AWS Well-Architected フレームワーク

IoT レンズ

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

IoT レンズ: AWS Well-Architected フレームワーク

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

Table of Contents

要約 1
要約 1
はじめに 2
定義 3
設計および製造レイヤー 3
エッジレイヤー 4
プロビジョニングレイヤー 4
通信レイヤー 5
取り込みレイヤー 6
分析レイヤー 6
ストレージサービス 6
分析と機械学習サービス 7
アプリケーションレイヤー 7
管理アプリケーション 7
ユーザーアプリケーション 8
データベースサービス – NoSQL および SQL 8
コンピューティングサービス 8
一般的な設計の原則 10
シナリオ 12
デバイスのプロビジョニング 12
デバイステレメトリ 13
デバイスコマンド 14
AWS IoT Device Shadow サービス 15
ファームウェアの更新 16
Well-Architected フレームワークの柱 18
運用上の優秀性の柱 18
設計の原則 18
定義 20
ベストプラクティス 20
主要な AWS のサービス 24
セキュリティの柱 25
設計の原則 25
定義 26
ベストプラクティス 27

主要な AWS のサービス 37
リソース 39
信頼性の柱 39
設計の原則 39
定義 40
ベストプラクティス 40
主要な AWS のサービス 46
リソース 46
パフォーマンス効率の柱 47
設計の原則 47
定義 48
ベストプラクティス 48
主要な AWS のサービス 55
リソース 56
コスト最適化の柱 57
設計の原則 57
定義 57
ベストプラクティス 58
主要な AWS のサービス 61
リソース 62
まとめ 63
寄稿者 64
改訂履歴 65
注意 66

IoT レンズ - AWS Well-Architected フレームワーク

公開日: 2019 年 12 月 (改訂履歴)

要約

本書では AWS Well-Architected フレームワークの AWS IoT レンズについて説明しています。これにより、お客様はクラウドベースのアーキテクチャを評価および改善し、設計上の決定がビジネスに及ぼす影響をより良く理解できるようになります。このドキュメントでは、一般的な設計原則と、Well-Architected フレームワークの 5 つの柱に関する特定のベストプラクティスとガイダンスについて説明しています。

はじめに

AWS Well-Architected フレームワークは、AWS でシステムを構築する際の決定における利点と欠点を理解するのに役立ちます。信頼性が高く、安全で、効率が良く、費用対効果が#いクラウド対応システムを設計して運#するために、アーキテクチャに関するベストプラクティスをこのフレームワークに従って学ぶことができます。このフレームワークにより、アーキテクチャをベストプラクティスに照らし合わせて一貫的に測定し、改善すべき点を特定する手段を提供します。Well-Architected システムによってビジネスの成功の可能性が大いに高まると当社は確信しています。

この「レンズ」では、AWS クラウドで IoT ワークロード (モノのインターネット) をデザイン、デプロイ、および設計する方法に焦点を当てます。Well-Architected IoT アプリケーションを実装するには、接続された物理的アセット (モノ) の調達から、安全で信頼性の高い自動化された方法でそれらの同じアセットを最終的に廃棄することまで、Well-Architected 原則に従う必要があります。このドキュメントでは、AWS クラウドのベストプラクティスに加えて、物理的アセットをインターネットに接続する際の影響、考慮事項、推奨事項についても説明しています。

このドキュメントでは、Well-Architected フレームワークからの IoT 固有のワークロードの詳細のみを取り上げます。AWS Well-Architected フレームワークホワイトペーパーを読み、その他のベストプラクティスと質問を検討することをお勧めします。

本書は、最高技術責任者 (CTO)、設計者、デベロッパー、技術エンジニア、運用チームメンバーなどの技術担当者向けです。このドキュメントを読むと、IoT アプリケーションに関する AWS のベストプラクティスと戦略を理解できます。

定義

AWS Well-Architected フレームワークは、運用上の優秀性、セキュリティ、信頼性、パフォーマンス効率、コスト最適化という 5 本の柱を基本としています。テクノロジーソリューションを設計するときは、ビジネスの状況に基づいて、柱間のトレードオフを十分な情報に基づいて行う必要があります。IoT ワークロードの場合、AWS では、アプリケーションの堅牢なアーキテクチャを設計できる複数のサービスを提供しています。IoT (モノのインターネット) アプリケーションは、多くのデバイス (またはモノ) で構成されており、エッジベースおよびクラウドベースの補完的なコンポーネントと安全に接続してやり取りし、ビジネス価値をもたらします。IoT アプリケーションは、接続されたデバイスによって生成されたデータを収集、処理、分析、および操作します。このセクションでは、IoT ワークロードを設計するためにこのドキュメント全体で使用される AWS コンポーネントの概要を示します。IoT ワークロードを構築する際に考慮すべき明確な論理レイヤーは 7 つあります。

トピック

- 設計および製造レイヤー
- エッジレイヤー
- プロビジョニングレイヤー
- 通信レイヤー
- 取り込みレイヤー
- 分析レイヤー
- アプリケーションレイヤー

設計および製造レイヤー

設計および製造レイヤーは、製品の概念、ビジネス要件と技術要件の収集、プロトタイピング、モジュールと製品のレイアウトと設計、コンポーネントのソーシング、製造で構成されます。各フェーズで行われた決定は、以下で説明されている IoT ワークロードの次の論理レイヤーに影響します。たとえば、IoT デバイスの作成者の中には、一般的なファームウェアイメージを契約製造元によって焼き付けおよびテストすることを希望するものもあります。この決定により、プロビジョニングレイヤー中に必要なステップが部分的に決まります。

さらに進み、製造中に各デバイスに固有の証明書とプライバシーキーを書き込むことができます。認証情報の種類は、その後のネットワークプロトコルの選択に影響を与える可能性があるため、この決定は通信レイヤーに影響する可能性があります。認証情報の有効期限が切れることがない場合は、発

行元の認証機関の侵害によるデータ損失のリスクが増大する可能性を犠牲にして、通信およびプロビジョニングレイヤーを簡素化できます。

エッジレイヤー

IoT ワークロードのエッジレイヤーは、デバイスの物理ハードウェア、デバイス上のプロセスを管理する組み込みオペレーティングシステム、デバイスファームウェア (IoT デバイスにプログラムされるソフトウェアと手順) で構成されます。エッジは、他の周辺機器を感知し、機能します。一般的なユースケースとしては、エッジデバイスに接続されたセンサーの読み取りや、モーションセンサーがアクティブ化されたときにライトを点灯するなどのユーザーアクションに基づいた周辺機器の状態の変更があります。

AWS IoT デバイス SDK では、プログラミング言語またはプラットフォームに合わせて調整された API を使用して、デバイスやアプリケーションで AWS IoT Core を簡単に使用できます。

Amazon FreeRTOS は、メモリ効率が良く、セキュアな埋め込みライブラリを活用しながら、小型で電力の少ないエッジデバイスをプログラムできる、マイクロコントローラー向けのリアルタイムオペレーティングシステムです。

AWS IoT Greengrass は、IoT デバイスの Linux オペレーションシステムを拡張するソフトウェアコンポーネントです。AWS IoT Greengrass は、デバイス間における MQTT ローカルルーティング、データキャッシング、AWS IoT シャドウ同期、ローカル AWS Lambda 関数、および機械学習アルゴリズムの実行を可能にします。

プロビジョニングレイヤー

IoT ワークロードのプロビジョニングレイヤーは、一意のデバイス ID の作成に使用されるパブリックキーインフラストラクチャ (PKI) と、デバイスに設定データを提供するアプリケーションワークフローで構成されます。プロビジョニングレイヤーは、継続的なメンテナンスと、時間の経過に伴うデバイスの最終廃棄にも関係します。IoT アプリケーションには、IoT アプリケーションでデバイスをスムーズに追加および管理できるように、堅牢で自動化されたプロビジョニングレイヤーが必要です。IoT デバイスをプロビジョニングするときは、一意の暗号化認証情報をデバイスにインストールする必要があります。

X.509 証明書を使用することで、デバイス用の信頼された ID を安全に作成するプロビジョニングレイヤーを実装できます。このレイヤーは、通信レイヤーに対する認証と承認に使用できます。X.509 証明書は、認証機関 (CA) と呼ばれる信頼されたエンティティによって発行されます。X.509 証明書は、メモリと処理要件のために制約のあるデバイス上のリソースを消費しますが、運用上のスケー

ラビリティと標準ネットワークプロトコルによる広範なサポートにより、理想的な ID メカニズムです。

AWS Certificate Manager プライベート CA は、API を使用して IoT デバイスのプライベート証明書のライフサイクルを管理するプロセスを自動化するのに役立ちます。X.509 証明書などのプライベート証明書は、プロビジョニング中に作成され、IoT アプリケーションに対するデバイスのアクセス許可を識別して承認するために使用できる長期的な ID をデバイスに付与するための安全な方法を提供します。

AWS IoT ジャストインタイム登録 (JITR) を使用すると、AWS IoT Core などのマネージド型 IoT プラットフォームで使用するデバイスをプログラムで登録できます。ジャストインタイム登録では、デバイスを AWS IoT Core エンドポイントに初めて接続するときに、証明書 ID の有効性を判断し、付与するアクセス許可を決定するワークフローを自動的にトリガーできます。

通信レイヤー

通信レイヤーは、接続、リモートデバイス間のメッセージルーティング、デバイスとクラウド間のルーティングを処理します。通信レイヤーでは、デバイスによる IoT メッセージの送受信方法や、デバイスがクラウド内で物理的な状態を表して保存する方法を設定できます。

AWS IoT Core では、デバイス間で IoT メッセージをパブリッシュおよびサブスクライブするための MQTT プロトコルの使用をサポートするマネージド型メッセージブローカーが提供されるため、IoT アプリケーションの構築に役立ちます。

AWS IoT デバイスレジストリは、モノの管理と運用に役立ちます。モノは、クラウド内の特定のデバイスまたは論理エンティティを表すものです。また、デプロイ後のアセットの識別、分類、検索に役立つカスタム定義の静的属性を持つこともできます。

AWS IoT Device Shadow サービスを使用すると、特定のデバイスの現在の状態を含むデータストアを作成できます。Device Shadow サービスは、AWS IoT に接続する各デバイスの仮想表現を個別の Device Shadow として維持します。各デバイスのシャドウは、対応するモノの名前によって一意に識別されます。

Amazon API Gateway を使用すると、IoT アプリケーションが HTTP リクエストを送信して IoT デバイスを制御できます。IoT アプリケーションには、リモート技術者用のダッシュボードなどの内部システム用の API インターフェイスと、ホームコンシューマー向けモバイルアプリケーションなどの外部システムが必要です。Amazon API Gateway を使用すると、基盤となるインフラストラクチャをプロビジョニングおよび管理することなく、一般的な API インターフェイスを作成できます。

取り込みレイヤー

IoT の主要なビジネス要因は、デバイスによって作成されたすべての異なるデータストリームを集約し、安全かつ信頼性の高い方法で IoT アプリケーションにデータを送信する機能です。取り込みレイヤーは、データフローとデバイス間の通信を疎結合化しながら、デバイスデータを収集する上で重要な役割を果たします。

AWS IoT ルールエンジンを使用すると、デバイスが AWS のサービスとインタラクションできるように IoT アプリケーションを構築できます。メッセージが受信された MQTT トピックストリームに基づいて、AWS IoT ルールが分析され、アクションが実行されます。

Amazon Kinesis はデータをストリーミングするためのマネージドサービスです。これにより、タイムリーな洞察を得て、IoT デバイスからの新しい情報に迅速に対応できます。Amazon Kinesis は AWS IoT ルールエンジンと直接統合し、MQTT を使用するデバイスの軽量デバイスプロトコルから、他のプロトコルを使用する内部 IoT アプリケーションとのブリッジングをシームレスに実現します。

Kinesis と同様に、Amazon Simple Queue Service (Amazon SQS) を IoT アプリケーションで使用して、通信層をアプリケーション層から疎結合化する必要があります。Amazon SQS は、メッセージの順序が不要な場合において、アプリケーションが IoT アプリケーションを 1 度処理する必要があるときに、イベント駆動型のスケーラブルな取り込みキューを有効にします。

分析レイヤー

IoT ソリューションを実装する利点の 1 つは、ローカル/エッジ環境で起こっていることに関する深い洞察とデータを得られることです。コンテキストに基づくインサイトを実現する主な方法は、IoT データを処理して分析できるソリューションを実装することです。

ストレージサービス

IoT ワークロードは、多くの場合、大量のデータを生成するように設計されています。この個別のデータは、永続的に保存しながら、安全に送信、処理、消費されるようにします。

Amazon S3 は、インターネット上のどこからでも任意の量のデータを保存および取得できるように設計されたオブジェクトベースのストレージです。Amazon S3 を使用すると、規制、ビジネスの進化、メトリクス、縦断研究、分析機械学習、組織のイネーブルメントなど、さまざまな目的で大量のデータを保存する IoT アプリケーションを構築できます。Amazon S3 は、コストの最適化とレイテンシーのためだけではなく、アクセスコントロールとコンプライアンスのためにも、データを管理する方法について幅広い柔軟性を提供します。

分析と機械学習サービス

IoT データが中央のストレージロケーションに到達したら、デバイスの動作に関する分析と機械学習を実装することで、IoT の価値を最大限に引き出すことができます。分析システムでは、分析に基づいてデータ駆動型の決定を行うことで、デバイスファームウェア、エッジロジックやクラウドロジックを改善できます。IoT システムは、分析と機械学習を使用して、予測メンテナンスや異常検出などの積極的な戦略を実装し、システムの効率性を向上させることができます。

AWS IoT Analytics を使用すると、IoT データのボリュームに対して高度な分析を簡単に実行できます。AWS IoT は基礎となる IoT データストアを管理し、独自の分析クエリまたは Jupyter ノートブックを使用してデータのさまざまなマテリアライズドビューを構築します。

Amazon Athena は、標準 SQL を使用して Amazon S3 でデータを簡単に分析できるようにするインタラクティブなクエリサービスです。Athena はサーバーレスであるため、管理するインフラストラクチャはなく、お客様は実行したクエリに対してのみ料金を支払います。

Amazon SageMaker AI は、クラウドとエッジレイヤーで機械学習モデルをすばやく構築、トレーニング、デプロイできる完全マネージド型プラットフォームです。Amazon SageMaker AI を使用すると、IoT アーキテクチャは今後の動作を推測するために、過去のデバイステレメトリのモデルを開発できます。

アプリケーションレイヤー

AWS IoT では、クラウドネイティブアプリケーションが IoT デバイスによって生成されたデータを簡単に消費する方法がいくつかあります。これらの接続機能には、サーバーレスコンピューティング、IoT データのマテリアライズドビューを作成するリレーショナルデータベース、IoT オペレーションの操作、検査、保護、管理のための管理アプリケーションなどがあります。

管理アプリケーション

管理アプリケーションの目的は、デバイスが現場にデプロイされたら、デバイスを操作するためのスケーラブルな方法を作成することです。デバイスの接続状態の検査、デバイス認証情報の正しい設定、現在の状態に基づくデバイスのクエリの実行など、一般的な運用タスクは、システムのトラブルシューティングに必要な可視性を確保するために、起動前に実行する必要があります。

AWS IoT Device Defender は、デバイスフリートの監査、異常なデバイスの動作の検出、セキュリティ上の問題のアラート送信、一般的な IoT セキュリティ問題の調査と軽減を支援する完全マネージド型サービスです。

AWS IoT Device Management では、IoT デバイスを規模に応じて、整理、モニタリング、管理することが容易になります。大規模に、お客様は複数の物理的な場所にまたがるデバイスのフリートを管理しています。AWS IoT Device Management を使用することで、デバイスをグループ化して管理を容易にすることができます。また、Device Management フリートのインデックス作成を使用して、デバイスの現在の状態に対するリアルタイム検索インデックス作成を有効にすることもできます。デバイスグループとフリートのインデックス作成は、更新する必要があるターゲットデバイスを決定する際に Over the Air Updates (OTA) で使用できます。

ユーザーアプリケーション

マネージド型アプリケーションに加えて、他の内部および外部システムでは、さまざまなアプリケーションを構築するために IoT データの異なるセグメントが必要になります。エンドコンシューマービュー、ビジネス運用ダッシュボード、および時間の経過とともに構築するその他のまったく新しいアプリケーションをサポートするには、接続および取り込みレイヤーから必要な情報を受け取り、他のシステムで使用できるようにフォーマットできる他のいくつかのテクノロジーが必要になります。

データベースサービス – NoSQL および SQL

データレイクは、未フォーマットの IoT 生成データのランディングゾーンとして機能しますが、IoT データ上でフォーマットされたすべてのビューをサポートするには、構造化および半構造化データストアでデータレイクを補完する必要があります。これらの目的のために、NoSQL データベースと SQL データベースの両方を活用する必要があります。このタイプのデータベースを使用すると、アプリケーションの個別のエンドユーザーに対して IoT データのさまざまなビューを作成できます。

