



JANOG48 Meeting

NaaS に関連する技術要素と "SP NaaS" を考える

v10_pub

Shunsuke Sasaki (shusasaki@cisco.com)
Technical Solutions Architect, Cisco Systems
Jul 15, 2021



自己紹介

- ・ 名前: 佐々木俊輔
- ・ 趣味: 最近乗っていないロードバイク、旅行
- ・ Twitter: @radiantmarch
- ・ Company Blog:
 - ・ <https://gblogs.cisco.com/jp/author/shusasaki/>
- ・ JANOGer 歴: JANOG36 (2015) あたりから参加
- ・ 業務経歴
 - ・ 2008- ネットワーク製品メーカー プリセールス SE (顧客領域: DC, SP WAN, Biz VPN, NFV, ...)
 - ・ 2017- 通信事業者様向けの管理・自動化ソフトウェア製品の提案を行うアーキテクト
 - ・ 2019- 次世代のネットワークや自動化・オーケストレーションのアーキテクトの検討や提案も
 - ・ ホワイトペーパー「シスコが考える 5G 時代のエンドツーエンドアーキテクト」 第4章・6章執筆 [1][2]
 - ・ 電子情報通信学会ネットワーク 第三十回 仮想化特別研究専門委員会 招待講演[3]



- [1] <https://gblogs.cisco.com/jp/2020/11/published-two-whige-papers-about-5g/>
[2] <https://gblogs.cisco.com/jp/2020/01/service-provider-end-to-end-architecture-6-end-to-end-automation-in-5g-era-1/>
[3] <https://www.ieice.org/cs/nv/conference/>

本日本話ししたいこと (12分)

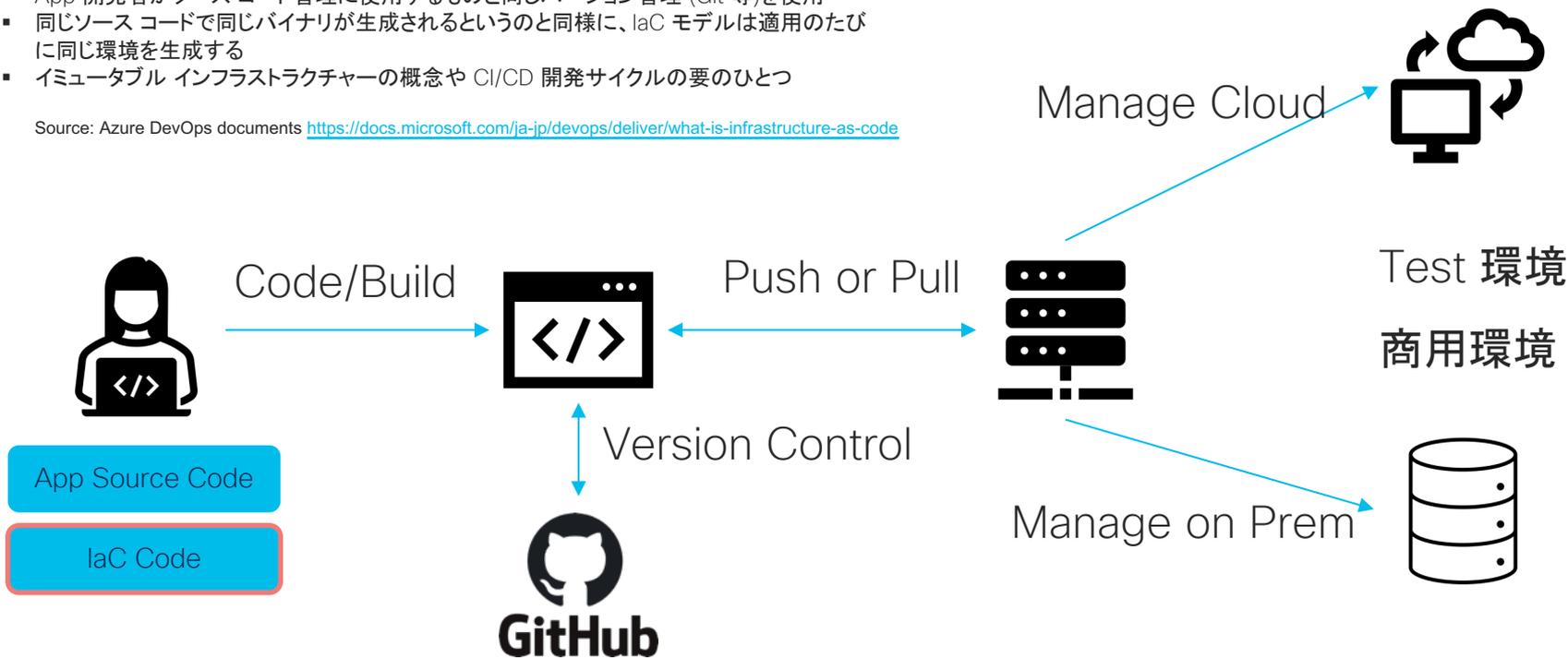
- NaaS に関連する技術要素としての [Infra as Code \(IaC\)](#)
- SP が NaaS を提供するとしたら？ 必要な技術要素の検討
 - 抽象化、サービスモデル、階層化、可視化 ([Observability](#))
- ディスカッション: "SP NaaS" サービスの実現性、課題、将来



