JUNIPER NETWORKS | Engineering Simplicity

ホワイトペーパー

JUNIPER APSTRA アーキテクチャ

ネットワークライフサイクル全体に特化した自動化でデータセンターの困難な問題を解決

目次

はじめに 3
主な課題：構成 3
知識は力なり：変更への確実な対応 5
ステートフルオーケストレーション 5
延伸性（Extensibility）：将来性のあるデータセンターネットワーク 12
拡張性（Scalability）：痛みを伴わない成長 14
Apstraアーキテクチャの概要 15
Apstraアーキテクチャのメリット 17
まとめ 18
ジュニパーネットワークスについて 18

概要

データセンターネットワークの構築、導入、運用、管理、トラブルシューティングは容易ではなく、高額な費用が発生し、リソースを大量に消費することがあります。2019年のGartnerレポートによると、ITインフラストラクチャの簡素化は、今日のビジネスにとって最優先の戦略です。Juniper® Apstraは、データセンターネットワークのライフサイクル全体を自動化するために設計されています。

このホワイトペーパーでは、今日のデータセンターネットワークチームが直面している最も困難な問題をいくつか取り上げ、Apstraのアーキテクチャによって、それぞれの問題にどのように対処しているかについて説明します。

はじめに

今日のデータセンターネットワークアーキテクチャおよび運用チームは、以下にあげる4つの主な課題に直面しています。

- **構成**：それぞれが異なる機能を備え、さまざまなベンダーのインフラストラクチャを使用して、一貫性のあるスムーズに機能するシステムを構築することは、決して容易なことではありません。Juniper® Apstraは、表現されたインテントに基づいてシステムを導入するために必要なすべての情報で構成される、一貫性のある全体像（ブループリント）を作成します。ブループリントは、物理インフラストラクチャにプッシュされるため、オペレーターは信頼性が高く、使いやすいサービスを提供できます。
- **確実な変更**：データセンターでは常に変更が発生します。オペレーターによる新しいサービスの追加や容量の拡張によって変更が生じる場合、または障害状態のインフラストラクチャから生じることもあります。いずれにせよ、オペレーターは変更に対処するために、より効果的な方法を見つけなければなりません。近年のさまざまな調査によると[1]、ネットワーク中断の障害の65～70%は、予定された変更の実行中に発生した人為的な構成エラーが原因です。
- **延伸性（Extensibility）**：イノベーションを促進し、テクノロジーでは避けることのできない急速な変化に対応するために、データセンターでは新しい機能を簡単かつスムーズに追加できる必要があります。テクノロジーの進化に伴い、ベンダーはイノベーションを推進し、オペレーターが競争力の維持に活用できる新機能を考案します。このような進化により、設計時に変更が発生し、導入するサービスを管理するための新たな動作規定（またはリファレンスデザイン）が必要になります。そのため、イノベーションを促進し、新しい要件に対応するためにも、拡張可能な構成でなければなりません。
- **拡張性（Scalability）**：大きな規模でこれらすべてに対処するために、データセンターネットワークは拡張が可能で、イノベーションや複雑化に対応できる必要があります。

Apstraアーキテクチャの主な目標は、このような問題に直接対処することです。

主な課題：構成

今日、コンピューティング/ネットワーク/ストレージインフラストラクチャを実行しているオペレーターが直面している主な課題は、構成です。構成とは、機能コンポーネントの単なる集合体よりも大きい一貫性のある全体像を作り上げ、避けることのできない変更が存在する場合も、その一貫性を長期的に維持し、ネットワークサービスを一般ユーザーに提供する機能です。つまり、Apstraは構成の課題を解決できるよう設計されています。

最新のデータセンターは容易にスケールアウト可能です。そのため、今日、ホストオペレーティングシステムが単一のマシンで提供しているものと同様の機能を提供するオペレーティングシステムが必要です。その機能とは、リソース管理とプロセス分離です。

コンピューティングの仮想化では、単一サーバーのコンピューティングに対して、すでにこれを行っています。ところが、スケールアウトとしてのデータセンターのコンピューティングでは、データセンターのオペレーターが最初にデータセンターを構成する必要があります。それによって初めて、リソースのパーティション分割ができるようになります。

[1] 次に例を示します。https://www.computerweekly.com/news/2240179651/Human-error-most-likely-cause-of-datacentre-downtime-finds-study、https://www.networkworld.com/article/3142838/top-reasons-for-network-downtime.html; https://www.ponemon.org/library/national-survey-on-data-center-outages

構成はなぜそこまで難しいのか？

構成がそうした課題になるには、主に2つの理由があります。1つ目は、一貫して全体を構成するのに使用される要素は、さまざまなベンダーから取得する場合、さまざまな機能を備えている場合、あるいはさまざまなAPIを介してしか利用できない場合があることです。

2つ目は、到達可能性、セキュリティ、エクスペリエンスの品質、可用性など、複数の機能面を備えたさまざまなサービスを提供できるよう、インフラストラクチャの構成が必要になる場合があることです。

複数のサービスインスタンスをインスタンス化すると、このようなサービスを実施するメカニズムにマッピングする方法を十分に理解していない限り、コンポーネント間のやり取りによってサービスが停止することがあります。インフラストラクチャの機能は次第に進化するため、既存のシステムを破壊することなく、新たなイノベーションを活用して導入できるようにする必要があります。

Apstraはリファレンスデザインの課題に取り組みます

Apstraアーキテクチャが構成の課題への対処に使用する重要な概念は、リファレンスデザインです。リファレンスデザインとは、ユーザーのビジネスインテントを実施メカニズムにマッピングする方法と、インテントが達成されると見なされるために満たさなければならない期待値について定義する動作規定です。

リファレンスデザインは、以下について規定します。

- 物理的および論理的コンポーネントの役割と責任
- サービスが実施メカニズムにどのようにマッピングされるか
- 満たす必要のある期待値（監視する状況など）

リファレンスデザインでは、物理的要素と論理的要素の役割と責任が明確に定義されているため、モデリングの範囲が限定され、最小限でありながら包括的なモデルの仕様が決まります。また、インテントが実施メカニズムにどのようにマッピングされるのかについても規定します。このマッピングを理解することで、トラブルシューティングの自動化が可能になり、ネットワークの発生事象をリバースエンジニアリングで解決するのではなく、システムの構成方法に関する知識に基づいた強力な分析を行うことができます。

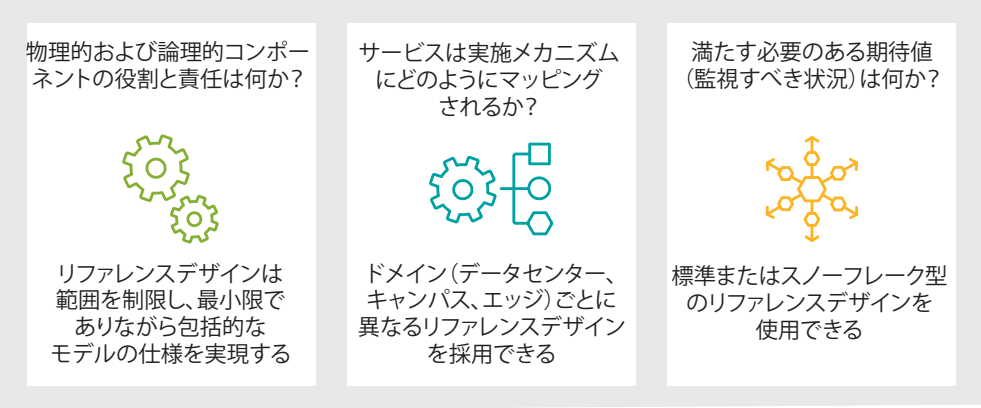

図1：リファレンスデザインは、統一した動作を規定します。

リファレンスデザインが異なる場合は、同一のインテントがより新しく、より革新的な別のメカニズムで実施されることがあります。このマッピングを理解することで、根本原因の特定が可能になり、サービスに影響を与える問題の原因をオペレーターに直接指摘することができます。

