이터의 암호화를 지원하는 Amazon S3 및 Amazon DynamoDB 와 같은 서비스와 직접 통합됩니다. 또한 AWS Key Management Service(AWS KMS)를 사용하면 암호화에 사용되는 키를 생성하고 제어할 수 있습니다. 디바이스에서는 Amazon FreeRTOS, AWS IoT Greengrass 또는 AWS IoT Embedded C SDK 와 같은 AWS 엣지 오퍼링을 사용하여 보안 통신을 지원할 수 있습니다.

인시던트 대응: AWS IoT Device Defender 를 사용하면 정상 디바이스 동작과의 편차를 탐지하고 AWS Lambda 를 포함한 자동화된 대응을 트리거하는 데 사용할 수 있는 보안 프로필을 생성할 수 있습니다. AWS IoT Device Management 를 사용하여 수정이 필요한 디바이스를 그룹화한 다음 AWS IoT Jobs 를 사용하여 디바이스에 수정 사항을 배포해야 합니다.

리소스

보안 관련 AWS 모범 사례에 대해 자세히 알아보려면 다음 리소스를 참조하십시오.

설명서 및 블로그

- IoT 보안 자격 증명
- AWS IoT Device Defender
- IoT 인증 모델
- 포트 443 의 MQTT
- Device Defender 를 사용하여 이상 탐지

백서

- MQTT 주제 설계

안정성 기반

안정성 기반은 장애를 예방하고 신속하게 복구하여 비즈니스 및 고객 요구 사항을 충족할 수 있는 기능에 중점을 둡니다. 주요 주제로는 설정 관련 기반 요소, 프로젝트 간 요구 사항, 복구 계획 및 변경 관리가 있습니다.

설계 원칙

전반적인 Well-Architected 프레임워크의 설계 원칙에 더해 클라우드의 IoT 에는 안정성을 위한 3 가지 설계 원칙이 있습니다.

- **프로덕션 규모에서 디바이스 동작 시뮬레이션**: 프로덕션 배포를 면밀하게 미러링하는 프로덕션 규모의 테스트 환경을 생성합니다. 라이브 시작 날짜 전에 더 큰 로드로 애플리케이션을 테스트할 수 있는 다단계 시뮬레이션 계획을 활용합니다. 개발 중에는 단일 테스트에 대한 전체 트래픽의 10%에서 시작하여 시간이 지남에 따라 시뮬레이션 테스트의 트래픽을 늘립니다(즉, 1 일차 디바이스 트래픽의 25%, 50% 및 100%). 시뮬레이션 테스트 중에 성능을 모니터링하고 로그를 검토하여 전체 솔루션이 예상대로 작동하는지 확인합니다.
- **스트림 또는 대기열을 사용하여 IoT 규칙 엔진의 메시지 전송 버퍼링**: 관리형 서비스를 활용하여 고처리량 원격 측정을 지원합니다. 고처리량 주제 뒤에 대기열 계층을 배치하면 IoT 애플리케이션에서 장애를 관리하고 메시징을 집계하고 다른 다운스트림 서비스를 확장할 수 있습니다.
- **장애 및 복원력을 고려한 설계**: 디바이스 자체의 복원력을 계획하는 것이 필수적입니다. 사용 사례에 따라 다르지만 복원력에는 간헐적 연결에 대한 강력한 재시도 로직, 펌웨어 업데이트를 롤백하는 기능, 다른 네트워킹 프로토콜로 장애 조치하거나 중요한 메시지 전송에 대해 로컬로 통신하는 기능, 이중화된 센서 또는 엣지 게이트웨이를 실행하여 하드웨어 오류에 대한 복원력을 갖추는 기능과 공장 재설정을 수행하는 기능이 포함될 수 있습니다.

정의

클라우드의 안정성에는 3 가지 모범 사례 영역이 있습니다.

1. 기반
2. 변경 관리
3. 오류 관리

안정성을 달성하려면 잘 계획된 기반과 모니터링 기능을 갖춘 시스템과 수요 변경 또는 요구 사항을 처리하거나 잠재적으로 무단 서비스 거부 공격을 방어할 수 있는 메커니즘이 필요합니다. 또한 장애를 탐지하고 자동으로 해결할 수 있는 시스템을 설계해야 합니다.

모범 사례

기반

IoT 디바이스는 네트워크 또는 클라우드 오류가 발생할 경우 일정 용량으로 계속해서 작동해야 합니다. 메모리 및 전력 제약에 민감한 방식으로 간헐적 연결 또는 연결 손실을 처리하도록 디바이스 펌웨어를 설계하십시오. 또한 IoT 클라우드 애플리케이션은 온라인과 오프라인 간 전환이 빈번한 원격 디바이스를 처리하여 데이터 일관성을 유지하고 시간이 지남에 따라 수평적 조정이 가능하도록 설계되어야 합니다. 전체 IoT 사용률을 모니터링하고 자동으로 용량을 확장하여 애플리케이션에서 피크 IoT 트래픽을 관리할 수 있는 메커니즘을 생성하십시오.

디바이스에서 불필요한 피크 트래픽이 생성되지 않도록 하려면 전체 디바이스 플릿에서 동일한 작업이 동시에 시도되는 것을 방지하는 디바이스 펌웨어를 구현해야 합니다. 예를 들어 경보 시스템으로 구성된 IoT 애플리케이션에서 모든 경보 시스템이 현지 시간으로 오전 9 시에 활성화 이벤트를 보내게 되면 전체 플릿에서 생성되는 즉각적인 스파이크가 IoT 애플리케이션에 발생합니다. 따라서 이 예약된 활동에 시간 제한 이벤트 및 지수 백오프와 같은 무작위 인수를 포함하여 IoT 디바이스가 일정 시간 내에 피크 트래픽을 보다 균등하게 분산할 수 있도록 하는 것이 좋습니다.

다음은 안정성 고려 사항에 중점을 둔 질문입니다.

IOTREL 1. IoT 애플리케이션의 피크에 대한 AWS Service Limits 를 어떻게 처리합니까?

AWS IoT 는 다양한 사용량 측면을 고려한 유동 및 고정 한도 세트를 제공합니다. AWS IoT 의 모든 데이터 플레인 한도는 IoT 한도 페이지에 명시되어 있습니다. 데이터 플레인 작업(예: MQTT 연결, MQTT 게시 및 MQTT 구독)을 수행하려면 디바이스 연결성이 중요합니다. 따라서 IoT 한도를 검토하고 애플리케이션이 데이터 플레인과 관련된 유동 한도를 준수하고 데이터 플레인을 통해 적용된 고정 한도를 초과하지 않는지 확인하는 것이 중요합니다.

IoT 조정 접근 방식에서 가장 중요한 부분은 고정 한도를 중심으로 아키텍처를 설계해야 한다는 것입니다. 조정 불가능한 한도를 초과하면 조절 및 클라이언트 오류 같은 애플리케이션 오류가 발생하기 때문입니다. 고정 한도는 단일 IoT 연결의 처리량과 관련됩니다. 애플리케이션이 고정 한도를 초과하는 경우 이러한 시나리오를 방지하기 위해 애플리케이션을 다시 설계하는 것이

좋습니다. 방법은 다양합니다. 예를 들어 MQTT 주제를 재구성하거나 메시지를 관심 디바이스에 전달하기 전에 메시지를 집계하거나 필터링하는 클라우드 측 로직을 구현할 수 있습니다.

AWS IoT 의 유동 한도는 일반적으로 단일 디바이스와 관계가 없는 계정 수준 한도와 관련됩니다. 계정 수준 한도의 경우 단일 디바이스의 IoT 사용량을 계산한 다음 이 사용량을 디바이스 수로 곱하여 초기 제품 출시에서 애플리케이션에 필요한 기준 IoT 한도를 결정해야 합니다. 증가 기간을 통해 현재 프로덕션 피크 사용량 및 추가 버퍼와 밀접하게 일치하는 범위 내에서 한도를 늘리는 것이 좋습니다. IoT 애플리케이션의 언더프로비저닝을 방지하려면 다음과 같이 하십시오.