Infrastructure as Code (IaC) とは

- 記述型モデルでのインフラストラクチャ (アプリ、コンテナ、仮想マシン、ロードバランサー、ネットワーク、接続トポロジ、etc..) の管理
- App 開発者がソースコード管理に使用するものと同じバージョン管理 (Git 等) を使用
- 同じソースコードで同じバイナリが生成されるというのと同様に、IaC モデルは適用のたびに同じ環境を生成する
- イミュータブル インフラストラクチャーの概念や CI/CD 開発サイクルの要のひとつ

Source: Azure DevOps documents <https://docs.microsoft.com/ja-jp/devops/deliver/what-is-infrastructure-as-code>



IaC ツールの一例: Terraform



- オープンソースのクラウド構成自動化ツール (HashiCorp 社による商用サポート)
- Terraform "Provider": リソースの API レイヤを抽象化するプラグイン
 - Provider を追加することで新しいインフラリソースをプロビ・管理することができる
 - 各種 CSP を始め、仮想化インフラベンダー、ネットワークベンダーなども自社製品に対する Provider を提供している



Google Cloud



vmware®



Terraform Provider Examples:

- AWS

The screenshot shows the Terraform Registry page for the AWS Provider. The page is titled "AWS Provider" and includes a navigation menu on the left with categories like "Guides", "ACM", "API Gateway", and "Access Analyzer". The main content area contains an overview of the provider, instructions on how to use it, and an "Example Usage" section with a code block showing the configuration for the AWS provider and a VPC resource.

```

terraform {
  required_providers {
    aws = {
      source = "hashicorp/aws"
      version = "~> 3.0"
    }
  }
}

# Configure the AWS Provider
provider "aws" {
  region = "us-east-1"
}

# Create a VPC
resource "aws_vpc" "example" {
  cidr_block = "10.0.0.0/16"
}
    
```

Source: <https://registry.terraform.io/providers/hashicorp/aws/latest/docs>

- Volterra

The screenshot shows the Terraform Registry page for the Volterra Provider. The page is titled "Resource volterra_tunnel" and includes a navigation menu on the left with categories like "Resources". The main content area contains an overview of the resource, a "Note" section, and an "Example Usage" section with a code block showing the configuration for the volterra_tunnel resource.

```

resource "volterra_tunnel" "example" {
  name = "acmecorp-web"
  namespace = "staging"

  local_ip {
    // One of the arguments from this list "intf_ip_address" must be set
  }

  ip_address {
    // One of the arguments from this list "ip_address auto" must be set
  }

  ipv4 {
    addr = "192.168.1.1"
  }

  virtual_network_type {
    // One of the arguments from this list "public_site_local_site_local_site_local_inside" must be set
  }

  remote_ip {
    // One of the arguments from this list "endpoints_ip" must be set
  }

  ip {
    // One of the arguments from this list "ipv4 ipv6" must be set
  }

  ipv4 {
    addr = "192.168.1.1"
  }

  tunnel_type = ["tunnel_type"]
}
    
```

Terraform コードの例 (AWS)

```

terraform {
  required_providers {
    aws = {
      source = "hashicorp/aws"
      version = "~> 3.27"
    }
  }

  required_version = ">= 0.14.9"
}

provider "aws" {
  profile = "default"
  region = "us-west-2"
}

resource "aws_instance" "app_server" {
  ami           = "ami-830c94e3"
  instance_type = "t2.micro"

  tags = {
    Name = "ExampleAppServerInst"
  }
}
    
```

main.tf

実行

```
$ terraform apply
```

An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
+ create

Terraform will perform the following actions:

```

# aws_instance.app_server will be created
+ resource "aws_instance" "app_server" {
  + ami           = "ami-830c94e3"
  + arn           = (known after apply)
  ##...
    
```

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Avail...
-	i-0fdd868a3468132cb	Pending	t2.micro	-	No alarms	ap-...

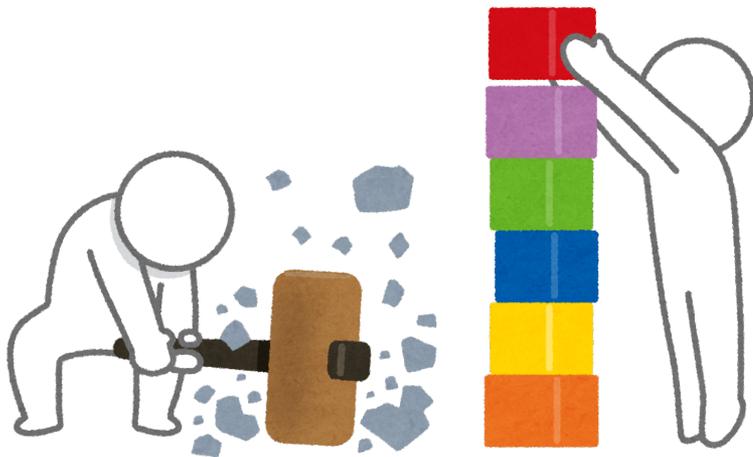
```

[10s elapsed]
[20s elapsed]
[30s elapsed]
ete after 36s [id=i-
changed, 0 destroyed.
    
```

