DNSのオブザーバビリティ向上開発でのAI活用

NTT東日本株式会社
先端テクノロジー部　ネットワーク技術部門 伊藤 雅子

発表する人 自己紹介（伊藤）

●部署：
NTT東日本
先端テクノロジー部
ネットワーク技術部門　ネットワーク制御技術担当

●お仕事：
DNSサーバー、RADIUSサーバーの開発/検証/導入

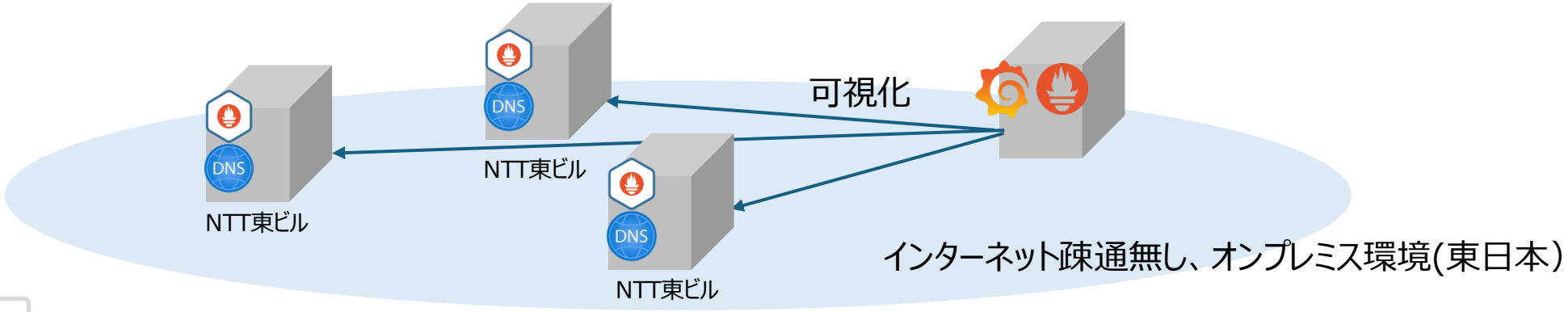
●名前：
伊藤　雅子

●プライベート(趣味など）：
子供（2人）とカブトムシの飼育、ポケ〇ン図鑑を読む、ドラ〇〇ボール鑑賞
Claudeで育児支援ゲームを作る

DNSのオブザーバビリティ開発とは、どんな開発をしているのか？

- これまでのDNSの保守では障害調査のたびに複数拠点のDNSサーバへログインして確認していた
- つらいので、ダッシュボードで一目で状態確認をできるようにしたい
- そのため、Prometheus ＋ Grafanaで、DNSのクエリ情報およびCPU使用率などの統計情報をダッシュボード上で**可視化する機能部(オブザーバビリティ機能部)を新規に開発**している。

メトリクス情報の流れ



開発個所

DNSのオブザーバビリティ開発でどんな課題感があったのか？

- 数年、既存システムの更改開発が中心のメンバーで、新規機能開発に取り組むことに
- **複数メンバー**での開発において**品質を一定以上**にするためには、**仕様書の作成は不可欠**
- 将来の維持管理を見据え、**Grafanaダッシュボードのコード化にも取り組みたかった**

⇒ **仕様整理とコード化、2つの課題が見えてきた**

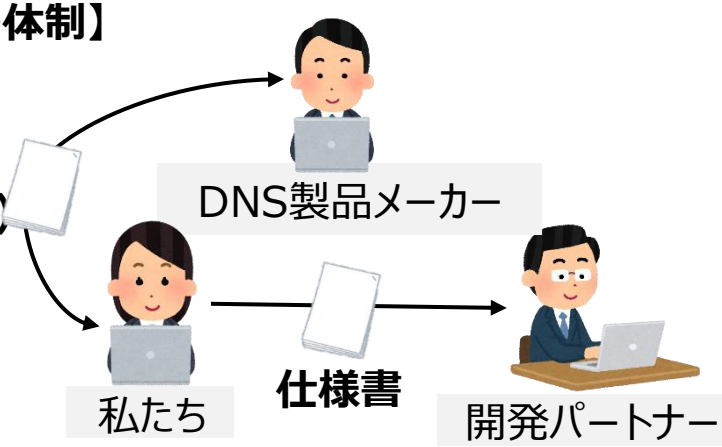
【本開発の流れ】

NTT東にて開発パートナー向けの要求仕様の作成 → 開発パートナーが要求仕様書をもとに開発 → 開発パートナーと協力して検証⇒修正を繰り返す → 商用環境へリリース

私たちシステム開発チームの領域

課題

【開発時の体制】



■新規機能開発を進める中で見えてきた**2つの課題**

課題①：仕様書をつくる大変さ

新規機能開発では、関係者間の広い範囲の**仕様整理が必要**だった。
また、検討内容はQA表に蓄積されていたが、必要な内容を抽出し、**仕様書に落とし込む作業が膨大**。