リファレンスデザインでは、検証対象の期待値も指定します。たとえば、リファレンスデザインでは、各リーフがそれぞれのスパインとのBGPセッションを確立する必要があると規定されている場合があります。この方法で、欠落や予期しないBGPセッションを簡単に特定できます。

リファレンスデザインの定義は、ネットワークの専門家が行うべき付加価値のあるアクティビティです。Apstraを使用すると、専門家だけが専門知識を独占するのではなく、システム全体で専門知識が明確になります。その結果、専門知識はハードコードする代わりに、モデル化することでシステムに挿入できます。

知識は力なり：変更への確実な対応

データセンターでは変更が常に発生するので、いかなる時も確実に対応することが最も重要です。変更は以下の2つから生じます。

> 「最も強いものが生き残るのではなく、最も賢い者が生き延びるのでもない。唯一、生き残るのは変化に最も対応できる者である」
>
> - チャールズ・ダーウィン、1809年

1. **オペレーターのインテント**：オペレーターは、仮想ネットワークなどの新しいサービスの追加、インフラストラクチャからのリソースの追加/削除、一部のポリシーの変更ができます。オペレーターは、変更の開始を制御できます。Apstraは、ステートフルオーケストレーションを使用して、オペレーターのインテントを確実に実現します。
2. **マネージドインフラストラクチャの障害**：マネージドインフラストラクチャに障害（過度なパケット損失やトラフィックの不均衡など）が発生することで変更が生じることもあります。オペレーターは、このような状況で生じた変更を制御できず、運用状態の量に圧倒されることがよくあります。Apstraは、オペレーターが運用状態の変化に対処するのに役立つインテントベースの分析を提供します。

上記の2つのタイプの変更に確実に対処するには、インフラストラクチャの状態に関する知識が必要です。以下の2つの条件を満たしている必要があります。

1. 知識はコンテキストで解釈する必要があります。つまり、「コンテキストモデル」セクションで説明しているように、状態と期待値の関係を解釈します。
2. 知識には適時性が必要なため、「リアルタイムの監視と通知」セクションで説明しているように、知識は現在の状態が反映されています。

ステートフルオーケストレーション

ステートフルオーケストレーションは、インテントを使用して、オペレーターによる変更の信頼性を高めます。ステートフルオーケストレーションフローのアクションを図2に示します。

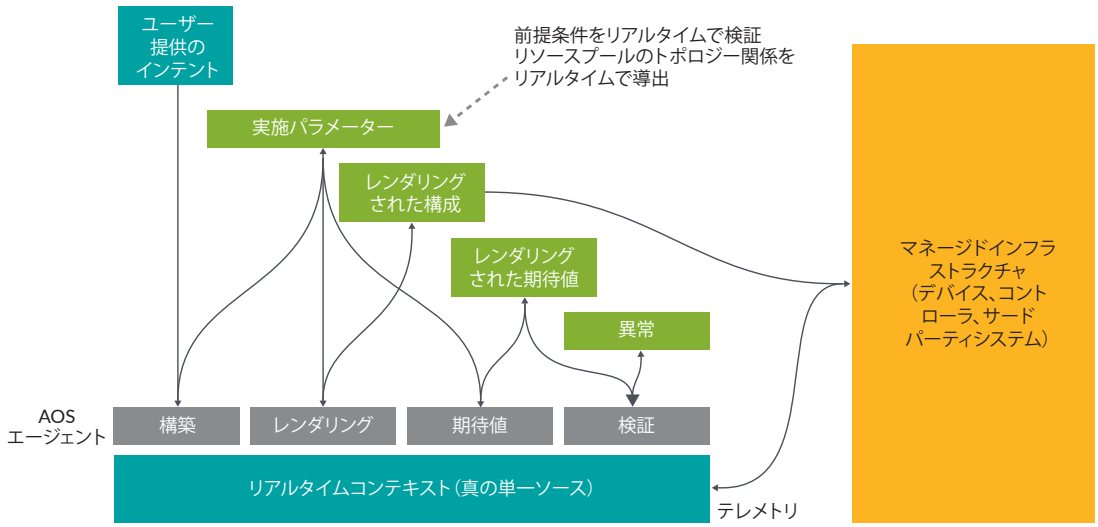

図2：ステートフルオーケストレーションのフロー

完全に機能する既存のシステムに変更を加える場合、ユーザーはステートフルオーケストレーションを使用することで、変更のインテントのみを提供できます。これは実装に依存しない方法で行われるため、インテントの指定がシンプルになり、エラーが発生しにくくなります。ステートフルオーケストレーションで実行される一連のステップは、以下のとおりです。

ステップ1：前提条件のリアルタイム検証 - この新しいリクエストは、一部のポリシーに違反することになるかどうか？たとえば、この仮想ネットワークの構築が許可されているかどうか、あるいはセキュリティホールが生じることになるか？または、他の一部のデバイスがすでにオフラインになっていて、このデバイスをオフラインにするとシステムが脆弱になることがわかっているので、デバイスをメンテナンスモードにできるかどうか？ステートフルオーケストレーションでは、このような質問がすべてリアルタイムで表示され、コンテキストグラフで検証されます。

ステップ2：実施パラメーターのリアルタイム導出 - インテントを実装するには、適切な実施メカニズムの特定パラメーターを指定して、特定の管理要素でアクティブ化する必要があります。Apstraで使用するリファレンスデザインでは、インテントの実装方法が詳しく記述されており、現在のコンテキストを理解することで、システムは実施パラメーターのリアルタイム導出を自動的に実行できます。そのため、オペレーターが誤って間違ったコマンド、インターフェイス、IPアドレス、VLANを挿入するようなことはありません。

ステップ3：マルチ構成のリアルタイム導入 - Apstraは、ベンダーやテクノロジーに固有のAPIが異なる場合の複数要素での構成導入を自動化し、マルチ構成の導入をリアルタイムで実行します。

ステップ4：期待値のリアルタイム生成 - 動作規定として機能するリファレンスデザインにより、Apstraはリアルタイムの期待値生成を実行できます。リファレンスデザインには、結果がインテントを満たしていることを宣言するのに必要な条件が記述されています。

ステップ5：期待値のリアルタイム検証 - 次に、Apstraはテレメトリとコンテキスト対応の運用分析のコレクションをトリガーして、期待値のリアルタイム検証を実行します。

ステップ6：検証済みサービス成果 - このプロセスの最後に、ユーザーは検証済みのサービス成果をはっきりと確認できます。インテントまたは予想される運用ステータスの変更はすべて、リアルタイムでコンテキストモデルに反映され、変更を認識する必要があるコンポーネントにもリアルタイムで通知されます。Apstraは、インテントベースの分析を使用して、未加工の運用データから関連する知識を抽出します。

ステートレスオーケストレーションでは、ステップの多くが欠落しています（図3を参照）。一般に、ステートレスオーケストレーションの場合、構成導入の結果は、構成を複数のシステムにプッシュし、構成がマネージド要素によって受け入れられたことを検証することがすべてです。そのため、ステートレスオーケストレーションでは、これ以上先に進むことができず、サービスの期待値や、期待値が満たされているかどうかについて検証するという概念はありません。サービス成果の自動検証はなく、代わりに、ユーザーには個々のシステムから生の運用状態に関する情報が「1つの画面」に表示されます。成果が達成されたかどうかを視覚的に確認するのはユーザーの責任です。ただし、1つの画面ですべての情報を表示する場合、一般に情報が多すぎて実装固有のコンテキストが欠如しているため、視覚的に確認することがきわめて難しく主観的なものになります。

