Juniper Networks | Engineering Simplicity

백서

주니퍼 APSTRA 아키텍처

네트워크 수명 주기 전반을 자동화하여 복잡한 데이터센터 문제 해결

목차

개요

데이터센터 네트워크의 설계, 구축, 운영, 관리 및 문제 해결은 어렵고 상당한 비용이 들며 많은 자원을 소모할 수 있습니다. 2019년 Gartner 보고서에 따르면 IT 인프라 단순화는 오늘날 기업이 최우선 순위에 두고 있는 전략적 과제입니다. Juniper® Apstra는 데이터센터 네트워크의 전체 수명 주기 관리를 자동화하도록 전문 설계된 솔루션입니다.

이 백서에서는 오늘날 데이터센터 네트워킹 팀이 직면한 가장 어려운 문제 몇 가지를 살펴보고 Apstra의 아키텍처를 통해 그러한 문제를 해결하는 방법을 설명합니다.

서론

오늘날 데이터센터 네트워킹 아키텍처 및 운영 팀은 다음과 같은 4가지 주요 과제에 직면해 있습니다.

- **설계**. 각기 다른 기능을 가진 서로 다른 벤더의 인프라 구성 요소를 사용하여 원활하게 작동하는 유기적인 시스템을 구성하는 것은 매우 어려운 일입니다. Juniper® Apstra는 표시된 인텐트를 기반으로 시스템 구축에 필요한 모든 정보가 포함된 유기적인 전체 청사진을 만듭니다. 운영자는 물리적 인프라에 푸시되는 이 청사진을 통해 안정적이고 간편하게 소비할 수 있는(easy-to-consume) 서비스를 제공할 수 있습니다.
- **안정적인 변경**. 데이터센터에서는 끊임없이 변경이 이루어집니다. 운영자가 새로운 서비스를 추가하거나 용량을 확장하는 경우 또는 인프라에서 장애 상태가 발생하는 경우 변경이 이루어질 수 있습니다. 어떤 경우이든 운영자는 변화에 대처하는 더 나은 방법을 찾아야 합니다. 최근 몇 년간의 여러 조사에 따르면[1], 네트워크 서비스 중단의 65~70%는 계획된 변경을 실행하는 과정에서 발생한 구성 상의 휴먼 에러로 인해 발생합니다.
- **연장성**. 혁신을 도모하고 피할 수 없는 급격한 기술 변화를 수용하려면 데이터센터에서 새로운 기능을 쉽고 원활하게 추가할 수 있어야 합니다. 벤더는 기술 변화에 따라 꾸준한 혁신을 통해 운영자가 경쟁력을 유지하는 데 활용할 수 있는 새로운 기능을 개발합니다. 이러한 변화로 인해 설계가 변경될 수 있으므로 구축하려는 서비스에 대한 새로운 행동 양식 변경 또는 레퍼런스 설계가 필요합니다. 다시 말해, 추가 혁신을 수용하거나 새로운 요구 사항을 충족하기 위해 설계에 대한 연장성을 갖춰야 합니다.
- **확장성**. 다양한 규모에서 이 모든 과제를 해결하려면 데이터센터 네트워크가 확장 가능하고 혁신에 따른 증가하는 복잡성을 수용할 수 있어야 합니다.

Apstra 아키텍처의 주요 목표는 이러한 문제를 직접 해결하는 것입니다.

주요 도전과제 : 구성(설계)

오늘날 컴퓨팅/네트워크/스토리지 인프라를 운영하는 운영자가 직면한 주요 과제는 구성입니다. 구성은 단순히 작동하는 구성 요소를 합쳐 놓은 것이 아니라 그 이상의 유기적인 완전체를 형성하고 시간 경과에 따라 불가피한 변경 사항이 있을 때 사용자에게 네트워크 서비스를 계속 제공하면서 일관성을 유지할 수 있어야 합니다. Apstra는 기본적으로 이러한 도전 과제를 해결하도록 마련되었습니다.

현대화된 데이터센터는 스케일 아웃 컴퓨팅으로 구현됩니다. 이러한 컴퓨터의 운영 체제에서는 오늘날 단일 시스템의 호스트 운영 체제와 비슷하게 리소스 관리와 프로세스 격리 기능이 제공되어야 합니다.

컴퓨팅 가상화로 이미 단일 서버 컴퓨팅에 이 같은 기능이 제공되고 있지만, 데이터센터 스케일 아웃 컴퓨팅의 경우 데이터센터 운영자가 먼저 설계 작업을 수행해야만 리소스 파티셔닝 등을 동작시킬 수 있다는 어려움이 있습니다.

[1] 대표적인 예는 다음과 같습니다. 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.

구성 단계가 어려운 이유는 무엇일까요?

구성이 어려운 데는 크게 두 가지 이유가 있습니다. 첫째, 일관성 있는 완전체를 구성하기 위해 사용하는 요소들이 다양한 벤더의 제품이거나 서로 다른 기능을 갖추었거나 서로 다른 API를 통해서 가능하거나 하는 이유 때문입니다.

둘째, 연결 가능성, 보안, 경험 품질, 가용성 같은 여러 기능적 측면을 갖는 다양한 서비스를 제공하도록 인프라를 구성해야 하기 때문입니다.

다양한 서비스를 실행 메커니즘에 매핑하는 방법을 확실히 이해하지 못한 채 여러 서비스 인스턴스를 생성하면 구성 요소 간의 상호 작용으로 인한 부하로 서비스가 중단될 수 있습니다. 시간 경과에 따라 인프라 기능이 변경될 때 기존 시스템을 중단하지 않고 새로운 혁신을 활용하고 도입할 수 있어야 합니다.

Apstra는 레퍼런스 설계로 이러한 도전 과제를 해결할 수 있습니다.

Apstra 아키텍처가 구성과 관계된 도전 과제를 해결하는 데 사용하는 핵심 개념은 레퍼런스 설계입니다. 레퍼런스 설계는 사용자의 비즈니스 인텐트가 실행 메커니즘에 매핑되는 방식과 인텐트가 이행된 것으로 간주되기 위해 충족해야 하는 기대치를 정의하는 행동 양식 변경입니다.

레퍼런스 설계는 다음과 같은 사항을 규정합니다.

- 물리적 및 논리적 구성 요소의 역할과 책임
- 서비스가 실행 메커니즘에 매핑되는 방식
- 충족해야 하는 기대치(즉, 관찰할 상황)

레퍼런스 설계에는 물리적 요소와 논리적 요소의 역할과 책임이 적절히 정의되어 있으며, 이는 모델링 범위와 최소 수준의 완결된 모델 사양을 규정합니다. 또한 레퍼런스 설계는 인텐트가 실행 메커니즘에 매핑되는 방식을 규정합니다. 이 매핑을 이해하면 문제 해결을 자동화할 수 있으며 네트워크에서 일어나는 일을 되돌리지 않고도 시스템이 어떻게 구성되어 있는지에 대한 지식에 기반하여 강력한 분석을 수행할 수 있습니다.

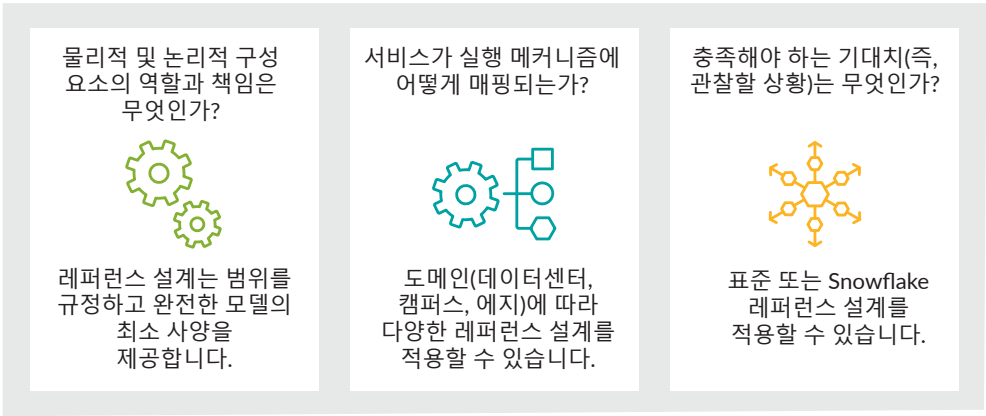

그림 1: 행동 양식 변경을 규범짓는 레퍼런스 설계

레퍼런스 설계에 따라 동일한 인텐트에 더 새롭고 혁신적인 메커니즘을 비롯한 다양한 메커니즘이 적용될 수 있습니다. 이 매핑을 이해하면 근본 원인 식별을 통해 서비스에 영향을 미치는 문제의 원인을 운영자가 직접 파악할 수 있습니다.

아울러 레퍼런스 설계는 검증해야 할 기대치를 지정합니다. 예를 들어, 레퍼런스 설계가 각 리프에 스파인별 BGP 세션을 설정하도록 규정할 수 있습니다. 이렇게 하면 누락되거나 예기치 않은 BGP 세션을 쉽게 식별할 수 있습니다.