Amazon DynamoDB は、IoT データ用の高速で柔軟な NoSQL データベースサービスです。IoT アプリケーションでは、お客様は多くの場合、信頼性の高いパフォーマンスとスループットキャパシティーの Auto Scaling を備えた柔軟なデータモデルを必要とします。

Amazon Aurora を使用すると、IoT アーキテクチャは、パフォーマンスが高く、費用対効果の高いオープンソースデータベースに構造化データを保存できます。事前定義された SQL クエリのために他の IoT アプリケーションからデータにアクセスする必要がある場合、リレーショナルデータベースは、取り込みレイヤーのデバイスストリームを最終的なビジネスアプリケーションから分離するためのもう 1 つのメカニズムを提供します。このメカニズムは、データの個別のセグメントに基づいて動作する必要があります。

コンピューティングサービス

多くの場合、IoT ワークロードでは、データの生成、取り込み、または消費/実現時にアプリケーションコードを実行する必要があります。コンピューティングコードをいつ実行する必要があるかにかか

わらず、サーバーレスコンピューティングは非常にコスト効率の高い選択肢です。サーバーレスコンピューティングは、エッジからコア、またコアからアプリケーションや分析まで活用できます。

AWS Lambda では、サーバーのプロビジョニングや管理を行わなくてもコードを実行できるようになります。IoT ワークロードの取り込みの規模により、AWS Lambda はステートレスでイベント指向 IoT アプリケーションをマネージドプラットフォームで実行するのに最適です。

一般的な設計の原則

Well-Architected フレームワークでは、IoT を使用したクラウドでの優れた設計を容易にするために、次の設計原則が識別されます。

- 取り込みを処理から疎結合化: IoT アプリケーションでは、取り込みレイヤーは、高速なストリーミングデバイスデータを処理できる高度なスケーラブルプラットフォームである必要があります。キュー、バッファ、メッセージングサービスを使用して高速取り込みをアプリケーションの処理部分から分離することで、IoT アプリケーションはデータを処理する頻度や関心のあるデータのタイプなど、デバイスに影響を与えることなく、いくつかの決定を下すことができます。
- オフライン動作の設計: 接続の問題や設定ミスなどが原因で、デバイスが予想よりも長期間オフラインになることがあります。長期間オフライン接続を処理するように組み込みソフトウェアを設計し、クラウドでメトリクスを作成して、定期的に通信していないデバイスを追跡します。
- エッジで無駄のないデータを設計し、クラウドを強化: IoT デバイスの性質に制約があるため、初期デバイススキーマは物理デバイスでのストレージと、デバイスから IoT アプリケーションへの効率的な送信用に最適化されます。このため、フォーマットされていないデバイスデータは、クラウドから推測できる静的なアプリケーション情報で強化されないことがよくあります。このような理由から、データがアプリケーションに取り込まれるときは、最初に人間が読み取れる属性でデータを強化し、デバイスがシリアル化されたフィールドを逆シリアル化または展開してから、アプリケーションの読み込み要件をサポートするように調整されたデータストアでデータをフォーマットすることをお勧めします。
- パーソナライゼーションの処理: Wi-Fi 経由でエッジまたはクラウドに接続するデバイスは、デバイスのセットアップ時に実行される最初のステップの 1 つとして、アクセスポイント名とネットワークパスワードを受け取る必要があります。通常、このデータは、機密およびサイト固有であるか、デバイスがまだ接続されていないクラウドからのデータなので、製造中にデバイスに書き込むことはできません。これらの要因により、パーソナライゼーションデータは、概念上アップストリームのデバイスクライアント証明書とプライベートキー、および概念的にはダウンストリームのクラウド提供のファームウェアや設定の更新とは区別されることがよくあります。パーソナライゼーションのサポートは、デバイス自体が直接データ入力のためのユーザーインターフェイスを必要とする可能性があるか、デバイスをローカルネットワークに接続するためのスマートフォンアプリケーションを提供する必要があるため、設計と製造に影響する可能性があります。
- デバイスが定期的にステータスチェックを送信していることを確認する: デバイスが長期間にわたって定期的にオフラインになっている場合でも、デバイスステータス情報を定期的な間隔で IoT アプリケーションに送信するよう設定するアプリケーションロジックがデバイスファームウェアに含まれていることを確認します。アプリケーションは、適切なレベルの可視性を確保するため

に、アクティブな参加者である必要があります。この定期的に発生する IoT メッセージを送信すると、IoT アプリケーションがデバイス全体のステータスの最新のビューを取得し、デバイスが想定期間内に通信しない場合にプロセスを作成することができます。

シナリオ

このセクションでは、IoT アプリケーションに関連する一般的なシナリオを取り上げて、各シナリオが IoT ワークロードのアーキテクチャに与える影響に焦点を当てます。これらの例は網羅的なものではありませんが、IoT の一般的なパターンをカバーしています。各シナリオのバックグラウンド、システムの設計に関する一般的な考慮事項、シナリオの実装方法のリファレンスアーキテクチャについて説明します。

トピック

- デバイスのプロビジョニング
- デバイステレメトリ
- デバイスコマンド
- ファームウェアの更新

デバイスのプロビジョニング

IoT では、デバイスのプロビジョニングは複数の連続したステップで構成されます。最も重要な側面は、各デバイスに一意の ID が与えられ、その後その ID を使用して IoT アプリケーションによって認証される必要があることです。

そのため、デバイスをプロビジョニングするための最初のステップは、ID をインストールすることです。デバイスの設計と製造における決定により、デバイスに本稼働対応ファームウェアイメージがあるかどうか、または顧客に到達するまでに固有のクライアント証明書があるかどうかが決まります。お客様の決定により、本稼働デバイス識別をインストールする前に実行する必要がある追加のプロビジョニング時間ステップがあるかどうかが決まります。

アプリケーションに IoT で X.509 クライアント証明書を使用する — 静的なパスワードよりも安全で、大規模な管理が容易になる傾向があります。AWS IoT Core では、デバイスは証明書と一意のモノの識別子を使用して登録されます。登録されたデバイスは IoT ポリシーに関連付けられます。IoT ポリシーを使用すると、デバイスごとにきめ細かなアクセス許可を作成できます。きめ細かなアクセス許可により、1 つのデバイスだけが独自の MQTT トピックおよびメッセージを操作するアクセス許可を持つことができます。

この登録プロセスにより、デバイスが IoT アセットとして認識され、生成されたデータが AWS IoT を通じて他の AWS エコシステムで使用できるようになります。デバイスをプロビジョニングするに

は、自動登録を有効にし、プロビジョニングテンプレートまたは AWS Lambda 関数を最初のデバイスプロビジョニングイベントに関連付ける必要があります。

この登録メカニズムは、IoT アプリケーション (この場合は AWS IoT) の認証に使用されるプロビジョニング (製造中または製造後に発生) の最中に一意の証明書を受け取るデバイスに依存します。このアプローチの利点の 1 つは、デバイスを別のエンティティに転送し、再プロビジョニングすることで、新しい所有者の AWS IoT アカウントの詳細で登録プロセスを繰り返すことができることです。

図 1: 登録フロー

1. データベースに製造デバイス識別子を設定します。
2. デバイスは API Gateway に接続し、CPM からの登録をリクエストします。リクエストが検証されます。
3. Lambda は、プライベート認証機関 (CA) から X.509 証明書をリクエストします。
4. プロビジョニングシステムによって CA が AWS IoT Core に登録されました。
5. API Gateway はデバイス認証情報をデバイスに渡します。
6. デバイスは AWS IoT Core で登録ワークフローを開始します。

デバイステレメトリ

多くのユースケース (インダストリアル IoT など) では、IoT の価値はマシンの動作に関するテレメトリを収集することにあります。例えば、このデータを使用して、予測メンテナンスを可能にし、コストのかかる予期しない機器の故障を防ぐことができます。テレメトリはマシンから収集し、IoT アプリケーションにアップロードする必要があります。テレメトリを送信するもう 1 つの利点は、クラウドアプリケーションが分析のためにこのデータを使用し、時間の経過とともにファームウェアに対して実行できる最適化を解釈できることです。

テレメトリデータは読み取り専用であり、収集されて IoT アプリケーションに送信されます。テレメトリデータはパッシブであるため、テレメトリメッセージの MQTT トピックが IoT コマンドに関連するトピックと重複しないようにします。例えば、data/device/sensortype というテレメトリトピックでは、「data」で始まる MQTT トピックはテレメトリトピックとみなされます。

論理的な観点から、デバイスデータテレメトリをキャプチャして操作するためのシナリオをいくつか定義しました。

図 2: テレメトリキャプチャのオプション

1. 1 つのパブリッシュトピックと 1 つのサブスクライバー。例えば、1 つのアプリケーションのみがサブスクライブできる 1 つのトピックに明るさのレベルをパブリッシュするスマート電球です。
2. 変数を含む 1 つのパブリッシュトピックと 1 つのサブスクライバー。例えば、類似しているがユニークな複数のトピックに明るさをパブリッシュするスマート電球のコレクションです。各サブスクライバーは、一意のパブリッシュメッセージをリッスンできます。
3. 単一のパブリッシュトピックと複数のサブスクライバー。この場合、家のすべての電球がサブスクライブするトピックにその値をパブリッシュする光センサーです。
4. 複数のパブリッシュトピックと 1 つのサブスクライバー。例えば、モーションセンサーを備えた電球のコレクションです。スマートホームシステムは、モーションセンサーを含むすべての電球トピックにサブスクライブし、明るさとモーションセンサーデータの複合ビューを作成します。

デバイスコマンド

IoT アプリケーションを構築するときは、コマンドを使ってリモートでデバイスを操作する機能が必要です。階層的な業界における例として、リモートコマンドを使用して、機器から特定のデータをリクエストします。スマートホーム業界での使用例として、リモートコマンドを使用してアラームシステムをリモートでスケジュールします。

AWS IoT Core では、MQTT トピックまたは AWS IoT Device Shadow を使用してコマンドを実装し、デバイスにコマンドを送信し、デバイスがコマンドを実行した通知を受け取ることができます。コマンドを実装するには、MQTT トピック経由で Device Shadow を使用します。Device Shadow には、標準の MQTT トピック (clientToken など) を使用してリクエストのオリジン、競合解決を管理するためのバージョン番号、デバイスがオフラインで発行時にコマンドを受信できない場合にクラウドにコマンドを保存する機能などを使用するよりもいくつかの利点があります。デバイスのシャドウは、デバイスが現在オンラインでない場合でも、コマンドをクラウドに保持する必要がある場合によく使用されます。デバイスがオンラインに戻ると、デバイスは最新のシャドウ情報をリクエストし、コマンドを実行します。

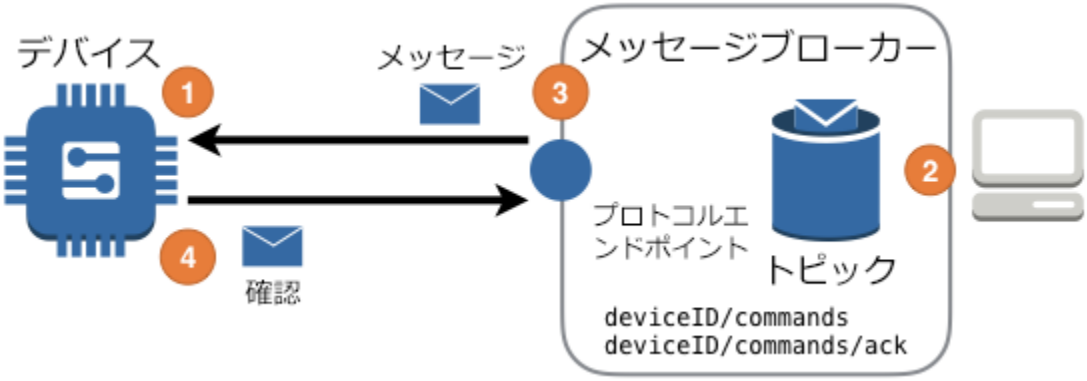

図 3: デバイスにコマンドを送信するためのメッセージブローカーの使用

AWS IoT Device Shadow サービス

AWS IoT Core の Device Shadow サービスを使用する IoT ソリューションでは、コマンドリクエストは、信頼性が高く、スケーラブルでわかりやすい方法で管理されます。Device Shadow サービスは、デバイス関連の状態の管理と状態の変更の伝達方法の両方について、規範的なアプローチに従います。このアプローチでは、Device Shadows サービスが JSON ドキュメントを使用して、デバイスの現在の状態、想定した今後の状態、および現在の状態と必要な状態の違いを保存する方法について説明します。

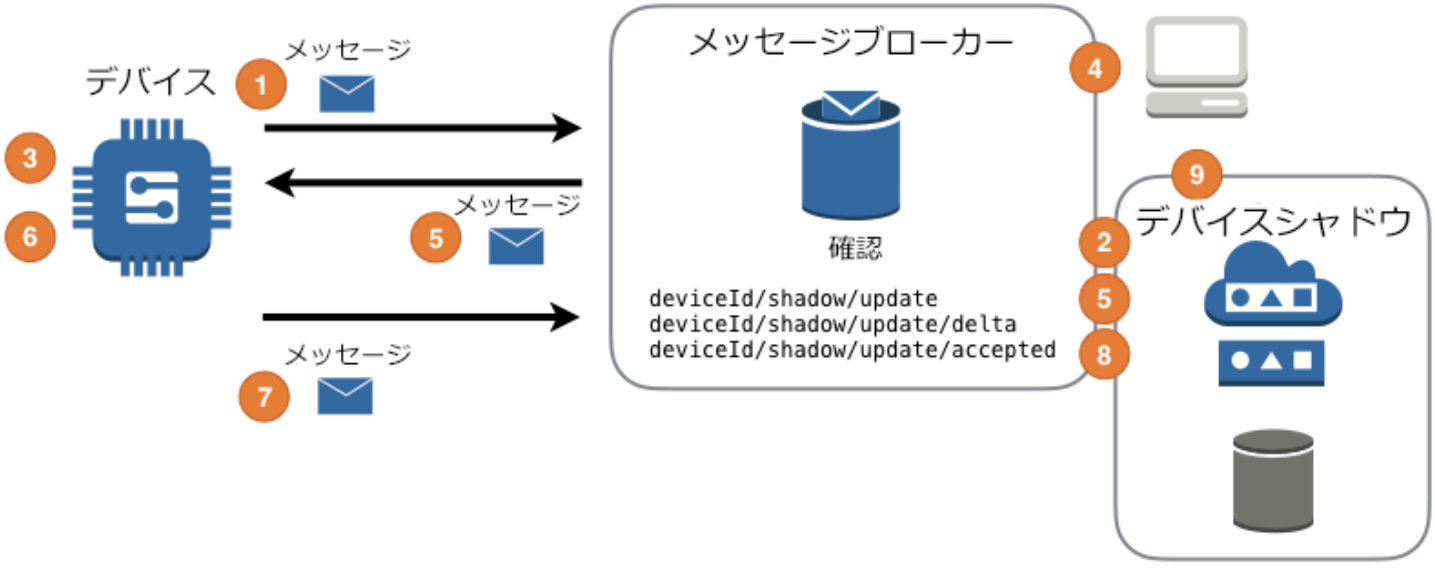

図 4: デバイスでの Device Shadow の使用

1. デバイスは、その状態をメッセージとして更新トピック deviceID/shadow/update にパブリッシュすることで、デバイスの初期状態を報告します。

2. Device Shadow はトピックからメッセージを読み取り、デバイスの状態を永続データストアに記録します。
3. デバイスは、デバイス関連の状態変更メッセージが到着するデルタメッセージングトピック deviceId/shadow/update/delta にサブスクライブします。
4. ソリューションのコンポーネントは、トピック deviceID/shadow/update に目的の状態メッセージをパブリッシュし、このデバイスを追跡する Device Shadow は、永続的なデータストアで目的のデバイスの状態を記録します。
5. Device Shadow は、トピック deviceId/shadow/update/delta にデルタメッセージをパブリッシュし、メッセージブローカーはメッセージをデバイスに送信します。
6. デバイスは差分メッセージを受信し、望ましい状態変更を実行します。
7. デバイスは、新しい状態を反映する確認メッセージを update トピック deviceID/shadow/update にパブリッシュし、このデバイスを追跡する Device Shadow は永続データストアに新しい状態を記録します。
8. Device Shadow は、deviceId/shadow/update/accepted トピックにメッセージをパブリッシュします。
9. ソリューションのコンポーネントは、Device Shadow から更新された状態をリクエストできるようになりました。

ファームウェアの更新

すべての IoT ソリューションでは、デバイスのファームウェアの更新が許可されている必要があります。セキュリティ、スケーラビリティ、新機能の提供には、人間による介入なしでファームウェアのアップグレードをサポートすることが不可欠です。

AWS IoT Device Management は、ファームウェアアップデートの実行とステータスの追跡など、IoT デプロイを管理するための安全で簡単な方法を提供します。AWS IoT Device Management は、AWS IoT メッセージブローカーと AWS IoT ジョブで MQTT プロトコルを使用して、ファームウェア更新コマンドをデバイスに送信するとともに、それらのファームウェア更新のステータスを経時的に受信します。