- 게시된 AWS IoT CloudWatch 지표에서 모든 한도를 참조합니다.
- AWS IoT Core 의 CloudWatch 지표를 모니터링합니다.
- CloudWatch 조절 지표에 대한 알림을 설정하여 한도 증가가 필요한 경우 알림을 받습니다.
- MQTT 연결, 게시, 구독, 수신 및 규칙 엔진 작업을 비롯하여 IoT 의 모든 임계값에 대한 경보를 설정합니다.
- 100% 용량에 도달하기 전에 적시에 한도 증가를 요청해야 합니다.

데이터 플레인 한도에 더해 AWS IoT 서비스에는 관리 API 에 대한 제어 플레인이 있습니다. 제어 플레인은 IoT 정책 및 보안 주체를 생성 및 저장하고, 레지스트리에 사물을 생성하며, 인증서 및 Amazon Cognito 연동 자격 증명 같은 IoT 보안 주체를 연결하는 프로세스를 관리합니다. 부트스트래핑 및 디바이스 등록은 전체 프로세스에서 중요한 역할을 하므로 제어 플레인 운영 및 한도를 계획하는 것이 중요합니다. 제어 플레인 API 호출은 초당 요청 수로 측정되는 처리량을 기반으로 합니다. 제어 플레인 호출의 단위는 일반적으로 초당 수십 개의 요청입니다. 최대 예상 등록 사용량에서 역방향으로 작업하여 제어 플레인 작업에 대한 한도 증가가 필요한지 여부를 결정하는 것이 중요합니다. IoT 한도 증가가 정기적인 일일 데이터 플레인 사용량과 일치하도록 온보딩 디바이스에 대한 지속 가능한 증가 기간을 계획하십시오.

제어 플레인 요청의 버스트를 방지하려면 아키텍처가 이러한 API 에 대한 액세스를 권한이 있는 사용자 또는 내부 애플리케이션으로 제한해야 합니다. 백오프 및 재시도 로직을 구현하고 인바운드 요청을 대기열에 추가하여 이러한 API 에 대한 데이터 비율을 제어하십시오.

IOTREL 2. 다른 애플리케이션에 대한 IoT 데이터의 수집 및 처리 처리량을 관리하기 위한 전략은 무엇입니까?

IoT 애플리케이션에서 통신은 다른 디바이스 간에만 라우팅되지만 메시지는 애플리케이션에서 처리되고 저장됩니다. 이러한 경우 나머지 IoT 애플리케이션은 수신 데이터에 응답할 준비가 되어 있어야 합니다. 해당 데이터를 사용하는 모든 내부 서비스에는 데이터 수집 및 처리를 원활하게 확장할 수 있는 방법이 필요합니다. 제대로 설계된 IoT 애플리케이션의 내부 시스템은 수집 계층을 통해 IoT 플랫폼의 연결 계층에서 분리됩니다. 수집 계층은 대기열 및 스트림으로 구성되는데 이러한 대기열 및 스트림은 수집 속도와 관계없이 컴퓨팅 리소스로 데이터를 처리할 수 있게 하는 동시에 내구성 있는 단기 스토리지를 지원합니다.

처리량을 최적화하려면 컴퓨팅 작업을 수행하기 전에 AWS IoT 규칙을 사용하여 Amazon Kinesis Data Streams, Amazon Kinesis Data Firehose 또는 Amazon Simple Queue Service 와 같은 서비스로 인바운드 디바이스 데이터를 라우팅합니다. 피크 용량을 처리할 모든 중간 스트리밍 지점을 프로비저닝해야 합니다. 이 접근 방식을 사용하면 업스트림 애플리케이션에서 데이터를 복원적으로 처리하는 데 필요한 대기열 계층이 생성됩니다.

IOTREL 3. 클라우드와 통신할 때 디바이스 안정성을 어떻게 처리합니까?

IoT 솔루션의 안정성에는 디바이스 자체도 포함되어야 합니다. 디바이스는 원격 위치에 배포되며 IoT 애플리케이션으로 제어할 수 없는 다양한 외부 요인으로 인해 발생하는 간헐적 연결 또는 연결 손실을 처리합니다. 예를 들어 ISP 가 몇 시간 동안 중단되는 경우 디바이스는 이러한 장기간의 잠재적 네트워크 중단에 어떻게 동작하고 대응합니까? 디바이스에 최소 임베디드 작업 세트를 구현하면 AWS IoT Core 에 대한 연결 및 통신 관리에서 미묘한 차이가 발생하더라도 그에 대한 복원력이 개선됩니다.

IoT 디바이스는 인터넷 연결 없이도 작동이 가능해야 합니다. 따라서 다음과 같은 기능을 제공하는 강력한 작업을 펌웨어로 구현해야 합니다.

- 중요한 메시지를 오프라인으로 안정적으로 저장하고 다시 연결되면 해당 메시지를 AWS IoT Core 로 전송하십시오.

- 연결 시도가 실패할 경우 지수 재시도 및 백오프 로직을 구현합니다.
- 필요한 경우 중요한 메시지를 AWS IoT 로 전송하기 위한 별도의 장애 조치 네트워크 채널을 마련합니다. 예를 들어 Wi-Fi 에서 대기 셀룰러 네트워크로 장애 조치하거나 연결된 디바이스 또는 게이트웨이로 메시지를 전송하기 위해 무선 개인 영역 네트워크 프로토콜(예: Bluetooth LE)로 장애 조치할 수 있습니다.
- NTP 클라이언트 또는 드리프트가 낮은 실시간 클록을 사용하여 현재 시간을 설정하는 방법을 사용하십시오. 디바이스는 AWS IoT Core 와의 연결을 시도하기 전에 시간이 동기화될 때까지 대기해야 합니다. 이것이 가능하지 않은 경우에는 사용자가 시스템에서 후속 연결이 설정할 수 있도록 디바이스 시간을 설정할 수 있습니다.
- 오류 코드 및 전체 진단 메시지를 AWS IoT Core 로 전송하십시오.

변경 관리

IOTREL 4. IoT 애플리케이션에 대한 변경 사항을 어떻게 롤아웃하고 롤백합니까?

롤아웃이 실패할 경우 이전 버전의 디바이스 펌웨어 또는 클라우드 애플리케이션으로 되돌릴 수 있는 기능을 구현하는 것이 중요합니다. 애플리케이션을 제대로 설계한다면 디바이스의 지표와 AWS IoT Core 및 AWS IoT Device Defender 에서 생성된 지표가 캡처됩니다. 또한 클라우드 측 변경 후 디바이스 카나리아가 예상 동작과 다를 경우 경보가 발생합니다. 운영 지표의 편차를 기반으로 다음을 수행할 수 있어야 합니다.

- Amazon S3 를 사용하여 모든 디바이스 펌웨어의 버전을 지정합니다.
- 디바이스 펌웨어에 대한 매니페스트 또는 실행 단계의 버전을 지정합니다.
- 오류 발생 시 디바이스를 대체할 수 있는 안전한 기본 펌웨어 버전을 구현합니다.
- 암호화 코드 서명, 버전 확인 및 다중 비휘발성 스토리지 파티션을 사용하여 소프트웨어 이미지를 배포하고 롤백하는 업데이트 전략을 구현합니다.
- CloudFormation 에 있는 모든 IoT 규칙 엔진 구성의 버전을 지정합니다.
- CloudFormation 을 사용하여 모든 다운스트림 AWS 클라우드 리소스의 버전을 관리합니다.

- CloudFormation 및 기타 코드형 인프라 도구를 사용하여 클라우드 측 변경 사항을 되돌리는 롤백 전략을 구현합니다.

