

IoT Edge Working Group

エッジネイティブ アプリケーション 設計指針 ホワイトペーパー

著者

Frank Brockners - Cisco Systems

Joel Roberts - Cisco Systems

Kate Goldenring - Fermion

Andy Anderson - KubeStellar / IBM Research

レビューワー

Brandon Wick - Aarna Networks

Herve Moyal - Cisco Systems

R. Prakash - eOTF

Tomoya Fujita - Sony

Steven Wong - VMWare



CLOUD NATIVE
COMPUTING FOUNDATION

出版

2023年10月23日（第1版）

目的

CNCF IoT Edge Working Groupによって開発された**Edge Native Application Principles Whitepaper**（2023年1月17日に最初に公開）に基づいて、この補足文書には、エッジ環境用のアプリケーションを開発するための設計指針を推奨することによって、実践に移すことができる原則が含まれています。

クラウドネイティブ アプリケーション設計のベストプラクティスは十分に確立されており、注目すべき例は“**Twelve-Factor App methodology**”です。エッジネイティブ アプリケーション設計は、クラウドネイティブ アプリケーション設計に基づいて構築されています。ただし、エッジとクラウドのいくつかの品質は異なります。その結果、エッジネイティブ アプリケーション設計には、いくつかのクラウドネイティブ設計原則が含まれていると同時に、エッジ固有の要件を満たすように拡張されてもいます。

目次

P3 1. エッジネイティブ アプリケーション設計

- a) エッジネイティブとクラウドネイティブのアプリケーション設計ドメイン
- b) エッジネイティブ アプリケーションのデプロイメント コンテキスト
 - エッジネイティブの制約

P5 2. エッジネイティブ アプリケーション設計指針

- 並行性/拡張性
- 依存関係とポリシー管理によるエッジの自律性
- 廃棄性
- 機能感度
- データの永続性
- メトリクス/ログ
- （エッジとノードの）オペレーション