Source: <https://learn.hashicorp.c>

ここから個人的な想像

- SP の様々なネットワークサービスを IaC コードで記述できるようになったら企業ユーザ(アプリ開発者) は嬉しい？



想像: "IaC 対応 SP" ができた場合

Terraform コードのイメージ

```

provider "SP-naas" {
  profile = "default"
  region = "tokyo"
}

resource "l3vpn" "vpn1" {
  vpn_id = "automatic"
}

resource "l3vpn" "endpoint1" {
  parent           = ${l3vpn.vpn1.id}
  pop_location     = "tokyo_pop_1"
  endpoint_type    = "physical_port"
  interface_id     = "allocated"
  interface_type   = "10G-SR"
  contract_bandwidth = "1G"
  burst_support    = "true"
}

resource "traffic-engineering" "te1" {
  headend = ${l3vpn.endpoint1.id}
  tailend = ${l3vpn.endpoint2.id}
  request_type = "low_latency"
  traffic_selection = {
    "destination": [ "10.1.1.0/24", "10.2.1.0/24" ],
  }
}
    
```

main.tf

provider を提供

L3VPN を構成

VPN エンドポイントを設定

Traffic Engineering の設定



```

resource "edge_vpc" "vpc1" {
  location = "edge_dc1"
}

...

resource "nfv" "firewall" {
  location = "edge_dc1"
  size     = "small"
  attached_endpoint = ${l3vpn.endpoint2.id}
}

...

resource "firewall_rules" "fw_rule1" {
}

...

resource "nfv" "upf" {
  location = "edge_dc1"
}

...

resource "servers" "vm1" {
  location = "edge_dc1"
}

...

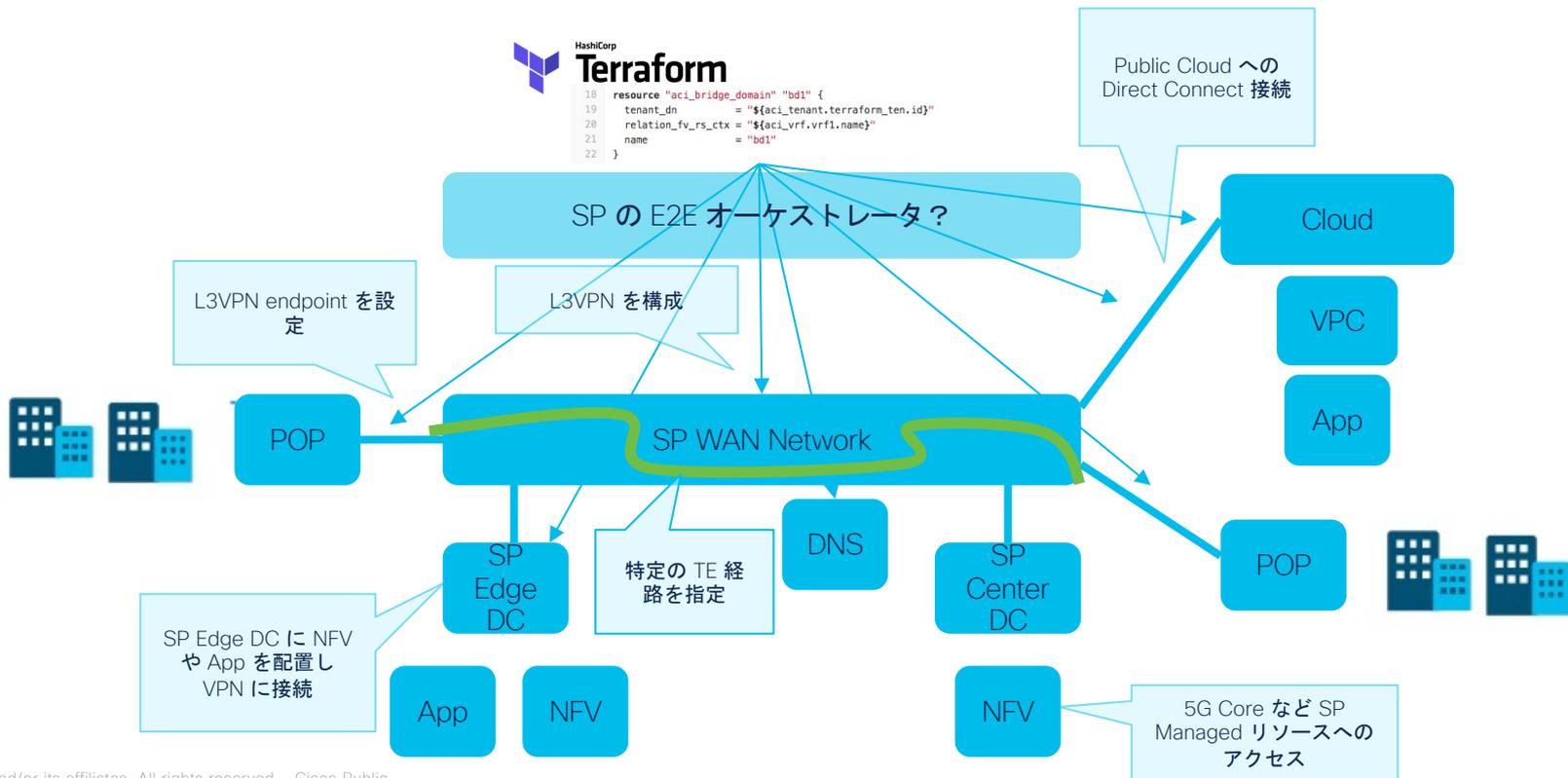
resource "dns" "dns1" {
    
```

