

[事前提出版]

Jupyter Notebook x データAIエージェントでつくる Agentic Loopなネットワーク検証プロセス

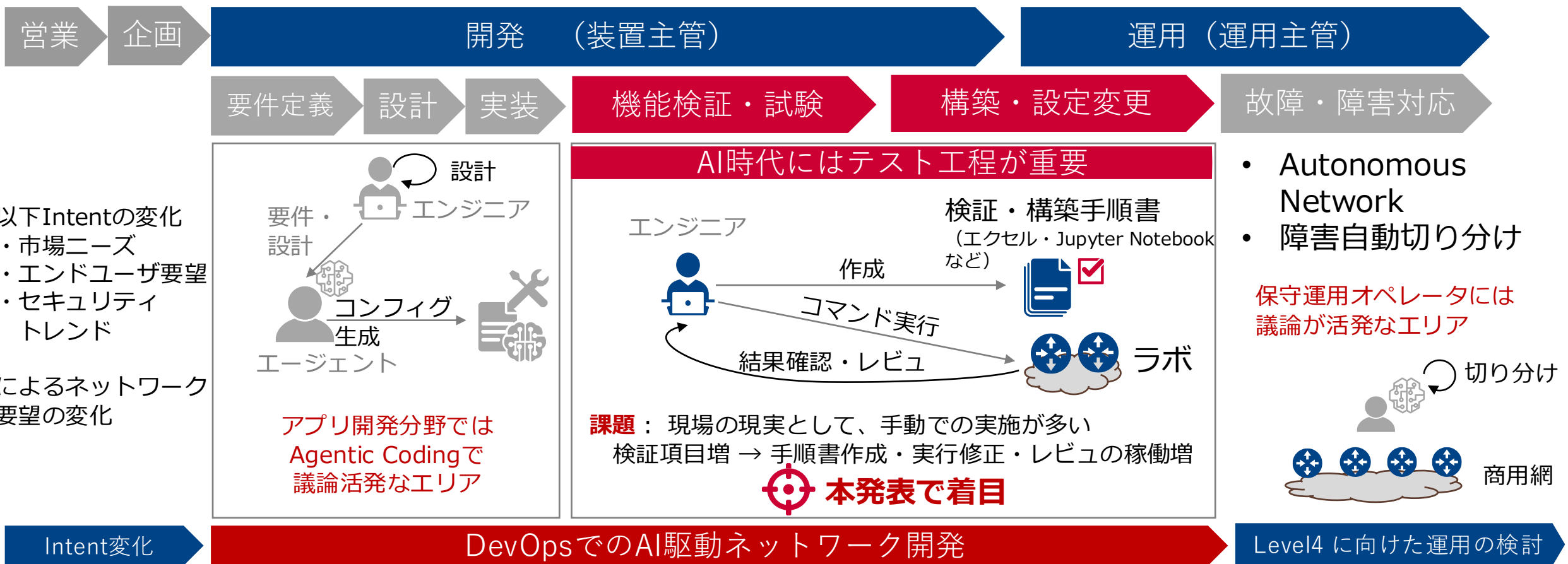


2026年6月26日
NTTドコモビジネス株式会社

AI駆動ネットワーク開発における課題

Autonomous Network Level 4のネットワーク運用に向けた議論が加速する中、マーケットニーズなどIntentの変化に対し、Analyze/Decision等ネットワーク設計変更迅速な対応が必要となり、上流のネットワーク開発工程から連携したAIエージェントによる効率化が求められる。

一方で、AI駆動の開発にはテストが欠かせず重要性が増し、検証・レビュー稼働の増加の課題に直面している。



AI駆動のネットワーク開発での工夫

検証工程における課題に対して、本発表では以下の工夫を紹介

- ① Human-in-the-loopでレビューしやすいJupyter Notebook手順書の生成AIによる試験要件ドリブンな作成
- ② Jupyter Notebook連携が得意な**データ分析用AIエージェント**を応用したAgentic Loopでの手順書改善・修正
- ③ AIエージェントによる検証用のデジタルツイン環境の生成と試験の拡充

開発 (装置主管)

運用 (運用主管)