P8 3. エッジネイティブ アプリケーション サンプル シナリオ

- a) まとめ&次のステップ
- b) 参加方法

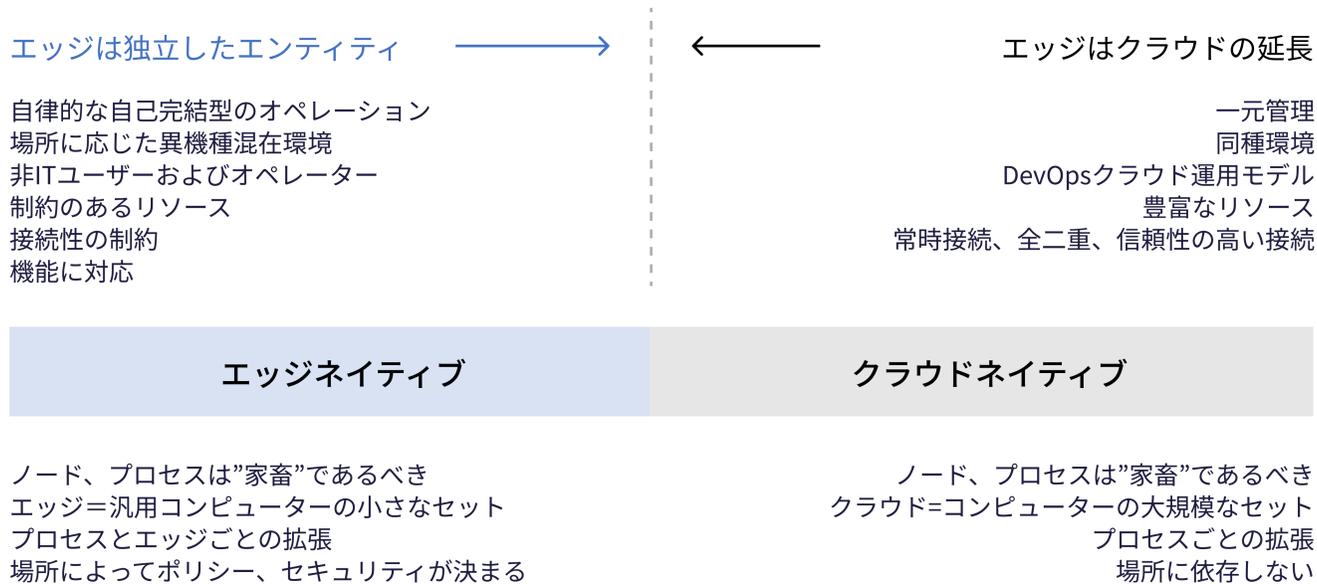


エッジネイティブ アプリケーション設計

1

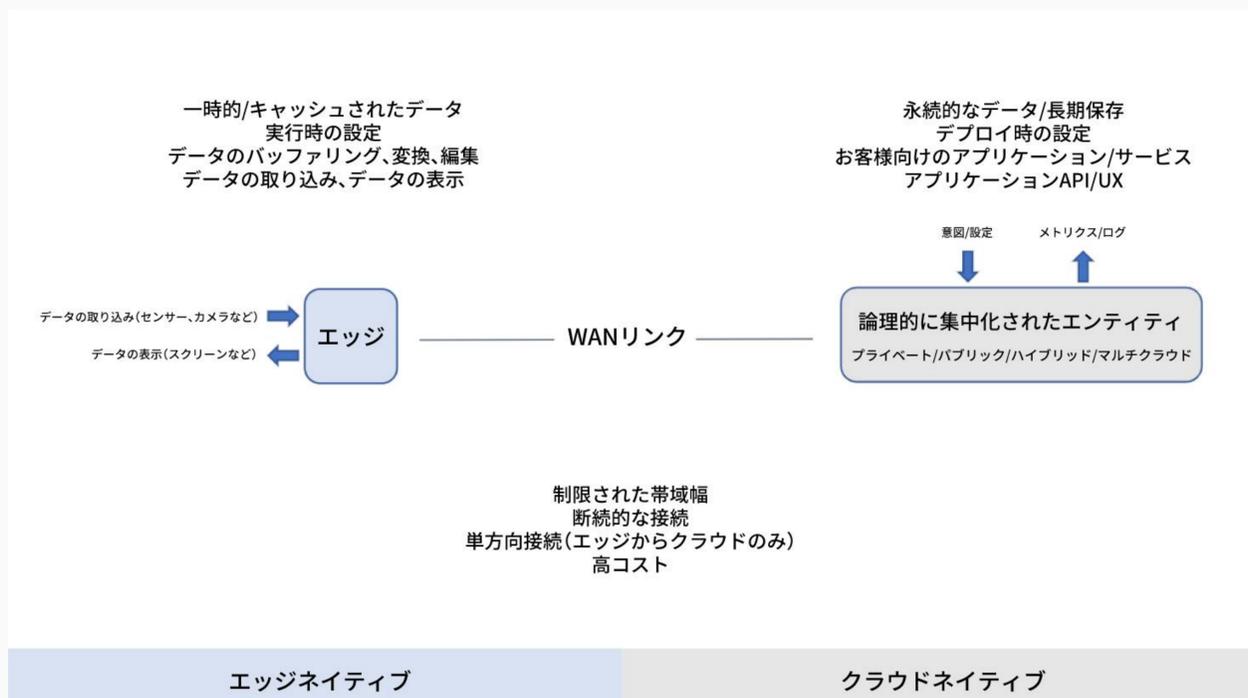
エッジネイティブとクラウドネイティブのアプリケーション設計 ドメイン

次の図は、「エッジネイティブ」設計指針が適用されるデプロイメントの概要を示しています。



エッジネイティブ アプリケーションのデプロイメント コン テキスト

次の図は、多くのエッジネイティブ アプリケーション シナリオに適用できるアプリケーション デプロイメントの参照を示しています。アプリケーションには、一連の分散エッジ コンポーネントと、論理的に集中化されたエンティティ（多くの場合、クラウドに配置されます）があり、分散エッジ コンポーネントは論理的に集中化されたエンティティを補完します。エッジ コンポーネントは、帯域幅消費の削減やロケーション ベースのポリシーの遵守など、エッジに存在する必要のある機能を処理します。エッジ コンポーネントの焦点は、多くの場合、データの取り込み、データのローカル変換、データのバッファリング、およびデータの表示にあります。このホワイトペーパーで説明するエッジ ネイティブ設計指針は、これらのタイプのデプロイメントを対象としています。