SP エッジ DC を構成に組み込む

SP NFV サービスのデプロイ

Private 5G 向けのリソースへもアクセス

イメージ: "IaC 対応 SP NaaS サービス"



"SP NaaS" 実現のための E2E オーケストレーション技術検討要素

クロスドメイン プロビジョニング

SP 視点

See Appendix

- テクノロジ領域が RAN, DC, NFV, WAN, モバイルコアなどにまたがり一つのシステムからすべてを管理することは難しい
- 階層化コントローラによるエンドツーエンド オーケストレーションの実現
- 3GPP/5G E2E スライシングの議論と共通
- コントローラのインターフェイス標準化への期待 (3GPP/IETF)

サービスの 抽象化

企業のアプリ開発者視点

See Appendix

- IaC コードで構成を宣言的に記述する
- ネットワーク機器の設定項目を個別にAPI として公開するだけではアプリ開発者の使い勝手は悪い
- ユーザが記述しやすいようにリソースを抽象化し、内部のロジックを隠蔽する仕組みが必要
- サービスモデル / YANG ベースのサービス開発 (例)

Today

Observability (可観測性)

企業のアプリ運用者視点

- アプリのユーザ体験に影響がある性能・品質問題を検知、分析できることがニーズ
- 単なるネットワーク監視ではなく、ネットワークの性能情報をアプリケーションの性能情報と紐付けられるような仕組みが必要