要件定義

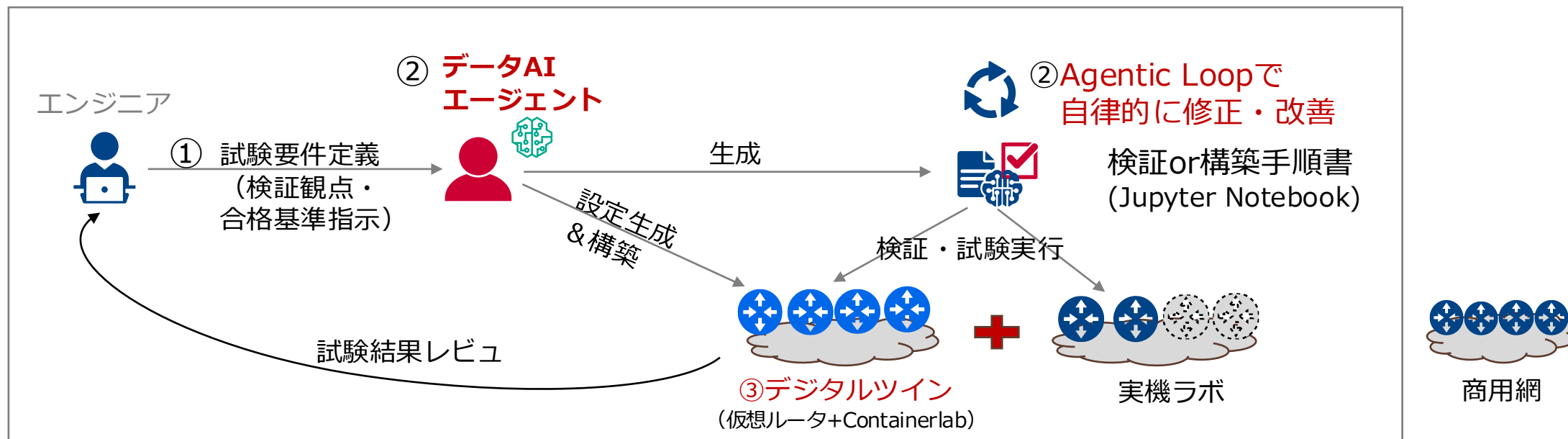
設計

実装

機能検証・試験

構築・設定変更 (の手順作成・試験)

故障・障害対応



AI駆動ネットワーク開発(要件定義～実装)の課題

ルータMCPサーバと連携したAIエージェント (参考: [NETCONをLLMで解く - JANOG57 Meeting in Osaka](#)) を活用し、現状のネットワーク構成の調査、新規ネットワーク要件に対する方式設計や詳細設計、コンフィグ生成までを実施

Session

新規セッション開始

過去セッション
session_20260622_234325_5d83...

Step Progress

- Step0 最小初期ヒアリング 承認済み
- Step1 現状構成整理と詳細ヒアリング 承認待ち
- Step2 要件定義 未開始
- Step3 基本方式設計 未開始
- Step4 詳細設計 未開始
- Step5 コンフィグ生成 未開始

Network Engineer Assistant Agent

Chat

現在の実行対象: step1 / Step1 現状構成整理と詳細ヒアリング

#現状

- 3つのエリアで太平洋と日本海側ルートの冗長経路を持つスクエア型のネットワークを組んでいる。
- ここで、MPLS-VPNのVPNテナントが2つある。
- この既存のネットワークに対して、新規要件を追加したい。#新規要件
- 2つのVPNを1エリアで相互接続を追加する
- ただし、アドレスが重複するため、双方向NATを行いたい
- NATは通信するサーバが固定で決まっているため、1:1 NATが良い

Step0 最小初期ヒアリングのArtifactを生成しました。Artifact保存先: sessions/session_20260622_234325_5d83eb92/artifacts/step0_initial_hearing_artifact.md

内容を確認し、承認または差戻してください。

es=9
es=5

120

10 25.4 PE ルータ上 (Junos の VRF間NAT機能) で
実施したい☑版で設定。サーバは1台

Chat / 差戻し理由

送信

承認して次ステップへ

差戻し

要件定義～設計～コンフィグ
生成までルータMCP+
AIエージェントで実施

課題：
AIエージェントの成果物利用では
ルータを用いたテストの重要性が増し、
手動で実施していた検証・レビュー稼働が増加
→ 本発表では検証・レビューの効率化例を紹介

Step1 現状構成整

概要・結論

Step0で確定した「MPLS-VPN×2本のスクエア型3エリア冗長網に対して、VPN間相互接続+双方向1:1 NATを1エリアに追加する」という要件を引き継ぎ、現状構成を推定図として可視化しました。設計着手には以下の●必須8項目の回答が必要です。回答受領後にStep2へ進行します。

1. 前Stepまでの概要 (Step0引継ぎ)