次の図は、一般的なエッジネイティブ デプロイメント コンテキストを示しています。分散エッジ（1つのエッジのみが示されています）は、ワイド エリア ネットワーク接続を介して論理的に集中化されたエンティティに接続されています。エッジは、（たとえば、センサーから）データを取り込み、（たとえば、画面に）データを表示するためにローカル デバイスとインターフェイスしています。全体的な設定/意図は、論理的に集中化されたエンティティを介してシステムに挿入されます。メトリクスとログは、論理的に集中化されたエンティティから取得されます。この図は、エッジと論理的に集中化されたエンティティの典型的な品質についても説明しています。これらについては、アプリケーション設計指針の一部として以下でさらに説明します。

論理的に集中化されたエンティティは、さまざまな層やさまざまな組織エンティティにわたって、さまざまな方法でデプロイされることができます。これは、Linux Foundation (LF) エッジ ホワイトペーパー “[Sharpening the Edge: Overview of the LF Edge Taxonomy and Framework](#)” に示されているように、エッジの連続体と見なすことができます。



LFエッジ連続体の中で参照される、考慮すべきエンティティ：

- 一元化されたデータセンター (CDC/ハブ/親)
- サービス プロバイダー エッジ (ESP/中間/親または子)
- ユーザー エッジ (UE/スポーク/子/ピア)

一般的に、一元化されたデータセンターの外部で、エッジネイティブの原則と動作が適用されます。言い換えれば、制約された環境は、ユーザー エッジからサービス プロバイダー エッジへの連続体にわたって可能です。全体的に、リソースのプロパティとそれに関連するポリシーは、エッジのタイプ（制約されたデバイス エッジ、スマート デバイス エッジなど）やエッジ デプロイメントの場所（ソブリン、単一ロケーションのスタンドアロン デプロイメント、地域デプロイメント、または複数地域デプロイメントなど）よりも、何を”エッジネイティブ”として分類するかの関係性を作成します。

エッジネイティブの制約

以下の箇条書きは、エッジ ネイティブ アプリケーションを設計する際に一般的に考慮する必要がある設計制約のカテゴリを定義しています。

- 接続性の制約（すなわち、転送中のデータまたはネットワークの制約）には、帯域幅の制限、断続的な接続性（”エアギャップモード”）、エッジとクラウドまたは他のエッジ間の遅延およびジッターが含まれます。さらに、接続性の制約には、転送中のデータに適用されるポリシー/セキュリティ関連のルールまたは変換を含めることができます。たとえば、特定のユースケースでは、データを別の場所に転送する前に、データを匿名化し、個人を特定できる情報（PII）を削除する必要があります。接続性の制約には、ネットワーク アドレス変換またはパケット フィルタリングを実行するミドルボックスによって課される制限も含めることができます。
- 保存データの制約には、セキュリティまたはポリシーの要件と、特定のデータ変換や保存を地域や国などの特定の場所で行うことを要求する関連規則が含まれます。
- リソースの制約には、電力、メモリー、スペース、計算能力など、エッジで利用可能となるリソースによる制限が含まれます。

一般に、さまざまなタイプの制約は、次のセクションで詳しく説明するさまざまな設計指針に適用できますが、適用可能性は配置によって異なります。

エッジネイティブ アプリケーション設計指針

2

”エッジネイティブ”アプリケーション設計とは、アプリケーションがエッジの制約に対処できるように、クラウドネイティブアプリケーション設計の方法論を進化させ、補完することを意味しています。本稿では、**12-factor methodology** をクラウドネイティブアプリケーション設計の参照方法論として考察します。データとコードの分離、ステートレスやシェアナッシング エンティティとしてのプロセスの概念、ビルド ステージと実行ステージの分離など、クラウドネイティブアプリケーション設計の基本原則は、エッジにも適用できます。以下に示す設計指針のリストは、エッジネイティブアプリケーションを設計する際に、進化した、再考された、または新規となる設計指針です。このリストは、エッジネイティブアプリケーションを構築する開発者のための参照として役立つはずですが、