課題②：コード化したいがイメージが伝わらない

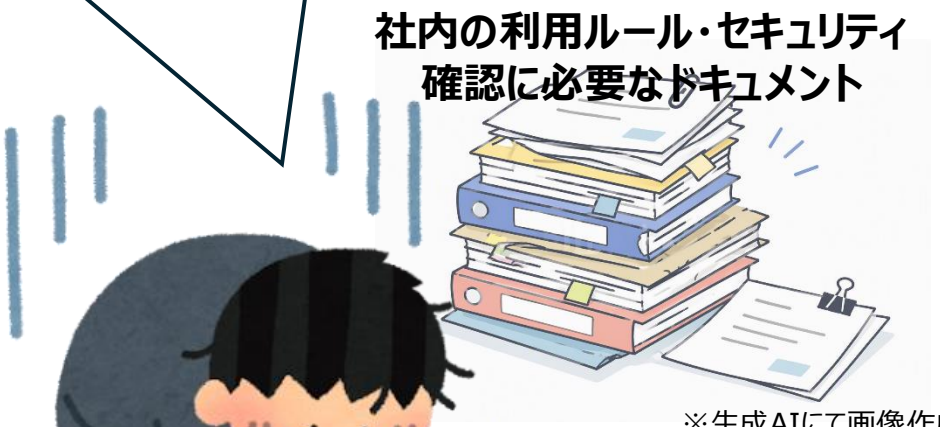
開発コスト・維持管理コストから、ダッシュボードはコード化(※)したいが開発パートナーに**知見がなく首を縦に振ってくれない・・**

※. GUI操作ではなく、Grafanaが提供しているソフトウェア開発キット (Grafana Foundation SDK)を用いたダッシュボードの開発

世の中には「Claude Code」とか「Codex」
とか賢いAIがあるらしい・・
これで解決するぞ～！！

しかし現実は
甘くなかった・・

クラウドサービス、AIエージェントを
開発検証環境に導入するには
時間がかかりそう・・・



※生成AIにて画像作成

でも

Copilot WebUI
ならある

課題①を解決するために実施したこと(Copilot WebUIの活用)

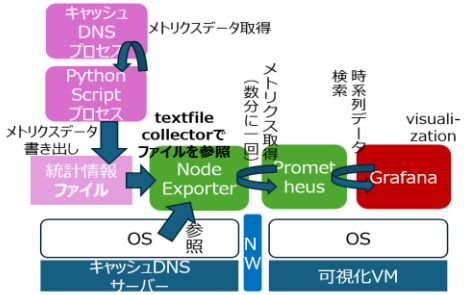
- 社内でCopilot WebUI(WebUIでOpusやChatGPT5.5の利用は可能）は利用可能
- DNS製品メーカーとのQA表では可視化の実装や要件について相談している（これを要求仕様書に盛り込めばよさそう)

⇒QA表をCopilot WebUI（LLMはOpusを利用）に読み込ませて仕様書のたたき台を作成(できた！)

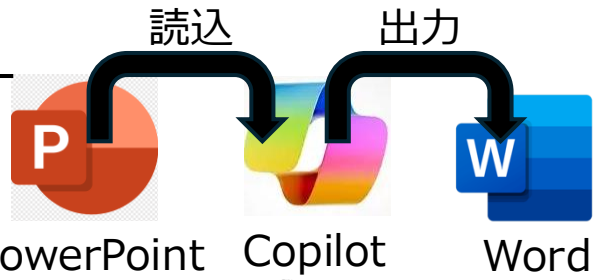
⇒仕様書のたたき台（骨子）をベースにチームでブラッシュアップ

No	分類	ご質問内容	記載者	ご回答内容	記載者
1	共通	DNS製品について、他社製品と比較した時の御社の強みを教えていただけないでしょうか。	NTT東	別途ご案内申し上げます。	XXX
2	共通	脆弱性情報についてはどのように取得すればよろしいでしょうか。 コミュニティに参加することで情報取得は可能なのでしょうか。	NTT東	ソフトウェアの脆弱性はベンダサポート・ポータル、及び、ベンダからお客様ダイレクトにメール通知されます。	XXX
3	共通	統計情報（現在どんなクエリがどれくらい来ているのか等）等は見られますでしょうか。 もしみられる場合は、具体的にどのような内容を見ることができるのかご教示いただけますでしょうか。別ページに出力する統計情報と全体イメージを記載しております。	NTT東	キャッシュ、権威共に 専用コマンドにて統計情報(クエリタイプ等)を取得可能です。 ＊ 積算値での出力となりますので、前回実行時の値と差分を計算する必要があります。 出力例については、QA12_統計情報例.txt 参照	XXX

【アーキテクチャー案】

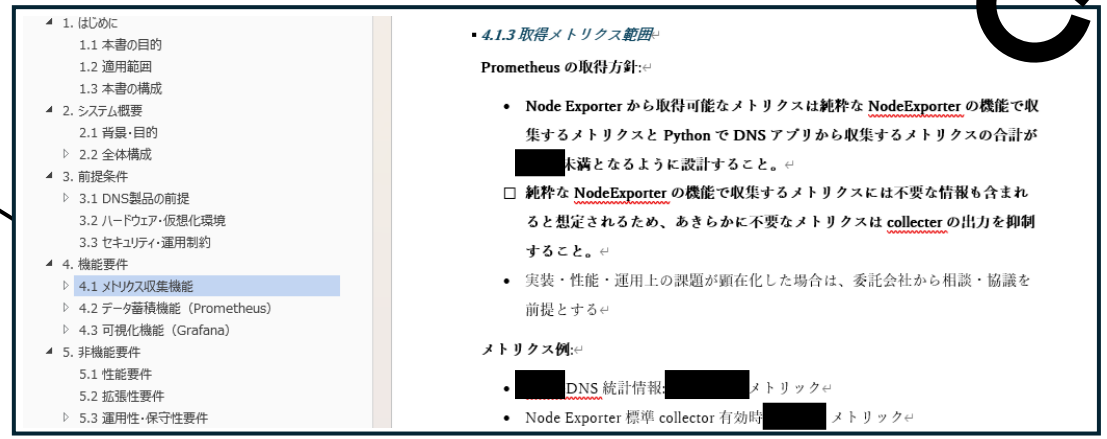


＜ QA表＞



記載する内容は
メンバーでチェックして
ブラッシュアップ

＜可視化機能部の要求仕様書＞



課題②を解決するために実施したこと(Copilot WebUIの活用)