AWS 에서 인프라를 코드로 처리하면 IoT 애플리케이션에 대한 모니터링 및 변경 관리를 자동화할 수 있습니다. 모든 디바이스 펌웨어 아티팩트의 버전을 업데이트하고 필요한 경우 업데이트를 확인, 설치 또는 롤백할 수 있는지 확인합니다.

오류 관리

IOTREL 5. IoT 애플리케이션의 내결함성은 어떻게 구현되었습니까?

IoT 는 이벤트 기반 워크로드이므로 이벤트가 애플리케이션으로 들어올 때 발생할 수 있는 알려진 오류 및 알려지지 않은 오류를 처리할 복원력이 애플리케이션 코드에 구현되어야 합니다. 제대로 설계된 IoT 애플리케이션에는 데이터 처리 시 오류를 기록하고 재시도하는 기능이 포함됩니다. IoT 애플리케이션은 모든 데이터를 원시 형식으로 보관합니다. 유효하거나 유효하지 않은 모든 데이터를 아카이빙하는 아키텍처에서는 데이터를 특정 시점으로 보다 정확하게 복원할 수 있습니다.

IoT 규칙 엔진을 사용하면 애플리케이션에서 IoT 오류 작업을 활성화할 수 있습니다. 작업을 호출할 때 문제가 발생하면 규칙 엔진이 오류 작업을 호출합니다. 그러면 기본 IoT 작업으로 전달할 수 없는 메시지를 캡처 및 모니터링하고 알림을 전송한 후 결과적으로 재시도할 수 있습니다. IoT 오류 작업은 기본 작업과 다른 AWS 서비스를 사용하여 구성하는 것이 좋습니다. 오류 작업에는 Amazon SQS 또는 Amazon Kinesis 등 내구력 있는 스토리지를 사용하십시오.

규칙 엔진을 사용하는 것에 더해, 처음에 대기열의 메시지를 처리하고 해당 메시지의 스키마가 올바른지 검증하는 애플리케이션 로직을 구현해야 합니다. 알려진 모든 오류를 포착하여 기록하고, 필요한 경우 추가 분석을 위해 이러한 메시지를 자체 DLQ 로 이동하는 애플리케이션 로직이 필요합니다. Amazon Kinesis Data Firehose 및 AWS IoT Analytics 채널을 사용하여 모든 원시 메시지와 형식이 지정되지 않은 메시지를 Amazon S3, AWS IoT Analytics 데이터 스토어 및 데이터 웨어하우징용 Amazon Redshift 의 장기 스토리지로 전송하는 포괄적인 IoT 규칙을 사용해야 합니다.

IOTREL 6. 물리적 자산의 다양한 하드웨어 오류 수준 모드를 어떻게 확인합니까?

IoT 구현은 디바이스 수준에서 여러 유형의 오류를 허용해야 합니다. 하드웨어, 소프트웨어, 연결 또는 예기치 않은 이상 조건으로 인한 오류가 발생할 수 있습니다. 사물의 오류에 대비하는 한 가지 방법은 가능한 경우 디바이스를 페어로 배포하거나 동일한 적용 범위 영역(메시)에 배포된 디바이스 플릿에 이중 센서를 배포하는 것입니다.

디바이스 장애의 근본적인 원인이 무엇이든, 클라우드 애플리케이션과의 통신이 가능하다면 해당 디바이스에서 진단 주제를 사용하여 하드웨어 오류에 대한 진단 정보를 AWS IoT Core 로 전송해야 합니다. 하드웨어 오류로 인해 디바이스가 연결이 끊긴 경우 플릿 인덱싱을 연결 상태와 함께 사용하여 연결 상태 변경을 추적합니다. 디바이스가 장기간 오프라인 상태인 경우 디바이스에 수정이 필요할 수 있음을 알리는 알림을 트리거합니다.

주요 AWS 서비스

Amazon CloudWatch 를 사용하여 실행 시간 지표를 모니터링하고 안정성을 보장하십시오. 3 가지 안정성 영역을 지원하는 다른 서비스 및 기능은 다음과 같습니다.

기반: AWS IoT Core 를 사용하면 기반 인프라를 관리할 필요 없이 IoT 애플리케이션을 확장할 수 있습니다. 계정 수준 한도 증가를 요청하여 AWS IoT Core 를 확장할 수 있습니다.

변경 관리: AWS IoT Device Management 를 사용하면 Amazon S3 를 사용하여 디바이스의 모든 펌웨어, 소프트웨어 및 업데이트 매니페스트의 버전을 관리하면서 현장의 디바이스를 업데이트할 수 있습니다. AWS CloudFormation 을 사용하면 IoT 인프라를 코드로 문서화하고 CloudFormation 템플릿을 사용하여 클라우드 리소스를 프로비저닝할 수 있습니다.

오류 관리: Amazon S3 를 사용하면 디바이스의 원격 측정을 오래 보관할 수 있습니다. AWS IoT 규칙 엔진의 오류 작업을 사용하면 기본 AWS 서비스가 오류를 반환할 때 다른 AWS 서비스로 대체할 수 있습니다.

리소스

안정성과 관련된 AWS 모범 사례에 대해 자세히 알아보려면 다음 리소스를 참조하십시오.

설명서 및 블로그

- 디바이스 시간을 사용하여 AWS IoT 서버 인증서 검증
- AWS IoT Core 한도
- IoT 오류 작업
- 플릿 인덱싱
- IoT Atlas

성능 효율성 기반

성능 효율성 기반은 컴퓨팅 리소스를 효율적으로 사용하는 데 중점을 둡니다. 이 기반에서는 워크로드 요구 사항을 기반으로 적합한 리소스 유형 및 크기를 선택하고, 성능을 모니터링하고, 비즈니스 및 기술 요구 사항의 변화에 따라 효율성을 유지할 수 있도록 정보에 근거한 의사 결정을 내리는 방법 등을 주로 다룹니다. 성능 효율성 기반은 컴퓨팅 리소스를 효율적으로 사용하여 요구 사항을 충족하고 수요 변경 및 기술 변화에 따라 이러한 효율성을 유지하는 데 중점을 둡니다.

설계 원칙

전반적인 Well-Architected 프레임워크의 성능 효율성 설계 원칙에 더해 클라우드의 IoT 에는 성능 효율성을 위한 3 가지 설계 원칙이 있습니다.

- **관리형 서비스 사용**: AWS 는 데이터베이스, 컴퓨팅 및 스토리지에 걸친 여러 관리형 서비스를 제공합니다. 이러한 서비스를 사용하면 아키텍처의 전반적인 안정성 및 성능을 개선하는 데 도움이 됩니다.
- **데이터를 배치로 처리**: IoT 애플리케이션의 연결 부분을 IoT 의 수집 및 처리 부분에서 분리하십시오. 수집 계층을 분리하면 IoT 애플리케이션에서 데이터를 집계하여 처리하고 여러 IoT 메시지를 한 번에 처리하여 좀 더 원활한 조정을 지원할 수 있습니다.

- **이벤트 기반 아키텍처 사용**: IoT 시스템은 디바이스의 이벤트를 게시하고 해당 이벤트를 IoT 애플리케이션의 다른 하위 시스템으로 전달합니다. 따라서 대기열, 메시지 처리, 멱등성, 배달 못한 편지 대기열 및 상태 시스템을 활용하는 등 이벤트 기반 아키텍처를 지원하는 메커니즘을 설계해야 합니다.

정의

클라우드의 성능 효율성에는 4 가지 모범 사례 영역이 있습니다.

1. 선택
2. 검토
3. 모니터링
4. 절충

고성능 아키텍처를 선택할 때는 데이터 기반 접근 방식을 사용하십시오. 개괄적 설계부터 리소스 유형 선택 및 구성에 이르는 아키텍처의 모든 측면에 대한 데이터를 수집합니다. 주기적으로 선택 사항을 검토함으로써 지속적으로 변화하는 AWS 플랫폼을 최대한 활용할 수 있습니다.
모니터링을 수행하면 예상 성능과의 차이를 인식하고 조치를 취할 수 있습니다. 압축 또는 캐싱을 사용하거나 일관성 요구 사항을 완화하는 등의 절충을 통해 아키텍처를 설계하면 성능을 개선할 수 있습니다.