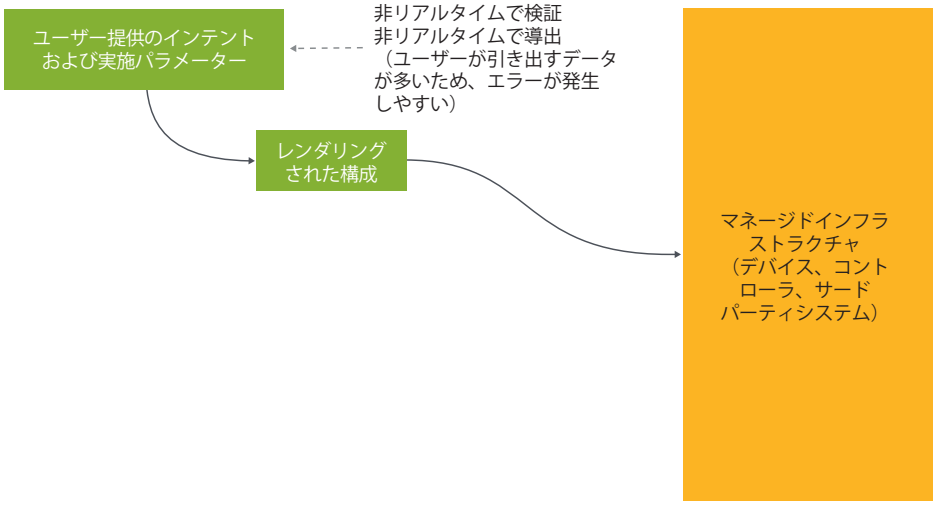

図3：一般的なステートレスオーケストレーション

インテントベースの分析

IBA（インテントベースの分析）は、未加工のテレメトリデータから知識を抽出することで、オペレーターがインフラストラクチャ運用状況の変更に対処できます。前述のように、IBAを使用するには、リアルタイムでクエリ可能なコンテキストが必要です。

知識の抽出は、以下にあげる2つのステップで構成されます。

1. **関心のある状態の検出**（監視する状況）。状態の検出は、IBAプローブによって行われます。
2. **関心のある状態の自動分類およびそれらの間の関係性の導出**。条件はセマンティックコンテンツによって異なります。つまり、コンテンツの一部は他のコンテンツより重要です。状態間の関係を理解することは重要です。それにより、重要な実行可能の状態（根本原因）を特定でき、重要な根本原因が解決されると消失するような単なる結果としての状態を理解できます。状態の分類と因果関係の導出は、IBAの根本原因特定するコンポーネントによって行われます。

関心のある状態のモデル化

IBAでは、4つのカテゴリーを使用して、関心のある状態をモデル化します。そのカテゴリーとは、異常、症状、影響、根本原因です。

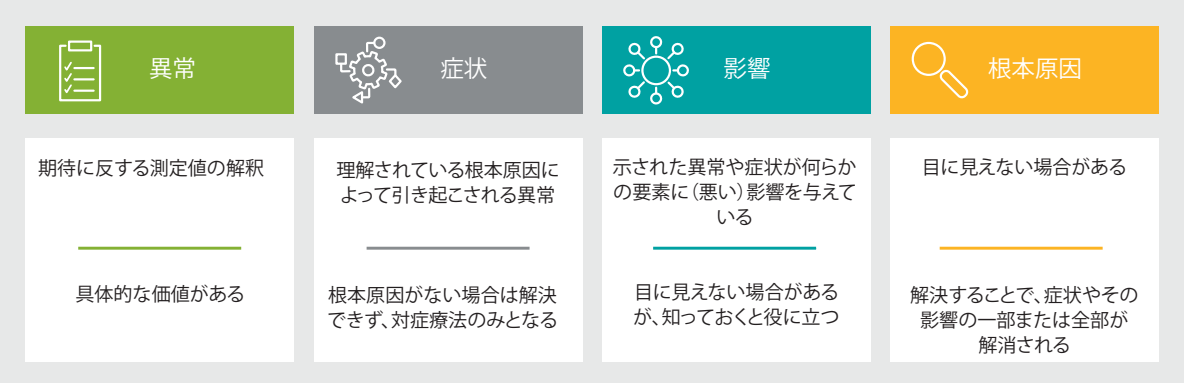

図4：状態の分類

異常

異常は、基本的には、期待に反する測定値の解釈、またはその集合体を表します。そのため、単純な測定値よりも具体的な価値があります。

症状

症状は、十分に理解されている根本原因によって引き起こされる異常です。一般的には簡単に目に見えるものですが、修正することはできません。それは、患者に解熱剤を与えて熱を下げる対症療法にしか過ぎず、病気自体を治療することにはなりません。つまり、病気そのものを治療しなければ、再び発熱することになります。重要なのは、症状は根本原因が解決されれば消えるので、症状は実行可能なものではなく、根本原因の診断に単に役立つものでしかないということです。

影響

影響は、異常の結果として生じたことを示します。影響は必ずしも目に見えるものとは限りませんが、知っておくことは有益です。たとえば、重要な顧客が苦情を申し立てる前に、デバイスの障害や過度のパケット損失の影響を受けることを知りたいと思う場合があります。

影響が目に見えない場合でも、インテントの知識とインテントの実装に使用される実施メカニズムに基づいて算定することはできます。影響を理解することで、オペレーターは運用に最も大きな影響を与える根本原因や異常の優先度を決めることができます。

根本原因

すべての状態の中で最も重要なのは根本原因です。根本原因は必ずしも目に見えるものとは限らないため、診断が難しくなりますが、多くの症状や影響を引き起こす原因です。根本原因を解決することで、それに起因する症状や影響が解消されることから、根本原因は実行可能なものです。

IBAプローブ：何を知っているのかを理解する

有名な格言は、データセンターネットワークにも応用できます。「知らない事が問題なのではない。知らないのに知っていると思い込むことが問題なのだ」-インテントは、常態、つまり明確に知っていることを特徴づけ、形式化するものです。IBAによって、知っていることは、実際にあるべき姿になります。

「知らない事が問題なのではない。知らないのに知っていると思い込むことが問題なのだ」 - マーク・トウェイン

IBAプローブは、関心のある状態を検出する役割を果たし、リファレンスデザインの一部である動作規定によって駆動されます。データを取得し、いくつかの処理を適用してから、結果を期待値と比較します。基本的に構成可能なデータ処理パイプラインであり、ユーザーが関心のある状態（監視する状況など）を設定できるようにします。

データソースとクエリ

通常、プローブの初期段階では、データソースが関与し、未加工のテレメトリデータを取得します。データソースステージの構成には、収集するデータを正確に表現できるクエリが含まれています。このクエリは非常に強力な機能です。

たとえば、オペレーターがファブリックインターフェイスにのみ適用されるECMP（等価コストマルチパス）ルーティングの不均衡を分析することに関心があり、特定のオペレーティングシステムのバージョンでECMPハッシュアルゴリズムにバグが発生したという報告があったとします。クエリでは、すべてのトップオブラックスイッチでインターフェイスカウンターを収集する必要性を表すことができますが、それはファブリックインターフェイスと、スイッチOSの特定バージョンx.y.zを実行しているスイッチに対してのみ有効です。このような状況を監視するよう設定すれば、オペレーターは変更について心配する必要はありません。基準に一致するスイッチに新しいファブリックリンクが追加されると、そのリンクは自動的に分析に含まれます。新しいスイッチが追加された場合も、自動的に含まれます。また、他のユーザーが別のスイッチをバージョンx.y.zにアップグレードした場合でも、自動的に含まれます。このように、IBAプローブはメンテナンスが不要です。