IoT ソリューションでは、この機能を提供するために、次の図に示す AWS IoT ジョブを使用してファームウェア更新を実装する必要があります。

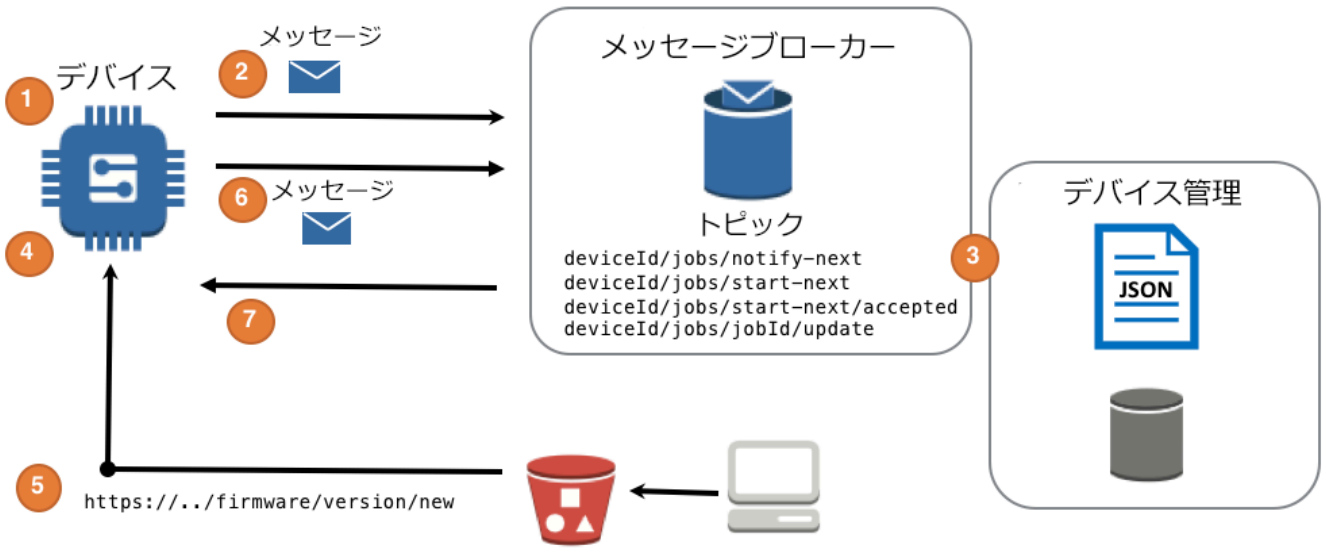

図 5: デバイスのファームウェアの更新

1. デバイスは、IoT ジョブ通知メッセージが到着する IoT ジョブ通知トピック deviceId/jobs/notify-next にサブスクライブします。
2. デバイスは deviceId/jobs/start-next にメッセージをパブリッシュして、次のジョブを開始し、次のジョブ、ジョブドキュメント、および statusDetails に保存された状態を含むその他の詳細を取得します。
3. AWS IoT Jobs サービスは、特定のデバイスの次のジョブドキュメントを取得し、サブスクライブされたトピック deviceId/jobs/start-next/accepted にこのドキュメントを送信します。
4. デバイスは、 deviceId/jobs/jobId/update MQTT トピックを使用してジョブドキュメントで指定されたアクションを実行し、ジョブの進行状況をレポートします。
5. アップグレードプロセス中に、デバイスは Amazon S3 の署名付き URL を使用してファームウェアをダウンロードします。Amazon S3 にアップロードするときに、コード署名を使用してファームウェアに署名します。エンドデバイスは、ファームウェアにコード署名することで、インストール前にファームウェアの信頼性を検証できます。Amazon FreeRTOS デバイスは MQTT 経由でファームウェアイメージを直接ダウンロードできるため、個別の HTTPS 接続が不要になります。
6. デバイスは、成功または失敗を報告するジョブトピック deviceId/jobs/jobId/update に更新ステータスメッセージをパブリッシュします。
7. このジョブの実行ステータスは最終状態に変更されているため、実行可能な次の IoT ジョブ (存在する場合) が変更されます。

Well-Architected フレームワークの柱

このセクションでは、各柱について説明し、AWS IoT のソリューションを設計する際に関連する定義、ベストプラクティス、質問、考慮事項、重要な AWS のサービスについて説明します。

トピック

- 運用上の優秀性の柱
- セキュリティの柱
- 信頼性の柱
- パフォーマンス効率の柱
- コスト最適化の柱

運用上の優秀性の柱

運用上の優秀性の柱には、本番ワークロードの管理に使用される運用プラクティスと手順が含まれます。運用上の優秀性は、計画された変更の実行方法と、予期しない運用イベントへの対応で構成されます。変更の実行と応答は自動化する必要があります。運用上優秀であるすべてのプロセスと手順は、文書化、テスト、および定期的に確認する必要があります。

トピック

- 設計の原則
- 定義
- ベストプラクティス
- 主要な AWS のサービス

設計の原則

Well-Architected フレームワークの運用上の優秀性設計原則に加えて、クラウドでの IoT の運用上の優秀性に関する 5 つの設計原則があります。

- デバイスプロビジョニングの計画: 安全な場所に初期デバイス ID を作成するように、デバイスプロビジョニングプロセスを設計します。一意の証明書を IoT デバイスに配布するパブリックキーインフラストラクチャ (PKI) を実装します。上記のように、事前に生成されたプライベートキーと証明書を持つ暗号化ハードウェアを選択すると、PKI の実行にかかる運用コストが排除されます。そ

れ以外の場合、PKI は、製造プロセスまたはデバイスのブートストラップ中に Hardware Security Module (HSM) を使用してオフラインで実行できます。クラウドで認証機関 (CA) と HSM を管理できるテクノロジーを使用します。

- デバイスのブートストラップを実装: (階層的な業界における) 技術者または (階層的になっている消費者における) ユーザーによるパーソナライズをサポートするデバイスも、プロビジョニングを受けることができます。例えば、Bluetooth LE 経由でデバイスと、Wi-Fi 経由でクラウドとやり取りするスマートフォンアプリケーションです。グローバルに分散されたブートストラップ API を使用して、デバイスが設定情報をプログラムで更新できるように設計する必要があります。ブートストラップ設計により、デバイスの新しい設定をクラウド経由でプログラムで送信できます。これらの変更には、通信する IoT エンドポイント、デバイス全体のステータスを送信する頻度、サーバー証明書などの更新されたセキュリティ設定などの設定が含まれます。ブートストラップのプロセスは初期プロビジョニングにとどまらず、クラウドを通じてデバイス設定を更新するプログラム的な方法を提供することで、デバイス運用において重要な役割を果たします。
- デバイス通信パターンの文書化: IoT アプリケーションでは、デバイスの動作はハードウェアレベルで手動で文書化されます。クラウドでは、運用チームは、デバイスのフリートにデプロイされたデバイスの動作をどのようにスケールするかを策定する必要があります。クラウドエンジニアは、デバイス通信パターンを確認し、デバイスデータの予想されるインバウンドトラフィックとアウトバウンドトラフィックの合計を推定し、クラウドでデバイスのフリート全体をサポートするために必要なインフラストラクチャを判断する必要があります。運用計画では、デバイスおよびクラウド側のメトリクスを使用してこれらのパターンを測定し、システムで想定される使用パターンが満たされるようにする必要があります。
- 無線 (OTA) アップデートの実装: ハードウェアへの長期投資のメリットを得るには、新しい機能を使用してデバイスのファームウェアを継続的に更新できる必要があります。クラウドでは、堅牢なファームウェア更新プロセスを適用できます。これにより、特定のデバイスのファームウェア更新、時間の経過に伴う変更の展開、更新の成功と失敗の追跡、KPI に基づいてファームウェアの変更のロールバックや停止を行うことができます。
- 物理的アセットに機能テストを実装: IoT デバイスのハードウェアとファームウェアは、現場にデプロイする前に厳格なテストを受ける必要があります。受諾と機能テストは、本稼働環境に向けて非常に重要です。機能テストの目的は、ハードウェアのパフォーマンスをプロファイリングしながら、周辺センサーの断続的または減少した接続や障害などの厳格なテストシナリオを通じて、ハードウェアコンポーネント、組み込みファームウェア、デバイスアプリケーションソフトウェアを実行することです。このテストでは、IoT デバイスがデプロイ時に想定どおりに動作することを確認します。

定義

クラウド内での運用の卓越性について 3 つの領域のベストプラクティスがあります。

1. 準備
2. オペレーション
3. 進化

プロセス、ランブック、ゲームの日に関する Well-Architected フレームワークの対象となるものに加えて、IoT アプリケーション内で運用上の優秀性を促進するために調べる必要がある特定の領域があります。

ベストプラクティス

トピック

- 準備
- 運用する
- 進歩する

準備

IoT アプリケーションの場合、さまざまな環境でハードウェアを調達、プロビジョニング、テスト、デプロイする必要があります。それは、運用上の優秀性のための準備を広げ、主に物理デバイスで実行され、クラウドでは実行されないデプロイをカバーする必要があるからです。ビジネス上の成果を測定して改善し、デバイスがこれらのメトリクスを生成して IoT アプリケーションに送信するかどうかを決定するには、運用メトリクスを定義する必要があります。また、さまざまな環境でデバイスがどのように動作するかをシミュレートできる、効率的な機能テストのプロセスを作成し、運用上の優秀性を計画する必要があります。

IoT ワークロードが障害に対して耐障害性を確保する方法、人が介入することなく問題からデバイスを自己復旧する方法、およびクラウドベースの IoT アプリケーションをスケールして接続ハードウェアの負荷が増え続けるニーズを満たす方法を尋ねることが不可欠です。

IoT プラットフォームを使用する場合は、IoT オペレーションを処理するための追加のコンポーネント/ツールを使用できます。これらのツールには、デバイスの動作のモニタリングと検査、接続メトリクスのキャプチャ、一意の ID を使用したデバイスのプロビジョニング、デバイスデータに対する長期分析の実行を可能にするサービスが含まれています。

IOTOPS 1.オペレーションの優先順位を左右する要因とは?

IOTOPS 2.IoT ワークロードのデバイスの運用をサポートする準備ができていることを確認するには、どうすればよいですか?

IOTOPS 3.新しくプロビジョニングされたデバイスが、必要な運用前提条件を満たしていることを確認するには、どうすればよいですか?

IoT とデータセンターの論理セキュリティは、どちらも主にマシンツーマシン認証を必要とする点で似ています。ただし、IoT デバイスは物理的に安全であるとみなされない環境に頻繁にデプロイされる点が異なります。IoT アプリケーションでは、一般的にインターネットを経由するために機密データも必要です。これらの考慮事項により、デバイスが ID を安全に取得する方法、ID を継続的に証明する方法、適切なレベルのメタデータでシードする方法、モニタリングのために整理および分類する方法、適切なアクセス許可セットで有効にする方法を決定するアーキテクチャを持つことが重要です。

スケーラブルな IoT アプリケーションを正常に実行するには、管理プロセスを自動化し、データ駆動型で、以前の、現在および予想されるデバイスの動作に基づいて行う必要があります。IoT アプリケーションは、増分ロールアウトとロールバック戦略をサポートする必要があります。これを運用効率計画の一部にすることで、耐障害性と効率性に優れた IoT アプリケーションを立ち上げることができます。

AWS IoT では、複数の機能を使用して、CA によって署名された個々のデバイス ID をクラウドにプロビジョニングできます。このパスでは、ID を使用してデバイスをプロビジョニングした後、ジャストインタイムプロビジョニング (JITP)、ジャストインタイム登録 (JITR)、または Bring Your Own Certificate (BYOC) を使用して、デバイス証明書をクラウドに安全に登録します。Route 53、Amazon API Gateway、Lambda、DynamoDB などの AWS のサービスを使用すると、簡単な API インターフェイスを作成して、デバイスのブートストラップでプロビジョニングプロセスを拡張できます。

運用する

IoT では、運用状態はクラウドアプリケーションの運用状態にとどまらず、アプリケーションの一部であるデバイスの測定、モニタリング、トラブルシューティング、修復を行う機能が拡張されますが、ローカルでのトラブルシューティングが困難または不可能な場所にリモートにデプロイされます。このリモート運用の要件は、これらのリモートデバイスから送信されるメトリクスを検査、分析、および実行できるようにするために、設計および実装時に考慮する必要があります。

IoT では、デバイスの動作の適切なベースラインメトリクスを確立し、デバイス間で発生する問題を集計および推測できるようにする必要があります。また、クラウドで実行されるだけでなく、デバイスファームウェアの一部も実行される堅牢な修復計画が必要です。一般的なデバイスインタラクションを本番システムに対して直接テストし続けるさまざまなデバイスシミュレーション Canary を実装する必要があります。Device Canary は、運用メトリクスが満たされない場合に調査する可能性のある領域を絞り込むのを支援します。Device Canary を使用して、Canary メトリクスが予想される SLA を下回ったときにプリエンプティブアラームを発生させることができます。

AWS では、AWS IoT Core のデバイスレジストリで物理デバイスごとに AWS IoT のモノを作成できます。レジストリでモノを作成することで、メタデータをデバイスに関連付けたり、デバイスをグループ化したり、デバイスのセキュリティアクセス許可を設定したりできます。AWS IoT のモノは、モノの関連付けられた Device Shadow に動的デバイスデータを保存しながら、モノのレジストリに静的データを保存するために使用する必要があります。デバイスのシャドウは、デバイスの状態情報の保存と取得に使用される JSON ドキュメントです。

運用プロセスの一環として、デバイスレジストリでデバイスの仮想表現を作成することに加えて、IoT デバイスを定義する同様の静的属性をカプセル化するモノのタイプを作成する必要があります。モノのタイプは、デバイスの製品分類に似ています。モノ、モノのタイプ、Device Shadow の組み合わせは、IoT オペレーションで使用される重要なメタデータを保存するための最初のエントリポイントとして機能します。

AWS IoT では、モノのグループを使用してデバイスをカテゴリ別に管理できます。グループには他のグループを含めることもできます。これにより、階層を構築できます。 IoT アプリケーションの組織構造により、関連するデバイスをデバイスグループ別にすばやく特定して対応できます。クラウドを活用すると、ビジネスロジックとデバイスのライフサイクルに基づいて、グループからのデバイスの追加や削除を自動化できます。

IoT では、デバイスはレジストリまたはデバイスのシャドウに保存されていないテレメトリまたは診断メッセージを作成します。代わりに、これらのメッセージは多数の MQTT トピックを使用して AWS IoT に配信されます。このデータを実行可能にするには、AWS IoT ルールエンジンを使用して

エラーメッセージを自動修復プロセスにルーティングし、IoT メッセージに診断情報を追加します。エラーステータスコードを含むメッセージをカスタムワークフローにルーティングする方法の例を以下に示します。ルールエンジンはメッセージのステータスを検査し、エラーの場合は Step Function ワークフローを開始して、エラーメッセージの詳細ペイロードに基づいてデバイスを修復します。

```
{
    "sql": "SELECT * FROM 'command/iot/response WHERE code = 'error'",
    "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"
        }
    }]
}
```

クラウドアプリケーションの運用上のインサイトをサポートするには、AWS IoT Core のデバイスブローカーから収集されたすべてのメトリクスのダッシュボードを生成します。これらのメトリクスは CloudWatch メトリクスを通じて利用できます。さらに、CloudWatch Logs には、インバウンド成功したメッセージの合計、アウトバウンドメッセージ、接続の成功、エラーなどの情報が含まれます。

本番用デバイスのデプロイを補強するには、複数の AWS リージョンにまたがるデバイス Canary として Amazon Elastic Compute Cloud (Amazon EC2) で IoT シミュレーションを実装します。これらのデバイス Canary は、長時間実行されるトランザクションなどのエラー条件のシミュレーション、テレメトリの送信、制御オペレーションの実装など、いくつかのビジネスユースケースをミラーリングします。デバイスシミュレーションフレームワークは、成功、エラー、レイテンシー、デバイスの順序など、広範なメトリクスを出力し、すべてのメトリクスをオペレーションシステムに送信する必要があります。

AWS IoT では、カスタムダッシュボードに加えて、Thing Registry および Device Shadow サービスから、AWS IoT フリートインデックス作成などの検索機能を通じて、フリートレベルおよびデバイスレベルのインサイトが提供されます。フリート全体を検索する機能により、IoT の問題を診断するための運用オーバーヘッドが軽減されます。デバイスレベルまたはフリート全体のレベルで発生したかどうかは関係ありません。

進歩する

IOTOPS 4.ダウンストリームの IoT デバイスへの影響を最小限に抑えて IoT アプリケーションを進化させるにはどうすればよいでしょうか?