레퍼런스 설계를 정의하는 것은 네트워킹 전문가가 수행해야 하는 고부가가치 활동입니다. Apstra를 사용하면 전문 지식이 전문가의 머릿속에만 머무르지 않고 시스템의 명시적인 부분으로 포함될 수 있습니다. 즉, 전문 지식을 하드코딩하는 것이 아니라 시스템에 모델링하고 삽입할 수 있습니다.

아는 것이 힘: 안정적인 변경 처리

데이터센터에서는 끊임없이 변화가 이루어지므로 안정적으로 변경을 처리하는 것이 무엇보다 중요합니다. 변경은 다음 두 가지 경위로 이루어집니다.

"살아남는 것은 가장 강하거나 가장 지능이 높은 종이 아니라 변화에 가장 잘 반응하는 종이다."

- 찰스 다윈, 1809

1. **운영자의 인텐트.** 운영자가 가상 네트워크와 같은 새로운 서비스를 추가하거나 인프라에서 리소스를 추가/제거하거나 일부 정책을 변경하려 할 수 있습니다. 이 경우 운영자는 개시되는 변경을 제어할 수 있습니다. Apstra는 상태 저장 오케스트레이션(Stateful Orchestration)을 사용해 운영자의 인텐트를 안정적으로 구현합니다.
2. **관리 대상 인프라의 장애.** 또한 변화는 관리 대상 인프라의 장애(예: 과도한 패킷 드롭 또는 트래픽 불균형)로 인해 발생할 수 있습니다. 운영자는 이러한 변화를 제어할 수 없으며 대개 수많은 운영 상태가 나타납니다. Apstra는 운영자가 운영 상태의 변화에 대처할 수 있도록 인텐트 기반 분석을 제공합니다.

두 가지 유형의 변화에 모두 안정적으로 대처하는 것은 인프라 상태에 대한 지식에 달려 있습니다. 다음 두 가지 조건이 충족되어야 합니다.

1. 지식은 컨텍스트 속에서 해석되어야 합니다. 즉, 컨텍스트 모델 섹션에 설명된 대로 상태와 기대치 사이의 관계를 이해해야 합니다.
2. 지식은 적시성이 있어야 합니다. 즉, 실시간 모니터링 및 알림 섹션에 설명된 대로 현재 상태를 반영해야 합니다.

상태 저장 오케스트레이션

상태 저장 오케스트레이션(Stateful Orchestration)은 인텐트 기반으로 운영자에 의한 변경을 안정적으로 만듭니다. 그림 2는 상태 저장 오케스트레이션 플로우를 보여줍니다.

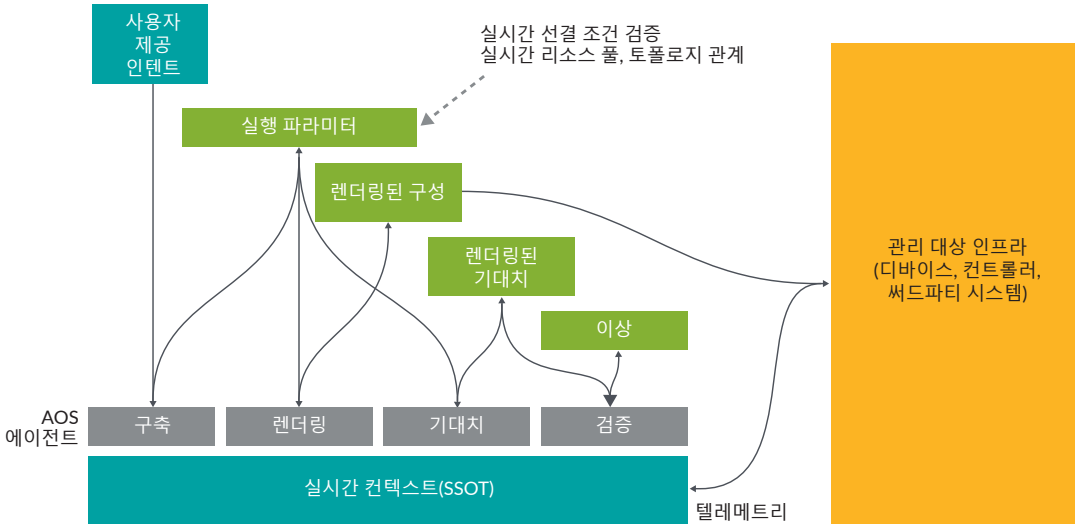

그림 2: 상태 저장 오케스트레이션 플로우

기존의 제대로 작동하는 시스템을 변경하려는 경우 상태 저장 오케스트레이션을 사용하여 변경 인텐트만 제공할 수 있습니다. 구현에 구애받지 않으므로 인텐트 사양을 더 간단하게 만들어 오류를 줄일 수 있습니다. 상태 저장 오케스트레이션 중에 실행되는 전체 단계는 다음과 같습니다.

1단계: 실시간 선결 조건 검증. 이 새로운 요청이 일부 정책을 위반합니까? 예를 들어 이 가상 네트워크를 만드는 것이 허용됩니까? 이 가상 네트워크로 인해 보안 허점이 생깁니까? 일부 다른 디바이스가 이미 오프라인 상태여서 이 디바이스를 오프라인으로 전환하면 시스템이 취약해질 것임을 알고도 디바이스를 유지 관리 모드로 전환할 수 있습니까? 상태 저장 오케스트레이션을 사용하면 컨텍스트 그래프를 기반으로 이러한 모든 질문이 실시간으로 제기되고 검증됩니다.

2단계: 실시간 실행 파라미터(Enforcement Parameters) 도출. 인텐트를 구현하려면 적절한 실행 메커니즘을 위한 특정 파라미터를 제공하고 특정 관리 요소에서 활성화해야 합니다. Apstra는 레퍼런스 설계를 사용하여 현재 컨텍스트에 대한 이해와 함께 인텐트 구현 방법을 지정하므로 시스템에서 실시간 실행 파라미터를 자동으로 도출할 수 있습니다. 운영자가 실수로 잘못된 명령, 인터페이스, IP 주소 또는 VLAN을 삽입할 가능성이 없습니다.

3단계: 실시간 다중 컨피그레이션 실행. Apstra는 다양한 벤더 및 기술별 API를 사용할 수 있는 여러 요소에 대한 컨피그레이션 구축을 자동화하여 실시간으로 다중 컨피그레이션을 실행합니다.

4단계: 실시간 기대치 생성. 행동 양식 변경 역할을 하는 레퍼런스 설계를 통해 Apstra는 결과가 인텐트를 이행했다고 선언하기 위해 충족해야 하는 상태를 설명하는 기대치를 실시간으로 생성할 수 있습니다.

5단계: 실시간 기대치 검증. 그런 다음 Apstra는 텔레메트리 및 컨텍스트 기반 운영 분석 모음을 트리거하여 실시간으로 기대치를 검증합니다.

6단계: 검증된 서비스 결과. 이 프로세스가 끝나면 사용자는 명확한 용어로 검증된 서비스 결과를 확인할 수 있습니다. 인텐트 또는 예상되는 운영 상태의 모든 변경 사항은 실시간으로 컨텍스트 모델에 반영되며 변경 사항을 인식해야 하는 모든 구성 요소에도 실시간으로 알림이 제공됩니다. Apstra는 인텐트 기반 분석을 사용하여 최초 운영 데이터에서 관련 지식을 추출합니다.

상태 비저장 오케스트레이션(Stateless Orchestration)에서는 많은 단계가 누락되게 됩니다. (그림 3 참조). 일반적으로 상태 비저장 오케스트레이션에서는 컨피그레이션 실행 결과가 전부입니다. 즉, 컨피그레이션을 여러 시스템으로 푸시하고 관리되는 요소에서 이를 수락했는지 검증합니다. 그리고 대개 여기서 상태 비저장 오케스트레이션이 끝납니다. 서비스 기대치 개념이나 기대치가 충족되었는지에 대한 검증이 없습니다. 자동화된 서비스 결과 검증이 수행되지 않으며 대신 별개의 여러 시스템에서 사용자에게 최초 운영 상태에 관한 "단일 창"을 제공할 뿐입니다. 결과가 달성되었는지 알아보기 위해 사용자가 직접 확인하는 역할을 수행해야 합니다. 이 단일 창은 일반적으로 너무 많은 정보를 포함하며 사용자가 직접 확인하는 과정을 거치더라도 관련 컨텍스트가 부족하므로 매우 어렵고 주관적으로 이루어집니다.