ステージプロセッサー

多くの場合、オペレーターは未加工のテレメトリデータに関する瞬時値ではなく、アグリゲーションやトレンドに関心があります。IBAには、平均、最小/最大、標準偏差の計算結果などの情報を集約するステージプロセッサが含まれています。次に、オペレーターはこれらの集計を期待値と比較して、集計メトリックが指定の範囲内にあるかどうかを特定できます。範囲外の場合は、異常値にフラグが付けられます。

オペレーターは、この異常値が特定のしきい値を超える期間に持続するかどうかを確認し、一過性または一時的な状態と見なされる異常値のフラグ付けを回避するために、しきい値を超えた場合にのみ、異常値にフラグを立てることができます。オペレーターは、後続のステージにタイム・イン・ステイト・プロセッサーと呼ばれるものを構成するだけで達成できます。

数値データを超えて

プローブは、数値データだけで動作するわけではありません。たとえば、コントロールプレーンとデータプレーンの正確さを検証するのに使用できます。たとえば、オペレーターはクエリを使用してデータソースプロセッサーを構成し、インテントに応じて、期待するルーティングテーブルまたは転送テーブルを作成できます。

EVPN（イーサネットVPN）環境の場合、インテントには、存在する仮想ネットワーク、エンドポイントの場所、実施メカニズムに関する情報などが含まれます。この予測されるテーブルは、不一致が見つかった場合にフラグが立てられたテレメトリおよび異常値のテーブルと比較できます。

プローブは、転送テーブルとルーティングテーブルのサイズの急変を追跡し、トレンドが期待どおりでない場合にアラートを出すよう構成することもできます。「期待値」のしきい値は、仮想ネットワークの数、エンドポイントの数、VTEP（仮想トンネルエンドポイント）の数、問題のあるVTEPの数などの関数である可能性があるため、インテントから動的に計算することもできます。可能性は無限大です。

プローブは、単純なREST呼び出しを使用するか、GUIを使用して、宣言的に作成し、アクティブ化/非アクティブ化できます。プローブのアクティブ化は、テレメトリのトリガーとしても機能します。つまり、特定のテレメトリデータは、それに関心のあるプローブが存在する場合にのみ収集されるため、プローブは基本的にテレメトリ収集構成メカニズムとして機能します。データソースプロセッサーのクエリは、必要に応じて構成をきめ細かく正確なものにし、何をすべきか明確な考えなしに大量のデータが収集されたときに発生する「データの囲い込み」を排除します。これにより、データの保存や処理にかかるコストが限度を超えて増加します。

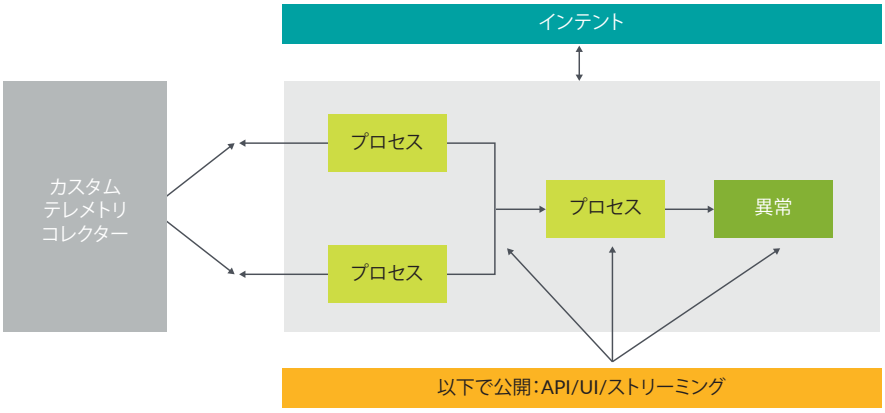

図 5：テレメトリの収集

プローブを構成することで、各ステージの出力結果をAPI、GUI、およびストリーミングエンドポイントを介して利用できるようになります。Apstraには組み込みのプローブセットが付属しており、GitHubにはオープンソースプローブのリポジトリがあります。ユーザーは、新しいプローブを最初から作成することもできます。

根本原因の特定

RCI（根本原因の特定）は、インフラストラクチャで特定された状態間の因果関係の分類と導出を自動化するメカニズムです。RCIの主なメリットは、根本原因による障害の結果にすぎない、実行不可能な大量の状態から、オペレーターのアクションを必要とする根本原因の状態を明らかにできることです。たとえば、オペレーターがインフラストラクチャを適切に計測し、「1つの画面ですべてを表示する」コンソールで異常な状態のログを監視しているとします（図6を参照）。赤い点は、多くの異なる要素で発生するインフラストラクチャ全体に散在する、さまざまなタイプの状態を示しています。

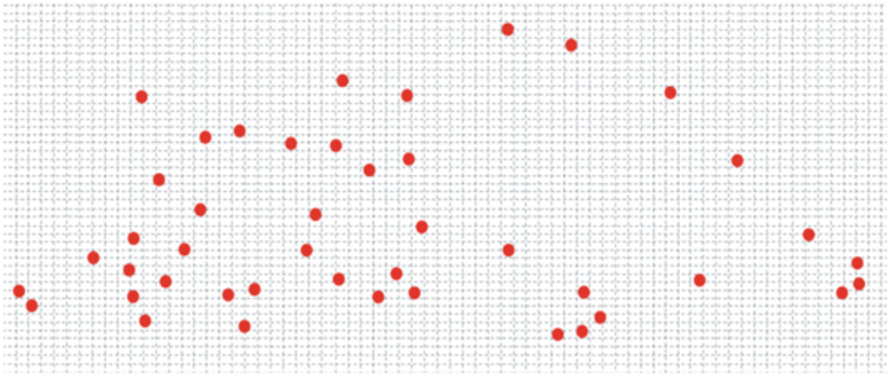

図 6：1つの画面ですべてを表示するコンソール

この画像から見える問題は、それが実行可能なものではなく、関係性の低い不要な情報でしかないということです。分類されていない場合、それぞれの状態はオペレーターの関心を引くものであっても、果たしてどの状態に対処すべきなのでしょうか。そこでRCIの出番です。RCIは、サービスの内容、サービスの実装方法と実施メカニズムへのマッピング方法、マネージドインフラストラクチャにおける要素の関連性などのコンテキストに関する知識を備えており、根本原因および関連する症状や影響を特定して分類します。図7に示すように、RCIを使用した分析の結果は以下のとおりです。

- 根本的な原因は、「spine_1」と呼ばれるスイッチのメモリリークでした（根本原因：大きい青緑色の点）。
- このメモリリークにより、メモリ不足のプロセスキラーが動作し、いくつかのプロセスが強制終了しました。そのうちの1つはルーティングプロセスでした（症状：濃い緑色の点）。
- これにより、BGPセッションがこのデバイスとピアリングデバイスで失敗し、期待されるルーティングテーブルエントリが欠落していました（症状：濃い緑色の点）。
- その結果、顧客Xと顧客Yに属するエンドポイントで接続の問題が発生しました（影響：薄い緑色の点）。