모범 사례

선택

Well-Architected IoT 솔루션은 디바이스, 연결, 데이터베이스, 데이터 처리 및 분석과 같은 다수의 시스템 및 구성 요소로 구성됩니다. AWS 에는 비즈니스 목표에 집중하면서 제대로 설계된 솔루션을 신속하게 구축할 수 있는 여러 IoT 서비스, 데이터베이스 제품 및 분석 솔루션이 있습니다. 워크로드에 가장 적합한 관리형 AWS 서비스를 혼합하여 활용하는 것이 좋습니다.
다음은 성능 효율성 고려 사항에 중점을 둔 질문입니다.

IOTPERF 1. 최고의 성능을 제공하는 IoT 아키텍처를 어떻게 선택합니까?

아키텍처에 대한 구현을 선택할 때는 장기적 운영 관점에서 데이터 기반 접근 방식을 사용하여 선택해야 합니다. IoT 애플리케이션은 이벤트 기반 아키텍처에 잘 맞습니다. IoT 애플리케이션을 위한 아키텍처에는 이벤트 기반 패턴(예: 알림, 데이터 게시 및 구독, 스트림 처리와 이벤트 기반 컴퓨팅)과 통합되는 여러 서비스가 사용됩니다. 다음 섹션에서는 아키텍처 구현을 선택할 때 고려해야 할 5 가지 주요 IoT 리소스 유형(디바이스, 연결, 데이터베이스, 컴퓨팅 및 분석)을 살펴봅니다.

디바이스

특정 시스템에 대한 최적의 임베디드 소프트웨어는 디바이스의 하드웨어 공간에 따라 달라집니다. 예를 들어 네트워크 보안 프로토콜은 데이터 프라이버시와 무결성을 유지하는 데 필요하지만 비교적 많은 RAM 공간을 차지할 수 있습니다. 인트라넷 및 인터넷 연결의 경우 강력한 암호 그룹 및 최소 공간의 조합과 함께 TLS 를 사용하십시오. AWS IoT 는 TLS 를 사용하여 AWS IoT 에 연결하는 디바이스에 대해 ECC(Elliptic Curve Cryptography)를 지원합니다. 디바이스에 대한 선택 기준 중에서 우선해야 할 것은 디바이스의 안전한 소프트웨어 및 하드웨어 플랫폼입니다. AWS 는 AWS IoT 에 안전하게 통합할 수 있는 하드웨어 솔루션을 제공하는 여러 IoT 파트너를 보유하고 있습니다.

적절한 하드웨어 파트너를 선택하는 것 외에 Amazon FreeRTOS 및 AWS IoT Greengrass 를 비롯한 여러 소프트웨어 구성 요소를 사용하여 디바이스의 애플리케이션 로직을 실행할 수도 있습니다.

IOTPERF 2. IoT 디바이스의 하드웨어와 운영 체제를 선택할 때는 어떻게 합니까?

IoT 연결

클라우드 통신에 사용할 펌웨어를 개발하기 전에 안전하고 확장 가능한 연결 플랫폼을 구현하여 디바이스의 장기적이고 지속적인 증가를 지원해야 합니다. 예상되는 디바이스 볼륨에 따라 IoT 플랫폼에서는 간단한 원격 측정 수집 또는 디바이스 간 명령 및 응답 통신과 같은 디바이스와 클라우드 간의 통신 워크플로를 확장할 수 있어야 합니다.

EC2 와 같은 AWS 서비스를 사용하여 IoT 애플리케이션을 구축할 수 있지만, 이 경우 IoT 오퍼링 안에 고유한 가치를 구축하려면 획일적이고 고된 작업을 수행해야 합니다. 따라서 AWS IoT Core 를 IoT 플랫폼으로 사용하는 것이 좋습니다.

AWS IoT Core 는 HTTP, WebSocket 및 MQTT(간헐적 연결을 허용하도록 설계된 경량의 통신 프로토콜)를 지원하고, 디바이스의 코드 공간을 최소화하며, 네트워크 대역폭 요구 사항을 줄여줍니다.

IOTPERF 3. 기본 IoT 플랫폼을 선택할 때는 어떻게 합니까?

데이터베이스

IoT 애플리케이션에는 데이터베이스에 데이터를 쓰는 빈도, 데이터베이스의 데이터를 읽는 빈도, 데이터 구조 및 쿼리 방법과 같은 속성에 대해 각각 선택된 다수의 데이터베이스가 포함됩니다. 데이터베이스 오퍼링을 선택할 때 고려해야 할 다른 기준도 있습니다.

- 데이터 볼륨 및 보존 기간.
- 내장 데이터의 구성 및 구조.
- 원시 데이터 또는 처리된 데이터를 소비하는 사용자 및 애플리케이션과 이러한 사용자 및 애플리케이션의 지리적 위치/분산.
- 기계 학습 또는 실시간 시각화와 같은 고급 분석 요구 사항.
- 다른 팀, 조직 및 사업 부서 간 데이터 동기화.
- 행, 테이블 및 데이터베이스 수준의 데이터 보안.
- 엔터프라이즈 애플리케이션, 드릴스루 대시보드 또는 시스템 상호 작용 등 기타 관련된 데이터 기반 이벤트와의 상호 작용.

AWS 에는 IoT 솔루션을 지원하는 여러 데이터베이스 제품이 있습니다. 구조적 데이터의 경우 고도로 확장 가능한 조직 데이터용 관계형 인터페이스인 Amazon Aurora 를 사용해야 합니다. 쿼리 시 지연 시간이 짧아야 하고 여러 소비자가 사용하는 반구조적 데이터의 경우 DynamoDB 를 사용하십시오. DynamoDB 는 완전관리형 다중 리전, 다중 마스터 데이터베이스로, 일관된 한 자릿수의 지연 시간을 제공하고 내장된 보안, 백업 및 복원 및 인메모리 캐싱을 제공합니다.

형식이 지정되지 않은 원시 이벤트 데이터를 저장하려면 AWS IoT Analytics 를 사용합니다. AWS IoT Analytics 는 분석을 위해 시계열 데이터 스토어에 저장할 IoT 데이터를 필터링하고 변환하고 보강합니다. Amazon SageMaker 를 사용하면 Greengrass Machine Learning Inference 와 같은

AWS IoT 서비스를 사용하여 클라우드와 엣지에서 IoT 데이터 기반의 기계 학습 모델을 구축하고 학습하고 배포할 수 있습니다. 원시 형식의 시계열 데이터는 Amazon Redshift 와 같은 데이터 웨어하우스 솔루션에 저장하는 것이 좋습니다. 형식이 지정되지 않은 데이터는 Amazon S3 및 Amazon Kinesis Data Firehose 를 통해 Amazon Redshift 로 가져올 수 있습니다. 형식이 지정되지 않은 데이터를 확장 가능한 관리형 데이터 스토리지 솔루션에 아카이빙하면 비즈니스 통찰력을 확보하고, 데이터를 탐색하며, 시간 경과에 따른 추세와 패턴을 파악할 수 있습니다.

IoT 데이터의 기록 추세를 저장하고 활용하는 것 외에도 디바이스의 현재 상태를 저장하고 모든 디바이스의 현재 상태를 쿼리할 수 있는 기능을 제공하는 시스템이 있어야 합니다. 이러한 시스템이 있으면 내부 분석 팀과 고객에게 IoT 데이터에 대한 보기를 제공할 수 있습니다.