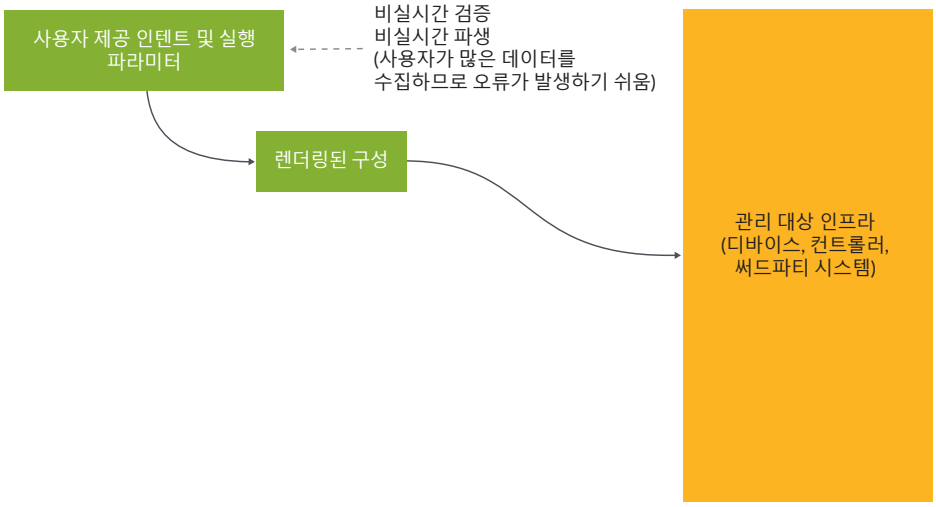

그림 3: 상태 비저장 오케스트레이션

인텐트 기반 분석

인텐트 기반 분석(Intent-Based Analytics, IBA)은 최초 텔레메트리 데이터(Raw Telemetry Data)에서 지식을 추출하여 운영자가 인프라의 운영 상태 변화에 대처하는 데 도움이 됩니다. 앞서 언급했듯이 IBA를 사용하려면 실시간 쿼리 가능 컨텍스트가 있어야 합니다.

지식 추출은 다음 두 단계로 이루어집니다.

1. **관심 상태 감지**(관찰할 상황). 상태 감지는 IBA 프로브를 통해 수행됩니다.
2. **자동으로 관심 상태 분류 및 관심 상태 간 관계 도출**. 상태의 의미론적 내용은 다양합니다. 다시 말해서, 일부 상태가 다른 상태보다 더 중요합니다. 상태 간 관계를 이해하여 중요한 조치 가능 상태(근본 원인)와 중요한 근본 원인이 해결되면 간단히 사라질 상태를 파악해야 합니다. 상태 분류 및 인과 관계 도출은 IBA의 근본 원인 식별 구성 요소를 통해 수행됩니다.

관심 상태(Conditions of Interest) 모델링

IBA는 관심 상태를 모델링하는 데 이상 징후, 증상, 영향, 근본 원인의 4가지 범주를 사용합니다.

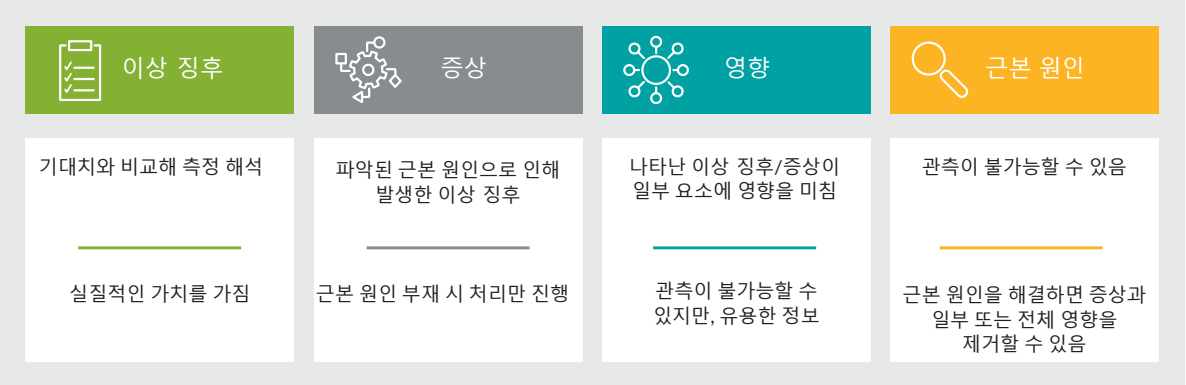

그림 4: 상태 범주화

이상 징후

이상 징후는 본질적으로 몇 가지 기대치에 대한 측정 해석 또는 집계를 나타내며, 이에 따라 단순한 측정보다 더 실질적인 가치를 갖습니다.

증상

증상은 잘 알려진 근본 원인으로 인해 발생하는 이상입니다. 증상은 일반적으로 쉽게 관찰되지만 고칠 수 없고 치료만 할 수 있습니다. 마치 환자에게 아스피린을 투여하면 열은 치료되지만 질병 자체는 치료되지 않는 것과 같습니다. 병을 치료하지 않으면 열이 돌아옵니다. 중요한 점으로, 증상은 근본 원인이 해결되면 사라지므로 조치를 취할 수 없고 단순히 근본 원인을 진단하는 데 유용합니다.

영향

영향은 이상 징후의 결과로 무언가가 발생했음을 나타냅니다. 영향은 항상 관찰할 수 있는 것이 아니지만 알아두면 유용합니다. 예를 들어, 중요한 고객이 전화로 불만을 제기하기 전에 디바이스 장애나 과도한 패킷 손실로 인해 영향을 받을 것이라는 선제적으로 파악할 수 있다면 운영자 입장에서 더할나위 없이 좋을 것입니다.

영향을 관찰할 수 없는 경우 인텐트에 대한 지식과 인텐트를 구현하는 데 사용되는 실행 메커니즘을 기반으로 영향을 산정할 수 있습니다. 영향을 이해하면 운영자가 작업에 가장 큰 영향을 미치는 근본 원인과 이상 징후가 무엇인지 파악하여 우선 순위를 정하는 데 도움이 됩니다.

근본 원인

모든 상태 중 가장 중요한 것은 근본 원인입니다. 근본 원인은 항상 관찰할 수 있는 것이 아니어서 진단하기 어려울 수 있지만 많은 증상과 영향을 야기합니다. 근본 원인을 해결하면 관련 증상과 영향이 사라지므로 근본 원인에 대한 조치가 가능합니다.

IBA 프로브: 알고 있는 지식 확인

마크 트웨인의 명언이 데이터센터 네트워크에도 적용됩니다. "곤경에 빠지는 것은 뭔가를 몰라서가 아니다. 뭔가를 확실하게 안다는 착각 때문이다." 인텐트는 우리가 알고 있는 정상 상태를 특징짓고 공식화합니다. IBA는 우리가 알고 있는 것이 실제로 그런지 확인합니다.

"곤경에 빠지는 것은 뭔가를 몰라서가 아니다. 뭔가를 확실하게 안다는 착각 때문이다." - 마크 트웨인

IBA 프로브는 관심 상태를 감지하는 역할을 하며 레퍼런스 설계의 일부인 행동 방식 변경에 의거하여 수행됩니다. IBA 프로브는 데이터를 가져오고 일부 처리를 적용한 다음 결과를 기대치와 비교합니다. IBA 프로브는 기본적으로 사용자가 관심 상태(즉, 관찰할 상황)를 설정할 수 있도록 하는 구성 가능한 데이터 처리 파이프라인입니다.

데이터 소스 및 쿼리

프로브의 초기 단계는 일반적으로 최초 텔레메트리 데이터를 가져오는 데이터 소스 단계입니다. 데이터 소스 단계의 컨피그레이션에는 수집할 데이터를 정확하게 표현할 수 있는 매우 강력한 기능인 쿼리가 포함되어 있습니다.

예를 들어, 운영자가 패브릭 인터페이스에만 적용되는 ECMP(Equal-Cost Multipath Routing) 불균형을 분석하는 데 관심이 있고 특정 운영 체제 버전에서 ECMP 해싱 알고리즘에 버그가 발생했다는 보고가 있다고 가정해 보겠습니다. 쿼리로 모든 TOR(Top-of-Rack) 스위치에서 인터페이스 카운터를 수집해야 함을 나타낼 수 있지만 패브릭 인터페이스와 스위치 OS의 특정 버전 x.y.z를 실행하는 스위치에 대해서만 수집하도록 표현할 수 있습니다. 이제 관찰할 상황이 설정되었으므로 운영자가 변경 사항에 대해 염려할 필요가 없습니다. 기준과 일치하는 스위치에 새 패브릭 링크가 추가되면 자동으로 분석에 포함됩니다. 새 스위치가 추가되면 자동으로 포함되고, 다른 스위치가 x.y.z 버전으로 업그레이드되면 자동으로 포함됩니다. IBA 프로브는 유지 관리가 필요 없습니다.

스테이지 프로세서(Stage Processors)

운영자라면 즉각적인 최초 텔레메트리 데이터 값 보다 집계된 데이터 혹은 트렌드 값에 더 관심이 있으실 것입니다. IBA에는 평균, 최소/최대, 표준 편차 등의 계산과 같이 정보를 집계하는 스테이지 프로세서가 포함되어 있습니다. 운영자는 이러한 집계를 예상치와 비교하여 집계 지표가 지정된 범위에 들었는지 지정된 범위를 벗어났는지를(이상 플래그가 지정됨) 식별할 수 있습니다.