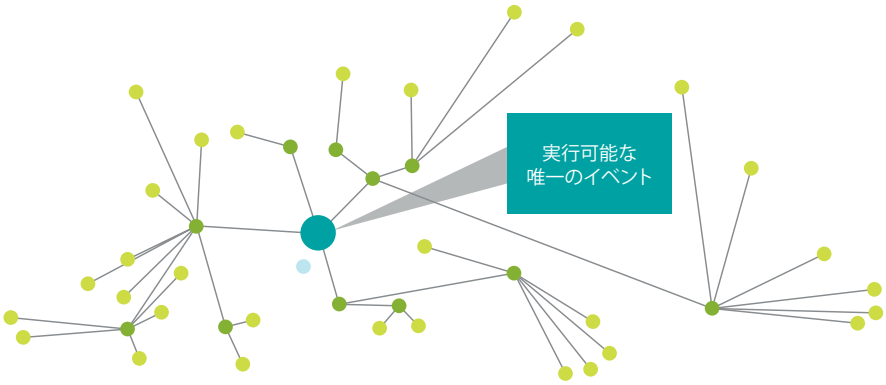

図 7：根本原因の特定コンソール

RCIは「1つの画面ですべてを表示する」ための複雑な思考プロセスを自動化します。1つの画面ですべてを表示する場合、膨大な情報によってオペレーターの負荷が高くなり、視覚的な確認や関連付けがオペレーター任せになります。RCIでは、その代わりに、シンプルな1つの画面で情報を表示するので、オペレーターは必要なアクションを簡単に識別できます。

コンテキストモデル

すべてのデータが揃っているからといって、すべての答えがそこにあるわけではありません。また、知識の抽出を自動化せずに大量のデータを収集することは、OpExの爆発的な増加につながります。専門家がデータの意味を理解するために貴重な時間を費やすことになるからです。データ自体は、そこから知識が引き出されるか、適切な質問への回答が得られることがない限り、その価値は限定的です。

一般に、未加工のテレメトリデータは、水平スケーリングやシャーディングに適したKey-Valueストアに保存されます。ただし、単純なキー検索以外のクエリには対応していません。知識を抽出するには、クエリのサポートを構築し、クエリ構築のコンテキストを用意する必要があります。

一方、SQLデータストアは、複雑なクエリにも対応します。問題は、SQLテーブルが、設計時に作成され、予想されるクエリを中心に設計されていることです。実行時に他の質問をしたい場合は、データベースを非正規化することもできますが、その場合、クエリ結果に多数の結合が発生し、パフォーマンスが低下して停止してしまう可能性があります。運用知識の抽出のカギとなるのは、実行時に発生するクエリです。グラフベースのコンテキストデータ実装が優れているのは、実行時に多数の「結合」を伴う複雑なクエリに対応できる点にあります。グラフモデルは、基本的な構成要素、ノード、および関係のインスタンスを追加するだけで、簡単に拡張できます。

クエリの有効活用について説明することにもつながるので、このコンテキストに含まれるものについて詳しく見てみましょう。

- **設計アーティファクト**：インテントグラフのコンテキストには、設計アーティファクトが含まれています。たとえば、データセンターネットワークを設計している場合、設計には目的のオーバーサブスクリプション係数、サーバー数、HA（高可用性）構成（シングル接続、デュアル接続）が含まれていることがあります。
- **リソース割り当て**：リソースの割り当てに関連する決定が含まれています。インフラストラクチャを「家畜」または「ペット」のどちらで管理するのかを決定します。
- **分離ポリシー**：分離ポリシーも指定します。特定リソース（IP、ASN、VLAN）の再利用を許可するかどうかを指定します。
- **セグメンテーションポリシー**：どのエンドポイントとワークロードが、どのような条件下で相互に通信できるのかを指定するセグメンテーションポリシーが含まれています。
- **実施情報**：セグメンテーションおよび到達可能性ポリシーが実際に実装された場所（物理アプライアンスまたは仮想アプライアンス）とその実装方法（使用するメカニズム）に関する実施情報が含まれています。
- **外部到達可能性ポリシー**：データセンターの場合、外部到達可能性ポリシーが含まれていることがあり、外部から見ることができる内部のエンドポイント（またはその逆）を指定します。また、どのテナントがどのセグメントを所有しているかに加え、各テナントに約束されているサービスレベルやサービスレベルの目標も含まれています。
- **ビジネスの視点**：サービスレベル合意や合意を果たせない場合のコストなどのビジネスの視点が含まれます。

8Dビュー

このようなアーティファクトのすべてが、1つのグラフで表示されたとしても、その信頼できる唯一の情報源は、論理的には単一の多次元空間であり、少なくとも以下にあげる8つの次元があります（ただし、次元の数はこの例に限定されるもので、これより多い場合もあります）。

- 設計
- リソース
- 分離
- セグメンテーション
- 実施
- 外部到達可能性
- サービスレベル
- ビジネスの視点

その結果、問題が発生した場合、この8Dビュー全体で1つのクエリを実行することで、以下のような複雑な質問に答えることができます。「要素に対する直近の障害状態を考えた場合、現在の設計目標はまだ有効か？」「目的の分離ポリシーによって、適切なリソースが正しい実施ポイントで使用されているか？」「セグメンテーションポリシーに何らかの影響を与えるか？」「サービスレベルの目標を守っているか？」「この失敗の代償はいくらか？」

このような8Dビューは、「あれば便利」なものではなく、信頼性の高い運用には必須です。Apstraのインテントベースシステムには、8Dビューが組み込まれています。Apstraがなければ、専門家しか持ち合わせていない、仲間うちだけで共有している知識を複数の信頼できる情報源から抽出し、このような8Dビューを非常に複雑なレイヤーで再構築しなければなりません。個々の信頼できる情報源が統合を念頭に構築されておらず、セマンティクスや動作が異なる環境で、このようなレイヤーを構築することは、最良の場合でもやみくもに負担が生じるだけで、最悪の場合は手に負えない悪夢のような状況に陥ります。

リアルタイムの監視と通知

Apstraは、インテントに関する知識の抽出と、その結果としてのインフラストラクチャの運用状態がすべてです。この知識は、タイムリーで、現在の状況を反映していれば、力になります。Apstraの中心にあるのは、ライブクエリを構成する機能です。この機能により、クライアントは関心のある状態にサブスクライブし、条件が満たされるとリアルタイムで通知を受け取ることができます。このコンテキストでの条件は、インテントと運用状態の両方に関連しています。これは、変更に確実に対処するための絶対的な前提条件です。

「アーキテクチャの概要」セクションでも説明しているように、ユーザーレベルで表示されるのと同じpub/subパラダイムは、これと同等の低レベルのpub/subメカニズムによってサポートされます。このメカニズムは、アプリケーションロジックを実装するApstraシステムプロセスのデータ中心の論理通信チャネルとして機能する、分散データストアによって実装されます。

セルフ運用

最後に、問題の修復、フォレンジック分析用のログ記録、次レベルのドリルダウン実行による根本原因分析に役立つテレメトリの収集を行うには、一部のイベントへの対応を自動化する必要があります。

自動対応による効果：

- 修復パラメーターが最新の信頼できる唯一の情報源から自動的に導出され、設定ミスや古いデータの影響を受けないため、リスクが軽減される
- 修復がタイムリーに行われるため、カスタマーエクスペリエンスが向上する
- 手動のトラブルシューティングや修復プレイブックの手動実行が不要になるため、運用コストが削減される

自己運用型ネットワークへのハードルは、必要な機能がすでに存在するため、技術的な制限よりも組織的または個人的な抵抗に起因します。

最終的には、自動対応により、ネットワークの自己運用と自己修復が可能になります。自己運用型ネットワークへのハードルは、必要な機能がすでに存在するため、技術的な制限よりも組織的または個人的な抵抗に起因します。

延伸性（Extensibility）：将来性のあるデータセンターネットワーク

延伸性には、以下にあげる3つの側面があります。

1. **リファレンスデザインの延伸性** - インフラストラクチャの要素がどのように連携してインテントを達成するのかを決定します。グラフモデルの変更、およびインテントがインフラストラクチャの実施メカニズムにどのようにマッピングされるかに関連する変更を可能にするプラグインが含まれています。
2. **分析の延伸性** - 監視する新しい状態や状況の定義に関連しており、ユーザーによる新しい検証とその分類および関連付けの方法の定義を可能にします。
3. **柔軟性のあるサービスAPI** - トップレベルのサービス定義を延伸する機能を提供します。