並行性/拡張性

CNCF IoT Edge WG “**Edge Native Application Principles Whitepaper**”で定義されているように、エッジネイティブアプリケーションは”スパンニング”です。地理的に拡張して複数のエッジにまたがることも、エッジ内のプロセスによって拡張して複数の障害ドメイン境界にまたがることもできます。

- **位置またはエッジにまたがる拡張**：アプリケーションは”エッジ”にデプロイされ、各”エッジ”は1つ以上の計算ノードになります。”エッジ”は、1つ以上のアプリケーションを実行し、小規模な多目的計算ノードで構成される計算ノードの物理的または論理的なグループを表します。多くのエッジが存在する可能性があります。同じアプリケーションのインスタンスを多くのエッジに同時にデプロイできます。
- **位置またはエッジ内の拡張**：アプリケーションは、複数のノードで並列に実行することでメリットを得られるように設計されています（12-factor methodologyでは、これを”プロセスモデルによるスケーリング”と呼んでいます）。エッジ内で必要なリソースは、単一の大規模な計算ユニットではなく、小規模で低コストのデバイスのグループを活用することで満たされます。複数のデバイスをグループ化/クラスタリングすることで、エッジローカル コントロール プレーンの高可用性メカニズムを活用することもできます（例えば、エッジでKubernetesクラスタが使用されている場合は、Kubernetesコントロール プレーンの高可用性メカニズムを使用できます）。マルチアーキテクチャ（x86、ARM）により、さまざまなユースケースに対応できます。エッジ全体が（1つの場所にあるすべてのデバイスと同様に）障害を起こす可能性があります。障害からの回復時に中央ハブ（クラウド ホスト）から取得できる状態を使用して再ブートストラップされます。

依存関係とポリシー管理によるエッジの自律性

クラウドへの接続が失われた場合でもアプリが動作できるように、すべての依存関係を事前に宣言します。

- 依存関係とポリシーの明示的な宣言と分離—エッジローカルとクラウド
- エッジは、ピアまたは親から情報を引き出し、それに応じて結果を投稿します。
- エッジは自律的に動作します—エッジの宣言された状態または意図、およびエッジで実行されているアプリケーションは、親またはピアで宣言され、それに応じて取得されます。
- エッジは、ノードが果たす役割、ノードへのジョブの割り当てやスケジューリングなどを自律的に決定します。
- エッジは、親またはピアから切断されても（1つのロケーション内および可能な場合はロケーション間で、いつでも）動作を継続します。
- エッジは、エンティティに到達してアクセスし、宣言された状態や意図を時々取得することができます。このエンティティは通常、親によってホストされますが、ピアからも可能です。エッジの外部からエッジへの接続（例えば、クラウドからエッジへ）が確立できるかどうかの仮定は行われていません。例えば、親がエッジとの接続を開始しないか、永続的なトンネル、または”常時接続モード”が想定されます。これにより、エッジが親またはピアから設定をプルしていることが保証されます。親またはピアへのプッシュはオプションですが、セキュリティと管理がより困難になります。
- エッジで実行されているアプリケーションおよびサービスのバックアップ サービスは、接続された、潜在的にリモートのリソースとして扱われます。



廃棄性

失敗を念頭に置いた設計：ノード、エッジ、アプリはいつでも失敗することが許されます。

- エッジによる堅牢性：エッジは迅速に起動し、適切にシャットダウンする必要があります
- エッジは、宣言されたステート（起動時と実行時の設定を含む可能性が高いです）を親またはピアから取得します。この親またはピアは、宣言されたステートまたは意図をホストし、中央エンティティに代わって仲介者またはエッジ サービス プロバイダーを含む階層化されたデプロイメント モデルを含む、クラウドホストされることができます。
- エッジに障害が発生した場合、そのエッジに機密データが含まれていて、自己防御モード/姿勢で意図的に”ブリック化”されていない限り、そのエッジが”ブリック化”されていない、または回復不能でないことを保証する必要があります。