- Grafana Foundation SDK未経験者であっても構築の流れや**具体的なコードのイメージ**を提示することで「想像できない」 ⇒ 「イメージが湧いた」という状態になって頂くことで、Grafana Foundation SDKを使って開発・修正・検証を実施する方向で了承いただくことができた

＜可視化機能部の要求仕様書＞

- 10.5 Grafana SDK 設定例

- ***10.5.1 本節の位置づけ***

本節では、Grafana Foundation SDK（Dashboard as Code）を用いて、Grafana ダッシュボード定義をコード化する場合の設定例を示す。

本設定例は、Grafana 上で手作業によりダッシュボードを作成するのではなく、Go 言語で書かれたコードから Grafana Dashboard を生成し、Ansible 等で登録・配布する方式を想定した参考実装例である。

初期構築時点で想定するダッシュボードは、(A) Overview（全体サマリ）、(B) QTYPE 別、(C) RCODE 別、(D) CPU／Thread Group 別の 4 種類とする。

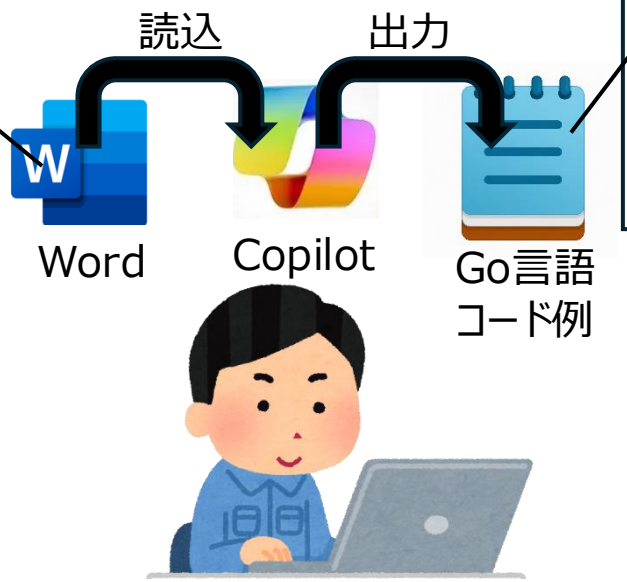
- ***10.5.2 共通前提***

各ダッシュボードでは、Prometheus を Grafana のデータソースとして利用する。

node 変数により表示対象ノードを選択し、node_uname_info から nodename を取得する。

instance 変数は、選択された node に対応する Prometheus 上の instance（IP:port）を取得するための内部変数として利用する。画面上の煩雑さを避けるため、instance 変数は hidden 設定とする。

各パネルでは、PromQL の rate()関数を用いて、累積カウンタ値から秒間値を算出する。



＜Go言語でのコード記述例＞

```
package main

import (
    "encoding/json"
    "fmt"
    "github.com/grafana/grafana-foundation-sdk/go/common"
    "github.com/grafana/grafana-foundation-sdk/go/dashboard"
    "github.com/grafana/grafana-foundation-sdk/go/prometheus"
    "github.com/grafana/grafana-foundation-sdk/go/timeseries"
    "github.com/grafana/grafana-foundation-sdk/go/units"
)

func main() {
    // Prometheusデータソース参照（データソース名 "Prometheus" を指定）
    promDS := prometheus.NewPrometheusDataSourceRef("Prometheus")

    // Dashboard変数: node（表示名）と instance（IP:port, 非表示）, node_uname_infoを使用[1](https://b0aa600df9934664bf4b-
my.sharepoint.com/personal/masako_itou_east_ntt_co_jp/_layouts/15/Doc.aspx?sourcedoc=%7BE234CAEF-1BA7-47C4-AE20-BD5A30470E7A%7D&file=NSCV%
28Prometheus_Grafana%29%E8%A6%81%E6%B1%82%E4%BB%95%E6%A7%98%E6%9B%B8_20260423r3.docx&action=default&mobileredirect=true)
    nodeVar := dashboard.NewVariableBuilder("node").
        Label("Node").
        DataSource(promDS).
        Type("query").
        Query(prometheus.NewPrometheusQuery(`label_values(node_uname_info, nodename)`)). // 各ノード名を取得
        IncludeAll(true). // すべて選択
        Build()
    instanceVar := dashboard.NewVariableBuilder("instance").
        Label("Instance").
        DataSource(promDS).
        Type("query").
        Query(prometheus.NewPrometheusQuery(`label_values(node_uname_info{nodename=~"$node"}, instance)`)). // node選択に対応するinstance
        Hide(common.VarHideLabelAndValue). // UI上非表示[1](https://b0aa600df9934664bf4b-my.sharepoint.com/personal/masako_itou_east_ntt_co_jp/
_layouts/15/Doc.aspx?sourcedoc=%7BE234CAEF-1BA7-47C4-AE20-BD5A30470E7A%7D&file=NSCV%28Prometheus_Grafana%29%E8%A6%81%E6%B1%82%E4%BB%95%E6%A7%98%E6%9B%B8
_20260423r3.docx&action=default&mobileredirect=true)
```