IoT ソリューションには、低電力デバイス、リモートロケーション、低帯域幅、断続的なネットワーク接続の組み合わせが頻繁に関係します。これらの各要因は、ファームウェアのアップグレードなど、通信の課題となります。したがって、ダウンストリームデバイスおよび運用への影響を最小限に抑える IoT 更新プロセスを組み込んで実装することが重要です。デバイスは、ダウンストリームへの影響を減らすことに加えて、断続的なネットワーク接続や停電など、ローカル環境に存在する一般的な課題に対して回復力が必要です。デプロイのために IoT デバイスをグループ化し、一定期間にわたるファームウェアアップグレードを驚異的に組み合わせて使用します。フィールドで更新されるデバイスの動作をモニタリングし、デバイスの割合が正常にアップグレードされた後にのみ続行します。

AWS IoT Device Management を使用して、デバイスのデプロイグループを作成し、無線による更新 (OTA) を特定のデバイスグループに配信します。アップグレード中、すべての CloudWatch Logs、テレメトリ、IoT デバイスのジョブメッセージを収集し、その情報を、アプリケーションの全体的な状態や長期実行の Canary のパフォーマンスを測定するために使用される KPI と組み合わせます。

ファームウェアの更新前と更新後に、ビジネスにまたがる参加者との運用メトリクスの遡及分析を行い、改善の機会と方法を決定します。AWS IoT Analytics や AWS IoT Device Defender などのサービスは、デバイス全体の動作の異常を追跡し、更新されたファームウェアの問題を示す可能性のあるパフォーマンスの逸脱を測定するために使用されます。

主要な AWS のサービス

IoT アプリケーションの運用上の優秀性を実現するために、いくつかのサービスを使用できます。AWS デバイス認定プログラムは、AWS IoT の相互運用性のために設計およびテストされたハードウェアコンポーネントを選択するのに役立ちます。適格なハードウェアにより、市場投入を迅速化し、運用上の摩擦を軽減できます。AWS IoT Core は、デバイスの初期オンボーディングを管理するために使用される機能を提供します。AWS IoT Device Management は、デバイスのグループ化や検索など、フリート全体にまたがる操作を実行する際の運用オーバーヘッドを削減します。さらに、Amazon CloudWatch は IoT メトリクスのモニタリング、ログの収集、アラートの生成、レスポンスのトリガーに使用されます。3 つの分野の運用の優秀性をサポートするサービスや機能は、次のとおりです。

- 準備: AWS IoT Core では、現場でのデバイスのプロビジョニングとオンボーディングがサポートされています。これには、ジャストインタイムプロビジョニング、ジャストインタイム登録、自分の証明書を使用したデバイス ID の登録が含まれます。その後、デバイスは、デバイスレジストリと Device Shadow を使用して、メタデータとデバイスの状態に関連付けることができます。
- 運用: AWS IoT モノのグループとフリートインデックス作成により、デバイスの組織構造をすばやく開発し、デバイスの現在のメタデータを検索して繰り返しデバイス運用を実行できます。Amazon CloudWatch では、デバイスとアプリケーションの運用状態をモニタリングできます。
- 応答: AWS IoT ジョブを使用すると、ファームウェアの更新やデバイス設定など、更新を 1 つ以上のデバイスにプロアクティブにプッシュできます。AWS IoT ルールエンジンを使用することで、最もきめ細かなレベルで、AWS IoT Core が IoT メッセージを受信するとそれを検査し、データに直ちに応答できます。AWS IoT Analytics と AWS IoT Device Defender では、AWS IoT Analytics を使用したリアルタイム分析、および Device Defender を使用したリアルタイムのセキュリティとデータしきい値に基づいて、通知や修復を事前にトリガーできます。

セキュリティの柱

セキュリティの柱には、ビジネス価値の提供と同時に、情報、システム、アセットを保護する機能が含まれています。

トピック

- 設計の原則
- 定義
- ベストプラクティス
- 主要な AWS のサービス
- リソース

設計の原則

Well-Architected フレームワークのセキュリティ設計原則全体に加えて、IoT セキュリティの具体的な設計原則があります。

- デバイスのセキュリティライフサイクルを全体的に管理: データセキュリティは設計フェーズから始まり、ハードウェアとデータの廃棄と破棄で終わります。競争優位性を維持し、顧客の信頼を維

持するために、IoT ソリューションのセキュリティライフサイクルにエンドツーエンドのアプローチを取ることが重要です。

- 最小限の権限を確認: すべてのデバイスには、デバイスが通信に使用できるトピックを制限する、きめ細かなアクセス許可が必要です。アクセスを制限することで、侵害された 1 つのデバイスが、他のデバイスに影響を与える機会を減らすことができます。
- 保管時のデバイス認証情報を保護: デバイスは、専用の暗号化要素やセキュアフラッシュなどのメカニズムを使用して、保管時の認証情報を安全に保存する必要があります。
- デバイス ID ライフサイクル管理を実装: デバイスは、作成からライフサイクルの終了まで、デバイス ID を維持します。適切に設計された ID システムは、デバイスの ID を追跡し、ID の有効性を追跡し、時間の経過とともに IoT アクセス許可を積極的に拡張または取り消します。
- データセキュリティの全体像を把握: リモートでデプロイされた多数のデバイスを含む IoT のデプロイは、データの盗難やプライバシーの損失に対する大きな攻撃対象領域となります。Open Trusted Technology Provider Standard などのモデルを使用して、リスクについてサプライチェーンとソリューション設計を体系的に確認し、適切な緩和策を適用します。

定義

クラウド内でのセキュリティには、5 つのベストプラクティス領域があります。

1. Identity and Access Management (IAM)
2. 発見的統制
3. インフラストラクチャ保護
4. データ保護
5. インシデント対応

インフラストラクチャとデータ保護には、IoT デバイスのハードウェアとエンドツーエンドのソリューションが含まれます。IoT の実装では、デバイスがハードウェアセキュリティのベストプラクティスを実装するようにセキュリティモデルを拡張する必要があります。また、IoT アプリケーションは、適切にスコープされたデバイスのアクセス許可や発見的統制などの要因について、セキュリティのベストプラクティスに従う必要があります。

セキュリティの柱では、情報とシステムの保護に焦点が当てられています。重要なトピックには、データの機密性と整合性、誰が何を実行できるかを権限管理で特定および管理、システムの保護、セキュリティイベントを検出するための統制の確立があります。

ベストプラクティス

トピック

- Identity and Access Management (IAM)
- 発見的統制
- インフラストラクチャ保護
- データ保護
- インシデント対応

Identity and Access Management (IAM)

IoT デバイスは多くの場合、信頼された ID でプロビジョニングされ、戦略的な顧客やビジネスデータ (ファームウェア自体など) を保存またはアクセスでき、インターネット経由でリモートにアクセスでき、直接的な物理的改ざんを受けやすいため、ターゲットとなります。不正アクセスに対する保護を提供するには、常にデバイスレベルでセキュリティを実装することから始める必要があります。ハードウェアの観点からは、デバイス上の機密情報の改ざんの攻撃対象領域を減らすために実装できるメカニズムがいくつかあります。

- ハードウェア暗号化モジュール
- セキュアフラッシュを含むソフトウェアサポートソリューション
- クローンできない物理関数モジュール
- PKCS #11 および TLS 1.2 を含む最新の暗号化ライブラリと標準

デバイスハードウェアを保護するには、プライベートキーと機密性の高い ID が固有であり、安全なハードウェアの場所にあるデバイスにのみ保存されるソリューションを実装します。AWS IoT との通信に使用されるプライベートキーへのアクセスを安全に保存および管理するハードウェアまたはソフトウェアベースのモジュールを実装します。ハードウェアのセキュリティに加えて、IoT デバイスには有効な ID が与えられなければなりません。この ID は IoT アプリケーションでの認証と承認に使用されます。

デバイスの有効期間中は、証明書の更新と失効を管理できる必要があります。デバイス上の証明書情報の変更を処理するには、まずフィールドでデバイスを更新できる必要があります。ハードウェアでファームウェアの更新を実行する機能は、Well-Architected IoT アプリケーションにとって重要な基盤となります。OTA 更新により、認証機関を含め、有効期限が切れる前にデバイス証明書を安全にローテーションします。

IOTSEC 1.デバイス証明書とデバイスのプライベートキーを安全に保存するにはどうすればよいですか?

IOTSEC 2.AWS IoT ID をデバイスに関連付けるにはどうすればよいですか?

例えば、AWS IoT では、まず X.509 証明書をプロビジョニングしてから、IoT への接続、メッセージのパブリッシュとサブスクライブ、および更新の受信のための IoT アクセス許可を個別に作成します。この ID とアクセス許可の分離により、デバイスのセキュリティを管理する柔軟性が得られます。アクセス許可の設定中に、各デバイスの MQTT アクションへのアクセスを制限する IoT ポリシーを作成することで、どのデバイスにも適切なレベルの ID と適切なレベルのアクセスコントロールがあることを確認できます。

AWS IoT で各デバイスに固有の X.509 証明書があり、デバイスが証明書を共有しないようにします (1 つのデバイスルールに 1 つの証明書)。デバイスごとに 1 つの証明書を使用することに加えて、AWS IoT を使用する場合、各デバイスには IoT レジストリに独自のモノが必要です。また、モノ名は MQTT 接続用の MQTT ClientID のベースとして使用されます。

1 つの証明書が AWS IoT Core で独自のモノとペアになっているこの関連付けを作成することで、侵害された証明書が誤って別のデバイスの ID を引き受けることを防ぐことができます。また、MQTT ClientID とモノの名前が一致すると、ClientID ログメッセージをその特定の通信部分に関連付けられているモノに関連付けることができるため、トラブルシューティングと修復が軽減されます。

デバイス ID の更新をサポートするには、OTA 通信とバイナリをデバイスに配信するためのマネージドプラットフォームである AWS IoT ジョブを使用します。AWS IoT ジョブは、AWS IoT に接続された 1 つ以上のデバイスに送信および実行される一連のリモート運用を定義するために使用されます。AWS IoT ジョブは、デフォルトで、相互認証と承認、更新の進行状況のデバイス追跡、特定の更新のフリート全体のメトリクスなど、いくつかのベストプラクティスを統合します。

AWS IoT Device Defender 監査を有効にして、デバイスの設定、デバイスポリシー、期限切れの証明書のチェックを自動化された方法で追跡します。例えば、Device Defender はスケジュールに基づいて監査を実行し、証明書の有効期限が切れる通知をトリガーできます。失効した証明書または失効保留中の証明書の通知を受信すると、証明書を事前に更新できる OTA を自動的にスケジュールできます。

IOTSEC 3.IoT アプリケーションへのユーザーアクセスの認証と承認はどのように行いますか?

多くのアプリケーションは IoT のモノの側面に重点を置いていますが、IoT のほぼすべての階層的な市場では、デバイスとの通信やデバイスからの通知の受信を必要とする人間のコンポーネントもあります。例えば、コンシューマー IoT では、通常、ユーザーがデバイスをオンラインアカウントに関連付けることでデバイスをオンボードする必要があります。インダストリアル IoT では、通常、ほぼリアルタイムでハードウェアテレメトリを分析する機能が必要です。いずれの場合も、アプリケーションが、特定のデバイスとのやり取りを必要とするユーザーを識別、認証、および承認する方法を決定することが不可欠です。

IoT アセットへのユーザーアクセスの制御は、ID から始まります。IoT アプリケーションには、ユーザーの ID と、その ID を使用してユーザーが認証する方法を追跡するストア (通常はデータベース) が必要です。ID ストアには、承認時に使用できる追加のユーザー属性 (ユーザーグループのメンバーシップなど) が含まれる場合があります。

IoT デバイスのテレメトリデータは、セキュリティ保護可能なアセットの一例です。このように処理することで、各ユーザーが持つアクセスを制御し、個々のユーザーのやり取りを監査できます。

AWS を使用して IoT アプリケーションユーザーを認証および承認する場合、ID ストアを実装するためのいくつかのオプションと、そのストアでユーザー属性を維持する方法があります。独自のアプリケーションでは、ID ストアに Amazon Cognito を使用します。Amazon Cognito は、承認の決定を行うためにアプリやその他の AWS のサービスによって直接使用される方法で、ID を表現し、ユーザーを認証する標準的なメカニズムを提供します。 AWS IoT を使用する場合、Amazon Cognito ID プール、AWS IoT ポリシー、AWS IoT カスタムオーソライザーなど、いくつかの ID および承認サービスから選択できます。

ユーザーのテレメトリの疎結合化ビューを実装するには、AWS AppSync や Amazon API Gateway などのモバイルサービスを使用します。これらの AWS のサービスでは、IoT データストリームをユーザーのデバイスデータ通知ストリームから疎結合化する抽象化レイヤーを作成できます。例えば、中間データストアで外部ユーザー用にデータの個別のビューを作成することによって可能です。Amazon DynamoDB または Amazon ElasticSearch Service では、AWS AppSync を使用して、中間ストアで許可されているデータのみに基づいてユーザー固有の通知を受信できます。AWS AppSync で外部データストアを使用することに加えて、IoT データの特定のビューを外部ユーザーにプッシュするために使用できるユーザー固有の通知トピックを定義できます。

外部ユーザーが AWS IoT エンドポイントに直接通信する必要がある場合は、そのユーザー ID が、承認された Amazon Cognito ロールおよびきめ細かい IoT ポリシーに関連付けられ承認された

Amazon Cognito フェデレーティッド ID であることを確認するか、AWS IoT カスタムオーソライザーを使用して承認が独自の承認サービスによって管理されることを確認します。どちらの方法でも、MQTT 通信に関して、ユーザーによる接続、パブリッシュ、サブスクライブ、およびメッセージ受信を制限する、きめ細かなポリシーを各ユーザーに関連付けます。

IOTSEC 4.IoT アプリケーションと通信するプリンシパルに、最小限の特権が適用されるようにするには、どうすればよいですか?

デバイスを登録してその ID を確立した後、モニタリング、メトリクス、テレメトリー、またはコマンドと制御に必要な追加のデバイス情報をシードする必要がある場合があります。各リソースには、アクセスコントロールルールの独自の割り当てが必要です。デバイスまたはユーザーがアプリケーションに対して実行できるアクションを減らし、各リソースを個別に保護することで、1 つの ID またはリソースが誤って使用された場合に発生する可能性のある影響を制限できます。

AWS IoT では、IoT レジストリの一貫した命名規則を使用して、きめ細かなアクセス許可を作成します。最初の規則では、MQTT ClientID および AWS IoT モノの名前と同じ一意の識別子を使用します。これらのすべての場所で同じ一意の識別子を使用することで、AWS IoT Thing Policy 変数を使用してすべてのデバイスに適用できる IoT アクセス許可の初期セットを簡単に作成できます。2 番目の命名規則は、デバイスの一意の識別子をデバイス証明書に埋め込むことです。このアプローチを続行して、証明書のサブジェクト名に CommonName として一意の識別子を保存し、証明書ポリシー変数を使用して IoT アクセス許可を各一意のデバイス認証情報にバインドします。