機能感度

エッジネイティブ アプリケーションは、エッジが環境やデプロイメント コンテキストを認識するための手段をプロビジョニングする必要があります。

機能感度とは、アプリケーションがその環境やデプロイメント コンテキストを認識すること、すなわち、利用可能な（ハードウェア）リソースや関連する、あるいは接続されたデバイスを認識することであり、エッジネイティブ アプリケーションのもう1つの原則です。[Edge Native Application Principles white paper](#) では、これらを”ハードウェア認識”と”外部リソースとの相互作用”のカテゴリの一部として説明しています。機能感度はクラウドアプリケーションでは一般的ではなく、12-factorの原則にも欠けています。

[Edge Native Application Principles Whitepaper](#) で説明されているように、エッジ アプリケーションは多くの場合、ハードウェア、外部デバイス、およびネットワークの可用性という形でその機能を認識しています。より具体的には、内部および可能であれば外部で利用可能なネットワーク帯域幅、計算能力、計算タイプ（CPU、GPUなど）、センサー、カメラ、アクチュエータ、ユーザーなどについて動的に”知識を持つ”必要があります。この”知識”がなければ、ローカル環境の値を設定入力として使用して、インストール時または実行時にカスタマイズを可能にしながら、大規模に適用できる共通の設定を作成することはできません。

データの永続性

永続化が必要なデータはすべて、専用のステートフルなバックアップ サービスに保存することが推奨されます。可能な限り、エッジはステートレスで、バッファ/キャッシュデータのみであるべきです。エッジ アプリケーションがステートレスであればあるほど、移植性と再利用性というエッジネイティブ アプリケーションの原則を満たすことができます。

- ストレージを必要とするデータのプロパティ（保持、機密性、サイズなど）に応じて、エッジネイティブ アプリケーションはデータをエッジ ロケーション上にローカルに保持することができます。アクション可能なデータについては、データを親またはピアに表示する必要があり、エッジ ロケーションから削除される場合があります。
- データ/ステート(構成、顧客データなど)は、通常、専用のステートフルなバックアップ サービス内の親またはピアに保持されます。エッジは、ローカル処理のニーズに応じて、また切断された操作の場合に、データをキャッシュまたはバッファすることができます。大量のデータは、通常、エッジで長期間保持されません（すなわち、親またはピアにデータと構成を格納することが望ましいです）。
- エッジで生成されたデータは、通常、一定の間隔でコントロール プレーンにポストされ、コントロール プレーンは、エッジがオフラインになったときを推測し、エッジからのデータをシーケンスまたは時系列だと期待しないようにします。
- 時系列アップストリームに適合させるためにデータを正確にスタンプするために、エッジで許容可能なレベルの時間同期が発生する必要があります。

(次ページへ続く)

- データはエッジの切断時に保存され、再接続時に転送されるべきです。
- すべてのエッジに対して宣言されたステートまたは意図を保持し、エッジによってポストされたデータの統合ポイントとしても機能する中央エンティティ（すなわち、コントロールプレーン）を保護するためのプロビジョンを追加する必要があります。この保護には、以前に切断されたエッジの大規模なセットがすべて同時にクラウドに接続しようとしても、コントロールプレーンに対するDoS攻撃が発生しないようにするためのプロビジョンが含まれます。
- エッジが切断モードの場合、ローカルストレージの可用性を考慮して、エージングデータの保存を制限する必要があります。限られたストレージを保持するために、データのエージングと破棄、フィルタリング、または要約を許可することをお勧めします。

メトリクス/ログ

クラウドネイティブアプリケーションと同様に、エッジネイティブアプリケーションは可能な限り中央で観測可能であるべきです。ログとメトリクスをオンデマンドストリームとして扱い、可能な限りメトリクスを使用します。