#	確定事項	内容
1	既存構成	3エリア (A/B/C)、太平洋・日本海側ルートによるスクエア型冗長網
2	VPNテナント	MPLS-VPN テナント×2本 (以下: VPN-α/VPN-βと仮称)
3	新規要件①	VPN-α⇄VPN-βの相互接続をいずれか1エリアに追加
4	新規要件②	両VPNのアドレスが重複→双方向 Static 1:1 NAT で解決
5	NATスコープ	通信対象サーバは固定→ホスト単位の1:1マッピング

検討内容

※ネットワーク設計変更の要件定義～コンフィグ生成のAIエージェント活用例は下記の弊社事例発表を参照。本発表では割愛
[Amazon Web Services ブログ: 【開催報告】通信事業者向けカスタム AI エージェントワークショップ開催しました! \(2026年4月23日\) - ネットワーク開発・運用を AI エージェントで変える](#)

工夫1: 試験要件ドリブンな生成AIでの手順書作成

つながり。驚きを。幸せを。



AIエージェントと試験観点や合格条件、想定差分や確認ポイントをブレストして、検証手順書を作成。
手順書を基に、ルータ操作・検証の自動実行コード入りのJupyter Notebookファイルを生成させる。

参考: ルータ作業自動化へのJupyter Notebookの活用例
[Janog44: Jupyter Notebookの拡張機能を使ってリモート機器にSSHして作業する](#)
[Janog 44: 簡単! Jupyter Notebook+Ansibleを使った作業手順書自動生成](#)
[Janog55: ネットワーク作業自動化への道: 信頼性と効率性の両立](#)

5. 12章 試験1: DHCP起動のCLI期待値

5.1 Kanto Main: show dhcp relay statistics routing-instance

コマンド

```
show dhcp relay statistics routing-instance
```

ルータコマンド

事前確認時の想定出力

```
Packets dropped:
Total 0
Due to missing interface 0
Due to missing route 0
```

```
Packets received:
BOOTREQUEST 100
DHCPDISCOVER 20
DHCPREQUEST 20

Packets forwarded:
To server 40
To client 40
```

想定コマンド出力

数値は例であり、重要なのは試験前後の増分である。

DHCP起動試験中の想定差分

NUCでDHCP renew/NIC disable-enable後、以下のようにカウンタ増加を期待する。

```
Packets received:
DHCPDISCOVER 20 -> 21以上

Packets forwarded:
To server 40 -> 41以上
```

想定結果・合格基準

確認ポイント

- DHCPDISCOVER 受信カウンタが増加すること。
- To server 転送カウンタが増加すること。
- Packets dropped が増加しないこと。

異常時との差分

```
Packets received:
DHCPDISCOVER 増加

Packets dropped:
```

検証手順書
(試験内容の意識合わせ用)

章7. 手順 6.5 DHCP server statistics 確

カウンタ増分を確認するため、baseline と after を分けて取得します。base

```
In [55]: step_id = "06_05_dhcp_server_statistics"
step_title = "章7. 手順 6.5 DHCP server statistics 確認"
cli_command = "show dhcp server statistics routing-instance"
command_md(cli_command)
```

```
show dhcp server statistics routing-instance
```

```
In [56]: # Baseline取得
baseline_cli_text, baseline_rpc_xml = collect_text_and_rpc(
print(baseline_cli_text[:2000])
```

```
<rpc-reply message-id="urn:uuid:41a301e2-5661-4f5d-a507-
<output>
Packets dropped:
Total 20
Send error 20
```

```
Delay:
DELAYED 0
INPROGRESS 0
TOTAL 0

Messages received:
DHCPDECLINE 0
DHCPDISCOVER 8
DHCPINFORM 45
DHCPRELEASE 1
DHCPREQUEST 7454
DHCPLEASEQUERY 0
DHCPBULKLEASEQUERY 0
DHCPACTIVELEASEQUERY 0
```

```
Messages sent:
BOOTREPLY 7452
DHCPPOFFER 8
DHCPACK 7444
DHCPNACK 0
DHCPFORCERENEW 0
DHCPLEASEUNASSIGNED 0
DHCPLEASEUNKNOWN 0
DHCPLEASEACTIVE 0
DHCPLEASEQUERYDONE 0
```

```
</output>
</rpc-reply>
```