AWS IoT 섀도우 서비스는 디바이스의 가상 표현을 클라우드에 저장하는 효과적인 메커니즘입니다. AWS IoT 디바이스 섀도우는 각 디바이스의 현재 상태를 관리하기에 매우 적합합니다. 또한 운영 목적으로 섀도우를 쿼리해야 하는 내부 팀의 경우 IoT 레지스트리 및 섀도우 메타데이터가 포함된 검색 가능한 인덱스를 제공하는 플릿 인덱싱의 관리형 기능을 활용할 수 있습니다. 소비자 애플리케이션처럼 다수의 외부 사용자에게 인덱스 기반 검색 또는 필터링 기능을 제공해야 하는 경우 IoT 규칙 엔진, Kinesis Data Firehose 및 Amazon ElasticSearch Service 를 함께 사용하여 섀도우 상태를 동적으로 아카이브하고 외부 사용자의 쿼리 액세스 권한을 세분화할 수 있는 형식으로 데이터를 저장합니다.

IOTPERF 4. IoT 디바이스 상태를 저장할 데이터베이스를 선택할 때는 어떻게 합니까?

컴퓨팅

IoT 애플리케이션은 메시지 스트림을 지속적으로 처리해야 하는 높은 수집 흐름에 적합합니다. 따라서 데이터를 저장할 때와 저장하기 전에 스트림 처리를 꾸준히 보강하면서 비즈니스 애플리케이션을 실행할 수 있는 컴퓨팅 서비스를 아키텍처에 선택해야 합니다.

IoT 에서 사용되는 가장 일반적인 컴퓨팅 서비스는 AWS Lambda 입니다. AWS Lambda 를 사용하면 원격 측정 데이터가 AWS IoT Core 또는 AWS IoT Greengrass 에 도달할 때 작업을 호출할 수 있습니다. IoT 전체의 서로 다른 지점에서 AWS Lambda 를 사용할 수 있습니다. AWS Lambda 로 비즈니스 로직을 트리거할 위치를 선택할 때는 특정 데이터 이벤트를 처리해야 하는 시간을 고려하십시오.

Amazon EC2 인스턴스는 다양한 IoT 사용 사례에 사용될 수 있습니다. 관리형의 관계형 데이터베이스 시스템과 웹, 보고 또는 기존 온프레미스 솔루션 호스팅과 같은 다양한 애플리케이션에 EC2 인스턴스를 사용할 수 있습니다.

IOTPERF 5. AWS IoT 이벤트를 처리하기 위한 컴퓨팅 솔루션을 선택할 때는 어떻게 합니까?

분석

비즈니스에서 IoT 솔루션은 주로 현장에 있는 디바이스의 성능 및 사용에 빠르게 대응할 목적으로 구현됩니다. 수신되는 원격 측정을 기반으로 직접적인 조치를 취함으로써 새 제품 또는 기능의 우선 순위를 정하거나 조직 내 워크플로의 운영 효율성을 개선하는 데 필요한 더 많은 정보를 얻을 수 있습니다. 분석 서비스를 선택할 때는 수행하는 분석의 유형에 따라 데이터에 대한 다양한 보기를 제공하는 서비스를 선택해야 합니다. AWS 는 시계열 분석, 실시간 지표, 아카이브 및 데이터 레이크 사용 사례 등 다양한 분석 워크플로에 적합한 여러 서비스를 제공합니다.

IoT 데이터를 사용하면 애플리케이션에서 스트리밍 데이터 메시지를 기반으로 시계열 분석을 생성할 수 있습니다. 기간별 지표를 계산한 다음 다른 AWS 서비스로 데이터를 스트리밍할 수 있습니다.

또한 AWS IoT Analytics 를 사용하는 IoT 애플리케이션에서는 시계열 데이터 스토어에 데이터를 저장하기 전에 데이터를 변환, 보강 및 필터링하는 작업으로 구성되는 관리형 AWS Data Pipeline 을 구현할 수 있습니다. AWS IoT Analytics 를 사용하면 기본적으로 QuickSight 및 Jupyter 노트북을 사용하여 시각화 및 분석을 수행할 수 있습니다.

검토

IOTPERF 6. IoT 애플리케이션의 기록 분석을 기반으로 아키텍처를 변경하려면 어떻게 합니까?

복잡한 IoT 솔루션을 구축하는 경우 비즈니스 결과와는 직접적인 관계가 없는 작업에 많은 시간을 할애하게 될 수 있습니다. 예를 들어 IoT 프로토콜 관리, 디바이스 자격 증명 보호,

디바이스와 클라우드 간에 원격 측정 전송 같은 IoT 의 측면은 중요하지만 가치를 차별화하는 데 직접적으로 연결되지 않습니다. IoT 의 혁신 속도에도 문제가 발생할 수 있습니다.

AWS 는 IoT 에서 흔히 발생하는 당면 과제를 고려하여 새로운 기능과 서비스를 주기적으로 출시합니다. 데이터를 정기적으로 검토하여 새로운 AWS IoT 서비스로 아키텍처의 현재 IoT 공백을 해결할 수 있는지, 중요한 비즈니스 차별화 요소가 아닌 아키텍처 구성 요소를 이러한 서비스로 대체할 수 있는지를 확인하십시오. IoT 데이터를 집계하고 데이터를 저장한 다음 나중에 데이터를 시각화하여 기록 분석을 구현하는 기능을 제공하도록 설계된 서비스를 활용하는 것이 좋습니다. AWS IoT Analytics 및 시간 기반 인덱싱과 같은 서비스와 IoT 디바이스에서 타임스탬프 정보를 전송하는 기능을 함께 활용하면 데이터를 연결된 타임스탬프 정보와 함께 아카이빙할 수 있습니다. AWS IoT Analytics 의 데이터는 디바이스의 추가 IT 또는 OT 운영 및 효율성 로그와 함께 자체 Amazon S3 버킷에 저장될 수 있습니다. 이와 같이 IoT 의 데이터를 아카이브 상태로 유지하고 여기에 더해 시각화 도구를 사용하면 데이터를 기반으로 새로운 AWS 서비스가 제공하는 추가 가치에 대한 의사 결정을 내리고 서비스를 통한 플릿 효율성 개선을 측정할 수 있습니다.

모니터링

IOTPERF 7. IoT 애플리케이션의 엔드투엔드 시뮬레이션 테스트를 어떻게 실행하고 있습니까?

테스트 디바이스로 설정된 프로덕션 디바이스(특정 테스트 MQTT 네임스페이스 사용)를 사용하거나 시뮬레이션된 디바이스를 사용하여 IoT 애플리케이션을 시뮬레이션할 수 있습니다. IoT 규칙 엔진을 사용하여 캡처되는 모든 수신 데이터는 프로덕션에 사용되는 것과 동일한 워크플로를 사용하여 처리됩니다.

엔드투엔드 시뮬레이션의 빈도는 특정 릴리스 주기 또는 디바이스 채택에 따라 결정해야 합니다. 오류 경로(오류 시에만 실행되는 코드)를 테스트하여 솔루션에 오류에 대한 복원력이 있는지 확인해야 합니다. 또한 프로덕션 및 사전 프로덕션 계정에 대해 디바이스 카나리아를 지속적으로 실행해야 합니다. 디바이스 카나리아는 시뮬레이션 테스트 중에 시스템 성능을 보여주는 주요 지표의 역할을 합니다. 테스트 출력을 문서화하고 수정 계획의 초안을 작성해야 합니다. 사용자 수락 테스트를 수행해야 합니다.

IOTPERF 8. IoT 구현에서 성능 모니터링을 어떻게 사용하고 있습니까?

IoT 배포와 관련된 성능 모니터링에는 디바이스, 클라우드 성능 및 스토리지/분석 등 여러 유형의 모니터링이 있습니다. 원격 측정 및 명령 데이터 로그에서 수집된 데이터를 사용하여 적절한 성능 지표를 생성하십시오. 기본적인 성능 추적에서 시작하고 비즈니스 핵심 역량이 확장되면 이러한 지표를 보강합니다.