トレンド: モニタリングからオブザーバビリティへ

デジタルトランスフォーメーション

開発・運用の方法論、文化

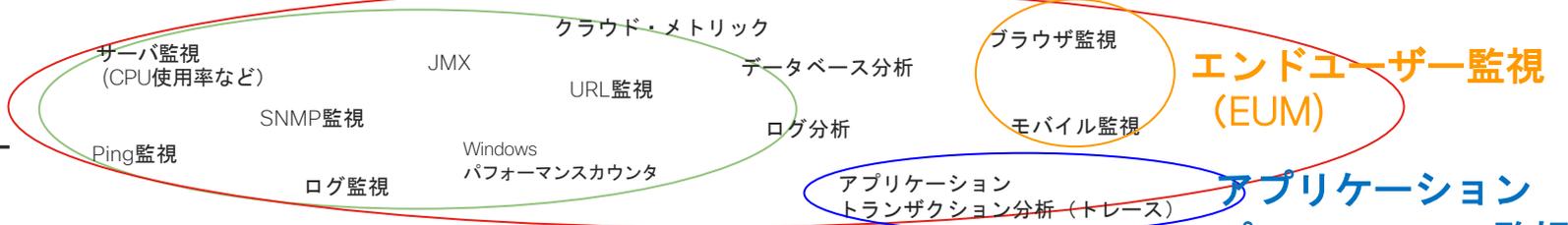
アジャイル SRE
DevOps **オブザーバビリティ**

監視の目的 **システムの可用性** 切分けと原因特定 (デバッグ) ユーザー体験可視化 **ビジネスへの貢献**

モニタリング

オブザーバビリティツール

監視
テクノロジー



ソフトウェア
テクノロジー

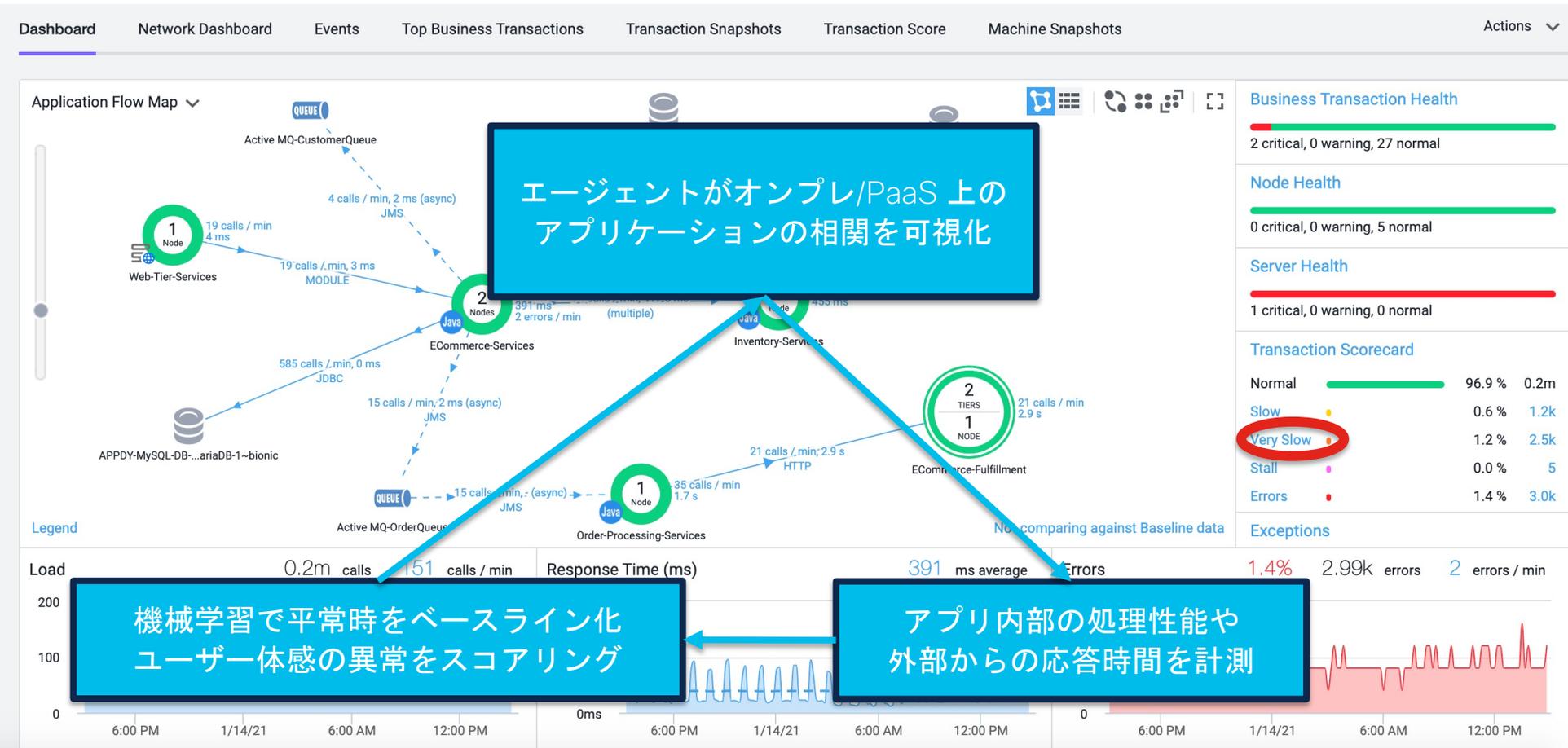


アプリケーション
アーキテクチャ



Application Performance Monitoring (APM) の例

source: AppDynamics



エージェントがオンプレ/PaaS 上のアプリケーションの相関を可視化

機械学習で平常時をベースライン化
ユーザー体感の異常をスコアリング

アプリ内部の処理性能や外部からの応答時間を計測

Application Performance Monitoring (APM) の例

source: AppDynamics

The screenshot displays the AppDynamics APM interface for an ECommerce application. A transaction with ID d7ab0b87-195f-4295-a431-51b2522235f0 is highlighted as 'User Experience: Very Slow' with an execution time of 1.4m. The interface shows a call graph starting from an HTTP call to api.shipping.com, which then calls ECommerce-Services (2 nodes), which in turn calls Inventory-Services (1 node). Inventory-Services then calls a JDBC connection to XE-Oracle-ORACL...4bit Production. A table below the call graph shows the stack trace for the slowest component, 'Spring Bean - queryExecutor.executeOracleQuery:61', which took 8,537 ms (self).

Very Slow判定されたトランザクションの一覧

Time	Exec Time (ms)	URL
01/14/21 1:12:18 PM	84,040	/app/
01/14/21 10:09:28 AM	75,262	/app/
01/14/21 3:07:58 PM	67,593	/app/
01/14/21 2:16:01 PM	67,479	/app/
01/14/21 12:08:11 PM	67,472	/app/
01/14/21 2:13:06 PM	67,461	/app/
01/14/21 1:04:02 PM	67,398	/app/
01/14/21 11:11:39 AM	67,363	/app/
01/14/21 11:13:01 AM	67,333	/app/
01/14/21 12:04:07 PM	67,317	/app/