でも、ちがう・・・・
本当にやりたいのはここからなんだ！

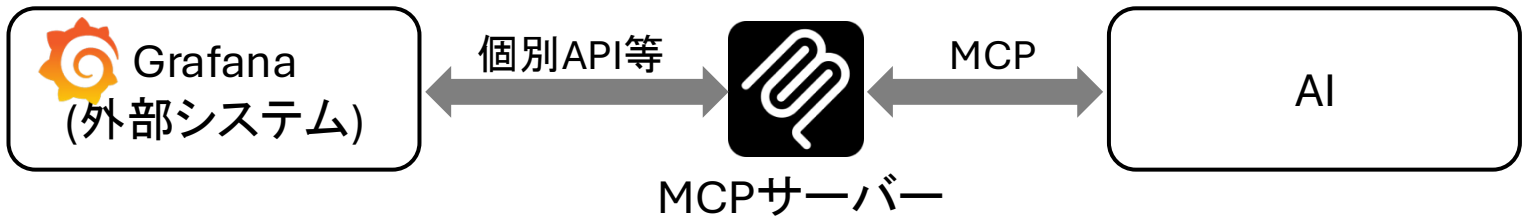
本当はやりたかった・・いや、今後やるんだ・・MCP連携

- 社内環境が未整備であるため、上司所有のサーバー上で実現したい内容のデモを実施いただいた
- デモ環境は、Grafana MCPを介してAIツールとGrafanaを連携し、Grafanaのメトリクス情報をAIに入力する構成とした
- AIは状況に応じて参照すべきメトリクスを選定し、必要な情報のみを抽出したダッシュボードをリアルタイムに生成する
- この構成について、実現性の検討と動作確認を実施した