운영자는 이상 징후가 특정 임계값을 초과하는 기간 동안 지속되는지 여부를 확인하고 임계값을 초과할 때만 이상에 플래그를 지정하여 일시적 또는 임시 상태에 대한 이상 플래그 지정을 방지할 수 있습니다. 이를 위해 운영자는 지정된 범위 내에 든 시간을 확인하는 time-in-state 프로세서를 포함하도록 후속 단계를 컨피그레이션하면 됩니다.

숫자 데이터를 넘어 다양한 처리 지원

프로브는 숫자 데이터에 대해서만 작동하는 것이 아닙니다. 컨트롤 플레인과 데이터 플레인의 정확성을 검증하는 등의 용도로도 프로브를 사용할 수 있습니다. 예를 들어, 운영자는 인텐트에 따라 예상 라우팅 또는 포워딩 테이블을 작성하도록 쿼리를 사용하여 데이터 소스 프로세서를 구성할 수 있습니다.

EVPN(Ethernet VPN) 환경의 경우 존재하는 가상 네트워크, 엔드포인트 위치, 실행 메커니즘 등에 대한 정보가 인텐트에 포함됩니다. 그런 다음 이 예상 테이블을 텔레메트리 데이터와 비교하여 불일치가 발견되면 이상 징후에 관한 플래그를 지정할 수 있습니다.

또한 포워딩 및 라우팅 테이블 크기의 급격한 변화를 추적하고 추세가 예상과 다를 때 경고하도록 프로브를 구성할 수 있습니다. "예상" 임계값은 가상 네트워크 수, 엔드포인트 수, VXLAN 터널 엔드포인트(VTEP) 수, 일부 문제가 있는 VTEP 수의 함수가 될 수 있으므로 인텐트에서 동적으로 계산할 수도 있습니다. 가능성은 무한합니다.

프로브는 간단한 REST 호출 또는 GUI 사용을 통해 선언적으로 생성, 활성화, 비활성화할 수 있습니다. 프로브 활성화는 텔레메트리의 트리거로도 작동합니다. 즉, 특정 텔레메트리 데이터는 해당 데이터에 관심을 가진 프로브가 있는 경우에만 수집되므로 프로브는 본질적으로 텔레메트리 수집 구성 메커니즘 역할을 합니다. 데이터 소스 프로세서의 쿼리를 통해 필요에 따라 구성을 세분화하고 정밀하게 만들 수 있으므로 어떻게 처리할지 명확한 계획 없이 수많은 데이터가 수집될 때 발생하는 "데이터 축적 문제(Data Hoarding Disorder)"를 해소할 수 있습니다. 데이터 축적 문제가 있으면 데이터 저장 및 처리 비용이 크게 증가합니다.

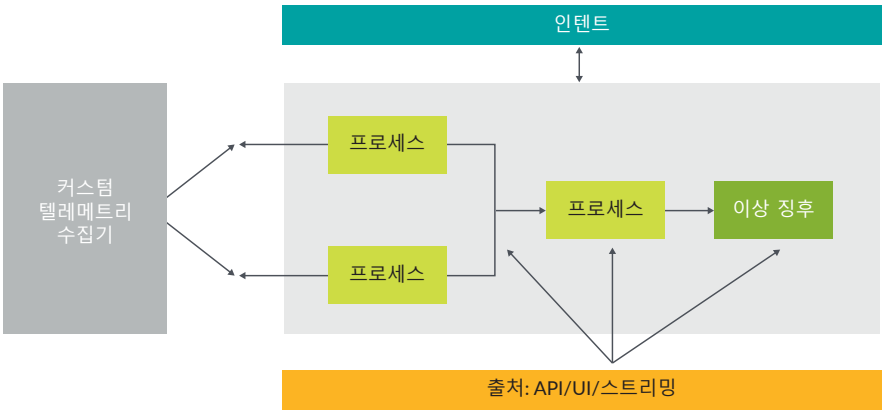

그림 5: 텔레메트리 수집

프로브가 구성되고 나면 API, GUI 및 스트리밍 엔드포인트를 통해 모든 단계의 출력을 사용할 수 있습니다. Apstra에는 기본 제공 프로브 세트가 포함되어 있으며, 오픈 소스 형태의 프로브 정보를 GitHub에서 확인할 수 있습니다. 또한 사용자가 처음부터 새로운 프로브를 생성할 수도 있습니다.

근본 원인 식별

RCI(Root-Cause Identification)는 인프라에서 식별된 여러 상태 간의 인과 관계를 자동으로 분류하고 도출하는 메커니즘입니다. RCI의 주요 이점은 근본 원인 장애의 결과일 뿐인 조치 불가능한 수많은 상태 가운데 운영자의 조치가 필요한 근본 원인 상태를 파악할 수 있다는 것입니다. 예를 들어 운영자가 인프라를 적절히 계측하고 "단일 창" 콘솔에서 이상 상태의 로그를 확인한다고 가정해 보겠습니다(그림 6 참조). 빨간색 점은 다양한 요소에서 발생하는 인프라 전반에 산재된 다양한 유형의 상태를 나타냅니다.

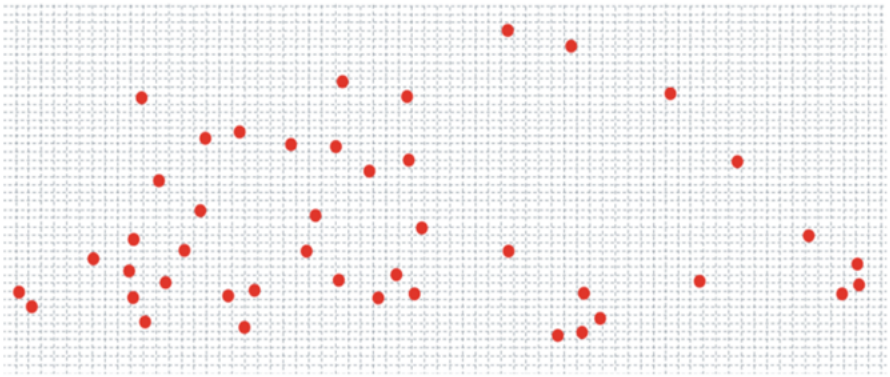

그림 6: 단일 창 콘솔

이 그림에서 문제는 노이즈만 많고 조치를 취할 수 없다는 것입니다. 분류가 없기 때문에 운영자의 주의를 끄는 상태는 많지만 어떤 상태에 대해 조치를 취해야 할지 알기 힘듭니다. 바로 RCI가 이에 대한 해결책을 제시합니다. RCI는 서비스가 무엇이며 어떻게 구현되고 실행 메커니즘에 매핑되는지, 관리되는 인프라의 요소들이 서로 어떻게 관련되어 있는지를 알려주는 컨텍스트 지식을 사용하여 근본 원인과 관련 증상 및 영향을 식별하고 분류합니다. 그림 7에 나온 것처럼 RCI를 이용한 분석 결과는 다음과 같습니다.

- 근본 원인은 "spine_1"이라는 스위치의 메모리 누수였습니다(근본 원인: 큰 빨간 점).
- 이 메모리 누수로 인해 메모리 부족 프로세스 킬러(Out-of-Memory Process Killer)가 몇 가지 프로세스를 중단시켰으며 그 중 하나가 라우팅 프로세스였습니다(증상: 짙은 파랑 점).
- 이로 인해 예상 라우팅 테이블 항목이 누락되고 이 디바이스와 피어링 디바이스에서 BGP 세션이 실패했습니다(증상: 짙은 파랑 점).
- 결과적으로 고객 X 및 Y에 속한 엔드포인트에서 연결 문제가 발생했습니다(영향: 옅은 파랑 점).

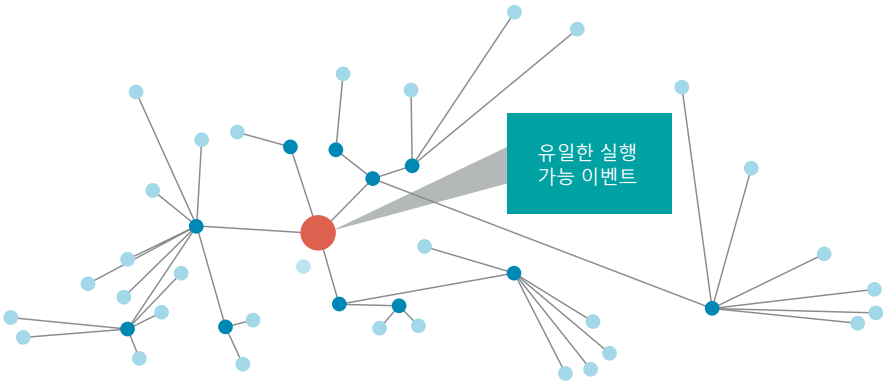

그림 7: 근본 원인 식별 콘솔

RCI는 산더미 같은 정보로 운영자에게 부담을 주고 운영자가 직접 눈으로 확인하여 상관 관계 분석을 수행해야 하는 "단일 창"의 복잡한 프로세스를 자동화하며 단순히 필요한 조치를 식별하는 간단한 창을 운영자에게 제공합니다.