該当トランザクションのE2Eの処理

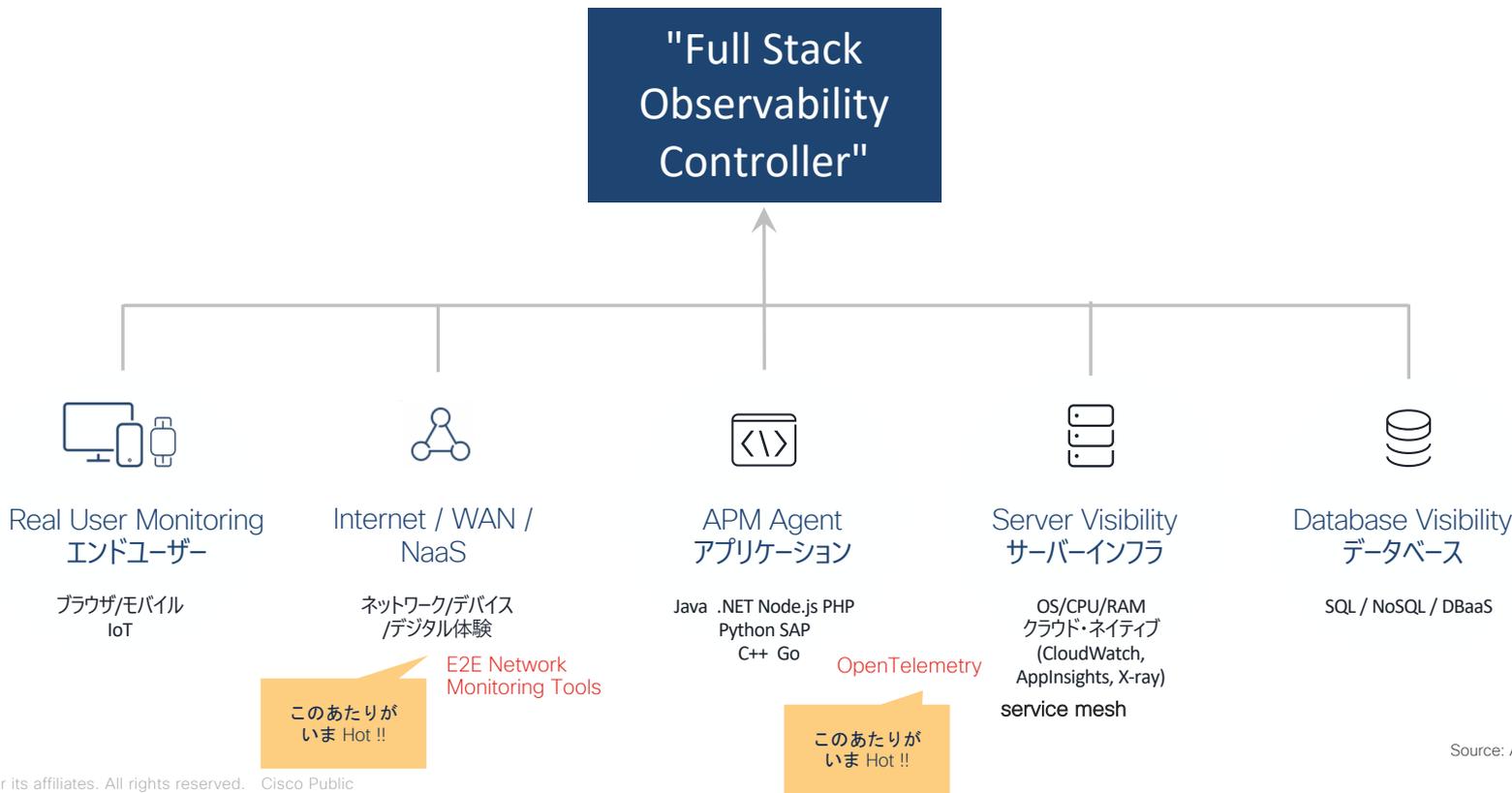
```

    graph LR
      HTTP[HTTP] -- 231ms HTTP --> ES[ECommerce-Services 2 Nodes]
      ES -- 161 ms (0.2%) 1 Call --> IS[Inventory-Services 1 Node]
      IS -- 4.5 ms (multiple) --> JDBC[JDBC]
      JDBC -- 1.4 m --> Oracle[XE-Oracle-ORACL...4bit Production]
      ES -.-> START((START))
      IS -.-> START
  
```

時間のかかった処理のスタックトレース, その性能と繋がり

Name	Execution Time	Wait Time	Block Time	Node
Servlet - jaxws-servlet.doPost	2 ms (self)	0 ms	0 ms	ECommerce_WS_NODE
Web Service - com.sun.xml.ws.server.InvokerTubeS2.invoke:149	0 ms (self)	0 ms	0 ms	ECommerce_WS_NODE
Spring Bean - orderService.createOrder:27	0 ms (self)	0 ms	0 ms	ECommerce_WS_NODE
Spring Bean - OrderDao.createOrder:59	0 ms (self)	0 ms	0 ms	ECommerce_WS_NODE
Spring Bean - queryExecutor.executeOracleQuery:61	8,537 ms (self)	0 ms	0 ms	ECommerce_WS_NODE