- 可能な場合は、メトリクスとログを親にストリーミングまたはプッシュします。ログはオンデマンドでのみ使用可能になります(エッジと親の間の帯域幅を節約するため)。
- ”実行可能な”メトリクスに焦点を当てます。”データ”ではなく”情報”を送ることが望ましいです。
- エッジでのログデータは、通常、一定の間隔でコントロールプレーンにポストされ、コントロールプレーンがエッジがオフラインになったときを推測し、エッジからのデータをシーケンスまたは時系列だと期待しないようにします。
- ログデータは、切断時に保存し、再接続時に転送する必要があります。ここでは、前述の”ストレージ”のための中央ログ/メトリクスサーバーを保護するための考慮事項が適用されます。
- エッジが切断モードの場合、使用可能なストレージの制約を考慮して、エージングログデータのストレージを制限する必要があります。制限されたストレージを保持するために、ログデータをエージングして破棄、フィルタリング、または要約できるようにすることが推奨されます。これは、上述した、”情報”を送信する代わりに”実行可能なメトリクス”を送信する、という点とも非常に関連しています。

(エッジとノードの) オペレーション

アプリの専門家ではないオペレータを想定します。アプリによって公開されるメトリクス/ログは実行可能である必要があります。

- 専門家ではないオペレーターを想定します。エッジロケーションの現場担当者にITスキルセットを想定すべきではありません。
- エッジとノードは使い捨てであり、すぐに開始または停止できる必要があります。
- エッジ、ノード、およびそれらが実行するプロセスのいずれも、適切に機能しない場合に”デバッグ”されません：エッジでの問題を解決するための”ファクトリーリセット再ブートストラップ”
- アプリケーションは通常、設定ミス回避のために、ゼロタッチプロビジョニングガイドラインと互換性があります。
- ラストノングッドステートへのグリーンブーティング（またはロールバック）は、問題の解決にも役立ち、ログを使用して更新またはアップグレードが失敗した理由を判断しながら、操作を再開できます。
- **コンフィグレーション**
 - アプリケーションおよびエッジのコンフィグレーションステートは、通常、論理的に集中化されたエンティティ（通常は親、階層型または分散型のデプロイメント、仲介またはエッジサービスプロバイダーなどの、この論理的に集中化されたエンティティに適用）に保持されます。
 - アプリケーションのスタートアップコンフィギュレーションまたはパラメータ化は、すべてのエッジに対して汎用である必要があります。エッジごとのランタイムコンフィギュレーションは、論理的に集中化されたエンティティから実行時にアプリケーションによって取得されます。
 - 可能な限り、限られた量の設定データしか必要とせず、どこに配置されていてもデフォルト値で適切に実行できるように、アプリケーションの開発を試みてください。これにより、多くのエッジにわたって簡単に型にはまった配置が可能になります。