컨텍스트 모델

모든 데이터가 있다고 해서 모든 답을 확보한 것은 아닙니다. 또한 자동으로 지식을 추출하지 않고 엄청난 양의 데이터를 수집하면 OpEx가 급격히 증가합니다. 따로 시간을 투입해 데이터를 이해해야 하기 때문입니다. 데이터가 지식을 제공하거나 질문에 적절히 답하지 않는다면 데이터 자체의 가치는 떨어집니다.

최초 텔레메트리 데이터는 일반적으로 수평적 확장 및 분할에 적합한 키-값(Key-Value) 저장소에 저장됩니다. 그러나 이러한 저장소에서는 단순 키 조회 이외의 쿼리는 지원되지 않습니다. 지식을 추출하려면 관련 쿼리를 지원하도록 구현하고 이러한 쿼리를 작성하기 위한 컨텍스트를 갖춰야 합니다.

SQL 데이터 저장소의 경우 복잡한 쿼리를 지원하지만, SQL 테이블은 설계 중에 생성되는 예상 쿼리를 중심으로 한다는 문제가 있습니다. 런타임에 다른 질문을 하고 싶다면 데이터베이스를 비정규화해야 하며, 그러지 않으면 쿼리에서 많은 조인이 발생해 성능이 저하되고 실행이 중단될 수 있습니다. 그러나 운영 지식의 추출은 모두 런타임에 이루어지는 쿼리를 통해 가능합니다. 바로 여기서 컨텍스트 데이터의 그래프 기반 구현이 진가를 발휘합니다. 이러한 구현은 런타임에 많은 수의 "조인(Joins)"이 있는 임의의 복잡한 쿼리를 지원합니다. 또한 그래프 모델은 기본 빌딩 블록, 노드 및 관계 인스턴스를 더 추가하는 간단한 방법으로 쉽게 확장할 수 있습니다.

쿼리의 활용 가치를 파악하는 데 도움이 되므로 이 컨텍스트에 포함되는 내용을 자세히 살펴보겠습니다.

- **설계 아티팩트**: 우선, 인텐트 그래프 컨텍스트에는 설계 아티팩트가 포함됩니다. 예를 들어 데이터센터 네트워크를 설계하는 경우 원하는 초과 할당(oversubscription) 비율, 원하는 서버 수, 원하는 고가용성(HA) 구성(단일 연결, 이중 연결)이 포함될 수 있습니다.
- **리소스 할당**: 인텐트 그래프 컨텍스트에는 리소스 할당과 관련된 결정이 포함됩니다. 이는 인프라를 "cattle"과 "pets" 중 어떤 모델로 관리할지를 결정합니다.
- **격리 정책**: 또한 인텐트 그래프 컨텍스트에는 격리 정책이 지정됩니다. 특정 리소스(IP, ASN, VLAN)를 재사용할 수 있는지 여부를 예로 들 수 있습니다.
- **세그먼트 분할 정책**: 인텐트 그래프 컨텍스트에는 어떤 엔드포인트와 워크로드가 어떤 조건에서 서로 통신할 수 있는지를 지정하는 세그먼트 분할(Segmentation) 정책이 포함됩니다.
- **실행 정보**: 인텐트 그래프 컨텍스트에는 세그먼트 분할 및 연결 가능성 정책이 실제로 구현된 위치(물리적 또는 가상 어플라이언스)와 방법(사용된 메커니즘)에 대한 실행 정보가 포함됩니다.
- **외부 연결 가능성 정책**: 데이터센터의 경우 인텐트 그래프 컨텍스트에는 외부 환경에서 볼 수 있는 내부 엔드포인트와 내부 환경에서 볼 수 있는 외부 엔드포인트를 지정하는 외부 연결 가능성 정책이 포함될 수 있습니다. 또한 각 테넌트가 소유한 세그먼트와 각 테넌트에 약속된 서비스 레벨 및 서비스 레벨 목표가 포함되어 있습니다.
- **비즈니스 관점**: 인텐트 그래프 컨텍스트에는 서비스 레벨 계약 및 서비스 레벨 계약을 준수하지 못할 경우 발생하는 비용 같은 비즈니스 관점이 포함됩니다.

8차원 뷰(8-D View)

이러한 모든 아티팩트가 단일 그래프에 표시되지만 이 단일 정보 소스는 논리적으로 최소한 다음 8개 차원(이 예에서 도출됨, 잠재적으로 더 많을 수 있음)을 갖는 단일 다차원 공간입니다.

- 설계
- 리소스
- 격리
- 세그먼트 분할
- 실행
- 외부 연결 가능성 정책
- 서비스 레벨
- 비즈니스 관점

결과적으로 문제가 발생할 경우 단일 쿼리를 이 8차원 뷰에 대해 실행하면서 다음과 같은 복잡한 질문에 답할 수 있습니다. "최근 요소 장애 상태를 감안할 때 설계 목표가 여전히 유효한가?" "원하는 격리 정책과 함께 올바른 실행 지점에서 올바른 리소스가 사용되는가?" "세그먼트 분할 정책에 어떤 식으로든 영향이 미치는가?" "서비스 레벨 목표가 충족되는가?" "이 장애로 인해 치르는 대가가 무엇인가?"

이 8차원 뷰는 "있으면 좋은 것"이 아니라 안정적인 운영을 위한 필수 요건입니다. Apstra 인텐트 기반 시스템에는 이 8차원 뷰가 기본으로 제공됩니다. Apstra가 없이는 전문가의 머릿속에 존재하는 지식과 여러 정보 소스에 걸쳐 있는 매우 복잡한 레이어를 사용하여 이 8차원 뷰를 재구성해야 합니다. 개별 정보 소스가 통합을 염두에 두고 작성되지 않고 서로 다른 의미 체계와 동작을 갖는 환경에서 이러한 레이어를 구축한다는 것은 잘해야 별 의미 없는 힘든 작업이 되며 잘못하면 통제할 수 없는 악몽과 같아집니다.

실시간 모니터링 및 알림

Apstra의 주요 목적은 인텐트와 인프라 운영의 결과 상태에 대한 지식을 추출하는 것입니다. 이 지식은 시기적절한 경우, 즉 현재 상황을 반영하는 경우에 유용합니다. Apstra의 핵심 기능은 클라이언트가 관심 상태를 구독하고 해당 상태 조건이 충족될 때 실시간으로 알림을 받을 수 있도록 라이브 쿼리를 구성하는 것입니다. 이 컨텍스트의 상태는 인텐트 및 운영 상태와 관련이 있습니다. 이는 변화에 안정적으로 대처하기 위한 필수 전제 조건입니다.

아키텍처 개요 섹션에서 볼 수 있듯이 사용자 수준에서 제공되는 동일한 게시/구독 패러다임이 동등한 하위 수준 게시/구독 메커니즘에서 지원됩니다. 이 메커니즘은 애플리케이션 논리를 구현하는 Apstra 시스템 프로세스의 데이터 중심적 논리 통신 채널 역할을 하는 분산 데이터 저장소를 통해 구현됩니다.

자동 운영

마지막으로 중요한 것은 일부 이벤트에 대한 대응을 자동화하여 문제를 해결하거나, 포렌식 분석을 위해 기록하거나, 드릴다운을 통해 근본 원인 분석에 도움이 되는 텔레메트리를 수집하는 등의 작업을 자동으로 수행하는 것입니다.

자동화된 대응에는 다음과 같은 이점이 있습니다.

- 문제 해결 파라미터가 최신 단일 정보 소스에서 자동으로 도출되고 잘못된 구성이나 오래된 데이터의 영향을 받지 않으므로 위험 감소
- 문제 해결이 적시에 이루어지므로 고객 경험 향상
- 수동 문제 해결 및 문제 해결 플레이북의 수동 실행이 필요 없기 때문에 운영 비용 절감

필요한 기능은 이미 존재하므로 자체 운영 네트워크에 대한 장애 요인은 기술적 한계보다는 조직 또는 개인의 저항에서 비롯됩니다.

궁극적으로 자동화된 대응을 통해 네트워크 자체 운영 및 자가 복구가 가능합니다. 필요한 기능은 이미 존재하므로 자체 운영 네트워크에 대한 장애 요인은 기술적 한계보다는 조직 또는 개인의 저항에서 비롯됩니다.

연장성: 미래에 대비한 데이터센터 네트워크 구축

연장성 측면에서는에는 다음 3가지 차원이 있습니다.

1. **레퍼런스 설계 연장성**. 이는 인프라 구성 요소들이 인텐트를 이행하기 위해 함께 작동하는 방식을 결정합니다. 여기에는 그래프 모델과 인프라의 실행 메커니즘에 인텐트가 매핑되는 방식을 수정할 수 있도록 지원하는 플러그인이 포함됩니다.
2. **분석 연장성**. 이는 관찰할 새로운 상태 또는 상황의 정의와 관련이 있으며 사용자가 새로운 검증과 분류 및 상관관계 파악 방법을 정의할 수 있도록 지원합니다.
3. **유연한 서비스 API**. 이는 최상위 서비스 정의를 확장할 수 있도록 지원합니다.