リファレンスデザイン

前述のように、リファレンスデザインとは、インテントを実施メカニズムにマッピングする方法と、インテントが達成されると見なされるために満たさなければならない期待値について定義する動作規定です。この規定のどの側面も変更または拡張できます。新しいノードと関係のタイプを定義でき、構成テンプレートとリソース割り当てメカニズムを変更できます。

分析の延伸性

新しい分析機能は、次の2つのメカニズムを介して導入できます。

1. 新しいIBAプローブは、関心のある新しい状態を検出するように定義できます。新しいテレメトリコレクター、および状態固有のデータ処理パイプラインが含まれる場合があります。プローブは公開して、後で公開リポジトリからインポートすることもできます。
2. 新しいRCIモデルは定義して、システムにロードできます。RCIモデルは、基本的に新しいタイプの根本原因を、それによって生じる一連の症状にマッピングすることです。このモデルが定義されてロードされると、Apstraシステムは、確認された症状に基づいて根本原因の特定を自動化します。

サービスAPI

Apstraは、サービスレベルのAPIをグループベースのポリシーの形式で公開し、実装に依存しない方法で幅広いサービスとポリシーを柔軟にサポートします。

グループベースのポリシーでは、インテントはグラフとして表され、一般的な動作のインテントを表す目的でグループ（メンバーの関係）に配置されるエンドポイントを表します。ポリシーはインスタンス化され、グループまたは個々のエンドポイントに関連付けられて、その動作を定義します。

ポリシーは、方向性のある方法（from/to関係）または方向性のない方法（applies-to）でグループに関連付けることができます。ポリシーはルールの集まりです。ルール間の順序が重要な場合、ルールは次のルールとの関係がある場合があります。グループは、他のグループ（階層）で構成できます。また、他のグループと関係を持って、制約を表すこともできます（「これらのエンドポイント/グループは、これらのポートグループの背後にあります」など）。エンドポイント、グループ、ポリシー、およびルールは、接続のインテントを表現するための構成要素と考えることができます。

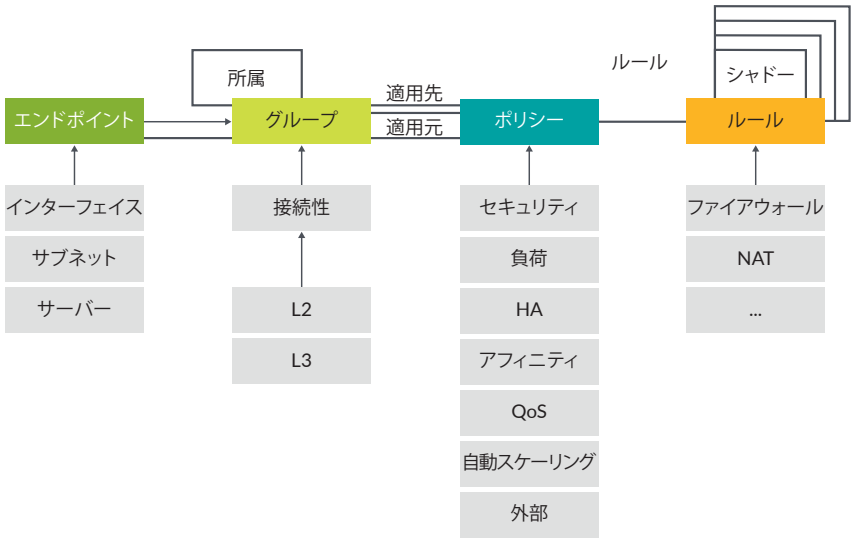

図 8：グループベースのポリシー

エンドポイントは、ポリシーの対象となるインフラストラクチャの要素であるため、その構成は非常に一般的です。インターフェイス（物理または仮想）、サーバー、仮想マシン（VM）、コンテナ、またはアプリケーションエンドポイントを表すことができます。図8の青い矢印は、「論理的な」継承を示しています。エンドポイントには、エンドポイントをより正確に定義するパラメーターが含まれています（インターフェイス名、ポート番号、シリアル番号、ホスト名、VM UUID、コンテナIP、アプリケーションプロトコル、UDP/TCPポート番号など）。また、Apstraによって管理されていない要素（外部エンドポイント）を表すこともできるため、Apstraが対話する必要のある外部システム（外部ルーターIP/ASNなど）からの制約を表すために使用されます。

エンドポイントはグループに配置されます。エンドポイントは複数グループのメンバーに指定でき、それぞれが意図された動作のさまざまな側面を表現します。たとえば、1つのグループは以下を示すことがあります。

- 「東部」地域のサーバー一式
- ロードバランシングに使用されるサーバーのグループ
- 冗長性グループを形成するサーバーのグループ

グループは、セマンティクスが過負荷状態の場合もあります。たとえば、L2ドメイングループのメンバーであるエンドポイントは、L2ブロードキャストドメイン（サブネット）のメンバーです。同様に、L3ドメインのメンバーであるエンドポイントには、L3到達可能性があります。

グループは、他のグループでも構成できます。グループのエンドポイントメンバーは明示的にインスタンス化される場合がありますが、グループ定義の一部として動的メンバーシップ仕様があり、エンドポイントがグループに存在する仕様から暗示される場合もあります。

たとえば、L3ドメインには、CIDR（クラスレスインタードメインルーティング）またはサブネットがプロパティとして含まれている場合があります。そのため、その範囲/サブネット内のIPを持つすべてのエンドポイントは、暗黙的にグループのメンバーになります。その場合、L3-to-the-serverリファレンスデザインは、外部/内部サブネットの数を指定します。エンドポイント（コンテナ）がApstraで明示的に指定・管理されていない場合でも、コンテナが指定されたL3ドメインに属していることを意味します。きめ細かいセグメンテーションをモデル化するために、コンテナエンドポイントとそれらがホストされているサーバーの明示的な仕様についても紹介します。

ポリシーは一般的な動作を定義し、その動作を正確に定義するためのパラメーターを含めることができます。ポリシーは、方向性のない方法でグループに適用できますが、セキュリティポリシーの場合などは、必要に応じて方向を示すこともできます。特定ポリシーの例には、セキュリティ、ロードバランシング、HA、アフィニティ（エンドポイントを同じ場所に配置または分散するなど）、およびQoS（サービス品質）が含まれます。

ポリシーは、必要に応じてルールを含むことができます。一般に、ルールは「条件の後にアクションが続く」パターンに従います。たとえば、セキュリティポリシーでは、「一致」は条件ステートメントで、アクションは「許可/拒否/ログ」です。

拡張性（Scalability）：痛みを伴わない成長

Apstraは、ネットワーク透過的な分散状態のアクセスと管理をサポートし、並列実行は個別のプロセスによってサポートされます。リアルタイムの実行は、実行のリアルタイムスケジューリングとともに、イベント駆動型の非同期実行モデルによってサポートされます。効率と予測可能性は、マシンレベルの効率性を達成するための中間言語としてのC++によるコンパイルによってサポートされます。スケーリングには、3つの側面があります。

状態のスケーリング

1つ目の側面は、状態のスケーリングです。Apstraデータストアは、サーバーのHAペアを追加することで水平方向に拡張されます。インテントデータストアとテレメトリデータストアは分離されており、必要に応じて個別に拡張できます。階層データストアのプロビジョニングでは、最上位のデータストアが、データストア全体の状態を相互に関連付けるのに必要なものとして、次のレベルのデータストアに関する状態の必須（設計者が指定した）サブセットのみにサブスクライブ（同期）します。

処理のスケーリング

2つ目の側面は、処理のスケーリングです。Apstraは、処理負荷の共有が必要な場合、処理エージェントの複数コピーを（エージェントタイプごとに）起動できます。エージェントをホストするサーバーを追加することで、エージェントを追加できます。エージェントのライフサイクルも管理します。