ポリシー変数を使用することで、最小限の権限を維持しながら、すべてのデバイス証明書に適用できるいくつかの IoT ポリシーを作成できます。例えば、以下の IoT ポリシーでは、デバイスの一意の識別子 (共通名で保存) を MQTT ClientID として使用し、証明書がデバイスにアタッチされている場合のみ、すべてのデバイスが接続するように制限されます。また、このポリシーは、個々のシャドウでのみパブリッシュするようにデバイスを制限します。

```
{
"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 を使用して、デバイス ID (証明書または Amazon Cognito フェデレーティッド ID) を AWS IoT レジストリのモノにアタッチします。

これらのシナリオは、独自のトピックおよび Device Shadow のセットと通信する 1 つのデバイスに適用されますが、1 つのデバイスが他のデバイスの状態またはトピックに基づいて動作する必要があるシナリオもあります。例えば、業界設定でエッジアプライアンスを運用する、ホームゲートウェイを作成してホームで調整オートメーションを管理する、またはユーザーが特定のロールに基づいて異なるデバイスセットにアクセスできるようにするなどです。このようなユースケースでは、グループ識別子やエッジゲートウェイの ID などの既知のエンティティを、ゲートウェイと通信するすべてのデバイスのプレフィックスとして活用します。すべてのエンドポイントデバイスに同じプレフィックスを使用させることで、IoT ポリシーでワイルドカード「*」を使用できます。このアプローチでは、MQTT トピックのセキュリティと管理可能性のバランスを取ります。

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

前の例では、IoT 演算子は、識別子 edgegateway123 を持つエッジゲートウェイにポリシーを関連付けます。このポリシーのアクセス許可により、エッジアプライアンスは、エッジゲートウェイによって管理される他の Device Shadow にパブリッシュできるようになります。これは、ゲートウェイに

接続されたすべてのデバイスに、ゲートウェイの識別子がプレフィックスとして付けられたモノの名前があることを強制することによって実現されます。例えば、ダウンストリームモーションセンサーは edgegateway123-motionsensor1 という識別子を持つため、アクセス許可を制限しながらエッジゲートウェイで管理できるようになりました。

発見的統制

IoT アプリケーションのデータ、メトリクス、ログの規模により、集約とモニタリングは Well-Architected IoT アプリケーションの不可欠な部分です。許可されていないユーザーは、IoT アプリケーションのバグを調査し、個々のデバイスを活用して、他のデバイス、アプリケーション、クラウドリソースにさらにアクセスすることを検討します。IoT ソリューション全体を運用するには、個々のデバイスだけでなく、アプリケーション内のデバイス群全体の検出制御を管理する必要があります。デバイスレベルおよびフリート全体のレベルで問題を検出するには、複数のレベルのログ記録、モニタリング、アラートを有効にする必要があります。

Well-Architected IoT アプリケーションでは、IoT アプリケーションの各レイヤーがメトリクスとログを生成します。少なくとも、アーキテクチャには、物理デバイス、デバイスの接続動作、デバイスごとのメッセージの入出力レート、プロビジョニングアクティビティ、承認試行、アプリケーション間のデバイスデータの内部ルーティングイベントに関連するメトリクスとログが必要です。

IOTSEC 5: クラウドとデバイス間でアプリケーションログとメトリクスをどのように分析していますか?

AWS IoT では、AWS IoT Device Defender、CloudWatch Logs、および CloudWatch メトリクスを使用して発見的統制を実装できます。AWS IoT Device Defender では、デバイスの動作やデバイスの接続動作に関連するログとメトリクスを処理します。また、AWS IoT Device Defender では、デバイスや AWS IoT Core からのセキュリティメトリクスを継続的にモニタリングして、各デバイスに適切な動作として定義したものからの逸脱を確認することもできます。

デバイスの動作または接続動作が通常のアクティビティから逸脱した場合のデフォルトのしきい値セットを設定します。

Amazon CloudWatch メトリクス、AWS IoT Coreによって生成された Amazon CloudWatch Logs、および Amazon GuardDuty を使用して、Device Defender メトリクスを拡張します。これらのサービスレベルのログは、AWS IoT プラットフォームサービスおよび AWS IoT Core プロトコルの使用に関連するアクティビティについての重要な洞察を提供するだけでなく、エンドツーエンドの IoT アプリケーションの重要なコンポーネントである AWS で実行されているダウンストリームアプリケー

ションについての洞察も提供します。すべての Amazon CloudWatch Logs は、すべてのソースにわたって口グ情報を関連付けるために一元的に分析する必要があります。

IOTSEC 6: IoT アプリケーションで無効な ID をどのように管理していますか?

セキュリティ ID は、IoT アプリケーションに対するデバイスの信頼と承認の重要なポイントです。証明書などの無効な ID を集中管理できることが重要です。無効な証明書は、失効、失効、非アクティブにすることができます。Well-Architected アプリケーションの一部として、無効な証明書をすべてキャプチャするプロセスと、証明書トリガーの状態に基づく自動応答が必要です。無効な証明書のイベントをキャプチャする機能に加えて、デバイスには IoT プラットフォームへの安全な通信を確立するための二次的な手段も必要です。前述のとおり、デバイスに 2 つの形式の ID が使用されるブートストラップパターンを有効にすることで、無効な証明書を検出し、デバイスまたは管理者が修復のために信頼された安全な通信を確立するためのメカニズムを提供するための、信頼性の高いフォールバックメカニズムを作成できます。

Well-Architected IoT アプリケーションは、失効したすべてのデバイス証明書または認証機関 (CA) を追跡する証明書失効リスト (CRL) を確立します。オンボーディングデバイスに独自の信頼できる CA を使用し、IoT アプリケーションに定期的に CRL を同期します。IoT アプリケーションは、有効でなくなった ID からの接続を拒否する必要があります。

AWS では、PKI 全体をオンプレミスで管理する必要はありません。AWS Certificate Manager (ACM) プライベート認証機関を使用して、クラウドで CA をホストします。または、APN パートナーと協力して、IoT デバイスのハードウェア仕様に事前設定されたセキュア要素を追加できます。ACM には、失効した証明書を S3 バケットのファイルにエクスポートする機能があります。同じファイルを使用して、AWS IoT Core に対する証明書をプログラムで取り消すことができます。

証明書のもう 1 つの状態は、有効期限が近いがまだ有効であることです。クライアント証明書は、少なくともデバイスのサービス有効期間にわたって有効である必要があります。有効期限に近いデバイスを追跡し、OTA プロセスを実行して、証明書を更新して後で有効期限のある新しい証明書に更新し、監査目的で証明書のローテーションが必要になった理由に関する口グ情報を提供するのはお客様の IoT アプリケーション次第です。

証明書と CA の有効期限に関連する AWS IoT Device Defender 監査を有効にします。Device Defender は、30 日以内に有効期限が切れるように設定された証明書の監査口グを作成します。証明書が有効でなくなる前にデバイスをプログラムで更新するには、このリストを使用します。独自の有効期限ストアを構築して、証明書の有効期限を管理し、プログラムによってデバイス証明書の交換または更新のための OTA のクエリ、識別、トリガーすることもできます。

インフラストラクチャ保護

設計時間は、デバイスとソリューションのライフサイクル全体にわたるインフラストラクチャ保護のセキュリティ要件を検討するための理想的なフェーズです。デバイスをインフラストラクチャの拡張として検討することで、デバイスのライフサイクル全体が、インフラストラクチャ保護の設計に与える影響を考慮できます。コストの観点からは、設計フェーズで行われた変更は、後で加えられた変更よりも安価です。有効性の観点からは、設計時に実装されるデータ損失の軽減は、遡及的緩和よりも包括的であると考えられます。したがって、設計時にデバイスおよびソリューションのセキュリティライフサイクルを計画すると、ビジネスリスクが軽減され、リリース前にインフラストラクチャセキュリティ分析を先行する機会が提供されます。

デバイスのセキュリティライフサイクルにアプローチする方法の 1 つは、サプライチェーン分析です。例えば、適度な規模の IoT デバイスメーカーやソリューションインテグレーターであっても、直接的または間接的にかかわらず、サプライチェーンを構成する多数のサプライヤーを持っています。ソリューションの存続期間と信頼性を最大化するには、正規のコンポーネントを受け取っていることを確認します。

ソフトウェアもサプライチェーンの一部です。デバイスの本番用ファームウェアイメージには、シリコンパートナー、GitHub や SourceForge などのオープンソースの集約サイト、以前のファーストパーティ製品、内部エンジニアリングによって開発された新しいコードなど、多くのソースのドライバとライブラリが含まれています。

ファーストパーティのファームウェアとソフトウェアのダウンストリームメンテナンスとサポートを理解するには、サプライチェーン内の各ソフトウェアプロバイダーを分析して、サポートが提供されているかどうか、およびパッチの配信方法を判断する必要があります。この分析は、接続されたデバイスにとって特に重要です。ソフトウェアのバグは避けられず、脆弱なデバイスがリモートから悪用される可能性があるため、顧客へのリスクを表します。IoT デバイスの製造元またはソリューションエンジニアリングチームは、これらのリスクを軽減するために、適切なタイミングでバグを学習し、パッチを適用する必要があります。

IOTSEC 7.サプライヤー、契約メーカー、その他のアウトソース関係をどのように評価していますか?

IOTSEC 8.IoT デバイスのセキュリティライフサイクルをどのように計画していますか?

IOTSEC 9.サードパーティーのファームウェアとソフトウェアコンポーネントのセキュリティバグをタイムリーに通知するには、どうすればよいですか?

AWS IoT サービスを使用する際に管理するクラウドインフラストラクチャはありませんが、AWS IoT Core がユーザーに代わって他の AWS のサービスとやり取りする統合ポイントがあります。例えば、AWS IoT ルールエンジンは、MQTT トピックストリームに基づいて他の AWS のサービスへのダウンストリームアクションをトリガーできる分析されたルールで構成されます。AWS IoT は他の AWS リソースと通信するため、アプリケーションに適切なサービスロールのアクセス許可が設定されていることを確認する必要があります。

データ保護

IoT アプリケーションを設計する前に、データ分類、ガバナンス、およびコントロールを設計し、文書化して、データをクラウドに保持する方法や、デバイス上でもデバイスとクラウド間でも、データを暗号化する方法を反映する必要があります。従来のクラウドアプリケーションとは異なり、データの機密性とガバナンスは、ネットワークの境界外のリモートの場所にデプロイされた IoT デバイスにも拡張されます。これらの手法は、デバイスから送信される個人を特定できるデータの保護と、規制上の義務の遵守をサポートしているため、重要です。

設計プロセス中に、デバイスの寿命終了時にハードウェア、ファームウェア、データをどのように処理するかを決定します。長期の履歴データをクラウドに保存します。現在のセンサー読み取り値の一部をデバイスにローカルに保存します。つまり、ローカル運用の実行に必要なデータのみを保存します。デバイスに必要な最小データのみを保存することで、意図しないアクセスのリスクが制限されます。

ローカルでデータストレージを減らすことに加えて、デバイスの寿命終了時に実装する必要があるその他の緩和策があります。まず、デバイスは、ハードウェアとファームウェアをデフォルトの工場出荷時バージョンにリセットできるリセットオプションを提供する必要があります。次に、IoT アプリケーションは、すべてのデバイスの最後のログオン時間に対して定期的なスキャンを実行できます。一定期間オフラインであったデバイス、または非アクティブな顧客アカウントに関連付けられているデバイスは、取り消すことができます。3 番目に、特定のデバイスに固有のキーを使用して、デバイスに保持する必要がある機密データを暗号化します。

IOTSEC 10: 転送中および保管中のデータの分類、管理、保護はどのように行いますか?

AWS IoT に出入りするすべてのトラフィックは、Transport Layer Security (TLS) を使用して暗号化する必要があります。AWS IoT では、セキュリティメカニズムによって、AWS IoT と他のデバイスや AWS のサービスの間で移動されるデータが保護されます。AWS IoT に加えて、デバイスのプライベートキーだけでなく、デバイスで収集および処理されたデータも保護するために、デバイスレベルのセキュリティを実装する必要があります。

組み込み開発の場合、AWS には、アプリケーションレイヤーのコンポーネントを抽象化し、デフォルトで AWS セキュリティのベストプラクティスをエッジに組み込むいくつかのサービスがあります。マイクロコントローラの場合、AWS は Amazon FreeRTOS の使用を推奨しています。Amazon FreeRTOS は、FreeRTOS カーネルを Bluetooth LE、TCP/IP、およびその他のプロトコル用のライブラリで拡張します。さらに、Amazon FreeRTOS には、AWS IoT と安全に通信する組み込みアプリケーションを作成できる一連のセキュリティ API が含まれています。

Linux ベースのエッジゲートウェイの場合、AWS IoT Greengrass を使用してクラウド機能をネットワークのエッジまで拡張できます。AWS IoT Greengrass には、接続されたデバイスとの相互 X.509 証明書ベースの認証、AWS IoT Greengrass とクラウドアプリケーション間の通信アクセス許可を管理するための AWS IAM ポリシーとロール、接続されたデバイスと Greengrass Core の間でデータをルーティングする方法や当該ルーティングの可否を判断するために使用されるサブスクリプションなど、いくつかのセキュリティ機能が実装されています。

インシデント対応

IoT でのインシデント対応に備えるには、IoT ワークロードの 2 種類のインシデントへの対応方法を計画する必要があります。最初のインシデントは、パフォーマンスを中断したり、デバイスの動作に影響を与えたりする試みで、個々の IoT デバイスに対する攻撃です。2 つ目のインシデントは、ネットワークの停止や DDoS 攻撃など、大規模な IoT イベントです。どちらのシナリオでも、IoT アプリケーションのアーキテクチャは、インシデントの診断、インシデント全体のデータの関連付け、その後自動化された信頼性の高い方法でランブックを影響を受けるデバイスに適用するまでの時間を決定する上で大きな役割を果たします。

IoT アプリケーションの場合、インシデント対応に関する以下のベストプラクティスに従います。

- IoT デバイスは、場所やハードウェアバージョンなどのデバイス属性に基づいて異なるグループに編成されます。
- IoT デバイスは、接続ステータス、ファームウェアバージョン、アプリケーションステータス、デバイスの状態などの動的な属性で検索できます。
- OTA 更新は、デバイスに対してステージングし、一定期間にわたってデプロイできます。デプロイのロールアウトはモニタリングされ、デバイスが適切な KPI を維持できない場合は自動的に中止できます。

- 更新プロセスはエラーに対して弾力性があり、失敗したソフトウェア更新からデバイスが復旧およびロールバックできます。
- 詳細なログ記録、メトリクス、およびデバイスのテレメトリを利用できます。これには、一定期間にデバイスが現在実行中および実行されている方法に関するコンテキスト情報が含まれます。
- フリート全体のメトリクスは、フリートの全体的な状態をモニタリングし、運用の KPI が一定期間満たされない場合にアラートします。
- 予想される動作から逸脱した個々のデバイスは、隔離、検査、およびファームウェアとアプリケーションの潜在的な侵害について分析できます。

IOTSEC 11: 単一のデバイスまたはデバイスのフリートに影響するインシデントに対応するには、どのように準備すればよいですか?

InfoSec チームが、修復が必要なデバイスをすばやく特定できる戦略を実装します。InfoSec チームに、ファームウェアのバージョン管理とデバイス更新のパッチ適用を考慮したランブックがあることを確認します。脆弱なデバイスがオンラインになったときにセキュリティパッチを予防的に適用する自動プロセスを作成します。

少なくとも、セキュリティチームはデバイスログと現在のデバイスの動作に基づいて、特定のデバイスでインシデントを検出できる必要があります。インシデントが特定されたら、次の段階はアプリケーションの隔離です。これを AWS IoT サービスで実装するには、より制限の厳しい IoT ポリシーとともに AWS IoT Things Groups を使用し、それらのデバイスのカスタムグループログ記録を有効にすることができます。これにより、トラブルシューティングに関連する機能のみを有効にし、さらに多くのデータを収集して根本原因と修復を理解できます。最後に、インシデントが解決されたら、ファームウェア更新をデバイスにデプロイして、既知の状態に戻せる必要があります。

主要な AWS のサービス

IoT で不可欠な AWS セキュリティサービスは、AWS IoT レジストリ、AWS IoT Device Defender、AWS Identity and Access Management (IAM)、および Amazon Cognito です。これらのサービスを組み合わせると、IoT デバイス、AWS のサービス、およびユーザーのリソースへのアクセスを安全に制御できます。次のサービスと機能は、セキュリティの 5 つの領域をサポートします。

設計: AWS デバイス認定プログラムは、AWS IoT との相互運用性について事前にテストされた IoT エンドポイントとエッジハードウェアを提供します。テストには、相互認証と、リモートパッチ適用の OTA サポートが含まれます。

AWS Identity and Access Management (IAM): デバイス認証情報 (X.509 証明書、IAM、Amazon Cognito ID プール、および Amazon Cognito user pools、またはカスタム承認トークン) を使用すると、AWS リソースへのデバイスおよび外部ユーザーのアクセスを安全に制御できます。AWS IoT ポリシーにより、IoT デバイスへのきめ細かいアクセスを実装することが可能になります。ACM プライベート CA は、デバイス証明書を作成および管理するためのクラウドベースのアプローチを提供します。AWS IoT モノのグループを使用して、IoT のアクセス許可を個別ではなくグループレベルで管理します。