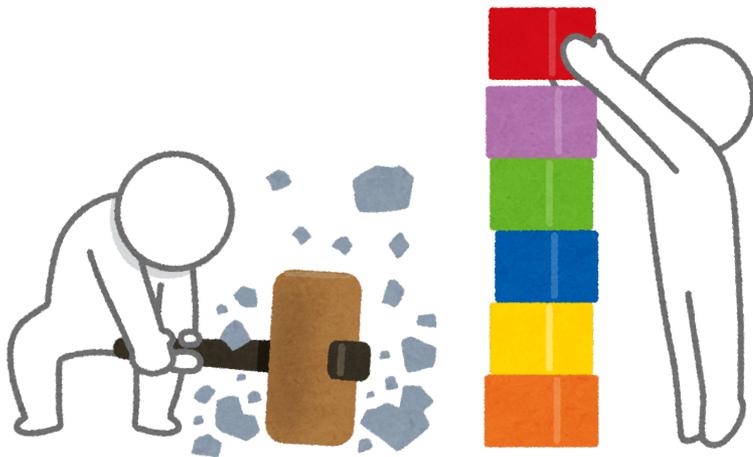
今後は "Full Stack" Observability が求められている



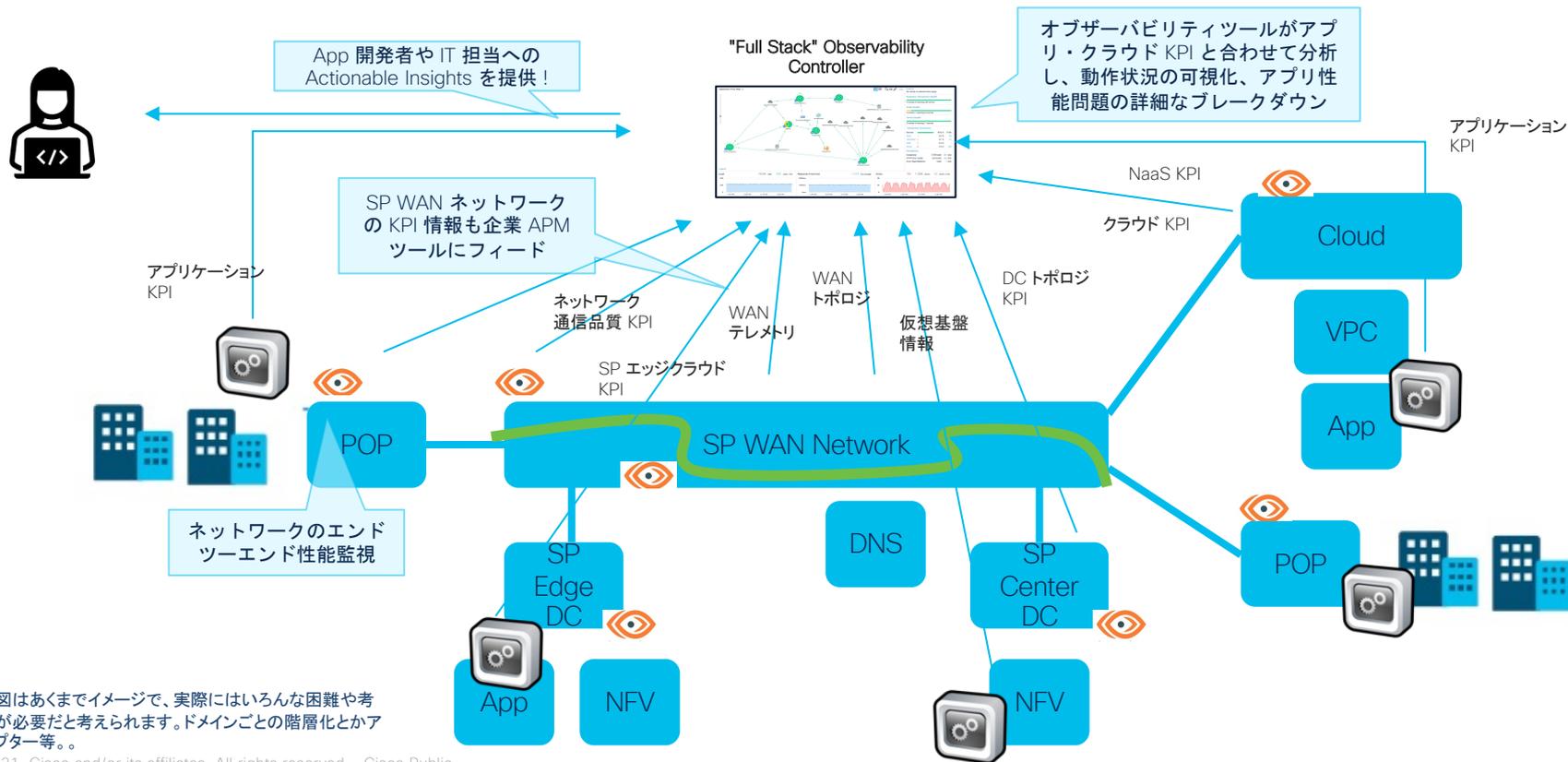
Source: AppDynamics

ふたたび個人的な想像

- SP が自身のネットワークの性能情報（テレメトリ）をオブザーバビリティツールにフィードできるようになったら、企業ユーザ(アプリ運用者) は嬉しい？



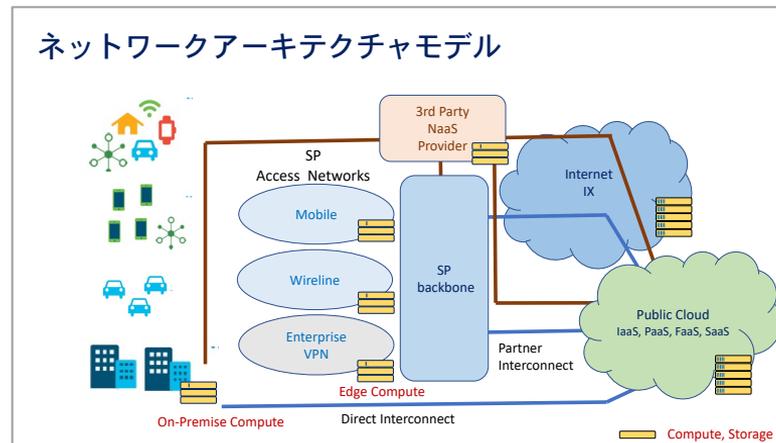
イメージ: "オブザーバビリティ対応 SP NaaS"



ここからは ディスカッション

JANOGer の皆様への問いかけ

- SP の皆様: "SP NaaS" は提供できる? 提供したい? すでに実現されてていたりする? そもそもニーズはある?
- NaaS ベンダーの皆様: NaaS 製品と SP がうまく組んでいる実例がありますか? どのようにパートナーシップを組むことが考えられますか?
- SaaS ベンダーや企業ユーザ、アプリ開発者の皆様: NaaS の利用・検討状況は? SP NaaS への期待は? Edge SP クラウドや、Private 5G/Local 5G などと絡めたユースケースなど
- Cloud SP の皆様: NaaS をどう見えていますか?
- IT 人材の視点: 高度なアプリケーション人材を SP/ネットワーク業界に呼び込み続けるには?