システムの状態ベースのpub/subアーキテクチャにより、エージェントは明確に定義された状態のサブセットに対応（アプリケーションロジックを提供）できます。インテント全体のカバレッジは、状態のさまざまなサブセットの処理を代行する個別のエージェントを通じて行われます。そのため、インテントや運用状態に変更が生じた場合、エージェントの対応は「増分変化」に対するものであり、状態全体のサイズとは無関係であることを意味します。

Apstraは、拡張とそれに関連した複雑さに対処するために従来のアプローチを採用しています。そのアプローチとは、分解です。「誰もがすべてを知っている」アプローチは、拡張性がありません。集権的な意思決定の必要性を回避するためには、目的の状態に関する知識を分散し、各エージェントにその状態に到達する方法を決定させる必要があります。Apstraはライブグラフクエリに対応し、UIなどのクライアントに必要なものを正確に要求して取得できます。そのため、バックエンドから取得されるデータの量をきめ細かく制御できます。

ネットワークトラフィックのスケーリング

3つ目の側面は、ネットワークトラフィックのスケーリングです。エージェントとデータストア間の通信は、最適化されたバイナリチャネルを使用しているため、テキストベースのプロトコルと比較してトラフィック量が大幅に削減されます。

耐障害性は、Apstraアプリケーションを複数のプロセスとして実行し、ネットワークで接続された個別のハードウェアデバイスで実行し、状態の再現と迅速な復旧のサポートで状態を処理から分離することで達成されます。

Apstraアーキテクチャの概要

このホワイトペーパーの最初の部分では、データセンターネットワークアーキテクチャの課題と、問題解決のためにApstraがどのように機能するかについて説明しました。ここでは、Apstraのアーキテクチャについて詳しく説明します。

Apstraは、分散型の状態管理インフラストラクチャに基づいて構築されており、水平方向の拡張性と耐障害性のあるインメモリデータストアを備えた、データ中心の通信ファブリックであるとも言えます。特定のリファレンスデザインアプリケーションのすべての機能は、ステートレスエージェントのセットを介して実装されます。エージェントは、論理的なパブリッシュ/サブスクライブベースの通信チャネルを介して相互に通信し、アプリケーションのロジックを実装します。

Apstraリファレンスデザインアプリケーションはすべて、上記のステートレスエージェントのコレクションにすぎません。大まかに言えば、エージェントには以下に挙げる3つのクラスがあります。

1. **インタラクション（Web）エージェント**は、ユーザーとのやり取り（ユーザー入力を取得し、データストアから関連するコンテキストをユーザーにフィードする）を行います。
2. **アプリケーションエージェント**は、入力エンティティにサブスクライブし、出力エンティティを生成することで、アプリケーションドメイン固有のデータ変換を実行します。
3. **デバイスエージェント**は、スイッチ、サーバー、ファイアウォール、ロードバランサー、さらにはコントローラーなど、管理対象の物理システムまたは仮想システム上に常駐します（またはプロキシとして機能します）。このエージェントは、構成の記述およびネイティブ（デバイス固有）のインターフェイスを介したテレメトリの収集に使用されます。

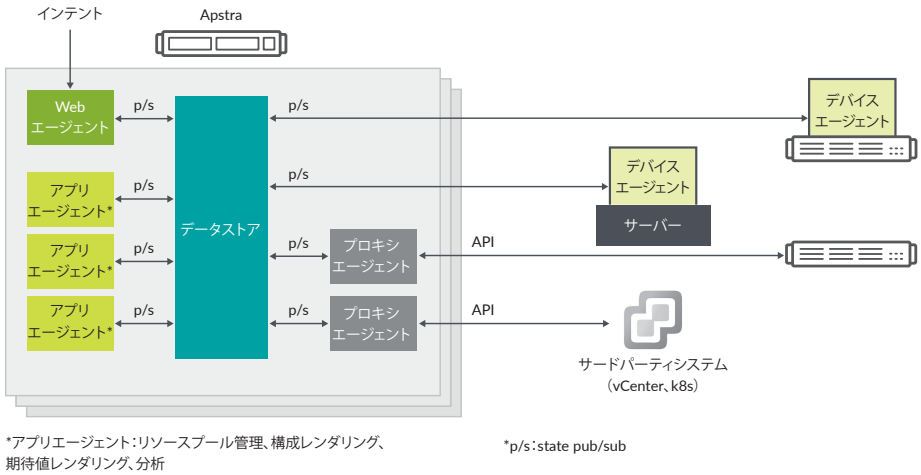

図 9：Apstraのエージェント

このようなやり取りは、Apstraのデータセンターネットワークを構築するリファレンスデザインアプリケーションを部分的に説明した例で説明できます。

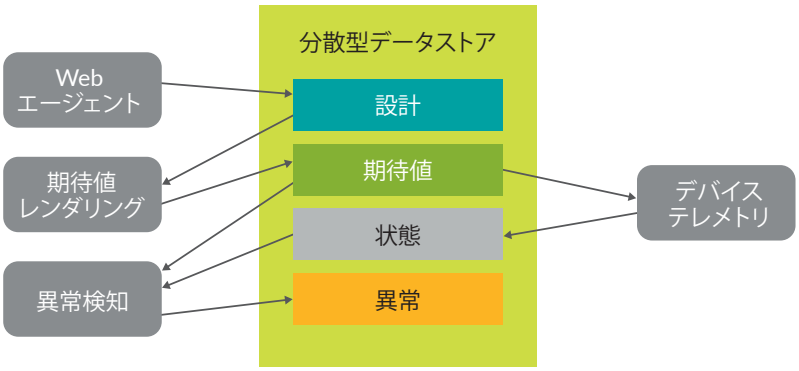

図 10：Apstraインテントベースシステムのデータセンターネットワーク構築リファレンスデザインアプリケーション

Webエージェントは、ユーザー入力を取得します。この場合は、スパイン、リーフ、相互間のリンク数、ファブリックIPとASN（Abstract Syntax Notation）番号に使用するリソースプールを含む、L3 Closファブリックの設計です。このインテントをグラフノードと関係、およびそれぞれに対応するプロパティのセットとしてデータストアに公開します。

ビルドエージェントは、このインテントにサブスクライブし、以下のことを行います。

- 正当性と完全性の検証を実行する
- リソースプールからリソースを割り当てる

検証に合格すると、ビルドエージェントはそのインテントをリソース割り当てとともにデータストアに公開します。

1. 構成レンダリングエージェントは、ビルドエージェントの出力にサブスクライブします。
2. 各ノードについて、リソースを含む関連データを取得し、それを構成テンプレートにマージします。
3. 待ち受けているエージェントは、ビルドエージェントの出力にもサブスクライブし、結果を検証するのに満たす必要のある期待値を生成します。
4. デバイステレメトリエージェントは、そのエージェントの出力にサブスクライブし、関連するテレメトリの収集を開始します。
5. IBAプローブは、未加工のテレメトリを処理し、それを期待値と比較して異常を公開します。
6. RCIエージェントは異常を分析し、それらを症状、影響、および特定された根本原因に分類します。

エージェントは、エンティティを公開し、エンティティの変更にサブスクライブすることで、属性ベースのインターフェイス（データ中心の方法）を介して通信します。データ中心とは、データ定義がフレームワークの一部であり、たとえばメッセージベースのシステムとは対照的に、エンティティを定義することで実装されることも意味します。

データ中心のパブリッシュ/サブスクライブシステムは、メッセージベースのシステムで起こる問題に悩まされることはありません。メッセージベースのシステムでは、遅かれ早かれ、メッセージの数がメッセージを保存または利用するシステムの容量を超えてしまいます。一貫性のある状態にするためには、メッセージの履歴を再生する必要があるため、この問題に対処するのは困難です。