結果レビュー用
CLI出力

判定結果

結果NGあり

開発時の検証手順だけでなく、
商用導入時の作業手順書にも活用可能

結果判定・検証
コード

```
In [59]: def extract_counter(text, name):
mobj = re.search(rf"{re.escape(name)}\s+(\d+)", text, re.IGNORECASE)
return int(mobj.group(1)) if mobj else None

counter_names = ["DHCPDISCOVER", "DHCPREQUEST", "DHCPPOFFER", "DHCPACK", "Total"]
rows = []
for name in counter_names:
b = extract_counter(baseline_cli_text, name)
a = extract_counter(after_cli_text, name)
delta = None if b is None or a is None else a - b
if name == "Total":
status = "PASS" if delta == 0 else ("WARN" if delta is not None
expected = "増加しない"
else:
status = "PASS" if delta is not None and delta >= 1 else "WARN"
expected = "1以上増加"
rows.append(make_result(step_id, name, f"statistics/{name}", delta,

save_text(step_dir(step_id)/"baseline_cli_text.txt", baseline_cli_text)
save_text(step_dir(step_id)/"baseline_rpc_xml.xml", baseline_rpc_xml)
save_text(step_dir(step_id)/"after_cli_text.txt", after_cli_text)
save_text(step_dir(step_id)/"after_rpc_xml.xml", after_rpc_xml)
df = display_and_record(step_id, rows, after_cli_text, after_rpc_xml)
```

	step_id	item	xml_key_or_source	actual	expected	status	message	source
0	06_05_dhcp_server_statistics	DHCPDISCOVER	statistics/DHCPDISCOVER	0	1以上増加	WARN	baseline=8, after=8	text
1	06_05_dhcp_server_statistics	DHCPREQUEST	statistics/DHCPREQUEST	0	1以上増加	WARN	baseline=7454, after=7454	text
2	06_05_dhcp_server_statistics	DHCPPOFFER	statistics/DHCPPOFFER	0	1以上増加	WARN	baseline=8, after=8	text
3	06_05_dhcp_server_statistics							
4	06_05_dhcp_server_statistics							

検証自動化コード入り
Jupyter Notebook手順書

工夫2: 手順書とデータ分析エージェントの活用

つながり。驚きを。幸せを。



クラウド各社において、Jupyter Notebookでのデータ分析エージェント開発が進められている

- [Amazon SageMaker Unified Studio の新しいワンクリックオンボーディングと組み込み AI エージェントを含むノートブック | Amazon Web Services ブログ](#)
- [データエージェントの新機能: AI ワークフローの強化 | Google Cloud 公式ブログ](#)

Jupyter Notebook全体の処理の流れを理解した対応が可能なAIエージェントで、Jupyter Notebookの解析や修正、トラブルシュートが可能。このデータ分析用の仕組みをルータの手順書修正に応用。

The screenshot shows a Jupyter Notebook environment. On the left, a notebook cell titled 'ステップ5-2: 自動化検証手順書 (Jupyter Notebook)' contains a Python script that uses pandas to process network configuration data. The script includes comments in Japanese and a function to print RPC results. Below the code, there are instructions for running the script. In the center, the execution results are displayed, showing network interface status and routing information. On the right, a Gemini AI assistant window is open, providing a detailed explanation of the code and the results, including a list of steps and a warning about an error in the previous execution.

ステップ5-2: 自動化検証手順書 (Jupyter Notebook)

本ノートブックは、[redacted] (2台)を用いたTwice NAT環境の構築 (設定投入)、およびデータプレーンの疎通・NAT正証するためのプログラムです。

- [redacted] 代わりに (dev.rpc.xxx({ 'text'})) を用いて人間レビュー用のCLIテキスト出力を取得します。
- 検証: Netconf (XML) 経由で取得した構造化データを (pandas.DataFrame) で表形式表示します。

```
!pip install python-dotenv junos-eznc xmlltodict pandas lxml
```

```
import os
```

```
# DataFrameの表示設定
```

```
pd.set_option('display.max_columns', None)
```

```
pd.set_option('display.max_colwidth', None)
```

```
def print_rpc_text(rpc_elem):
```

```
    """
```

```
    RPCの結果(format: 'text')として返されるXML Elementから
```

```
    テキスト部分を抽出して出力するヘルパー関数
```

```
    """
```

```
    if rpc_elem is not None and rpc_elem.text:
```

```
        print(rpc_elem.text.strip())
```

```
    else:
```

```
        print(etree.tostring(rpc_elem, encoding='unicode'))
```

手順書の実行結果 (ルータコマンド 実行結果)

Interface	Admin	Link	Proto	Local	Remote
si-0/0/0	up	up			
si-0/0/0.1	up	up	ip		
si-0/0/0.2	up	up	ip		

自動化コード 入りの手順書

手順書と実行結果に対する AIエージェントによる解説

1. CLI出力の取得: (get_inline_nat_statistics) および (get_nat_pool_information) というRPC (リモートプロシージャコール) を使って、人間が目視確認しやすいテキスト形式 (CLIでコマンドを打った時と同じ出力) で情報を取得・表示しようとしています。
2. データの自動検証: もう一度XML形式で統計データを取得し、(fpc-slot), (pic-slot), (nat-hits) (NATの変換ヒット数) という項目を抽出して、表形式 (DataFrame) で表示します。
3. ヒット数の確認: (nat-hits) (変換されたパケット数) が0より大きいかを確認し、ハードウェアでNAT処理が行われているかを自動判定します。