■MCPとは？

・MCP（Model Context Protocol）は、AIが外部システムを使うための共通規格

・今回はGrafana MCPを使い、AIからGrafanaメトリクス情報の参照やダッシュボードの生成を行う



※Grafana MCP

・Grafana LabsがOSSで公開しているMCP

・https://github.com/grafana/mcp-grafana

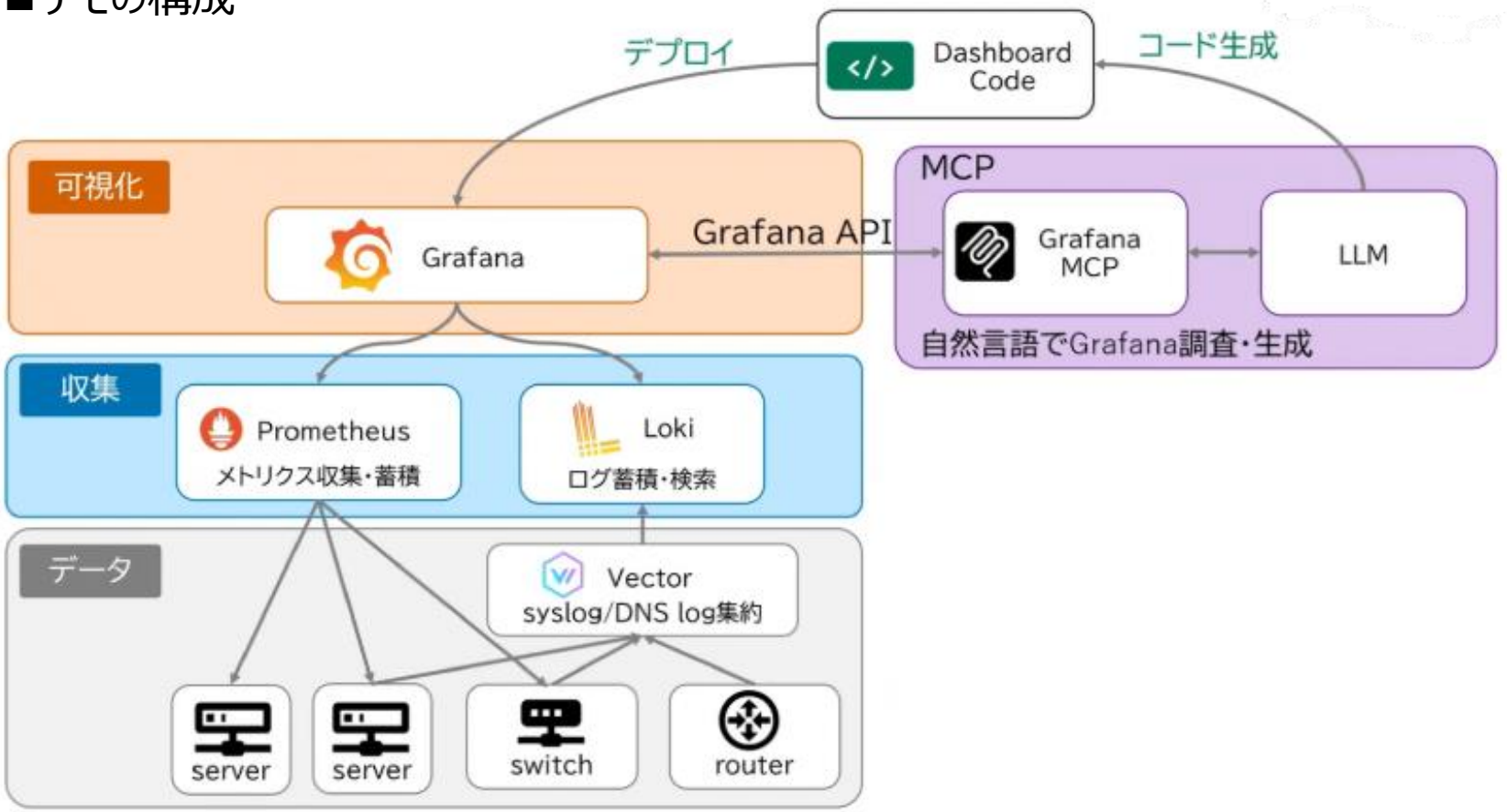
本当はやりたかった・・いや、今後やるんだ・・MCP連携（構成とデモ概要）

- デモの構成については以下の通り
- AI(LLM)がGrafana MCP経由で関連ダッシュボード、メトリクス、ログ、アラートを学習、推論し、
  状況の要約や原因候補の整理、詳細ダッシュボード案の生成をどの程度の精度で実施できるのかをデモしてみた

■デモの流れ

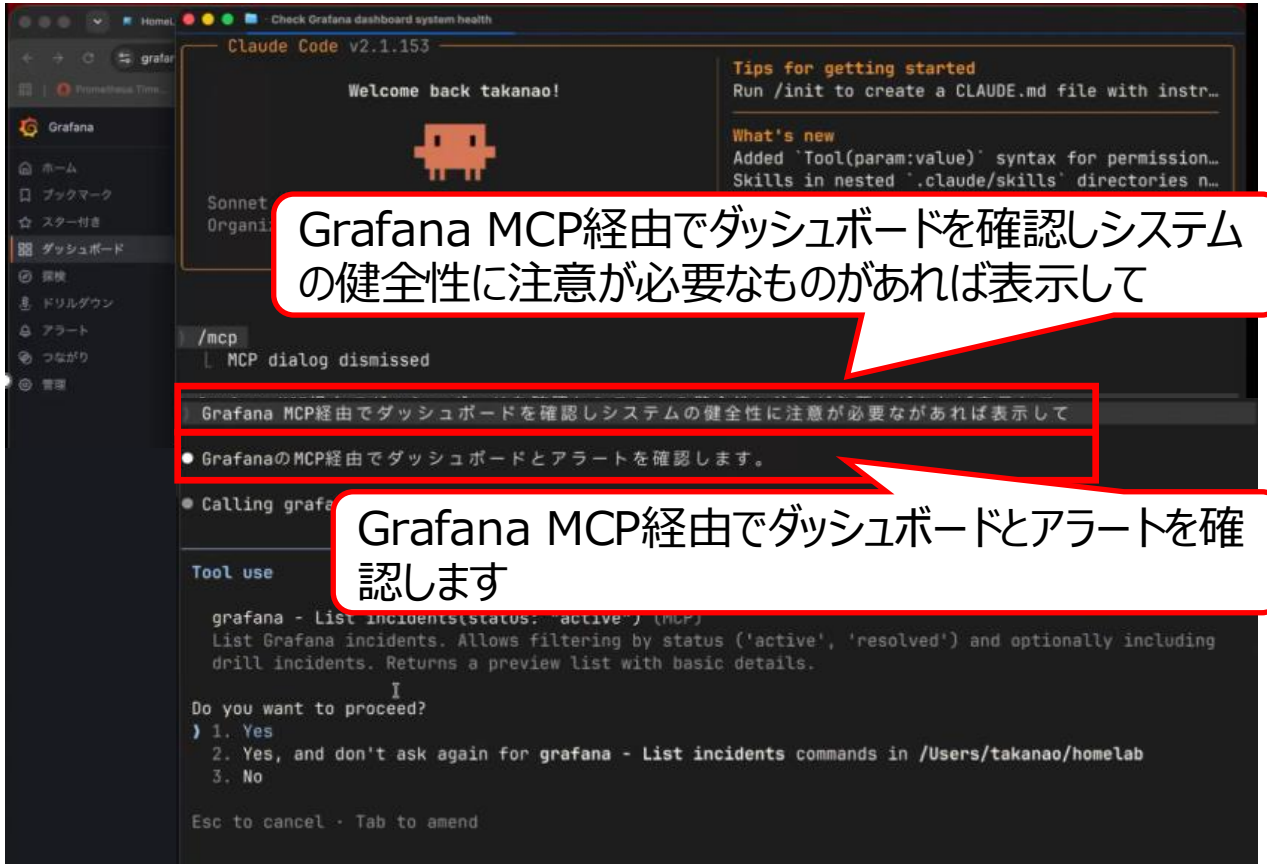
1. 健全性の確認
2. 気になるメトリクスを確認
3. 必要なダッシュボード案を生成