CloudWatch Logs 지표 필터를 활용하면 정규식 패턴 일치를 통해 IoT 애플리케이션의 표준 출력을 사용자 지정 지표로 변환할 수 있습니다. 애플리케이션의 사용자 지정 지표를 기반으로 CloudWatch 경보를 생성하여 IoT 애플리케이션의 동작을 빠르게 파악해야 합니다.

특정 사물 그룹을 추적하도록 세분화된 로그를 설정하십시오. IoT 솔루션의 개발 중에는 DEBUG 로깅을 활성화합니다. 이렇게 하면 디바이스에서 메시지 브로커 및 규칙 엔진으로 전달되는 각 IoT 메시지에 대한 이벤트의 진행 상황을 분명하게 파악할 수 있습니다. 프로덕션에서는 로깅을 ERROR 및 WARN 으로 변경합니다.

클라우드 계층에 더해 배포 전에 디바이스에서 계측을 실행하여 디바이스가 로컬 리소스를 가장 효율적으로 사용하고 있고 펌웨어 코드로 인해 메모리 누수 같은 원치 않는 시나리오가 발생하지 않는지 확인해야 합니다. 제약된 디바이스에 맞게 고도로 최적화된 코드를 배포하고 임베디드 애플리케이션에서 AWS IoT 에 게시하는 디바이스 진단 메시지를 사용하여 디바이스의 상태를 모니터링합니다.

절충

IoT 솔루션은 운영, 고객 관리, 재무, 영업, 마케팅 등 중요한 엔터프라이즈 기능의 방대한 영역에 걸쳐 풍부한 분석 기능을 구동합니다. 이와 동시에 IoT 솔루션은 엣지 게이트웨이의 효율적인 송신 지점으로 사용될 수도 있습니다. 디바이스의 데이터 및 분석이 클라우드로 푸시되고 클라우드에서 디바이스 게이트웨이로 기계 학습 알고리즘을 가져오는 고도로 효율적인 IoT 구현을 설계할 때는 충분한 고려가 필요합니다.

지정된 네트워크를 통해 개별 디바이스에 지원되는 처리량에는 제약이 있습니다. 데이터의 교환 빈도는 전송 계층과 필요에 따라 데이터를 저장, 집계 및 클라우드로 전송할 수 있는 디바이스의 기능과 균형을 이루어야 합니다. 디바이스의 데이터를 클라우드로 전송하는 시기는 백엔드

애플리케이션에서 데이터를 처리하고 조치를 취하는 데 필요한 시간에 맞춰져야 합니다. 예를 들어 1 초 단위로 데이터를 확인해야 하는 경우에는 디바이스에서 1 초보다 더 잦은 빈도로 데이터를 전송해야 합니다. 반대로, 애플리케이션이 시간당 요금으로 데이터를 읽기만 하는 경우 엣지에서 데이터 요소를 집계하고 30 분마다 데이터를 전송하여 성능을 절충할 수 있습니다.

IOTPERF 9. 비즈니스 및 운영 체제에서 IoT 디바이스의 데이터를 사용할 준비가 되었는지 확인하려면 어떻게 합니까?

IOTPERF 10. 디바이스의 데이터는 얼마나 자주 IoT 애플리케이션으로 전송됩니까?

엔터프라이즈 애플리케이션, 비즈니스 및 운영에서 IoT 원격 측정 데이터를 봐야 하는 속도에 따라 IoT 데이터의 가장 효율적인 처리 지점이 결정됩니다. 하드웨어가 제한되지 않는 네트워크 제약 환경에서는 AWS IoT Greengrass 와 같은 엣지 솔루션을 사용하여 클라우드에서 오프라인으로 데이터를 운영하고 처리하십시오. 네트워크와 하드웨어가 모두 제약된 경우에는 바이너리 형식을 사용하고 유사한 메시지를 단일 요청으로 그룹화하여 메시지 페이로드를 압축할 수 있는 기회를 찾아야 합니다.

시각화를 원한다면 Amazon Kinesis Data Analytics 를 사용하여 거의 실시간으로 데이터를 지속적으로 읽고 처리하고 저장하는 SQL 코드를 신속하게 작성할 수 있습니다. 스트리밍 데이터의 경우 표준 SQL 쿼리를 사용하여 데이터를 변환하고 통찰력을 제공하는 애플리케이션을 구성할 수 있습니다. Kinesis Data Analytics 를 사용하면 스트리밍 분석을 위해 IoT 데이터를 노출할 수 있습니다.

주요 AWS 서비스

성능 효율성을 위한 주요 AWS 서비스는 AWS IoT Core, AWS IoT Device Defender, AWS IoT Device Management, AWS Lambda 및 DynamoDB 를 비롯한 다수의 IoT 서비스와 통합되는 Amazon CloudWatch 입니다. Amazon CloudWatch 는 애플리케이션의 전반적인 성능 및 운영 상태에 대한 가시성을 제공합니다. 다음과 같은 서비스도 성능 효율성을 지원합니다.

선택

디바이스: AWS 하드웨어 파트너는 IoT 애플리케이션의 일부로 사용할 수 있는 프로덕션용 IoT 디바이스를 제공합니다. Amazon FreeRTOS 는 마이크로컨트롤러용 소프트웨어 라이브러리가

포함된 운영 체제입니다. AWS IoT Greengrass 를 사용하면 엣지에서 로컬 컴퓨팅, 메시징, 데이터 캐싱, 동기화 및 ML 을 실행할 수 있습니다.

연결: AWS IoT Core 는 디바이스 통신을 위한 경량의 게시 및 구독 프로토콜인 MQTT 를 지원하는 관리형 IoT 플랫폼입니다.

데이터베이스: Amazon DynamoDB 는 10 밀리초 미만의 지연 시간 요청을 지원하여 IoT 데이터의 다양한 보기를 빠르게 검색할 수 있게 하는 완전관리형 NoSQL 데이터 스토어입니다.

컴퓨팅: AWS Lambda 는 서버를 프로비저닝하지 않고도 애플리케이션 코드를 실행할 수 있는 이벤트 기반 컴퓨팅 서비스입니다. Lambda 는 기본적으로 AWS IoT Core 또는 Amazon Kinesis 및 Amazon SQS 와 같은 업스트림 서비스에서 트리거된 IoT 이벤트와 통합됩니다.

분석: AWS IoT Analytics 는 IoT 원격 측정을 위한 시계열 데이터 스토어를 제공하는 동시에 디바이스 수준 분석을 운영하는 관리형 서비스입니다.

검토: AWS 웹 사이트의 AWS IoT 블로그 섹션은 AWS IoT 의 일부로 새로 출시된 기능에 대해 알아볼 수 있는 리소스입니다.

모니터링: Amazon CloudWatch 지표 및 Amazon CloudWatch Logs 는 기존 모니터링 솔루션에 통합할 수 있는 지표, 로그, 필터, 경보 및 알림을 제공합니다. 이러한 지표를 디바이스 원격 측정으로 보강하여 애플리케이션 모니터링에 사용할 수 있습니다.

절충: AWS IoT Greengrass 와 Amazon Kinesis 는 IoT 애플리케이션의 서로 다른 위치에서 데이터를 집계하고 배치 처리하여 보다 효율적인 컴퓨팅 성능을 제공하는 서비스입니다.

리소스

성능 효율성과 관련된 AWS 모범 사례에 대해 자세히 알아보려면 다음 리소스를 참조하십시오.

설명서 및 블로그

- AWS Lambda 시작하기
- DynamoDB 시작하기
- AWS IoT Analytics 사용 설명서

- Amazon FreeRTOS 시작하기
- AWS IoT Greengrass 시작하기
- AWS IoT 블로그

비용 최적화 기반

비용 최적화 기반에는 전체 수명 주기 동안 시스템을 개선하고 구체화하는 지속적인 프로세스가 포함됩니다. 최초 개념 증명의 초기 설계부터 지속적인 프로덕션 워크로드 운영에 이르는 전체 과정에 이 백서의 사례를 도입하면 비용을 최소화하면서 원하는 비즈니스 결과를 달성할 수 있는 비용 인식 시스템을 구축하고 운영하여 기업의 투자 수익을 최대화할 수 있습니다.