ただし、直前の実行結果を見るとエラーになっています。(bad_element: get_inline_nat_statistics) というエラーが出ているため、今回使用しているvMX (仮想ルーター) のバージョンでは、この特定のRPCコマンド (show services inline nat statistics) がサポートされていないか、別のコマンドを使う必要があるようです。

工夫2のデモ動画

当日の発表時にデモ動画投映予定
(掲載用に以降のスライドにて静止画版を記載)

工夫2 : Agentic Loopな検証プロセス(1/4)

データ分析AIエージェントの仕組みをルータ検証に応用し、
Agentic Loopなルータの検証プロセスを実現。試験手順書や設定導入手順書の効率的な修正や改善

The screenshot shows a development environment with a code editor on the left and a Gemini chat window on the right. The code editor displays a Python script for verifying NAT statistics on a Netconf device. The script includes a function `verify_nat_statistics()` and a `def investigate_nat_rpc()` function. The output shows an error: `[ERROR] NAT統計検証失敗: RpcError(severity: error, bad_element: get-inline-nat-statistics, message: syntax error)`. The Gemini chat window shows the AI agent's response to the error, providing a list of possible causes and suggesting alternative commands.

①手順書実行でルータからのデータ取得に失敗

```
print(f"[INFO] Netconfでの統計取得スキーマが一致しませんでした。CLI結果を目視確認してください。: {rpc_e}")

except Exception as e:
    print(f"[ERROR] NAT統計検証失敗: {e}")

verify_nat_statistics()

[CLIレビュー用] Inline NAT 統計情報
[ERROR] NAT統計検証失敗: RpcError(severity: error, bad_element: get-inline-nat-statistics, message: syntax error)
```

②エラーの原因調査を指示

エラーを元に原因調査して。

エラー `bad_element: get-inline-nat-statistics` の原因を調査するため、機器 (vMX) に対して直接CLIコマンドのヘルプやRPC構造を問い合わせるコードを作成します。

これにより、以下のいずれが原因か特定できます。

- そもそも `show services inline nat statistics` コマンドがこの環境でサポートされていない。
- コマンドに対応する正しいRPCタグ名が `get-inline-nat-statistics` ではない。

以下のセルを実行して、結果を確認させてください。

処理しています...

③エージェントが原因を2つ推測

```
def investigate_nat_rpc():
    try:
        with Device(host=IP, port=NETCONF_PORT, ...) as dev:
            print("1. 'show services inline nat statistics' のRPC名を調査します")
            try:
                # <command> を使ってCLIコマンドを実行し、出力を取得する
                res = dev.execute(etree.XML('<command>show services inline nat statistics | display rpc</command>'))
                print("[実行結果]")
                print(res.text if res is not None else "出力なし (コマンドが不正な可能性)")
            except Exception as e:
                print(f"[エラー]: {e}")

            print("\n2. 代替コマンド 'show services nat statistics' の確認")
            try:
                res = dev.execute(etree.XML('<command>show services nat statistics | display rpc</command>'))
                print("[実行結果]")
                print(res.text if res is not None else "出力なし")
            except Exception as e:
                print(f"[エラー]: {e}")

            print("\n3. 参考: 'show services nat pool' の確認")
            try:
                res = dev.execute(etree.XML('<command>show services nat pool</command>'))
                print("[実行結果]")
                print(res.text if res is not None else "出力なし")
            except Exception as e:
                print(f"[エラー]: {e}")
```

何を構築しますか?

Gemini 3.1 Pro

Gemini は不正確な情報を表示することがあるため、内容を再確認し、コードは注意して使用してください。

工夫2 : Agentic Loopな検証プロセス(2/4)