레퍼런스 설계

앞에서 언급했듯이 레퍼런스 설계는 인텐트가 실행 메커니즘에 매핑되는 방식과 인텐트가 이행된 것으로 간주되기 위해 충족해야 하는 기대치를 정의하는 동작 계약입니다. 계약의 모든 측면을 수정하거나 확장할 수 있습니다. 새로운 노드 및 관계 유형을 정의할 수 있으며, 구성 템플릿 및 리소스 할당 메커니즘을 변경할 수 있습니다.

분석 연장성

다음 두 가지 메커니즘을 통해 새로운 분석 기능을 도입할 수 있습니다.

1. **새로운 IBA 프로브**를 정의하여 새로운 관심 상태를 감지할 수 있습니다. 여기에는 새로운 텔레메트리 수집기와 상태별 데이터 처리 파이프라인이 포함될 수 있습니다. 또한 프로브를 게시한 후 공개 리포지토리에서 가져올 수 있습니다.
2. **새로운 RCI 모델**을 정의하고 시스템에 로드할 수 있습니다. RCI 모델은 본질적으로 새로운 유형의 근본 원인을 이 근본 원인으로 인한 일련의 증상에 매핑하는 것입니다. 이 모델이 정의되고 로드되면 Apstra 시스템은 관찰된 증상을 기반으로 근본 원인을 자동으로 식별합니다.

서비스 API

Apstra는 그룹 기반 정책의 형태로 서비스 레벨 API를 노출하여 구현에 구애받지 않는 방식으로 광범위한 서비스 및 정책을 지원할 수 있는 유연성을 제공합니다.

그룹 기반 정책에서 인텐트 그래프는 그룹(구성원 관계)으로 배치되는 엔드포인트를 나타내어 일부 공통 동작에 대한 인텐트를 표현합니다. 해당 동작을 정의하기 위해 정책이 인스턴스화되고 그룹 또는 개별 엔드포인트와 관련지어집니다.

정책은 방향성(시작/끝 관계) 또는 비방향성(적용 대상) 방식으로 그룹과 관련지어질 수 있습니다. 정책은 규칙의 모음입니다. 규칙 간의 순서가 중요한 경우 규칙은 다음 규칙 관계를 가질 수 있습니다. 그룹은 다른 그룹들(계층 구조)로 구성될 수 있습니다. 또한 그룹은 일부 제약 조건을 표현하기 위해 다른 그룹과 관계를 가질 수 있습니다(예: "이 엔드포인트/그룹은 이 포트 그룹 뒤에 있음"). 엔드포인트, 그룹, 정책 및 규칙은 연결 인텐트를 표현하기 위한 빌딩 블록으로 생각할 수 있습니다.

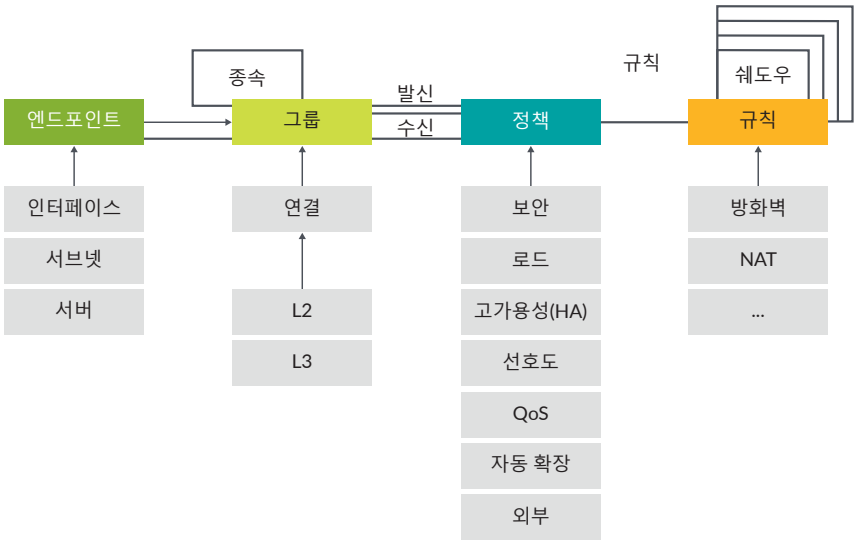

그림 8: 그룹 기반 정책

엔드포인트는 정책의 적용을 받는 인프라 요소이며, 이에 따라 구성 면에서 상당히 일반적입니다. 인터페이스(물리적 또는 가상), 서버, VM(Virtual Machine), 컨테이너 또는 애플리케이션 엔드포인트를 나타낼 수 있습니다. 그림 8의 파란색 화살표는 "논리적" 상속을 나타냅니다. 엔드포인트에는 엔드포인트를 보다 정확하게 정의하는 파라미터(예: 인터페이스 이름, 포트 번호, 일련 번호 및 호스트 이름, VM UUID, 컨테이너 IP, 애플리케이션 프로토콜, UDP/TCP 포트 번호)가 포함됩니다. 또한 엔드포인트는 Apstra로 관리되지 않는 요소(외부 엔드포인트)를 나타낼 수도 있으며 이 경우 Apstra가 상호 작용해야 하는 외부 시스템(예: 외부 라우터 IP/ASN)의 제약 조건을 표현하는 데 사용됩니다.

엔드포인트는 그룹으로 배치됩니다. 개별 엔드포인트는 각각 의도된 동작의 다른 측면을 표현하는 여러 그룹의 구성원이 될 수 있습니다. 예를 들어 한 그룹은 다음을 나타낼 수 있습니다.

- "East" 지역의 서버 집합
- 로드 밸런싱에 사용될 서버 그룹
- 중복 그룹을 형성하는 서버 그룹

그룹에는 의미 체계가 오버로드될 수도 있습니다. 예를 들어, L2 도메인 그룹의 구성원인 엔드포인트는 L2 브로드캐스트 도메인(서브넷)의 구성원입니다. 마찬가지로, L3 도메인의 구성원인 엔드포인트는 L3 연결이 가능합니다.

또한 그룹은 다른 그룹들로 구성될 수 있습니다. 그룹의 엔드포인트 구성원은 명시적으로 인스턴스화될 수도 있고, 그룹 정의의 일부인 동적 구성원 사양에서 엔드포인트가 암시될 수도 있습니다.

예를 들어, L3 도메인에는 CIDR(Classless Interdomain Routing) 또는 서브넷이 속성으로 포함될 수 있습니다. 따라서 해당 범위/서브넷에 IP가 있는 모든 엔드포인트는 암시적으로 그룹의 구성원입니다. 이 경우 L3-서버 레퍼런스 설계는 여러 외부/내부 서브넷을 지정합니다. 엔드포인트(컨테이너)는 Apstra에서 명시적으로 지정되고 관리되지 않지만, 컨테이너가 지정된 L3 도메인에 속한다는 것이 암시됩니다. 그러나 세분화된 세그먼트 분할을 모델링하기 위해 컨테이너 엔드포인트와 이를 호스팅하는 서버의 명시적 사양도 도입할 예정입니다.

정책은 공통 동작을 정의하며, 해당 동작을 정확하게 정의하기 위한 파라미터를 포함할 수 있습니다. 정책은 비방향성 방식으로 그룹에 적용되거나 보안 정책에서와 같이 필요한 경우 방향이 지정될 수 있습니다. 구체적인 정책의 예로는 보안, 로드 밸런싱, HA, 선호도(예: 엔드포인트를 코로케이션하거나 배포하고자 함), QoS(Quality of Service) 등이 있습니다.

정책에는 필요에 따라 규칙이 포함될 수 있습니다. 규칙은 일반적으로 "조건과 그 뒤의 작업" 패턴을 따릅니다. 예를 들어 보안 정책에는 "match"라는 조건문과 "allow/deny/log"라는 작업이 있을 수 있습니다.

확장성: 원활한 확장

Apstra는 네트워크 트랜스페어런트 분산 상태 액세스(Network-Transparent Distributed-State Access) 및 관리를 지원하며, 별개의 프로세스들을 통해 병렬 실행을 지원합니다. 실시간 실행 스케줄링과 함께 이벤트 기반 비동기 실행 모델을 통해 실시간 실행을 지원하고, 머신 레벨 효율성을 위해 중간 언어로 C++를 사용하여 컴파일함으로써 효율성과 예측 가능성을 지원합니다. 확장에는 다음 세 가지 차원이 있습니다.