発見的統制: AWS IoT Device Defender は、AWS IoT Core Core からのデバイス通信とクラウド側のメトリクスを記録します。AWS IoT Device Defender は、Amazon Simple Notification Service (Amazon SNS) を介して内部システムまたは管理者に通知を送信することにより、セキュリティ応答を自動化できます。AWS CloudTrail は、IoT アプリケーションの管理アクションをログに記録します。Amazon CloudWatch は、AWS IoT Core と統合されたモニタリングサービスであり、セキュリティ応答を自動化するために CloudWatch Events をトリガーできます。CloudWatch は、IoT エッジコンポーネントとクラウドサービス間の接続およびセキュリティイベントに関連する詳細なログをキャプチャします。

インフラストラクチャ保護: AWS IoT Core は、接続されたデバイスがクラウドアプリケーションやその他のデバイスと簡単かつ安全にやり取りできるようにするマネージド型サービスです。AWS IoT Core にある AWS IoT のルールエンジンは、IAM アクセス許可を使用して他のダウンストリームの AWS のサービスと通信します。

データ保護: AWS IoT には、転送中のデータを保護するための TLS を介したデバイスの暗号化機能が含まれています。AWS IoT は、保存中の暗号化をサポートする Amazon S3 や Amazon DynamoDB などのサービスと直接統合します。さらに、AWS Key Management Service (AWS KMS) は、暗号化に使用されるキーを作成および制御する機能をサポートします。デバイスでは、Amazon FreeRTOS、AWS IoT Greengrass、AWS IoT Embedded C SDK などの AWS エッジサービスを使用して、安全な通信をサポートできます。

インシデント対応: AWS IoT Device Defender では、通常のデバイス動作からの逸脱を検出し、AWS Lambda を含む自動応答をトリガーするために使用できるセキュリティプロファイルを作成できます。AWS IoT Device Management は、修復が必要なデバイスをグループ化し、AWS IoT ジョブを使用してデバイスに修正をデプロイするために使用する必要があります。

リソース

当社のセキュリティのベストプラクティスの詳細については、以下のリソースをご参照ください。

ドキュメントとブログ

- IoT セキュリティ ID
- AWS IoT Device Defender
- IoT 認証モデル
- ポート 443 での MQTT
- Device Defender で異常を検出する

ホワイトペーパー

- MQTT トピックの設計

信頼性の柱

信頼性の柱では、ビジネスやお客様の要求に応えるための障害の防止や、障害からの迅速な復旧を行う能力について焦点を当てます。主要なトピックには、設定、プロジェクト間の要件、復旧計画、および変更の管理を中心とする基礎的な要素が含まれます。

トピック

- 設計の原則
- 定義
- ベストプラクティス
- 主要な AWS のサービス
- リソース

設計の原則

Well-Architected フレームワーク全体の設計原則に加えて、クラウドでの IoT の信頼性に関する 3 つの設計原則があります。

- 本稼働規模でのデバイスの動作のシミュレーション: 本稼働デプロイメントを厳密に反映した本稼働規模のテスト環境を作成します。複数ステップのシミュレーションプランを活用して、ライブ開始日より大きな負荷でアプリケーションをテストできます。開発中、シミュレーションテストを 1 回のテストで全体のトラフィックの 10% から開始し、時間の経過とともに増加します (つまり、25%、50%、1 日目の 1 デバイストラフィックの 100%)。シミュレーションテスト中に、パフォーマンスをモニタリングし、ログを確認して、ソリューション全体が期待どおりに動作することを確認します。
- ストリームまたはキューを使用して IoT ルールエンジンからのメッセージ配信をバッファ: マネージドサービスを活用すると、高スループットのテレメトリが可能になります。高スループットのトピックの背後にキューイングレイヤーを挿入することで、IoT アプリケーションは障害を管理し、メッセージングを集約し、他のダウンストリームサービスをスケールできます。
- 障害と回復性を考慮した設計: デバイス自体の回復性を計画することが不可欠です。ユースケースによっては、断続的な接続のための堅牢な再試行ロジック、ファームウェアの更新をロールバックする機能、重要なメッセージ配信のためにローカルに通信する機能、冗長センサーまたはエッジゲートウェイの実行、ハードウェア障害に対する回復力のある冗長センサーまたはエッジゲートウェイの実行、工場出荷時のリセットの実行能力が必要になる場合があります。

定義

クラウド内で信頼性のあるベストプラクティスには 3 つの領域があります。

1. 基盤
2. 変更管理
3. 障害の管理

信頼性を達成するには、システムは、需要や要件の変化を処理するためのメカニズムを備えた、十分に計画された基盤とモニタリングが必要です。また、サービス妨害攻撃を防御するメカニズムも必要になります。障害を検出し、自動的に修復できるシステムを設計することが必要です。

ベストプラクティス

トピック

- 基盤
- 変更管理
- 障害の管理

基盤

IoT デバイスは、ネットワークまたはクラウドのエラーが発生した場合でも、ある程度の容量で動作し続ける必要があります。デバイスファームウェアを設計して、断続的な接続や接続の損失をメモリや電源の制約の影響を受けやすい方法で処理します。IoT クラウドアプリケーションは、データの整合性を維持し、時間の経過とともに水平にスケールするために、オンラインとオフラインの間で頻繁に移行するリモートデバイスを処理するように設計する必要があります。IoT の全体的な使用率をモニタリングし、自動的に容量を増やすメカニズムを作成して、アプリケーションがピークの IoT トラフィックを管理できるようにします。

デバイスが不必要なピークトラフィックを生成しないようにするには、デバイスのフリート全体が同時に同じオペレーションを試行することを防止するデバイスファームウェアを実装する必要があります。例えば、IoT アプリケーションがアラームシステムで構成されており、すべてのアラームシステムが現地時間の午前 9 時にアクティベーションイベントを送信する場合、IoT アプリケーションにはフリート全体からの即時のスパイクが流入します。代わりに、タイミングイベントやエクスポネンシャルバックオフなど、スケジュールされたアクティビティにランダム化係数を組み込み、IoT デバイスが時間枠内でピークトラフィックをより均等に分散できるようにします。

以下の質問は、信頼性のに関する考慮事項に焦点を当てています。

IOTREL 1.IoT アプリケーションのピークに対する AWS のサービスの制限をどのように処理しますか?

AWS IoT は、さまざまな使用態様のために、一連のソフト制限とハード制限を提供します。AWS IoT は、IoT 制限ページですべてのデータプレーン制限の概要を説明します。データプレーン運用 (MQTT Connect、MQTT Publish、MQTT Subscribe など) は、デバイス接続のプライマリドライバーです。したがって、IoT の制限を確認し、アプリケーションがデータプレーンに関連するソフト制限に準拠していることを確認し、データプレーンによって課されるハード制限を超えないようにすることが重要です。

IoT スケーリングアプローチの最も重要な部分は、ハード制限を適切に設計することです。調整できない制限を超えると、スロットリングやクライアントエラーなどのアプリケーションエラーが発生するためです。ハード制限は、単一の IoT 接続でのスループットに関連しています。アプリケーションがハード制限を超えている場合は、このようなシナリオを避けるためにアプリケーションを再設計することをお勧めします。これは、MQTT トピックの再構築、関心のあるデバイスにメッセージを配信する前にメッセージを集約またはフィルタリングするクラウド側ロジックの実装など、いくつかの方法で行うことができます。

従来、AWS IoT のソフト制限は、単一のデバイスに依存しないアカウントレベルの制限に関連しています。アカウントレベルの制限について、1 つのデバイスの IoT 使用量を計算し、その使用量にデバイス数を掛けて、アプリケーションが最初の製品起動に必要な基本の IoT 制限を決定する必要があります。AWS では、制限の引き上げが現在の本稼働ピーク使用量と密接に連携し、バッファを追加することをお勧めします。IoT アプリケーションがプロビジョニングされていないことを確認するには:

- すべての制限については、公開されている AWS IoT CloudWatch メトリクスをご参照ください。
- AWS IoT Core で CloudWatch メトリクスをモニタリングします。
- CloudWatch スロットルメトリクスに関するアラート。制限の引き上げが必要な場合に通知します。
- MQTT 接続、パブリッシュ、サブスクライブ、受信、ルールエンジンアクションなど、IoT のすべてのしきい値にアラームを設定します。
- 100% の容量に達する前に、制限の引き上げをタイムリーにリクエストしてください。

データプレーンの制限に加えて、AWS IoT サービスには管理 API 用のコントロールプレーンがあります。コントロールプレーンは、IoT ポリシーとプリンシパルの作成と保存、レジストリでのモノの作成、および証明書や Amazon Cognito フェデレーティッド ID などの IoT プリンシパルの関連付けのプロセスを管理します。ブートストラップとデバイス登録はプロセス全体にとって重要であるため、コントロールプレーンの運用制限を計画することが重要です。コントロールプレーン API 呼び出しは、1 秒あたりのリクエスト数で測定されたスループットに基づいています。コントロールプレーンの呼び出しは、通常、1 秒あたり数十のリクエストです。登録使用量が予想されるピーク時からさかのぼって、コントロールプレーン運用の制限の引き上げが必要かどうかを判断することが重要です。オンボーディングデバイスの持続的なランプアップ期間を計画し、IoT の制限が日常的なデータプレーンの使用量に合わせて引き上げられるようにします。

コントロールプレーンリクエストのバーストから保護するには、アーキテクチャでこれらの API へのアクセスを許可されたユーザーまたは内部アプリケーションのみに制限する必要があります。バックオフロジックと再試行ロジックを実装し、これらの API へのデータレートを制御するインバウンドリクエストをキューに入れます。

IOTREL 2.他のアプリケーションへの IoT データの取り込みと処理のスループットを管理するための戦略はどのようなものですか?

IoT アプリケーションには、他のデバイス間でのみルーティングされる通信がありますが、アプリケーションで処理および保存されるメッセージがあります。このような場合、IoT アプリケーションの残りの部分は受信データに応答する準備を整える必要があります。そのデータに依存するすべての内部サービスには、データの取り込みと処理をシームレスにスケールする方法が必要です。Well-Architected IoT アプリケーションでは、内部システムは取り込みレイヤーを通じて IoT プラットフォームの接続レイヤーから疎結合化されます。取り込みレイヤーは、耐久性のある短期ストレージを可能にするキューとストリームで構成されており、取り込み速度とは関係なくコンピューティングリソースでデータを処理できます。

スループットを最適化するには、コンピューティングオペレーションを実行する前に、AWS IoT ルールを使用して、インバウンドデバイスデータを、Amazon Kinesis Data Streams、Amazon Data Firehose、Amazon Simple Queue Service などのサービスにルーティングします。すべての中間ストリーミングポイントがピーク容量を処理するようにプロビジョニングされていることを確認します。このアプローチにより、アップストリームアプリケーションがデータを弾力的に処理するために必要なキューイングレイヤーが作成されます。

IOTREL 3.クラウドと通信するときのデバイスの信頼性をどのように処理しますか?

IoT ソリューションの信頼性には、デバイス自体も含まれる必要があります。デバイスはリモートの場所にデプロイされ、IoT アプリケーションが制御できないさまざまな外部要因により、断続的な接続または接続の喪失に対処します。例えば、ISP が数時間中断された場合、デバイスはこれらの長期間にわたり発生する可能性のあるネットワーク停止に対してどのように動作し、対応しますか? デバイスに、最小限の組み込みオペレーションを実装して、AWS IoT Core への接続と通信の管理のニュアンスに対する耐障害性を高めます。

IoT デバイスはインターネットに接続せずに動作できる必要があります。ファームウェアに堅牢な運用を実装する必要があります。次の機能を提供します。

- 重要なメッセージをオフラインで永続的に保存し、再接続したら AWS IoT Core に送信します。
- 接続試行が失敗したときに、エクスポネンシャル再試行とバックオフロジックを実装します。
- 必要に応じて、重要なメッセージを AWS IoT に配信するための別のフェイルオーバーネットワークチャネルを用意します。これには、Wi-Fi からスタンバイセルラーネットワークへのフェイルオーバーや、接続されたデバイスまたはゲートウェイにメッセージを送信するためのワイヤレスパーソナルエリアネットワークプロトコル (Bluetooth LE など) へのフェイルオーバーが含まれます。

- NTP クライアントまたは低ドリフトリアルタイムクロックを使用して現在の時刻を設定する方法があります。デバイスは、AWS IoT Core との接続を試みる前に、その時刻が同期されるまで待機する必要があります。これが不可能な場合、システムは、後続の接続が成功できるように、デバイスの時間を設定するための手段を提供します。
- エラーコードと全体的な診断メッセージを AWS IoT Core に送信します。

変更管理

IOTREL 4.IoT アプリケーションへの変更のロールアウトとロールバックはどのように行いますか?

ロールアウトが失敗した場合に、デバイスファームウェアまたはクラウドアプリケーションの以前のバージョンに戻す機能を実装することが重要です。アプリケーションが Well-Architected の場合は、デバイスからメトリクスと、AWS IoT Core と AWS IoT Device Defender によって生成されたメトリクスをキャプチャします。また、クラウド側の変更後、デバイスの Canary が想定どおりの動作から逸脱した場合にも警告されます。運用メトリクスの逸脱に基づいて、次の機能が必要です。

- Amazon S3 を使用して、すべてのデバイスファームウェアをバージョニングします。
- デバイスファームウェアのマニフェストまたは実行ステップをバージョニングします。
- エラー発生時にデバイスがフォールバックするように、既知の安全なデフォルトのファームウェアバージョンを実装します。
- 暗号化コード署名、バージョンチェック、および複数の不揮発性ストレージパーティションを使用して更新戦略を実装し、ソフトウェアイメージをデプロイしてロールバックします。
- CloudFormation ですべての IoT ルールエンジン設定をバージョニングします。
- CloudFormation を使用して、すべてのダウンストリームの AWS クラウドリソースをバージョニングします。
- CloudFormation や他のインフラストラクチャをコードツールとして使用して、クラウド側の変更を元に戻すためのロールバック戦略を実装します。

AWS でインフラストラクチャをコードとして処理することで、IoT アプリケーションのモニタリングと変更管理を自動化できます。すべてのデバイスファームウェアアーティファクトをバージョニングし、必要に応じて更新を確認、インストール、またはロールバックできることを確認します。

障害の管理

IOTREL 5.IoT アプリケーションは障害にどのように耐えられますか?

IoT はイベント指向のワークロードであるため、アプリケーションコードは、アプリケーションを通じてイベントが浸透したときに発生する可能性のある既知および未知のエラーを処理するために回復力を持つ必要があります。Well-Architected IoT アプリケーションには、データ処理のエラーを記録して再試行する機能があります。IoT アプリケーションは、すべてのデータを raw 形式でアーカイブします。有効なデータと無効なすべてのデータをアーカイブすることで、アーキテクチャは特定の時点にデータをより正確に復元できます。

IoT ルールエンジンを使用すると、アプリケーションは IoT エラーアクションを有効にできます。アクションを呼び出すときに問題が発生した場合、ルールエンジンはエラーアクションを呼び出します。これにより、プライマリ IoT アクションに配信できなかったメッセージをキャプチャ、モニタリング、アラート、最終的には再試行できます。IoT エラーアクションは、プライマリアクションとは異なる AWS のサービスを使用して設定することをお勧めします。Amazon SQS または Amazon Kinesis などのエラーアクションには、耐久性のあるストレージを使用してください。

ルールエンジンから、アプリケーションロジックは最初にキューからのメッセージを処理し、そのメッセージのスキーマが正しいことを検証する必要があります。アプリケーションロジックは、既知のエラーをキャッチして記録し、必要に応じてそれらのメッセージを独自の DLQ に移動してさらに分析する必要があります。Amazon Data Firehose と AWS IoT Analytics チャネルを使用して、未加工のメッセージと未フォーマットのすべてのメッセージを Amazon S3、AWS IoT Analytics データストア、およびデータウェアハウジングの Amazon Redshift の長期ストレージに転送する IoT ルールがあります。

IOTREL 6.物理アセットに対して異なるレベルのハードウェア障害モードを検証するには、どうすればよいですか?

IoT の実装では、デバイスレベルで複数のタイプの障害に対応する必要があります。障害は、ハードウェア、ソフトウェア、接続、または予期しない不利な条件が原因である可能性があります。モノの失敗を計画する 1 つの方法は、可能であればデバイスをペアでデプロイするか、同じカバレッジエリアにデプロイされたデバイス群にデュアルセンサーをデプロイすることです (メッシュ)。

デバイス障害の根本的な原因にかかわらず、デバイスがクラウドアプリケーションと通信できる場合は、診断トピックを使用して、ハードウェア障害に関する診断情報を AWS IoT Core に送信する必要があります。ハードウェア障害のためにデバイスが接続を失った場合、接続性ステータスでフリートのインデックス作成を使用して、接続ステータスの変更を追跡します。デバイスが長期間オフラインになっている場合は、デバイスが修復を必要とする可能性があるというアラートをトリガーします。

主要な AWS のサービス

Amazon CloudWatch を使用してランタイムメトリクスをモニタリングし、信頼性を確保します。信頼性の 3 つの領域をサポートするその他のサービスと機能は次のとおりです。

基礎: AWS IoT Core では、基盤となるインフラストラクチャを管理することなく、IoT アプリケーションをスケールできます。アカウントレベルの制限の引き上げをリクエストすることで、AWS IoT Core をスケールできます。

変更管理: AWS IoT Device Management は、Amazon S3 を使用して、すべてのファームウェア、ソフトウェア、および更新マニフェストのバージョンを管理しつつ、現場でデバイスを更新できます。AWS CloudFormation により、IoT Infrastructure as Code を記録し、CloudFormation テンプレートを使用してクラウドリソースをプロビジョニングすることが可能になります。