クラウド各社が実装を進めるJupyter Notebookのビルトインのデータ分析AIエージェントの仕組みをルータの検証や導入手順書のコード修正や改善に応用

The screenshot shows a Jupyter Notebook environment. The left pane contains Python code for interacting with a network device via Netconf. The code includes functions to execute commands, handle errors, and log results. The right pane shows a Gemini AI chat interface with a user query and a detailed response explaining a syntax error in the code and providing a corrected version.

```

with Device(host=IP, port=NETCONF_PORT, ...) as dev:
    print("1. 'show services inline nat statistics' のRPC名を調査します")
    print("1. 'show services inline nat statistics' をパイプなしで実行します")
    try:
        # <command> を使ってCLIコマンドを実行し、出力を取得する
        res = dev.execute(etree.XML('<command>show services inline nat statistics | display rpc</command>'))
        res = dev.execute(etree.XML('<command>show services inline nat statistics</command>'))
        print("[実行結果]")
        print(res.text if res is not None else "出力なし (コマンドが不正な可能性)")
        print(res.text.strip() if res is not None and res.text else "{出力なし / 該当データなし}")
    except Exception as e:
        print(f"[エラー]: {e}")

    print("\n2. 代替コマンド 'show services nat statistics' の存在を調査します")
    print("\n2. 'show services nat statistics' をパイプなしで実行します")
    try:
        res = dev.execute(etree.XML('<command>show services nat statistics | display rpc</command>'))
        res = dev.execute(etree.XML('<command>show services nat statistics</command>'))
        print("[実行結果]")
        print(res.text if res is not None else "出力なし")
        print(res.text.strip() if res is not None and res.text else "{出力なし / 該当データなし}")
    except Exception as e:
        print(f"[エラー]: {e}")

    print("\n3. 参考: 'show services nat pool' の確認")
    print("\n3. 'show services nat pool' を実行します")
    try:
        res = dev.execute(etree.XML('<command>show services nat pool</command>'))
        print("[実行結果]")
        print(res.text if res is not None else "出力なし")
        print(res.text.strip() if res is not None and res.text else "{出力なし / 該当データなし}")
    except Exception as e:
        print(f"[エラー]: {e}")

```

⑤切り分けコードを自律的に実行、原因を推察

Gemini

エラーを元に原因調査して。

エラー `bad_element: get-inline-nat-statistics` の原因を調査するため、機器 (vMX) に対して直接CLIコマンドのヘルプやRPC構造を問い合わせるコードを作成します。

これにより、以下のいずれが原因か特定できます。

- そもそも `show services inline nat statistics` コマンドがこの環境でサポートされていない。
- コマンドに対応する正しいRPCタグ名が `get-inline-nat-statistics` ではない。

以下のセルを実行して、結果を確認させてください。

エラーの原因がわかりました。

JunosのNetconf (XML API) 経由で `<command>` タグを使ってコマンドを送る際、`| display rpc` のようなパイプ (|) 処理を含めることは仕様上許可されておらず、構文エラー (syntax error) になります。

そのため、パイプを使わずに直接コマンドのみを実行し、このvMX機器上でそもそもこれらのコマンドがどのように認識されるか (サポートされているか) を確認するようにコードを修正します。以下の修正内容で再度実行させてください。

⑥AIエージェントが切り分けコードを改善。ルータからデータ取得を試行錯誤して原因調査

1. 'show services inline nat statistics' のRPC名を調査します
[エラー]: RpcError(severity: error, bad_element: rpc, message: syntax error, expecting <command>)

2. 代替コマンド 'show services nat statistics' の存在を調査します

切り分け Agentic Loop1

切り分け Agentic Loop2

工夫2 : Agentic Loopな検証プロセス(3/4)

つながる。驚きを。幸せを。

The screenshot shows a code editor with Python code for verifying NAT statistics. The code is divided into sections: initial setup and printing, a PyEZ RPC call to get inline NAT statistics, and a PyEZ RPC call to get NAT pool information. A comment indicates a fix for a PyEZ RPC mapping error by using the <command> tag. The code then prints the results and performs a final verification using the <command> tag. The Gemini chat window on the right provides a detailed explanation of the error, identifying the issue as a 'bad_element' error caused by the router's XML output format. It suggests using the <command> tag to bypass PyEZ's XML parsing and directly execute the Netconf command. A confirmation dialog is visible at the bottom of the chat window.

```
def verify_nat_statistics():
    try:
        with Device(host=... as dev:
            print("=====")
            print(" [CLIレビュー用] Inline NAT 統計情報")
            print("=====")
            txt_stats = dev.rpc.get_inline_nat_statistics({'format': 'text'})
            print_rpc_text(txt_stats)

            txt_pool = dev.rpc.get_nat_pool_information({'format': 'text'})
            if txt_pool is not None and txt_pool.text:
                print("\n[NAT Pool 情報]")
                print_rpc_text(txt_pool)

            # PyEZのRPCマッピングエラーを回避するため、<command>で明示的に発行
            cmd_txt = etree.XML('<command format="text">show services inline nat statistics</command>')
            txt_stats = dev.execute(cmd_txt)
            if txt_stats is not None and txt_stats.text:
                print(txt_stats.text.strip())

            print("=====")
            print(" [Netconfデータ検証] NAT統計")
            print("=====")

        try:
            nat_stats = dev.rpc.get_inline_nat_statistics()

            fpc = nat_stats.findtext('./fpc-slot') or "0"
            pic = nat_stats.findtext('./pic-slot') or "0"
            hits = nat_stats.findtext('./nat-hits') or "0"
            cmd_xml = etree.XML('<command>show services inline nat statistics</command>')
            nat_stats = dev.execute(cmd_xml)

            # 取得したXML構造から該当データを抽出 (空白や改行を取り除く)
            pic_name = (nat_stats.findtext('./srv-pic-name') or "").strip()
            nat_pkts = (nat_stats.findtext('./si-nat-ipv4-pkts') or "0").strip()
            denat_pkts = (nat_stats.findtext('./si-denat-ipv4-pkts') or "0").strip()

            df_stats = pd.DataFrame([{"FPC": fpc, "PIC": pic, "NAT Hits": hits}])
            total_hits = int(nat_pkts) + int(denat_pkts)
```