データ中心のシステムは、基本的に最後の状態にのみ依存するため、状態変化の急増に対して耐障害性があります。この状態は、重要なコンテキストをキャプチャし、それにつながる可能性のあるイベント（および無関係なイベント）のシーケンスをすべて抽象化します。ステートマシンパラダイムを使用して記述されたコードは、読み取り、保守、デバッグがより簡単にできます。

解決が難しい問題（柔軟性や耐障害性など）は、すべてのエージェントに代わって一度だけ解決されます。一般的なアーキテクチャは、障害が発生した場合に再起動でき、SysDB（システムデータベース）からサブスクライブしている状態を再読み取りするだけで、中断したところから再開できる、多数のステートレスエージェントで構成されます。

Apstraアーキテクチャのメリット

このホワイトペーパーですでに説明したように、Apstraアーキテクチャには、データセンターネットワークに関連した難問を解決できるという大きなメリットがあります。アーキテクチャの特長は、以下のとおりです。

- **オペレーターが変更に確実に対処するのに役立ちます。**リアルタイムのクエリ可能なインテントと運用コンテキストを通じて可能になります。
- **ネットワークサービスのライフサイクルにおけるすべての側面を簡素化します。**ライフサイクルには、Day0からDay2までの運用が含まれます。また、シンプルさにより、オペレーターによるエラーの発生が減少します。
- **ステートフルオーケストレーションで運用リスクが軽減されます。**このオーケストレーションでは、前提条件の検証、事後条件の検証、自動構成レンダリング、および期待の自動検証を活用します。

根本原因の特定とインテントベース分析の特別なメリット

Apstraの運用分析コンポーネント（根本原因の特定とインテントベースの分析）により、以下のことが可能になります。

- **修復までの平均時間と特定分野の専門家（SME）のワークロードを削減します。**運用分析を簡素化することで、最もスキルの高いリソースが解放され、目先の問題を解決する代わりに、改善やイノベーションに時間を投入できるようになります。
- **より多くの知識を抽出します。**収集するデータや保存するデータを少なくすることで、抽出する知識を増やします。Apstraは、リファレンスデザインの動作規定を利用して、期待するものを認識し、その情報のみを収集します。このような即時処理により、ストレージのニーズや関心のないデータの後処理が桁違いに削減されます。そのため、インフラストラクチャの実行効率が高まり、コストを抑えながら競争力を維持できます。
- **問題をすばやく簡単に確認できます。**Apstraは、「シンプルな1つの画面」で重要度の高い実行可能なイベントを識別し、特定された根本原因の単なるアーティファクトでしかない症状のノイズを排除します。
- **ワークフローを自動化します。**Apstraシステムでは、コンテキストが豊富なトラブルシューティングワークフローを自動化できます（それ以外の場合は、面倒で効果がなく、時間とコストがかかります）。
- **ゼロタッチメンテナンスを実現します。**インテントと常に同期しているため、変更が発生しても、ゼロタッチでゼロコストのメンテナンスを可能にします。そのため、耐障害性に優れ、変更に自動対応し、データ処理パイプラインの維持に関連する莫大なコストを節約できます。
- **コストのかかるDIY（自身で手がける）開発を排除します。**データ処理パイプラインを統合するためのDIY開発は、コストがかかり、脆弱なため、コアビジネスに投入する時間とリソースを奪い、特定分野の専門家に多くの負担を強いることになります。
- **高精度の結果を提供します。**機械学習/人工知能のアプローチと比較して精度の高い結果を提供します。
- **ベンダーに依存しないアプローチを採用します。**最適なベンダーや機能の中から自由に選択できます。
- **共通のAPIを使用します。**パブリック/プライベート/ハイブリッドのクラウドインフラストラクチャ全体で共通のAPIを使用します。

導入Day 1から拡張できる：導入事例

Apstraは、導入Day 1から拡張することを念頭に置いて構築されています。Apstraを使用したお客様が、広範な検証を実行することで得られた素晴らしい結果は、以下のとおりです。

- **物理インフラストラクチャ：**6,000回のインターフェイスステータス検証、1,000回のケーブル検証、36,000回のエラーカウンター、10,000回の電力/温度/電圧メトリック検証
- **L2/L2データプレーン：**12,000回のキュードロップカウンター、数百回のMLAG健全性チェック、3,000回のSTP（スパニングツリープロトコル）検証、ステータス変更の通知
- **コントロールプレーン：**1,500回のBGPセッションヘルスチェック、予想される最大500のネクストホップ/デフォルトルート
- **容量計画：**ルーティングテーブルを使用するために設定されたしきい値による最大500のトレンド分析、ARP（アドレス解決プロトコル）テーブル、VRF（仮想ルーティングおよび転送）ごとのマルチキャストテーブル、6,000回のリンク使用状況の検証

- **コンプライアンス**：予想されるOSバージョンが100個程度のスイッチすべてで実行されていることを確認
- **マルチキャスト**：2,500回の予想されるPIM（物理インターフェイスモジュール）ネイバー検証、300回のランデブーポイントチェック、25回のランデブーポイントでのソース数/グループ数/ソースとグループのペア数の異常なパターンの検出に関する検証

基本的には、82,000個のエントリーを1つの画面に表示する代わりに、以下に示す顧客の仕様に従って情報をダッシュボードに分類し、異常のみをシンプルな1つの画面に表示しました。

- 100%正確な情報（統計的な推定値ではない）
- トポロジおよびインテントと常に同期しているので、継続的なメンテナンスは不要
- 関連性の高い実行可能なコンテキスト（逸脱の内容、逸脱の性質、望ましい状態）を提供
- ストレージと処理のニーズを軽減（9 GBのRAM、96 GBのディスクのみ必要）
- 複雑で脆弱なデータ処理パイプラインを記述することで、内部ロックインとレガシーの危険性を排除

まとめ

ITインフラストラクチャに確実な変更を加える能力がないことは、成長とイノベーションにとって大きな障害になります。Apstraは、そのような不安を取り除き、誰も関わりたくない「レガシーインフラストラクチャ」を永久に根絶することを可能にし、確実に変更を加えることができるようにします。そのため、イノベーションが確実に促進され、競争力を維持することができます。

ジュニパーネットワークスについて

ジュニパーネットワークスは、世界をつなぐ製品、ソリューション、サービスを通じて、ネットワークを簡素化します。エンジニアリングのイノベーションにより、クラウド時代のネットワークの制約や複雑さを解消し、お客様およびパートナーの皆様が日々直面している困難な課題を解決します。ジュニパーネットワークスは、世界に変革をもたらす知識の共有や人類の進歩のリソースとなるのはネットワークであると考えています。私たちは、ビジネスニーズにあわせた、拡張性の高い、自動化されたセキュアなネットワークを提供するための革新的な方法の創造に取り組んでいます。

米国本社
Juniper Networks, Inc.
1133 Innovation Way
Sunnyvale, CA 94089 USA
電話番号：888.JUNIPER（888.586.4737）
または +1.408.745.2000
www.juniper.net

アジアパシフィック、
ヨーロッパ、中東、アフリカ
Juniper Networks International B.V.
Boeing Avenue 240
1119 PZ Schiphol-Rijk
Amsterdam, The Netherlands
電話番号：+31.0.207.125.700

日本
東京本社
ジュニパーネットワークス株式会社
〒163-1445 東京都新宿区西新宿3-20-2
東京オペラシティタワー45階
電話番号：03-5333-7400
FAX：03-5333-7401
西日本事務所
〒530-0001 大阪府大阪市北区梅田2-2-2
ヒルトンプラザウエストオフィスタワー18階
www.juniper.net/jp

JUNIPER NETWORKS | Engineering Simplicity