상태 확장

첫 번째 차원은 상태를 확장하는 것입니다. Apstra 데이터 저장소는 더 많은 고가용성(HA) 서버 페어를 추가하여 수평으로 확장됩니다. 인텐트와 텔레메트리 데이터 저장소는 분리되어 필요에 따라 독립적으로 확장할 수 있습니다. 계층적 데이터 저장소도 제공됩니다. 여기서 최상위 데이터 저장소는 데이터 저장소 전반의 상태 상관관계를 파악하는 데 필요한 대로 다음 수준 데이터 저장소의 필수(설계자가 지정) 상태 하위 집합만 구독(동기화)합니다.

처리 확장

두 번째 차원은 처리를 확장하는 것입니다. Apstra는 필요에 따라 처리 부하를 공유할 여러 복사본의 처리 에이전트를 에이전트 유형별로 시작할 수 있습니다. 에이전트를 호스팅할 서버를 더 추가하여 더 많은 에이전트를 추가할 수 있습니다. 또한 Apstra는 에이전트의 수명 주기를 관리합니다.

시스템의 상태 기반 게시/구독 아키텍처를 통해 에이전트는 적절히 정의된 상태 하위 집합에 대응(애플리케이션 논리 제공)할 수 있습니다. 전체 인텐트의 적용은 서로 다른 상태 하위 집합을 처리하도록 위임된 별개의 에이전트들을 통해 수행됩니다. 즉, 인텐트 또는 운영 상태에 변화가 있을 때 에이전트의 대응은 "점진적 변경"이며 전체 상태의 규모와 무관합니다.

Apstra는 규모 및 관련 복잡성을 처리하기 위해 기존 접근 방식인 분해를 사용합니다. "모두가 모든 것을 안다"는 접근 방식은 확장성이 없습니다. 원하는 상태에 대한 지식을 배포하고 각 에이전트가 해당 상태에 도달하는 방법을 결정하게 하여 중앙 집중식 의사 결정이 필요 없도록 해야 합니다. Apstra는 라이브 그래프 쿼리를 지원하므로 UI와 같은 클라이언트가 정확히 원하는 것만 요청하고 정확히 필요한 것만 얻을 수 있습니다. 따라서 백엔드에서 가져올 데이터의 양을 세밀하게 제어할 수 있습니다.

네트워크 트래픽 확장

세 번째 차원은 네트워크 트래픽을 확장하는 것입니다. 에이전트와 데이터 저장소 간의 통신은 최적화된 바이너리 채널을 사용하므로 텍스트 기반 프로토콜에 비해 트래픽 양이 크게 줄어듭니다.

내결함성(Fault Tolerance)은 Apstra 애플리케이션을 네트워크로 연결된 별도의 하드웨어 디바이스에서 실행하고 복제된 상태(Replicated State) 및 빠른 상태 복구를 지원하여 프로세스에서 상태를 분리하는 다수의 프로세스로 실행함으로써 달성됩니다.

Apstra 아키텍처 개요

이 백서의 첫 부분에서 데이터센터 네트워크 아키텍처에 관한 몇 가지 과제와 Apstra를 통해 이러한 문제를 해결하는 방법을 알아봤습니다. 여기서는 Apstra의 아키텍처에 대해 자세히 살펴보겠습니다.

Apstra는 분산 상태 관리 인프라를 기반으로 합니다. 이 인프라는 수평으로 확장 가능하고 장애 허용 능력을 갖춘 인메모리 데이터 저장소가 있는 데이터 중심적 통신 패브릭으로 설명할 수 있습니다. 특정 레퍼런스 설계 애플리케이션의 모든 기능은 상태 비저장 에이전트 세트를 통해 구현됩니다. 에이전트는 논리적 게시-구독(Publish-Subscribe) 기반 통신 채널을 통해 서로 통신하고 기본적으로 애플리케이션의 논리를 구현합니다.

모든 Apstra 레퍼런스 설계 애플리케이션은 위에서 설명한 대로 단순히 상태 비저장 에이전트(Stateless Agents)의 모음입니다. 일반적으로 에이전트에는 다음 세 가지 부류가 있습니다.

1. **상호 작용 (웹) 에이전트**는 사용자와의 상호 작용을 담당합니다. 즉, 사용자 입력을 받고 데이터 저장소에서 관련 컨텍스트를 사용자에게 제공합니다.
2. **애플리케이션 에이전트**는 입력 개체를 구독하고 출력 개체를 생성하여 애플리케이션 도메인별 데이터 변환을 수행합니다.
3. **디바이스 에이전트**는 스위치, 서버, 방화벽, 로드 밸런서 또는 컨트롤러와 같은 관리되는 물리적 또는 가상 시스템에 상주하거나 이 시스템의 프록시입니다. 이 에이전트는 구성을 작성하고 기본 (디바이스별) 인터페이스를 통해 텔레메트리를 수집하는 데 사용됩니다.

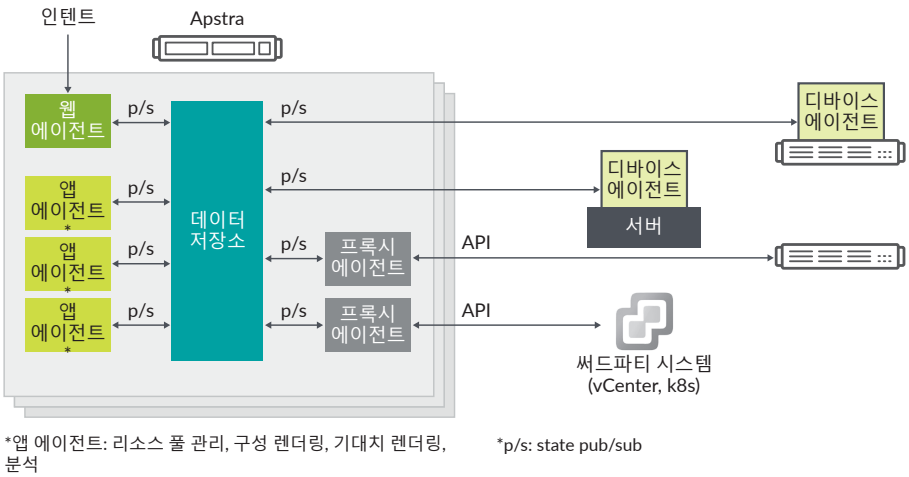

그림 9: Apstra 에이전트

이 상호 작용은 Apstra의 데이터센터 네트워킹 레퍼런스 설계 애플리케이션의 일부를 설명하는 예로 나타낼 수 있습니다.

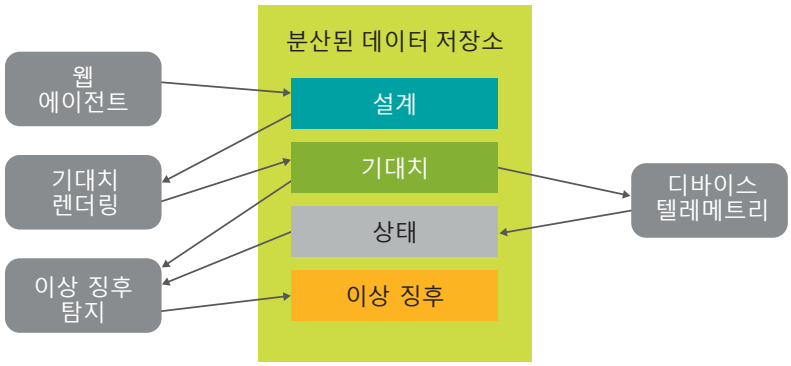

그림 10: Apstra 인텐트 기반 시스템의 데이터센터 네트워킹 레퍼런스 설계 애플리케이션

웹 에이전트는 사용자 입력을 받습니다. 이 경우 스파인, 리프 및 이들 간의 링크 수를 포함하는 L3 Clos 패브릭과 패브릭 IP 및 ASN(Abstract Syntax Notation) 번호에 사용할 리소스 풀에 대한 설계가 포함됩니다. 웹 에이전트는 이 인텐트를 그래프 노드 및 관계와 해당 속성 집합으로 데이터 저장소에 게시합니다.

빌드 에이전트는 이 인텐트를 구독하고 다음 작업을 수행합니다.

- 정확성 및 완전성 검증
- 리소스 풀에서 리소스 할당

검증에 통과했다고 가정하면 그 후에 빌드 에이전트는 리소스 할당과 함께 해당 인텐트를 데이터 저장소에 게시합니다.

1. 구성 렌더링 에이전트가 빌드 에이전트의 출력을 구독합니다.
2. 각 노드에 대해 구성 에이전트가 리소스를 포함한 관련 데이터를 가져와 구성 템플릿과 병합합니다.
3. 또한 기대치 에이전트가 빌드 에이전트의 출력을 구독하고 결과를 검증하기 위해 충족되어야 하는 기대치를 생성합니다.
4. 디바이스 텔레메트리 에이전트가 기대치 에이전트의 출력을 구독하고 관련 텔레메트리 수집을 시작합니다.
5. IBA 프로브가 최초 텔레메트리를 처리하고 기대치와 비교하여 이상을 게시합니다.
6. RCI 에이전트가 이상을 분석하여 증상, 영향 및 식별된 근본 원인으로 분류합니다.