エッジネイティブ アプリケーション サンプル シナリオ

3

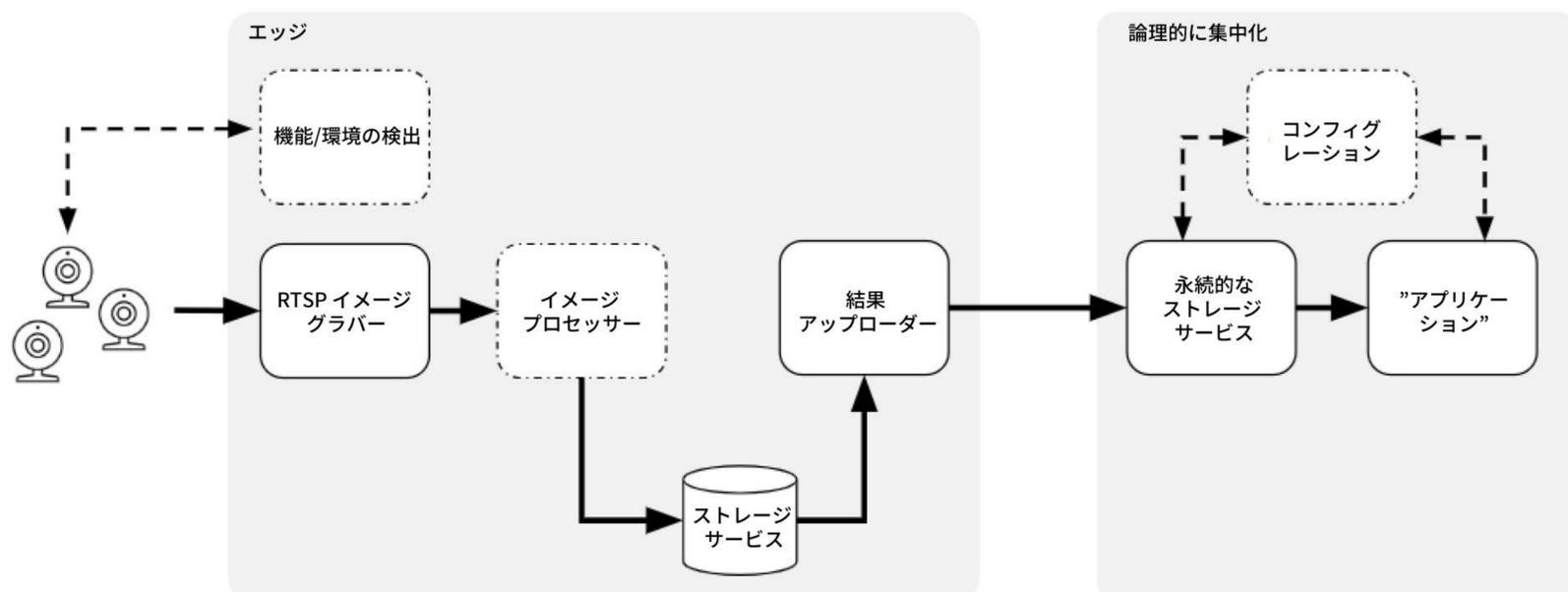
Reliable Video Capture and Transformation (VCAT) は、セキュリティやコンプライアンスを目的とした監視、顧客の行動や人出の分析など、多くのエッジユースケースに共通するコンポーネントです。これらのユースケースは、小売、クイックサービスレストラン、輸送、物流など、多くの業種で見られます。1つ以上のカメラが、後で参照するために保存するか、情報を抽出するために処理する必要があるビデオをキャプチャします。カメラの場所とクラウド間の接続は制限されているため、処理または保存のためにカメラ ストリームをクラウドに送信できません。接続に必要な帯域幅がないか、コストがかかりすぎるか、停止の可能性があるため、クラウドへのアップロードが信頼できない場合があります。また、セキュリティまたはポリシー上の理由により、クラウドへのアップロードが実行できない場合があります。たとえば、クラウドに送信される画像に個人識別情報 (PII) を含めることができないとします。ビデオ関連の制約に加えて、エッジ サイトは、ネットワーク アーキテクチャおよび関連するネットワーク セキュリティによる追加の接続制約に直面する可能性があります。

ソリューションの例

Reliable Video Capture and Transformation (VCAT) のソリューション例は、次の機能を提供します。

- 複数のカメラからビデオストリームを受信してサンプリングします。
- 受信したフレームを処理します。処理には、次のような複数の変換を含めることができます。
 - 特定のユーザ定義のフレーム レートで着信フレームをサンプリングします。
 - イメージのサイズをユーザー定義のサイズに変更します。
 - イメージ内のオブジェクトを検出してマーキングします。
 - 画像からPIIを除去します(例えば、顔をぼかします)。
- イメージをエッジにローカルにバッファ/キャッシュし、エッジからクラウドへの接続時にすぐに、またはユーザーが設定した時間に、それらの処理結果をアップロードします。
- 画像をクラウドに保持します(長期保存)。
- アプリケーションによってクラウドにアップロードされたイメージをさらに処理します。これは、監視、分析など、特定のユースケースを実装する任意のタイプのアプリケーションに行うことができます。