⑦原因を特定して、元の手順のルータデータ取得のPythonコードを修正

切り分け Agentic Loop3

切り分け Agentic Loop4

工夫2 : Agentic Loopな検証プロセス(4/4)

修正前

1. 'show services inline nat statistics' のRPC名を調査します
[エラー]: RpcError(severity: error, bad_element: rpc, message: syntax error, expecting <command>)
2. 代替コマンド 'show services nat statistics' の存在を調査します

修正後

The screenshot shows a development environment with a Python script and its output. The script is as follows:

```
print(f"[INFO] Netconfでの統計取得スキーマ解析: {rpc_e}")  
except Exception as e:  
    print(f"[ERROR] NAT統計検証失敗: {e}")  
verify_nat_statistics()
```

The output shows the following NAT statistics table:

[CLIレビュー用] Inline NAT 統計情報			
Service	PIC	Name	
Control Plane Statistics			
Received IPv4 packets			0
ICMPv4 error packets pass through			0
ICMPv4 error packets locally generate			0
Dropped IPv4 packets			0
Received IPv6 packets			0
ICMPv6 error packets pass through for NPTv6			0
ICMPv6 error packets locally generated for NPTv6			0
Dropped IPv6 packets			0
Data Plane Statistics	Packets	Bytes	
IPv4 NATed packets	0	0	
IPv4 deNATed packets	0	0	
IPv4 error packets	0	0	
IPv4 skipped packets	0	0	
IPv6 NATed packets	0	0	
IPv6 deNATed packets	0	0	
IPv6 error packets	0	0	
IPv6 skipped packets	0	0	

Below the table is a summary table:

Interface	Source NAT Packets	Dest NAT Packets	Total NAT Hits
0 si-0/0/0	0	0	0

[!] 警告: NATヒット数が0です。Pingを実行していないか、トラフィックがNATエンジンを通していません。

The callout box contains the following text:

⑧失敗していた手順書のPythonコードの修正が完了して、正常実行

The Gemini chat window on the right contains the following text:

原因がはっきりと分かりました！
ルーターは show services inline nat statistics の結果として、ちゃんとNATの統計情報 (si-nat-ipv4-pkts などのXMLタグ) を返しています。しかし、PyEZライブラリが内部で用意している get_inline_nat_statistics() という関数が、このルーターのバージョン (vMX 25.4) と相性が悪く、存在しないRPCタグを送ってしまっていたために bad_element エラーが発生していました。
解決策: PyEZの関数に頼らず、先ほどの調査コードと同様に <command> タグを使って直接 Netconfコマンドを発行し、その結果のXMLを解析するようにメインの検証コード (セル 21SRN25e9CkV) を修正します。
また、先ほど取得したXMLの構造を見ると、パケット数は <si-nat-ipv4-pkts> (Source NAT) と <si-denat-ipv4-pkts> (Destination NAT) というタグで出力されることが分かったため、これを抽出してDataFrameに表示するように直します。
エラーが完全に解消され、無事にNATの統計情報が取得できました！
出力結果を見ると、PyEZのエラーに邪魔されることなく、CLI形式でのテキスト出力と、XMLから抽出した表形式 (DataFrame) の出力の両方が正常に行われています。
現在、表の中の Total NAT Hits (NAT変換されたパケット数) が 0 となっており、「警告: NATヒット数が0です...」というメッセージが表示されています。これはプログラムのエラーではなく、「まだルーターのNAT機能を使って通信 (Pingなど) が行われていない」ことを示しています。
次のステップ: 一つ上の手順である「3. 正常動作の検証: Ping疎通確認 (Twice NATの通過)」のセル (verify_ping_connectivity()) を実行して顧客側機器からPingトラフィックを発生させてください。その後、もう一度この NAT統計確認セルを実行すると、パケット数がカウントアップされ、NATが正常に機能していることが確認できるはずです！