障害管理: Amazon S3 では、デバイスから永続的にテレメトリをアーカイブできます。AWS IoT ルールエンジンの Error アクションを使用すると、プライマリ AWS のサービスがエラーを返すときに、他の AWS のサービスにフォールバックできます。

リソース

信頼性に関するベストプラクティスの詳細については、以下のリソースをご参照ください。

ドキュメントとブログ

- Device Time を使用した AWS IoT Server 証明書の検証
- AWS IoT Core 制限
- IoT エラーアクション
- フリートのインデックス作成
- IoT Atlas

パフォーマンス効率の柱

パフォーマンス効率の柱では、コンピューティングリソースの効率的な利用に焦点を当てます。主要トピックには、必要なワークロードに基づく適切なリソースのタイプやサイズの選択、パフォーマンスのモニタリング、ビジネスとテクノロジーのニーズの変化に応じて効率性を維持するための情報に基づく意思決定などが含まれます。パフォーマンス効率の柱は、コンピューティングリソースを効率的に使って要件を満たし、需要が変化し技術が発展するのに合わせて効率性を維持する能力に焦点を当てます。

トピック

- 設計の原則
- 定義
- ベストプラクティス
- 主要な AWS のサービス
- リソース

設計の原則

Well-Architected フレームワークのパフォーマンス効率設計原則全体に加えて、クラウドでの IoT のパフォーマンス効率化には 3 つの設計原則があります。

- マネージド型サービスを使用: AWS では、データベース、コンピューティング、ストレージにまたがる複数のマネージドサービスが提供されており、アーキテクチャ全体の信頼性とパフォーマンスを向上させることができます。
- データをバッチで処理: IoT アプリケーションの接続部分を IoT の取り込みおよび処理部分から疎結合化します。取り込みレイヤーを分離することで、IoT アプリケーションは集計データを処理し、複数の IoT メッセージを一度に処理することで、よりシームレスにスケールできます。
- イベント指向のアーキテクチャを使用: IoT システムはデバイスからイベントをパブリッシュし、IoT アプリケーションの他のサブシステムにそれらのイベントを透過します。キュー、メッセージ処理、べき等性、デッドレターキュー、ステートマシンの活用など、イベント指向のアーキテクチャに対応できるメカニズムを設計します。

定義

クラウドでのパフォーマンス効率を向上させるには、次の 4 つのベストプラクティス領域があります。

1. 選択
2. 確認
3. モニタリング
4. トレードオフ

高性能なアーキテクチャを選択する際は、データ駆動型のアプローチを選択します。ハイレベルな設計から、リソースタイプの選択と設定まで、アーキテクチャのあらゆる側面に関するデータを収集します。選択した内容を定期的にレビューすることで、絶えず進化し続けている AWS プラットフォームを活用できているかを確認できます。モニタリングにより、予想されるパフォーマンスからの逸脱が確実に認識され、対応できるようになります。さらに、圧縮やキャッシュを使用したり、整合性に関する要件を緩和したりするなど、アーキテクチャにおけるトレードオフを行ってパフォーマンスを向上させることができます。

ベストプラクティス

トピック

- 選択
- デバイス
- IoT 接続性
- データベース
- コンピューティング
- 分析
- 確認
- モニタリング
- トレードオフ

選択

Well-Architected IoT ソリューションは、デバイス、接続、データベース、データ処理、分析など、複数のシステムおよびコンポーネントで構成されています。AWS には、IoT サービス、データベー

スサービス、分析ソリューションがいくつかあり、お客様はビジネス目標に集中しながら、Well-Architected ソリューションをすばやく構築できます。AWS では、ワークロードに最適なマネージド型の AWS のサービスを組み合わせて活用することをお勧めします。以下の質問は、パフォーマンスの効率に関するこれらの考慮事項に焦点を当てています。

IOTPERF 1.最も良いパフォーマンスの IoT アーキテクチャをどのように選択していますか?

アーキテクチャの実装を選択するときは、運用の長期的なビューに基づくデータ駆動型アプローチを使用します。IoT アプリケーションは、イベント指向のアーキテクチャに自然に適合します。アーキテクチャは、通知、データのパブリッシュとサブスクライブ、ストリーム処理、イベント指向のコンピューティングなどのイベント指向のパターンと統合するサービスを組み合わせます。以下のセクションでは、考慮すべき 5 つの主要な IoT リソースタイプ (デバイス、接続、データベース、コンピューティング、分析) について説明します。

デバイス

特定のシステムに最適な組み込みソフトウェアは、デバイスのハードウェアフットプリントによって異なります。例えば、データのプライバシーと整合性を維持するために必要になるネットワークセキュリティプロトコルは、比較的大きい RAM フットプリントを持つことができます。イントラネットおよびインターネット接続の場合は、強力な暗号スイートと最小限のフットプリントを組み合わせてた TLS を使用します。AWS IoT では、TLS を使用して AWS IoT に接続するデバイスに対して Elliptic Curve Cryptography (ECC) がサポートされています。デバイスの選択基準では、デバイス上の安全なソフトウェアおよびハードウェアプラットフォームが優先されます。AWS には、AWS IoT と安全に統合できるハードウェアソリューションを提供する多数の IoT パートナーもあります。

適切なハードウェアパートナーの選択に加えて、Amazon FreeRTOS や AWS IoT Greengrass など、さまざまなソフトウェアコンポーネントを使用してデバイス上でアプリケーションロジックを実行することもできます。

IOTPERF 2.IoT デバイス用のハードウェアとオペレーティングシステムはどのように選択しますか?

IoT 接続性

クラウドと通信するためにファームウェアを開発する前に、デバイスの長期的成長をサポートする、安全でスケーラブルな接続プラットフォームを実装します。IoT プラットフォームは、デバイスの予測量に基づいて、デバイスとクラウド間の通信ワークフローをスケールできる必要があります。これは、テレメトリの取り込みやデバイス間のコマンドと応答の通信の簡単な取り込みである場合です。

Amazon EC2 などの AWS のサービスを使用して IoT アプリケーションを構築することはできますが、IoT サービスに独自の価値をもたらすため、差別化につながらない手間がかかります。したがって、AWS では IoT プラットフォームに AWS IoT Core を使用することをお勧めします。

AWS IoT Core は、HTTP、WebSockets、および MQTT をサポートしています。断続的な接続を許容し、デバイスのコードフットプリントを最小限に抑え、ネットワーク帯域幅の要件を軽減するために設計された軽量通信プロトコルです。

IOTPERF 3.プライマリ IoT プラットフォームはどのように選択しますか?

データベース

IoT アプリケーションには複数のデータベースがあり、それぞれのデータベースへのデータの書き込み頻度、データベースからのデータの読み取り頻度、データの構造とクエリの方法などの属性に対して選択されます。データベース提供を選択する際に考慮すべき他の条件があります。

- データのボリュームと保持期間。
- 組み込みデータの編成と構造。
- データを消費するユーザーとアプリケーション (未加工または処理済み) とその地理的場所/分散。
- 機械学習やリアルタイムの可視化など、高度な分析のニーズ。
- 他のチーム、組織、およびビジネスユニット間でのデータの同期。
- 行、テーブル、およびデータベースレベルでのデータのセキュリティ。
- エンタープライズアプリケーション、ドリルスルーダッシュボード、インタラクションシステムなど、他の関連するデータ駆動型イベントとのやり取り。

AWS には、IoT ソリューションをサポートするいくつかのデータベースサービスがあります。構造化データの場合は、組織データへの高度にスケーラブルなリレーショナルインターフェイスである Amazon Aurora を使用する必要があります。クエリに低レイテンシーが必要で、複数のコンシュー

マーによって使用される半構造化データには、完全マネージド型、マルチリージョン、マルチマスターデータベースである DynamoDB を使用します。このデータベースは、ミリ秒単位の安定したレイテンシーを実現し、組み込みのセキュリティ、バックアップと復元、インメモリキャッシュを提供します。

未加工で、未フォーマットのイベントデータを保存するには、AWS IoT Analytics を使用します。AWS IoT Analytics では、分析のために IoT データを時系列データストアに保存する前に、フィルター処理、変換、および強化を行います。 Amazon SageMaker AI を使用して、Greengrass Machine Learning Inference などの AWS IoT サービスを使用して、クラウドとエッジで、IoT データに基づく機械学習モデルの構築、トレーニング、デプロイを行います。未加工でフォーマット済みの時系列データを Amazon Redshift などのデータウェアハウスソリューションに保存することをご検討ください。フォーマットされてないデータは、Amazon S3 および Amazon Data Firehose を介して Amazon Redshift にインポートできます。フォーマットされていないデータをスケーラブルなマネージド型データストレージソリューションにアーカイブすることで、ビジネスインサイトの取得、データの調査、時間の経過に伴う傾向やパターンの識別を開始できます。

IoT データの履歴傾向を保存して活用することに加えて、デバイスの現在の状態を保存し、すべてのデバイスの現在の状態に対してクエリを実行するシステムが必要です。これにより、IoT データに対する内部分析と顧客向けビューがサポートされます。

AWS IoT Shadow サービスは、デバイスの仮想表現をクラウドに保存するための効果的なメカニズムです。AWS IoT Device Shadow は、各デバイスの現在の状態を管理するのに最適です。さらに、運用上のニーズについてシャドウに対してクエリを実行する必要がある内部チームには、フリートインデックス作成のマネージド型機能を活用します。これにより、IoT レジストリとシャドウメタデータを組み込んだ検索可能なインデックスが提供されます。コンシューマーアプリケーションなど、多数の外部ユーザーにインデックスベースの検索やフィルタリング機能を提供する必要がある場合は、IoT ルールエンジン、Firehose、Amazon ElasticSearch Service を組み合わせてデータを動的にアーカイブし、外部ユーザーにきめ細かなクエリアクセスを許可する形式でデータを保存します。

IOTPERF 4.IoT デバイスの状態のデータベースはどのように選択しますか?

コンピューティング

IoT アプリケーションは、メッセージのストリームに対する継続的な処理を必要とする大量の取り込みに適しています。したがって、アーキテクチャは、データストレージ中およびデータストレージ前の安定したストリーム処理とビジネスアプリケーションの実行をサポートするコンピューティングサービスを選択する必要があります。

IoT で使用される最も一般的なコンピューティングサービスは AWS Lambda で、テレメトリデータが AWS IoT Core または AWS IoT Greengrass に到達したときにアクションを呼び出すことができます。AWS Lambda は IoT 全体のさまざまな場所で使用できます。AWS Lambda でビジネスロジックをトリガーすることを選択した場所は、特定のデータイベントを処理する時間の影響を受けます。

Amazon EC2 インスタンスは、さまざまな IoT ユースケースにも使用できます。マネージド型リレーショナルデータベースシステムや、ウェブ、レポート、既存のオンプレミスソリューションのホストなど、さまざまなアプリケーションに使用できます。

IOTPERF 5.AWS IoT イベントを処理するためのコンピューティングソリューションはどのように選択しますか?

分析

IoT ソリューションを実装する主なビジネスケースは、現場におけるデバイスの動作および使用態様に迅速に対応することです。受信テレメトリに直接行動することで、ビジネスは、優先すべき新製品や機能、または組織内のワークフローをより効率的に運用する方法について、より多くの情報に基づいた決定を下すことができます。分析サービスは、実行する分析のタイプに基づいてデータのさまざまなビューを表示できるように選択する必要があります。AWS では、時系列分析、リアルタイムメトリクス、アーカイブおよびデータレイクのユースケースなど、さまざまな分析ワークフローに合わせてさまざまなサービスを提供します。

IoT データを使用すると、アプリケーションはストリーミングデータメッセージの上に時系列分析を生成できます。時間枠でメトリクスを計算し、他の AWS のサービスに値をストリーミングできます。

さらに、AWS IoT Analytics を使用する IoT アプリケーションでは、時系列データストアにデータを保存する前に、データ変換、強化、フィルタリングで構成されるマネージド型 AWS Data Pipeline を実装できます。さらに、AWS IoT Analytics では、QuickSight と Jupyter ノートブックを使用してネイティブに可視化と分析を実行できます。

確認

IOTPERF 6.IoT アプリケーションの履歴分析に基づいて、アーキテクチャをどのように進化させますか?

複雑な IoT ソリューションを構築する場合、ビジネスの成果に直接影響を与えない取り組みにかなりの時間を割くことができます。例えば、IoT プロトコルの管理、デバイス ID の保護、デバイスとクラウド間のテレメトリの転送などです。IoT のこれらの側面は重要ですが、差別化価値に直接つながるわけではありません。IoT におけるイノベーションのペースも課題となります。

AWS では、IoT の一般的な課題に基づいて、新しい機能とサービスを定期的にリリースしています。データを定期的に見直して、新しい AWS IoT サービスがアーキテクチャの現在の IoT ギャップを解決できるかどうか、またはコアビジネスの差別化要因ではないアーキテクチャのコンポーネントを置き換えることができるかどうかをご確認ください。IoT データを集約し、データを保存し、後でデータを可視化して履歴分析を実装するために構築されたサービスを活用します。IoT デバイスからのタイムスタンプ情報の送信と、AWS IoT Analytics や時間ベースのインデックス作成などのサービスを活用して、関連するタイムスタンプ情報とともにデータをアーカイブできます。AWS IoT Analytics のデータは、デバイスからの追加の IT または OT の運用および効率ログと共に、お客様独自の Amazon S3 バケットに保存できます。IoT のデータのアーカイブ状態を可視化ツールと組み合わせることで、新しい AWS のサービスがどのように付加価値を提供できるかについて、データに基づく決定を下し、サービスがフリート全体の効率性を向上させるかを測定できます。

モニタリング

IOTPERF 7.IoT アプリケーションのエンドツーエンドのシミュレーションテストをどのように実行していますか?

IoT アプリケーションは、テストデバイスとしてセットアップされた実稼働デバイス (特定のテスト MQTT 名前空間を持つ) を使用するか、シミュレートされたデバイスを使用してシミュレートできます。IoT ルールエンジンを使用してキャプチャされたすべての受信データは、本番稼働用と同じワークフローを使用して処理されます。

エンドツーエンドのシミュレーションの頻度は、特定のリリースサイクルまたはデバイスの導入によって駆動される必要があります。障害パス (障害時にのみ実行されるコード) をテストして、ソリューションが エラーに対して耐障害性を確保する必要があります。また、本番稼働用アカウントと本番稼働前のアカウントに対して Device Canary を継続的に実行する必要があります。デバイス Canary は、シミュレーションテスト中にシステムパフォーマンスの重要な指標として機能します。テストの出力を文書化し、修復計画を下書きする必要があります。ユーザー受け入れテストを実行する必要があります。

IOTPERF 8.IoT 実装でパフォーマンスモニタリングをどのように使用していますか?

IoT デプロイに関連するパフォーマンスモニタリングには、デバイス、クラウドパフォーマンス、ストレージ/分析など、いくつかの主要なタイプがあります。ログから収集されたデータと、テレメトリとコマンドデータを使用して、適切なパフォーマンスメトリクスを作成します。ベーシックパフォーマンス追跡から開始し、ビジネスコアコンピテンシーが拡大するにつれ、メトリクスに基づいて構築します。

CloudWatch Logs のメトリクスフィルタを活用して、regex (正規表現) パターンマッチングを通じて IoT アプリケーションの標準出力をカスタムメトリクスに変換します。アプリケーションのカスタムメトリクスに基づいて CloudWatch アラームを作成し、IoT アプリケーションの動作をすばやく把握します。

特定のモノのグループを追跡するためのきめ細かなログをセットアップします。IoT ソリューション開発中に、DEBUG ログ記録を有効にして、メッセージブローカーとルールエンジンを介してデバイスから渡される各 IoT メッセージに関するイベントの進行状況を明確に把握します。本稼働環境では、ログ記録を ERROR および WARN に変更します。

クラウド計測に加えて、デプロイ前にデバイスで計測を実行して、デバイスがローカルリソースを最も効率的に使用できるようにし、ファームウェアコードがメモリリークなどの不要なシナリオにつながらないようにする必要があります。制約のあるデバイスに高度に最適化されたコードをデプロイし、組み込みアプリケーションから AWS IoT にパブリッシュされたデバイス診断メッセージを使用してデバイスの状態をモニタリングします。

トレードオフ

IoT ソリューションは、運用、カスタマーケア、金融、販売、マーケティングなど、重要なエンタープライズ機能の広範囲にわたる豊富な分析機能を駆動します。同時に、エッジゲートウェイの効率的な出力ポイントとして使用できます。データおよび分析がデバイスによってクラウドにプッシュされ、機械学習アルゴリズムがクラウドからデバイスゲートウェイにプルダウンされる高効率な IoT 実装のアーキテクチャ設計には、慎重に検討する必要があります。

個々のデバイスは、特定のネットワークでサポートされているスループットによって制限されます。データが交換される頻度は、トランスポートレイヤーと、オプションでデータを保存、集計、クラウドに送信するデバイスの能力とのバランスを取る必要があります。バックエンドアプリケーションがデータを処理してアクションを実行するために必要な時間に合わせたタイミング間隔で、デバイスからクラウドにデータを送信します。例えば、1 秒単位でデータを表示する必要がある場合、デバイス

は 1 秒よりも頻繁にデータを送信する必要があります。逆に、アプリケーションが 1 時間あたりの料金でのみデータを読み取る場合、エッジでデータポイントを集約し、30 分ごとにデータを送信することで、パフォーマンスのトレードオフをすることができます。

IOTPERF 9.IoT デバイスのデータをビジネスシステムや運用システムで使用できるようにするには、どうすればよいでしょうか?

IOTPERF 10.デバイスから IoT アプリケーションにデータが送信される頻度はどれくらいですか?