설계 원칙

전반적인 Well-Architected 프레임워크의 비용 최적화 설계 원칙에 더해 클라우드의 IoT 에는 비용 최적화를 위한 1 가지 설계 원칙이 있습니다.

- **제조 비용 절충 관리**: 제조 비용에 있어서는 비즈니스 파트너십의 기준, 하드웨어 구성 요소 선택, 펌웨어 복잡성 및 배포 요구 사항 모두 중요한 역할을 합니다. 이러한 비용을 최소화하면 여러 제품 세대에 걸쳐 제품 출시의 성공 여부를 결정하는 데 도움이 됩니다. 그러나 구성 요소 및 제조업체를 선택할 때 빠른 길을 택한다면 다운스트림 비용이 증가할 수 있습니다. 예를 들어 평판이 좋은 제조업체와 파트너십을 맺으면 다운스트림 하드웨어 오류와 고객 지원 비용이 최소화됩니다. 전용 암호화 구성 요소를 선택하면 BOM(Bill Of Material) 비용이 증가할 수 있지만 다운스트림 제조 및 프로비저닝 복잡성은 줄어듭니다. 부품에 이미 온보드 프라이빗 키 및 인증서가 포함될 수 있기 때문입니다.

정의

클라우드의 비용 최적화에는 4 가지 모범 사례 영역이 있습니다.

1. 비용 효율적인 리소스
2. 수요와 공급 일치

3. 지출 인식
4. 시간별 최적화

최적화를 수행하려면 절충을 고려해야 합니다. 예를 들어 출시 시간의 최적화와 비용 최적화를 절충해야 합니다. 어떤 경우에는 선결제 비용 최적화에 투자하는 것보다 출시 시간을 단축하거나 새로운 기능을 배포하거나 단순히 납기를 준수하는 등 속도를 기준으로 최적화하는 것이 가장 좋습니다. 최적화된 비용의 배포에 대한 벤치마킹에 시간을 쓰기보다 과잉 지출을 하는 것이 더 쉽기 때문에 경험적인 데이터에 기반을 두지 않고 서둘러 결정을 내리게 되는 경우도 있습니다. 이렇게 하면 배포가 과다하게 프로비저닝되고 최적화되지 않습니다. 다음 섹션에는 배포의 초기 및 지속적인 비용 최적화에 대한 기술 및 전략적 지침이 나와 있습니다.

모범 사례

비용 효율적인 리소스

IoT 애플리케이션에서 생성할 수 있는 디바이스 및 데이터의 규모를 고려할 때, 비용 절감의 핵심은 시스템에 적합한 AWS 서비스를 사용하는 것입니다. IoT 아키텍트는 IoT 솔루션의 전체 비용 외에 BOM 비용의 관점에서도 연결을 살펴보는 경우가 많습니다. BOM 을 계산하려면 디바이스의 생애에 걸쳐 IoT 애플리케이션에 대한 연결을 관리하는 데 드는 장기 비용을 예측하고 모니터링해야 합니다. AWS 서비스를 활용하면 초기 BOM 비용을 계산하고, 이벤트 기반의 비용 효율적인 서비스를 활용하며, 아키텍처를 업데이트하여 전체 수명 동안 연결 비용을 지속적으로 낮출 수 있습니다.

리소스의 비용 효율성을 높일 수 있는 가장 간단한 방법은 IoT 이벤트를 배치로 그룹화하고 데이터를 집합적으로 처리하는 것입니다. 그룹으로 이벤트를 처리하면 각 개별 메시지에 대한 전체 컴퓨팅 시간을 줄일 수 있습니다. 집계를 사용하여 데이터를 유지하기 전에 압축하고 아카이브하면 컴퓨팅 리소스를 절약하고 솔루션을 지원하는 데 도움이 됩니다. 이 전략을 활용하면 데이터 손실이나 데이터 쿼리 기능의 저하 없이 전체 스토리지 공간을 줄일 수 있습니다.

COST 1. IoT 플랫폼에서 다른 서비스로 전송되는 배치, 보강 및 집계 데이터에 대한 접근 방식을 선택할 때는 어떻게 합니까?

AWS IoT 는 즉각적인 소비 또는 기록 분석을 위해 데이터를 스트리밍하는 시나리오에 가장 적합합니다. AWS IoT Core 에서 다른 AWS 서비스로 전송되는 데이터를 배치 처리하는 방법에는 여러 가지가 있는데 원시 데이터를 있는 그대로 배치 처리하거나 데이터를 보강한 후 배치 처리할 수 있다는 점에서 차별화됩니다. 수집 중에 또는 수집 직후에 IoT 원격 측정 데이터를 보강, 변환 및 필터링하려는 경우 가장 좋은 방법은 데이터를 Kinesis Data Streams, Kinesis Data Firehose, AWS IoT Analytics 또는 Amazon SQS 로 전송하는 AWS IoT 규칙을 생성하는 것입니다. 이러한 서비스를 사용하면 여러 데이터 이벤트를 한 번에 처리할 수 있습니다.

이 배치 파이프라인에서 원시 디바이스 데이터를 처리할 때는 AWS IoT Analytics 및 Amazon Kinesis Data Firehose 를 사용하여 데이터를 S3 버킷 및 Amazon Redshift 로 전송할 수 있습니다. Amazon S3 의 스토리지 비용을 절감하려면 애플리케이션에서 Amazon S3 Glacier 와 같은 저렴한 스토리지에 데이터를 아카이브하는 수명 주기 정책을 활용하면 됩니다.

수요와 공급 일치

공급과 수요가 최적의 방식으로 일치하면 시스템 비용이 절감됩니다. 그러나 IoT 워크로드는 데이터 버스트에 취약하기 때문에 동적으로 확장 가능하고 피크 용량을 고려하여 리소스를 프로비저닝하는 솔루션이 필요합니다. 이벤트 기반 데이터 흐름을 사용하면 피크 용량에 일치하는 AWS 리소스를 자동으로 프로비저닝한 다음 확장하고 트래픽이 낮은 것으로 알려진 기간에는 축소할 수 있습니다.

다음은 비용 최적화 고려 사항에 중점을 둔 질문입니다.

COST 2. 디바이스 수요에 리소스 공급을 맞추려면 어떻게 합니까?

AWS Lambda 및 API Gateway 와 같은 서버리스 기술은 아키텍처의 확장성 및 복원력을 개선하는 데 도움이 됩니다. 게다가 애플리케이션에서 이러한 서비스를 사용할 때만 요금이 부과됩니다. AWS IoT Core, AWS IoT Device Management, AWS IoT Device Defender, AWS IoT Greengrass 및 AWS IoT Analytics 도 사용량에 따라 비용을 지불하며 유휴 컴퓨팅 파워에

대해서는 요금을 부과하지 않는 관리형 서비스입니다. 관리형 서비스의 이점은 AWS 가 리소스의 자동 프로비저닝을 관리한다는 것입니다. 관리형 서비스를 사용하는 경우 사용자는 AWS 서비스의 한도 증가에 대한 알림을 모니터링하고 설정해야 합니다.

수요와 공급을 일치시키도록 아키텍처를 설계하려면 시간대별 예상 사용량과 초과될 가능성이 가장 높은 한도를 사전에 계획해야 합니다. 이러한 한도 증가를 향후 계획에서 고려합니다.

시간별 최적화
새로운 AWS 기능을 평가하면 디바이스의 성능을 분석하여 비용을 최적화하고 디바이스와 IoT 의 통신 방법을 변경할 수 있습니다.