에이전트는 개체를 게시하고 개체의 변경 사항을 구독하여 속성 기반 인터페이스(따라서 데이터 중심이라는 용어 사용)를 통해 통신합니다. 또한 데이터 중심은 데이터 정의가 프레임워크의 일부이며 예를 들어 메시지 기반 시스템이 아닌 개체를 정의하여 구현됨을 의미합니다.

데이터 중심 게시-구독 시스템은 메시지 기반 시스템의 문제를 겪지 않습니다. 메시지 기반 시스템에서는 조만간 메시지 수가 시스템에서 저장하거나 소비할 수 있는 용량을 초과하게 됩니다. 일관된 상태에 도달하기 위해 메시지 기록을 재생해야 하므로 이를 처리하는 것은 어렵습니다.

데이터 중심 시스템은 근본적으로 마지막 상태에만 의존하기 때문에 상태 변화의 급증에 탄력적입니다. 이 상태는 중요한 컨텍스트를 캡처하고 해당 상태로 이어진 모든 가능한(및 관련 없는) 이벤트 시퀀스를 추상화합니다. 상태 머신 패러다임을 사용하여 작성된 코드는 읽고 유지 관리하고 디버그하기가 더 쉽습니다.

어려운 문제(예: 탄력성 및 장애 허용 능력)는 모든 에이전트를 위해 한 번에 해결됩니다. 일반적인 아키텍처는 장애가 발생한 경우 다시 시작할 수 있으며 시스템 데이터베이스(SysDB)에서 구독하는 상태를 단순히 다시 읽어 중단된 부분부터 다시 시작하는 여러 상태 비저장 에이전트로 구성됩니다.

Apstra 아키텍처의 이점

Apstra 아키텍처는 이 백서에서 살펴본 가장 어려운 데이터센터 네트워킹 문제 중 일부를 해결하여 상당한 이점을 제공합니다. 이 아키텍처의 이점은 다음과 같습니다.

- **운영자가 변화에 안정적으로 대처하도록 지원**. 이는 실시간 쿼리 가능한 인텐트 및 운영 컨텍스트를 통해 가능합니다.
- **네트워크 서비스 수명 주기의 모든 측면 단순화**(Day 0, 1, 2 운영 포함). 그리고 단순성은 운영자의 오류 가능성을 줄입니다.
- **상태 저장 오케스트레이션을 통해 운영 위험 감소**. 사전 조건 검증, 사후 조건 검증, 자동화된 구성 렌더링 및 자동화된 기대치 검증을 활용합니다.

근본 원인 식별 및 인텐트 기반 분석의 특별한 이점

Apstra의 운영 분석 구성 요소(근본 원인 식별 및 인텐트 기반 분석)를 사용하면 다음과 같은 이점이 있습니다.

- **평균 복구 시간격(MTTR) 단축 및 SME(Subject Matter Expert) 워크로드 감소**. 단순화된 운영 분석을 통해 실현되는 이러한 이점을 통해 숙련된 인력이 문제 해결이 아닌 향상과 혁신에 시간을 할애할 수 있습니다.
- **더 많은 지식 추출**. 더 적은 데이터를 수집하고 저장하면서 이것이 가능합니다. 레퍼런스 설계 동작 계약을 기반으로 Apstra는 무엇을 찾아야 할지 알고 해당 정보만 수집합니다. 이 즉각적인 처리로 인해 필요하지 않은 데이터의 스토리지 요구량과 사후 처리가 5~6배 감소할 수 있습니다. 이를 통해 인프라를 보다 효율적으로 운영하고 비용을 통제하면서 경쟁력을 유지할 수 있습니다.
- **문제를 빠르고 쉽게 확인.** Apstra는 "단순한 창"에서 중요하고 조치 가능한 이벤트를 식별하며, 단지 식별된 근본 원인의 결과인 증상 같은 노이즈를 제거합니다.
- **복잡한 워크플로우 자동화.** Apstra 시스템을 사용하면 컨텍스트가 풍부한 문제 해결 워크플로우를 자동화할 수 있습니다(그렇지 않으면 번거롭고 비효율적이며 시간과 비용이 많이 듦).
- **제로 터치 유지 관리 달성.** Apstra는 인텐트와 지속적으로 동기화되기 때문에 변화가 있을 때 제로 터치 제로 비용 유지 관리가 가능해집니다. 따라서 복원력이 있고 변경 사항에 자동으로 대응하며 데이터 처리 파이프라인 유지 관리와 관련된 막대한 비용을 절감할 수 있습니다.
- **비용이 많이 드는 DIY(Do-It-Yourself) 개발 불필요**. 데이터 처리 파이프라인 통합의 DIY 개발은 비용이 많이 들고 오류가 발생하기 쉬우며, 핵심 비즈니스에 투입할 시간과 리소스를 빼앗고, 많은 중소기업에 큰 부담으로 작용합니다.
- **고도로 정확한 결과 제공**. 머신러닝/인공지능 접근 방식에 비해 한층 정확합니다.
- **벤더 독립적인 접근 방식을 사용**하여 최고의 벤더 및 기능 중에서 선택할 수 있습니다.
- **공통 API 사용.** 퍼블릭/프라이빗/하이브리드 클라우드 인프라 전반에서 동일합니다.

운영 Day 1 부터 확장 가능 : 사례 연구

Apstra는 Day 1부터 확장을 염두에 두고 설계되었습니다. Apstra를 사용하여 광범위한 검증을 수행하고 인상적인 결과를 얻은 다음과 같은 고객이 있습니다.

- **물리적 인프라:** 6000개 인터페이스 상태 검증, 1000개 케이블 연결 검증, 36,000개 오류 카운터; 10,000개 전력, 온도, 전압 지표 검증
- **L2/L2 데이터 플레인**: 12,000개 대기열 드롭 카운터, 수백 개 MLAG의 상태, 3000개 스패닝 트리 프로토콜(STP) 검증, 상태 변경 알림
- **컨트롤 플레인**: 1500개 BGP 세션 상태, 약 500개의 예상 다음 홉/기본 경로
- **용량 계획**: 라우팅 테이블 사용을 위해 구성된 임계값으로 약 500개 추세 분석, ARP(Address Resolution Protocol) 테이블, VRF(Virtual Routing and Forwarding)당 멀티캐스트 테이블, 6000개 링크 사용 검증
- **규제 준수**: 약 100개의 스위치 모두에서 예상 OS 버전이 실행되도록 보장

- **멀티캐스트**: 예상되는 PIM(Physical Interface Module) 인접 항목에 대한 2500개 검증, 300개 집합 지점 확인, 집합 지점에서 소스, 그룹, 소스-그룹 페어 수의 비정상적인 패턴 감지에 관한 25개 검증

기본적으로 82,000개의 항목이 있는 단일 창 대신 고객 사양에 따라 대시보드로 분류된 이상만 표시하는 다음과 같은 간단한 창이 고객에게 제공되었습니다.

- 100% 정확함(통계적으로 추론되지 않음)
- 고객의 토폴로지 및 인텐트와 지속적으로 동기화되므로 지속적인 유지 관리가 필요 없음
- 관련 조치 가능한 컨텍스트 제공(범위 일탈 항목 및 특성, 원하는 상태)
- 저장 및 처리 요구량 절약(9GB RAM, 96GB 디스크만 필요)
- 복잡하고 취약한 데이터 처리 파이프라인 작성으로 인한 내부 종속 및 레거시 위험 해소

결론

IT 인프라를 안정적으로 변경할 수 없다는 것은 성장과 혁신의 주된 장애물입니다. Apstra는 이러한 우려를 해소하고 문제 많은 "레거시 인프라"로 비롯되는 리스크를 근절하여 안정적으로 인프라를 변경함으로써 안정적으로 혁신하고 경쟁력을 유지할 수 있게 도와드립니다.

주니퍼 네트웍스에 대하여

주니퍼 네트웍스는 세상을 연결하는 제품, 솔루션, 서비스를 통해 네트워크를 간소화합니다. 주니퍼는 엔지니어링 혁신을 통해 클라우드 시대에 네트워킹의 복잡성과 제약을 없애고 고객과 파트너가 일상적으로 직면하는 가장 어려운 과제들을 해결해나가고 있습니다. 주니퍼 네트웍스는 네트워크가 세상을 변화시키는 정보와 인재의 발전을 공유하는 근간이 되는 자원이라고 믿습니다. 주니퍼는 혁신적이고 획기적인 방식으로 빠르게 변화하는 비즈니스의 속도에 맞추어 확장 가능하고 자동화를 지원하는 안전한 네트워크를 제공할 것을 약속합니다.

본사
Juniper Networks, Inc.
1133 Innovation Way
Sunnyvale, CA 94089 USA
전화: 888.JUNIPER(888.586.4737)
또는 +1.408.745.2000
www.juniper.net

한국주니퍼네트웍스
서울 강남구 테헤란로 142
아크플레이스 19층
우편번호 06236
www.kr.juniper.net
전화: 02-3483-3400

JUNIPER NETWORKS | Engineering Simplicity