エンタープライズアプリケーション、ビジネス、運用が IoT テレメトリデータを可視化する必要がある速度によって、IoT データを処理するための最も効率的なポイントが決まります。ハードウェアに制限がないネットワーク制約のある環境では、AWS IoT Greengrass などのエッジソリューションを使用して、クラウドからオフラインでデータを運用および処理します。ネットワークとハードウェアの両方に制約がある場合には、バイナリ形式を使用し、同様のメッセージを 1 つのリクエストにまとめることで、メッセージペイロードを圧縮する機会を探します。

可視化のために、Amazon Managed Service for Apache Flink を使用すると、ほぼリアルタイムで継続的にデータを読み取り、処理、保存する SQL コードをすばやく作成できます。ストリーミングデータに対して標準的な SQL クエリを使用すると、データを変換して洞察を提供するアプリケーションを構築できます。Managed Service for Apache Flink を使用すると、ストリーミング分析用の IoT データを公開できます。

主要な AWS のサービス

パフォーマンス効率を実現するための主要な AWS のサービスは Amazon CloudWatch です。これは、AWS IoT Core、AWS IoT Device Defender、AWS IoT Device Management、AWS Lambda、DynamoDB などのいくつかの IoT サービスと統合されます。Amazon CloudWatch は、アプリケーションの全体的なパフォーマンスと運用状態を可視化します。以下のサービスでは、パフォーマンス効率もサポートしています。

選択

デバイス: AWS ハードウェアパートナーは、IoT アプリケーションの一部として使用できる、プロダクション対応の IoT デバイスを提供します。Amazon FreeRTOS は、マイクロコントローラー用のソフトウェアライブラリを備えたオペレーティングシステムです。AWS IoT Greengrass では、エッ

ジでローカルのコンピューティング、メッセージング、データキャッシュ、同期、および ML を実行できます。

接続性: AWS IoT Core は、MQTT (デバイス通信用の軽量のパブリッシュおよびサブスクライブプロトコル) をサポートするマネージド型 IoT プラットフォームです。

データベース: Amazon DynamoDB は完全マネージド型 NoSQL データストアで、数ミリ秒未満のレイテンシーリクエストをサポートし、IoT データのさまざまなビューを簡単に取得できます。

コンピューティング: AWS Lambda は、サーバーをプロビジョニングせずにアプリケーションコードを実行できるイベント駆動型のコンピューティングサービスです。Lambda は、AWS IoT Core または Amazon Kinesis や Amazon SQS などのアップストリームサービスからトリガーされる IoT イベントとネイティブに統合します。

Analytics: AWS IoT Analytics は、IoT テレメトリの時系列データストアを提供しながら、デバイスレベルの分析を運用するマネージド型サービスです。

確認: AWS ウェブサイトの「AWS IoT ブログ」セクションは、AWS IoT の一部として新しくリリースされたことを学習するためのリソースです。

モニタリング: Amazon CloudWatch メトリクスと Amazon CloudWatch Logs は、既存のモニタリングソリューションと統合できるメトリクス、ログ、フィルター、アラーム、通知を提供します。これらのメトリクスは、デバイスのテレメトリで拡張して、アプリケーションをモニタリングできます。

トレードオフ: AWS IoT Greengrass と Amazon Kinesis は、IoT アプリケーションのさまざまな場所でデータを集約およびバッチ処理できるサービスで、より効率的なコンピューティングパフォーマンスを提供します。

リソース

当社のパフォーマンス効率性に関連するベストプラクティスの詳細については、以下のリソースをご参照ください。

ドキュメントとブログ

- AWS Lambda の開始方法
- DynamoDB の開始方法
- AWS IoT Analytics ユーザーガイド
- Amazon FreeRTOS の開始方法
- AWS IoT Greengrass の開始方法

- AWS IoT ブログ

コスト最適化の柱

コスト最適化の柱には、システムを全ライフサイクルにわたり洗練および改善する継続的プロセスが含まれます。最初の PoC (実証支援) の初期設計から、製品ワークロードの継続運用まで、コスト意識の高いシステムを構築および運用する際に紙形式のプラクティスを採用すれば、ビジネス上の成果を上げながら、コストの最小化と投資利益率の最大化の両方を達成できるようになります。

トピック

- 設計の原則
- 定義
- ベストプラクティス
- 主要な AWS のサービス
- リソース

設計の原則

Well-Architected フレームワーク全体のコスト最適化設計原則に加えて、クラウドでの IoT のコスト最適化には 1 つの設計原則があります。

- 製造コストのトレードオフを管理: ビジネスパートナーとしての基準、ハードウェアコンポーネントの選択、ファームウェアの複雑さ、および配布要件はすべて、製造コストにとって重要な役割を果たします。このコストを最小限に抑えることで、複数の製品世代にわたって製品を市場に投入できるかどうかを判断できます。ただし、コンポーネントや製造元の選択にショートカットを使用すると、ダウンストリームのコストが増加する可能性があります。たとえば、信頼できる製造元と提携することで、ダウンストリームのハードウェア障害とカスタマーサポートコストを最小限に抑えることができます。専用の暗号化コンポーネントを選択すると、部品表 (BOM) のコストは増加しますが、ダウンストリームの製造とプロビジョニングの複雑さが軽減されます。これは、部品にすでにオンボードプライベートキーと証明書が付属している場合があるためです。

定義

4 つの分野のベストプラクティスなど、クラウド内でのコストの最適化:

1. 費用対効果の高いリソース

2. 需要と供給の一致
3. 費用認識
4. 時間の経過に伴う最適化

考慮すべきトレードオフがあります。例えば、最適化したいのは市場投入までのスピードですか、それともコストですか? 市場投入のスピードを向上する、新しい機能を導入する、締め切りに間に合わせるといったケースでは、前払いコストの投資を最適化するよりも、スピードを重視して最適化することが最善なことがあります。設計上の決定は、時には急いで行われます。即ち基準とは対照的に、実際的なデータではなく、コストを最適化するために ベンチマークではなく、最もコストの最適なデプロイメントに時間をかけます。その結果、プロビジョニングは過剰に、最適化デプロイは過小になります。以下のセクションでは、デプロイの初期段階でのコスト最適化と継続的なコスト最適化について、そのテクニックと戦略的ガイダンスを紹介します。

ベストプラクティス

トピック

- 費用対効果の高いリソース
- 需要と供給の一致
- 時間の経過に伴う最適化

費用対効果の高いリソース

IoT アプリケーションで生成できるデバイスやデータの規模を考えると、システムに適した AWS のサービスを使用することは、コスト削減の鍵となります。IoT ソリューション全体のコストに加えて、IoT アーキテクトは多くの場合、BOM コストのレンズを通じて接続性を調べます。BOM 計算では、IoT アプリケーションへの接続を管理するための長期的なコストを予測し、モニタリングする必要があります。AWS のサービスを活用すると、初期 BOM コストを計算し、イベント駆動型の費用対効果の高いサービスを活用し、アーキテクチャを更新して、接続にかかる全体の寿命コストを削減できます。

リソースのコスト効率を高める最も簡単な方法は、IoT イベントをバッチにグループ化し、データをまとめて処理することです。グループでイベントを処理することで、個々のメッセージの全体的なコンピューティング時間を短縮できます。集約を使用すると、コンピューティングリソースを節約し、永続化する前にデータを圧縮およびアーカイブするときにソリューションを実現できます。この戦略は、データを失ったり、データのクエリ能力を損なうことなく、ストレージフットプリント全体を減らします。

IOTCOST 1.IoT プラットフォームから他のサービスに配信されるバッチデータ、エンリッチデータ、集計データのアプローチをどのように選択しますか?

AWS IoT は、即時消費または履歴分析のストリーミングデータに最適です。AWS IoT Core から他の AWS のサービスにデータをバッチ処理する方法は複数あります。差別化要因は、未加工データ (そのまま) をバッチ処理するか、データをエンリッチ化してバッチ処理することです。取り込み中 (または直後に) IoT テレメトリデータの強化、変換、フィルタリングは、Kinesis Data Streams、Firehose、AWS IoT Analytics、または Amazon SQS にデータを送信する AWS IoT ルールを作成することによって実行するのが最適です。これらのサービスを使用すると、複数のデータイベントを一度に処理できます。

このバッチパイプラインから raw デバイスデータを処理する場合、AWS IoT Analytics と Amazon Data Firehose を使用して S3 バケットと Amazon Redshift にデータを転送できます。Amazon S3 のストレージコストを削減するために、アプリケーションは Amazon Glacier などの低コストのストレージにデータをアーカイブするライフサイクルポリシーを活用できます。

需要と供給の一致

需要と供給を最適にマッチングすることで、システムのコストを最低限に抑えることができます。ただし、IoT ワークロードがデータバーストの影響を受けているため、ソリューションは動的にスケーラブルであり、リソースをプロビジョニングするときにピークキャパシティーを考慮する必要があります。イベント指向のデータフローでは、ピーク容量に合わせて AWS リソースを自動的にプロビジョニングし、既知の低トラフィック期間中にスケールアップまたはスケールダウンすることができます。

以下の質問は、コストの最適化に関する考慮事項に焦点を当てています。

IOTCOST 2.リソースの供給とデバイスの需要をどのように一致させていますか?

AWS Lambda や API Gateway などのサーバーレステクノロジーは、よりスケーラブルで耐障害性の高いアーキテクチャを作成するのに役立ちます。お支払いいただくのは、アプリケーションがこれらのサービスを利用した分のみです。AWS IoT Core、AWS IoT Device Management、AWS IoT Device Defender、AWS IoT Greengrass、および AWS IoT Analytics もマネージド型サービスであり、使用量に応じて料金が発生し、アイドル状態のコンピューティングキャパシティーに対して課金されることはありません。マネージドサービスの利点は、AWS がリソースの自動プロビジョニング

を管理することです。マネージドサービスを使用する場合は、AWS のサービスの制限の引き上げをモニタリングしてアラートを設定する責任があります。

需要と供給を照合するように設計するときは、時間の経過とともに予想される使用量と、超える可能性が高い制限を事前に計画します。これらの制限の引き上げを今後の計画に考慮します。

時間の経過に伴う最適化

AWS の新機能を評価することで、デバイスのパフォーマンスを分析し、デバイスと IoT との通信方法を変更することでコストを最適化できます。

デバイスファームウェアの変更によってソリューションのコストを最適化するには、AWS IoT などの AWS のサービスの料金コンポーネントを確認し、特定のサービスの請求計測しきい値を下回っているかどうかを判断してから、コストとパフォーマンスのトレードオフを比較検討する必要があります。

IOTCOST 3.デバイスと IoT プラットフォーム間のペイロードサイズを最適化するにはどうすればよいですか?

IoT アプリケーションは、エンドデバイスで実現できるネットワーキングスループットと、IoT アプリケーションでデータを処理する最も効率的な方法のバランスを取る必要があります。IoT デプロイでは、デバイスの制約に基づいてデータ転送を最初に最適化することをお勧めします。まず、デバイスからクラウドに個別のデータイベントを送信し、1 つのメッセージで複数のイベントのバッチ処理を最小限に抑えます。後で、必要に応じて、シリアル化フレームワークを使用して IoT プラットフォームに送信する前にメッセージを圧縮できます。

コストの観点からは、MQTT ペイロードサイズは AWS IoT Core の重要なコスト最適化要素です。IoT メッセージは、5 KB 単位で請求されます (最大 128 KB) 。このため、各 MQTT ペイロードサイズは最大 5 KB に近くなる必要があります。例えば、現在 6 KB のサイズのペイロードは、10 KB のペイロードほどコスト効率が良くありません。これは、1 つのメッセージが他のメッセージよりも大きいにもかかわらず、そのメッセージのパブリッシュにかかる全体的なコストが同じであるためです。

ペイロードサイズを利用するには、データを圧縮するか、データをメッセージに集約する機会を探ります。

- 値を読みやすくしながら短くする必要があります。5 桁の精度で十分であれば、ペイロードに 12 桁を使用しないでください。

- IoT ルールエンジンのペイロード検査が不要な場合は、シリアル化フレームワークを使用してペイロードをより小さいサイズに圧縮できます。
- データを送信する頻度を低く抑え、請求可能な増分内でメッセージを集約できます。例えば、1 つの 2 KB のメッセージを 1 秒おきに送信することは、2 つの 2 KB のメッセージを 2 秒おきに送信することにより、より低い IoT メッセージのコストで実現できます。

このアプローチには、実装前に考慮する必要があるトレードオフがあります。デバイスに複雑さや遅延を追加すると、予期せず処理コストが増加する可能性があります。IoT ペイロードのコスト最適化の演習は、ソリューションが本稼働状態になった後にのみ行われ、データ駆動型のアプローチを使用して、AWS IoT Core へのデータ送信方法の変更によるコストへの影響を判断できます。

IOTCOST 4.IoT デバイスの現在の状態を保存するコストをどのように最適化しますか?

Well-Architected IoT アプリケーションには、クラウド内のデバイスの仮想表現があります。この仮想表現は、マネージド型データストアまたは専門的な IoT アプリケーションデータストアで構成されます。いずれの場合も、デバイス状態の変更を IoT アプリケーションに効率的に送信するようにエンドデバイスをプログラムする必要があります。たとえば、デバイスの完全な状態が同期していない可能性があり、現在の設定をすべて送信することで最適な一致がファームウェアロジックで指示されている場合にのみ、デバイスは完全なデバイス状態を送信する必要があります。個々の状態の変化が発生すると、デバイスは、その変更をクラウドに送信する頻度を最適化する必要があります。

AWS IoT では、Device Shadow およびレジストリ運用は 1 KB 単位で計測され、課金は 100 万回のアクセス/変更オペレーションごとに課金されます。シャドウには、各デバイスの目的の状態または実際の状態が保存され、レジストリはデバイスの名前と管理に使用されます。

デバイスシャドウとレジストリのコスト最適化プロセスは、実行されるオペレーションの数と各オペレーションのサイズの管理に重点を置いています。運用がシャドウ運用およびレジストリ運用にコスト依存する場合は、シャドウ運用を最適化する方法を探す必要があります。たとえば、シャドウの場合、報告された各変更を個別に送信する代わりに、レポートされた複数のフィールドを 1 つのシャドウメッセージ更新に集約できます。シャドウの更新をグループ化すると、サービスの更新を統合することで、シャドウの全体的なコストが削減されます。

主要な AWS のサービス

コスト最適化をサポートする AWS の主な機能は、コスト配分タグです。これは、システムのコストを把握するのに役立ちます。以下のサービスと機能は、コスト最適化の 3 つの分野で重要です。

- コスト効率の高いリソース: Amazon Kinesis、AWS IoT Analytics、および Amazon S3 は、コンピューティングリソースのコスト効率を向上させるために、1 回のリクエストで複数の IoT メッセージを処理できる AWS のサービスです。
- 需要と供給の一致: AWS IoT Core は、接続、クラウドへのデバイスセキュリティ、メッセージングルーティング、デバイスの状態を管理するためのマネージド型 IoT プラットフォームです。
- 時間の経過に伴う最適化: AWS ウェブサイトの「AWS IoT ブログ」セクションは、AWS IoT の一部として新しく発表された内容について学習するためのリソースです。

リソース

コスト最適化に関する AWS のベストプラクティスの詳細については、以下のリソースをご参照ください。

ドキュメントとブログ

- AWS IoT ブログ

まとめ

AWS Well-Architected フレームワークの目的は、効率が良く、費用対効果が#く、安全で信頼のおける IoT アプリケーションに対するクラウド対応システムを設計して運#するための柱のアーキテクチャに関するベストプラクティスを提供することです。このフレームワークには、既存の IoT アーキテクチャまたは提案された IoT アーキテクチャを評価するための一連の質問と、それぞれの柱についての一連の AWS ベストプラクティスも提供しています。このフレームワークをアーキテクチャに適用し、安定した効率のよいシステムを構築することにより、機能面の要件に注力できます。

寄稿者

本ドキュメントは、次の人物および組織が寄稿しました。

- Olawale Oladehin 氏、IoT、ソリューションアーキテクトスペシャリスト
- Dan Griffin 氏、IoT、ソフトウェア開発エンジニア
- Catalin Vieru 氏、IoT、ソリューションアーキテクトスペシャリスト
- Brett Francis 氏、IoT、製品ソリューションアーキテクト
- Craig Williams 氏、IoT、パートナーソリューションアーキテクト
- Philip Fitzsimons、シニアマネージャー (Well-Architected)、アマゾン ウェブ サービス

改訂履歴

このホワイトペーパーの更新に関する通知を受け取るには、RSS フィードをサブスクライブしてください。

変更	説明	日付
ホワイトペーパーの更新	IoT SDK の使用、ブートストラップ、デバイスライフサイクル管理、IoT に関する追加のガイダンスが含まれるように更新されました。	December 23, 2019
初版発行	IoT レンズが初めて公開されました。	November 1, 2018

注意

お客様は、この文書に記載されている情報を独自に評価する責任を負うものとします。本書は、(a) 情報提供のみを目的としており、(b) AWS の現行製品と慣行ついて説明していますが、予告なしに変更されることがあり、(c) AWS およびその関連会社、サプライヤーまたはライセンサーからの契約上の義務や保証をもたらすものではありません。AWS の製品やサービスは、明示または暗示を問わず、一切の保証、表明、条件なしに「現状のまま」提供されます。お客様に対する AWS の責任は、AWS 契約により規定されます。本書は、AWS とお客様の間で行われるいかなる契約の一部でもなく、そのような契約の内容を変更するものでもありません。

© 2019 Amazon Web Services, Inc. or its affiliates.All rights reserved.