디바이스 펌웨어 변경을 통해 솔루션 비용을 최적화하려면 AWS IoT 와 같은 AWS 서비스의 요금 구성 요소를 검토하고 해당 서비스에 대한 청구 계측 임계값보다 낮은 위치를 확인한 다음 비용과 성능 중에서 절충해야 합니다.

COST 3. 디바이스와 IoT 플랫폼 간 페이로드 크기를 어떻게 최적화합니까?

IoT 애플리케이션은 최종 디바이스에서 실현할 수 있는 네트워킹 처리량과 IoT 애플리케이션에서 가장 효율적으로 데이터를 처리하는 방법 사이에서 균형을 이루는 지점을 찾아야 합니다. 처음에는 디바이스 제약을 감안하여 IoT 배포에서 데이터 전송을 최적화하는 것이 좋습니다. 디바이스에서 클라우드로 개별 데이터 이벤트를 전송하는 것에서 시작하고 단일 메시지로 여러 이벤트를 배치 처리하는 작업은 최소한만 사용하십시오. 나중에 필요한 경우 직렬화 프레임워크를 사용하여 IoT 플랫폼으로 전송하기 전에 메시지를 압축할 수 있습니다.

비용 관점에서 MQTT 페이로드 크기는 AWS IoT Core 의 비용 최적화에서 중요한 요소입니다. IoT 메시지 요금은 5KB 단위에서 시작하여 최대 128KB 단위로 청구됩니다. 따라서 각 MQTT 페이로드 크기는 5KB 에 최대한 근접하는 것이 좋습니다. 예를 들어 현재 크기가 6KB 인 페이로드는 10KB 인 페이로드와 마찬가지로 비용 효율성이 떨어집니다. 한 메시지가 다른 메시지보다 크더라도 해당 메시지를 게시하는 데 드는 전체 비용은 동일하기 때문입니다.

페이로드 크기를 활용하려면 데이터를 압축하거나 데이터를 메시지로 집계할 수 있는 기회를 찾으십시오.

- 값을 읽기 쉬운 상태로 유지하면서 줄여야 합니다. 5 자리 자릿수가 충분하다면 페이로드에 12 자리 숫자를 사용하지 마십시오.
- IoT 규칙 엔진 페이로드 검사가 필요하지 않은 경우 직렬화 프레임워크를 사용하여 페이로드를 더 작은 크기로 압축할 수 있습니다.
- 데이터 전송 빈도를 줄이고 청구 가능한 단위 내에서 메시지를 집계할 수 있습니다. 예를 들어 초당 2KB 메시지 하나를 전송하는 경우 2 초당 2KB 메시지 2 개를 전송하여 IoT 메시지 비용을 절감할 수 있습니다.

이 접근 방식에는 구현 전에 고려해야 할 절충이 있습니다. 디바이스에 복잡성 또는 지연이 추가되면 처리 비용이 예기치 않게 증가할 수 있습니다. IoT 페이로드에 대한 비용 최적화는 솔루션이 프로덕션으로 이동한 후 수행되어야 하며 데이터 중심의 접근 방식을 사용하여 AWS IoT Core 로 데이터가 전송되는 방식을 변경할 때의 비용 영향을 확인할 수 있습니다.

비용 4. IoT 디바이스의 현재 상태 저장 비용을 최적화하려면 어떻게 합니까?

Well-Architected IoT 애플리케이션은 클라우드에서 디바이스를 가상으로 표현합니다. 이 가상 표현은 관리형 데이터 스토어 또는 특수 IoT 애플리케이션 데이터 스토어로 구성됩니다. 두 경우 모두 최종 디바이스는 디바이스 상태 변경을 IoT 애플리케이션에 효율적으로 전송하는 방식으로 프로그래밍되어야 합니다. 예를 들어 전체 디바이스 상태가 동기화되지 않았고 모든 현재 설정을 보내 조정하는 것이 최선임을 명시하는 펌웨어 로직이 있는 경우에만 전체 디바이스 상태를 전송하도록 디바이스를 프로그래밍해야 합니다. 개별 상태 변경이 발생하는 경우 디바이스에서 이러한 변경 사항을 클라우드로 전송하는 빈도를 최적화할 수 있어야 합니다.

AWS IoT 에서 디바이스 섀도우 및 레지스트리 작업은 1KB 단위로 측정되며 액세스/수정 작업 1 백만 개를 기준으로 요금이 청구됩니다. 섀도우에는 각 디바이스의 원하는 또는 실제 상태가 저장되며 레지스트리는 디바이스의 이름을 지정하고 관리하는 데 사용됩니다.

디바이스 섀도우 및 레지스트리에 대한 비용 최적화 프로세스는 수행되는 작업 수와 각 작업의 크기를 관리하는 데 중점을 둡니다. 작업이 섀도우 및 레지스트리 작업 비용에 민감한 경우 섀도우 작업을 최적화하는 방법을 찾아야 합니다. 예를 들어 섀도우의 경우 보고된 각 변경 사항을 개별적으로 전송하는 대신 보고된 여러 필드를 하나의 섀도우 메시지 업데이트로 통합할 수 있습니다. 섀도우 업데이트를 그룹화하면 업데이트를 서비스에 통합하여 섀도우의 전체 비용을 절감할 수 있습니다.

주요 AWS 서비스

비용 최적화를 지원하는 핵심 AWS 기능은 시스템 비용을 파악하는 데 도움이 되는 비용 할당 태그입니다. 다음은 비용 최적화의 3 가지 영역에서 중요한 역할을 하는 서비스와 기능입니다.

- **비용 효율적인 리소스**: Amazon Kinesis, AWS IoT Analytics 및 Amazon S3 같은 AWS 서비스를 사용하면 여러 IoT 메시지를 단일 요청으로 처리하여 컴퓨팅 리소스의 비용 효율성을 개선할 수 있습니다.
- **공급과 수요 일치**: AWS IoT Core 는 연결, 클라우드에 대한 디바이스 보안, 메시징 라우팅 및 디바이스 상태 관리를 위한 관리형 IoT 플랫폼입니다.
- **시간별 최적화**: AWS 웹 사이트의 AWS IoT 블로그 섹션은 AWS IoT 의 일부로 새로 출시된 기능에 대해 알아볼 수 있는 리소스입니다.

리소스

비용 최적화 관련 AWS 모범 사례에 대해 자세히 알아보려면 다음 리소스를 참조하십시오.

설명서 및 블로그

- AWS IoT 블로그

결론

AWS Well-Architected 프레임워크는 클라우드에서 IoT 애플리케이션을 위한 안정적이고 안전하며 효율적이고 경제적인 시스템을 설계하고 운영하기 위한 기반에 대한 아키텍처 모범 사례를 제공합니다. 이 프레임워크는 기존 또는 제안된 IoT 아키텍처를 검토하는 데 사용할 수 있는 일련의 질문과 각 기반에 대한 일련의 AWS 모범 사례를 제공합니다. 아키텍처에 이 프레임워크를 사용하면 기능적 요구 사항에 초점을 둔 안정적이고 효율적인 시스템을 구축할 수 있습니다.

기고자

다음은 본 문서 작성에 도움을 준 개인 및 조직입니다.

- Olawale Oladehin: 솔루션스 아키텍트 전문가, IoT
- Dan Griffin: 소프트웨어 개발 엔지니어, IoT
- Catalin Vieru: 솔루션스 아키텍트 전문가, IoT
- Brett Francis: 제품 솔루션스 아키텍트, IoT
- Craig Williams: 파트너 솔루션스 아키텍트, IoT
- Philip Fitzsimons: Well-Architected 부문 선임 관리자, Amazon Web Services

문서 개정

날짜	설명
2019 년 12 월	IoT SDK 사용, 부트스트래핑, 디바이스 수명 주기 관리 및 IoT 에 대한 추가 지침을 포함하도록 업데이트됨
2018 년 11 월	초판