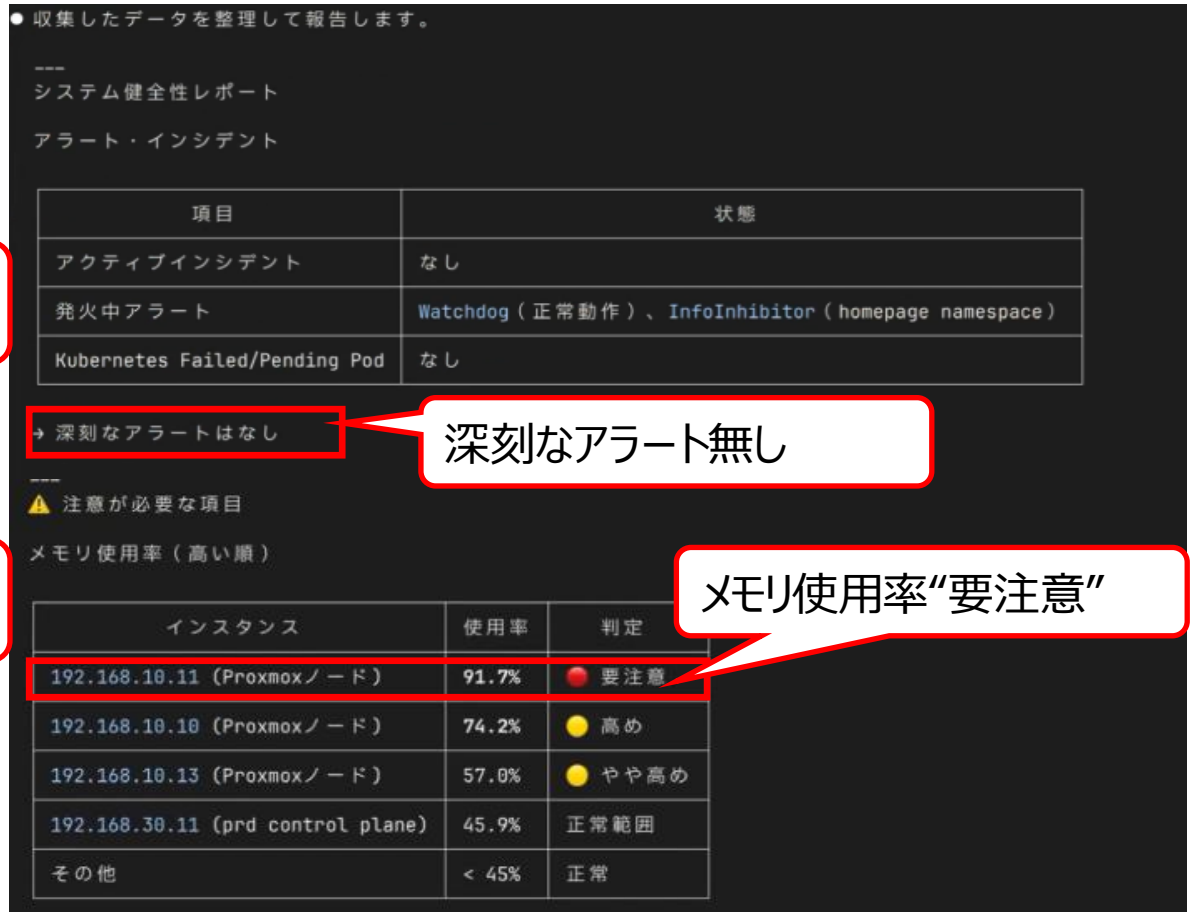
■デモの構成



本当はやりたかった・・いや、今後やるんだ・・MCP連携（デモ）

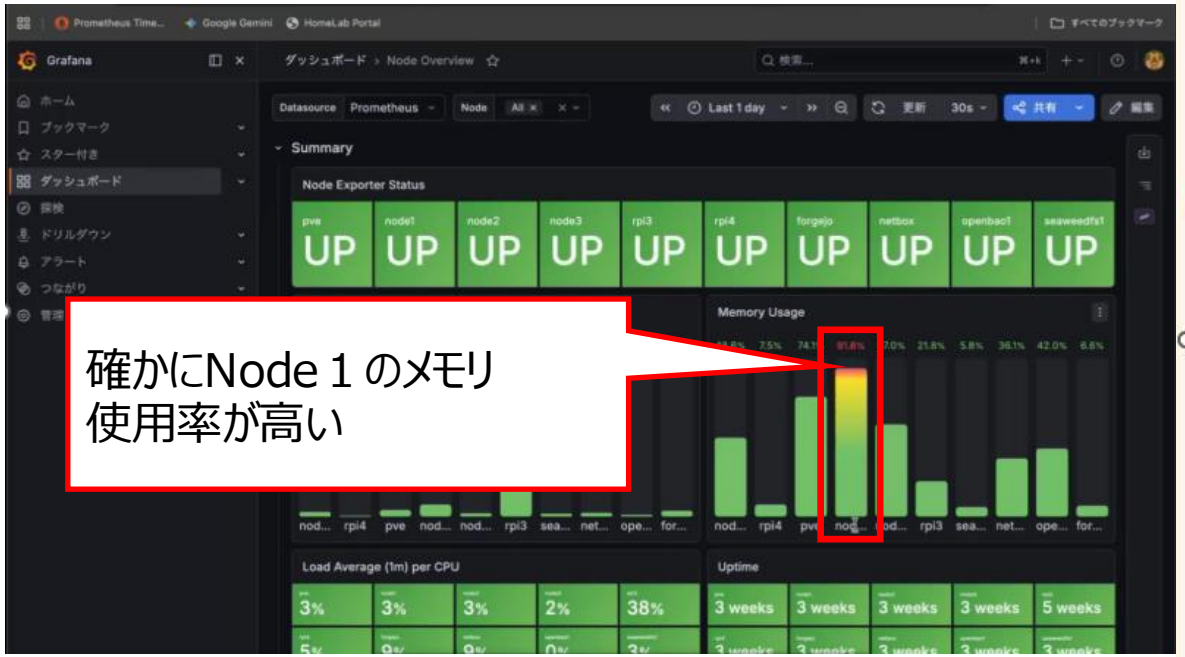
- デモではLLMはClaude Codeを用いて実施した
- 自然言語（日本語）で指示を出して正しく結果を返してくれるのかをデモしてみた