工夫3: 仮想ルータによる試験拡充

実機ラボ試験に加えて、仮想ルータ + Containerlabの設定 + 構築手順書を生成させ、デジタルツインを構築
網全体の事前迂回・経路確認手順など、模擬範囲が限られる実機ラボで実施が難しい試験を補完

Proceeded with Implementation Plan

すべてのタスクの実行とドキュメンテーションの更新が正常に完了しました。重複アドレス空間 (10.100.1.10/32) を持つサービスAとサービスB間の双方向・冗長化NATシステム設計、仮想検証環境、および自動検証環境を完備し、ユーザーへの引き渡し準備が整いました。

プロジェクト全体の進捗状況は task.md にすべて完了 ([x]) として整理し、最終検証の成果報告および操作手順書を walkthrough.md に取りまとめました。

最終成果物の概要とファイル配置

- 新規要件に対するネットワーク詳細設計およびルータコンフィグ (Junos 25.4)
 - juniper_detailed.
 - 構成: PE-1 (太平洋)
 - NAT/ルーティング:
- Containerlab 仮想検証環境定義
 - containerlab_topology.clab.yml (Step 6)
 - 構成: vr-pan/juniper_vjunosrouter:latest イメージを用いた 4 ノードの最小検証トポロジー定義。Linuxの eth ポートと Junos 論理インターフェイス (ge-0/0/0 ~ 2) を厳密にマッピング。
- 手動検証手順書 (Manual Verification)
 - verification_manual.md (Step 7)
 - 内容: VAL-01 (仮想IF活性化) から VAL-06 (PE障害切替) まで、6つの検証シナリオを定義。
 - 構成: 実行 show コマンド、想定テキスト出力、NETCONF XMLブロック、およびチェック箇所のポイント。
- 自動検証用 Jupyter Notebook (PyEZ & NETCONF)
 - juniper_nat_verification.ipynb (Step 8)
 - 特徴: Pythonの junos-eznc および ncclient を用いて、実機 (PE/CE) から直接ステータス (サービスIF、

Artifact

```

yml
name: juniper_vpn_nat_lab

topology:
  nodes:
    PE-1:
      kind: vr-pan/juniper_vjunosrouter
      image: vr-pan/juniper_vjunosrouter:latest
      mgmt-ipv4: 172.20.20.11
    PE-2:
      kind: vr-pan/juniper_vjunosrouter
      image: vr-pan/juniper_vjunosrouter:latest
      mgmt-ipv4: 172.20.20.12
    CE-A:
      kind: vr-pan/juniper_vjunosrouter
      image: vr-pan/juniper_vjunosrouter:latest
      mgmt-ipv4: 172.20.20.21
    CE-B:
      kind: vr-pan/juniper_vjunosrouter
      image: vr-pan/juniper_vjunosrouter:latest
      mgmt-ipv4: 172.20.20.22

  links:
    # -----
    # PE-1 ⇄ PE-2 コアバックボーン (ge-0/0/0 ⇄ ge-0/0/0)
    # -----
    - endpoints: ["PE-1:eth1", "PE-2:eth1"]
    # -----
    # CE-A ⇄ PE デュアルホーム接続
    # CE-A:ge-0/0/1 ⇄ PE-1:ge-0/0/1
    # CE-A:ge-0/0/2 ⇄ PE-2:ge-0/0/1
    # -----
    - endpoints: ["CE-A:eth1", "PE-1:eth1"]
    - endpoints: ["CE-A:eth2", "PE-2:eth1"]
    # -----
    # CE-B ⇄ PE デュアルホーム接続
    # CE-B:ge-0/0/1 ⇄ PE-1:ge-0/0/2
    # CE-B:ge-0/0/2 ⇄ PE-2:ge-0/0/2
    # -----
    - endpoints: ["CE-B:eth1", "PE-1:eth3"]
    - endpoints: ["CE-B:eth2", "PE-2:eth3"]
  
```

Containerlabの仮想ルータ環境設定生成

Containerlabの仮想ルータ環境設定

仮想ルータ環境で補う試験範囲の手順書生成

ラボの実機と仮想ルータを組み合わせることで試験両者の得意分野に応じた試験の割り振りが重要

つなごう。驚きを。幸せを。

 ^{NTT} docomo **Business**