[河野さん JANOG 48 発表資料](#)



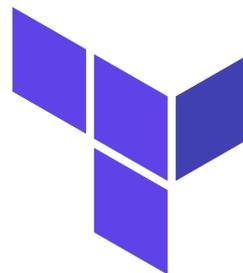
The bridge to possible

Appendix

Ansible vs Terraform



- Very much focused towards infrastructure automation
- Modules (written in Python)
- Define playbooks in YAML
- Stateless (*by default*) – push out the intent of a playbook on run
- Imperative

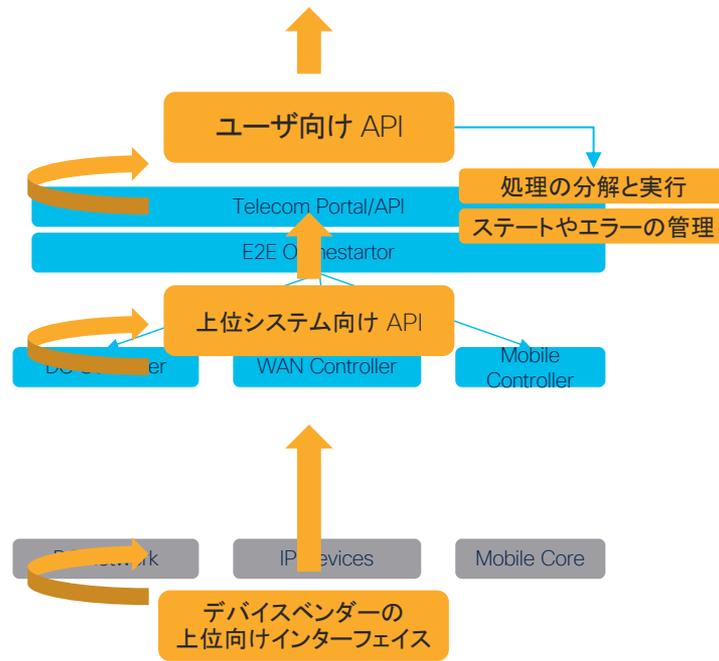


- More focused to cloud automation
- Lends itself better to API first platforms
- Providers (written in Go)
- Define a Terraform config in HCL (HashiCorp configuration language)
- Stateful – keeps a state and looks to ensure the config matches the state
- Declarative

"SP NaaS" オーケストレーション の検討事項

クロスドメイン プロビジョニング

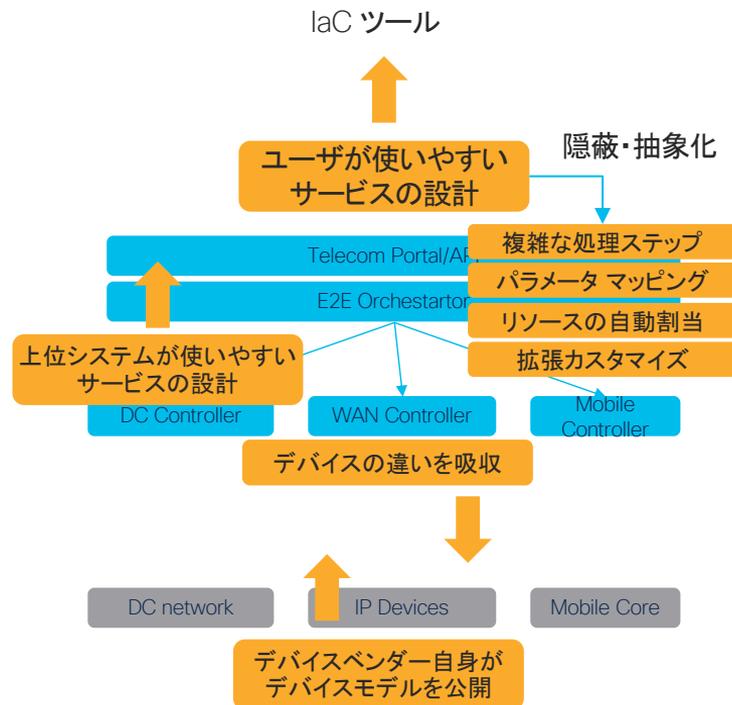
- ドメインコントローラがそのドメインを代表してプロビジョニング リクエストを受け付ける
- ドメイン内部でのクローズド ループ
- 階層的なシステムとなるがコントローラ間のインテグレーションを容易にする仕組みが必要
- 各コントローラで必要なサービスモデルをゼロから開発するのは手間であり、業界標準やコミュニティの成果をマッシュアップ（再利用・組み合わせ）できることが望ましい



"SP NaaS" オーケストレーション の検討事項

サービスの抽象化

- laC コードで「どう設定したいか (what)」をシンプルに記述できるようなサービス設計
- ネットワーク機器の設定項目をそのまま API として公開するのではなく抽象化
- ユーザ入力を具体的なデバイス設定にマッピングするためのロジックの開発
- サービス差別化のためにカスタマイズ開発が可能なプラットフォーム
- サービス設計 = ユーザ向けインターフェイス設計はセンスが問われサービスの差別化要素となる (Public CSP がそうであるように)





The bridge to possible