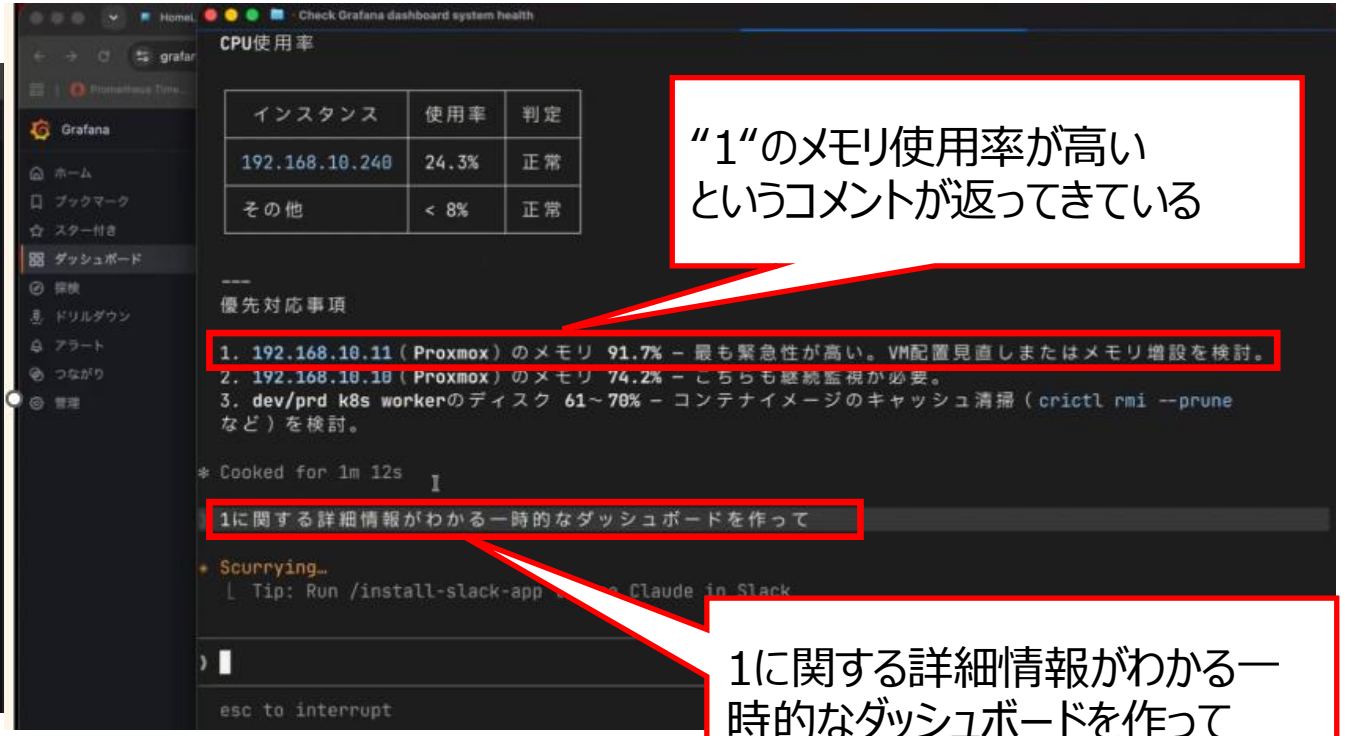




本当はやりたかった・・いや、今後やるんだ・・MCP連携（デモ）

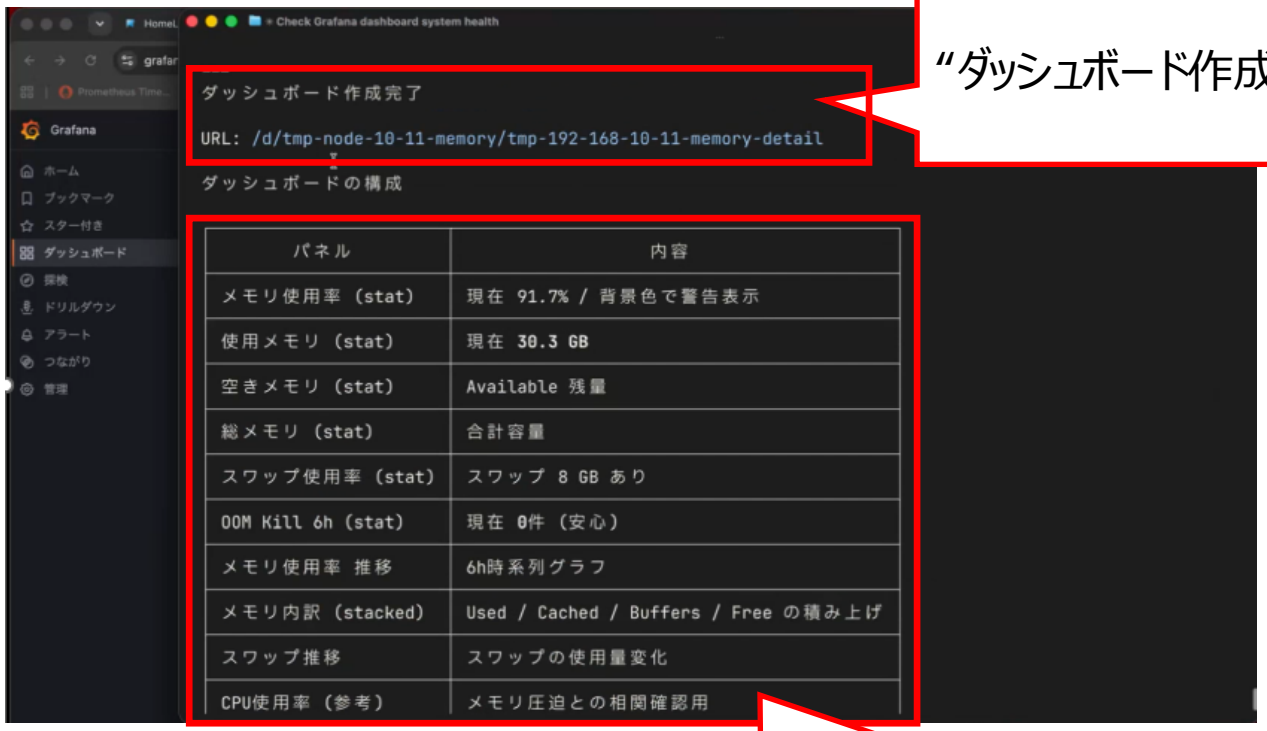
- AI画面で表示された結果が事実とマッチしているのか確認したところ実際のGrafanaダッシュボードでも内容が一致していることが確認できた
- 次に詳細ダッシュボード案の生成を自然言語で指示してみた