次の図は、エッジに常駐する機能とクラウドに常駐する機能を含む、サンプル ソリューションのアーキテクチャーの概要を示しています。



この図は、アプリケーションが3台のカメラからのビデオ ストリームを処理するVCATアプリケーション実装のシナリオを示しています。アプリケーションのエッジ部分には、カメラとその機能を自動的に検出するコンポーネントを含めることができます（点線で囲まれた”機能/環境の検出”ボックスを参照）。エッジ アプリケーションには、ビデオ ストリームを受信し、ストリームをイメージのシーケンスに変換するRTSPイメージ グラバー コンポーネントが含まれます。”イメージ プロセッサ”コンポーネントは、イメージをさらに処理します（たとえば、イメージを編集してPIIを削除します）。処理後、イメージはローカル キャッシュ サービスに渡されます。結果 アップローダー コンポーネントは、ローカル ストレージ サービスからイメージを取得し、そのイメージをアプリケーションの論理的に集中化されたエンティティに送信します。ローカル ストレージ サービスは、データの取得とデータのアップロードを分離します。エッジが論理的に集中化されたエンティティに接続されていない場合、イメージはローカル ストレージ サービスによってバッファされます。論理的に集中化されたエンティティは、エッジからイメージを受信し、永続的なストレージ サービスに格納します。イメージのその他の使用方法および全体的な設定の構成は、”コンフィグレーション”および”アプリケーション”として示される汎用コンポーネントに抽象化されます。

エッジ ネイティブ アプリケーションの設計指針はどのように適用されますか？

エッジネイティブ アプリケーションの設計指針は、次のようにVCATアプリケーションのVCAT-エッジおよびVCAT-Cloud部分に適用されます。

- 並行性/拡張性：VCATソリューションのエッジ部分（VCAT-エッジ）は、各エッジ サイトに配置された一連のコンピュータ ノードに導入されます。各エッジ サイトは、同じデプロイメント コンフィグレーションで導入されます。
- 依存関係/ポリシー/エッジの自律性：VCAT-エッジは、クラウドへの接続が失われても動作を継続するように設計されています。接続が失われた場合、イメージは常に最初にローカル ボリュームに書き込まれるため、ビデオ ストリームのキャプチャが継続され、情報が失われることはありません。
- 廃棄性:VCAT-エッジ インスタンスに障害が発生すると、自動的に再起動され、VCAT-Cloudインスタンスから必要なデプロイメント コンフィグレーションが取得されます。データはVCAT-Cloudインスタンスに保持されるため、障害が発生した場合はエッジでキャッシュされたデータのみが失われます。
- 機能感度：各VCAT-エッジ インスタンスは、そのローカル環境を”検出”します。すなわち、カメラが表すRTSPエンドポイントのIPアドレスを特定します。ローカル機能または環境検出とは、VCAT-エッジ インスタンスが、明示的な設定を必要とせずに、その動作をローカル環境に自動的に適応させることを意味します。明示的な設定は、大規模で絶えず変化する環境では管理が困難です。
- データ ストレージ：すでに説明したように、VCAT-エッジ インスタンスはデータをキャッシュ/バッファするだけです。
- メトリクス/ログ：運用環境では、VCAT-エッジは、専門家ではないオペレータが実行できる一連のメトリクスを提供します。これらのメトリクス/ログは、たとえば、VCAT-エッジ アプリケーションまたはエッジ全体を再起動する必要があるかどうか（すなわち、専門家ではないオペレータが実行できるアクション）を示します。

オペレーション：VCAT-エッジは、サイト固有の設定を必要としないように設計されています。少なくとも、検出操作やAPI呼び出しなどによって自動的に設定できないサイト固有の設定は必要ありません。

まとめ & 次のステップ

本論文は初版であり、改訂される可能性があります。本論文のサブセクションに関連する将来の論文が予想されます。フィードバック、コラボレーションリクエスト、または質問をするためには、[CNCF Runtime TAG GitHub page](#)でイシューを作成してください。

参加方法

CNCF IoT Edge Working Groupには、定期的なミーティング、メーリングリスト、Slackがあります。最新情報については、ワーキンググループのGitHubページの[Communication section](#)を参照してください。エッジ関連のプロジェクトを紹介したり、グループの作業領域に関するアイデアを紹介したり、このホワイトペーパーの改訂やフォローアップペーパーのドラフト作成を支援したりすることで、読者が参加することを歓迎します。

この日本語レポートは、以下の文書の参考訳です。

[Edge Native Application Design Behaviors Whitepaper](#)

翻訳協力：橋本修太