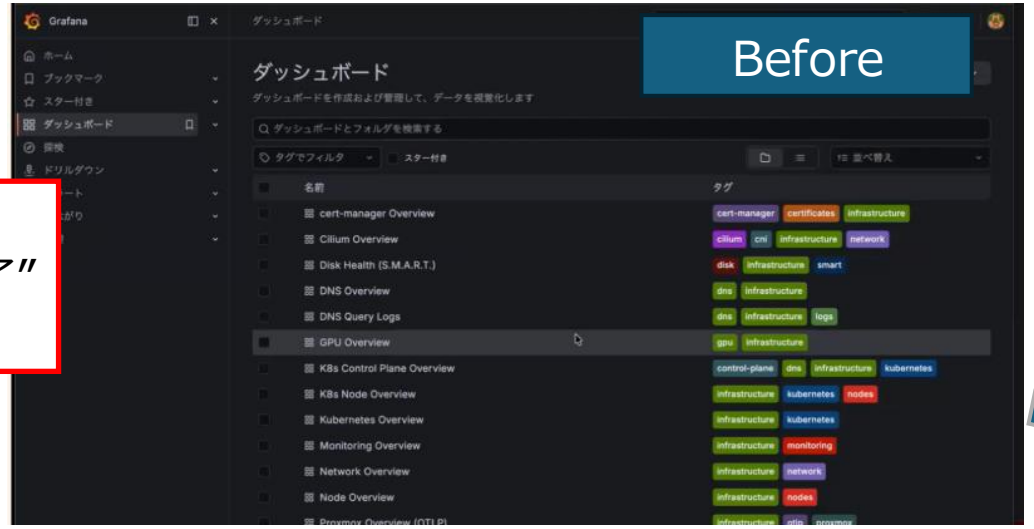
本当はやりたかった・・いや、今後やるんだ・・MCP連携（デモ）

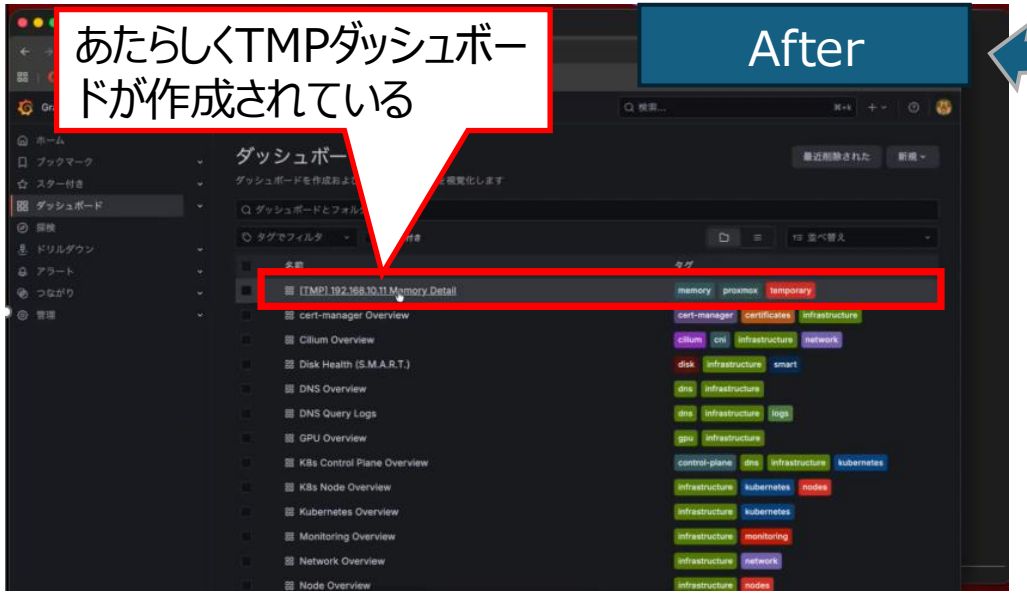
- 指示通り、一時ダッシュボードが作成されたことが確認できた



“ダッシュボード作成完了”

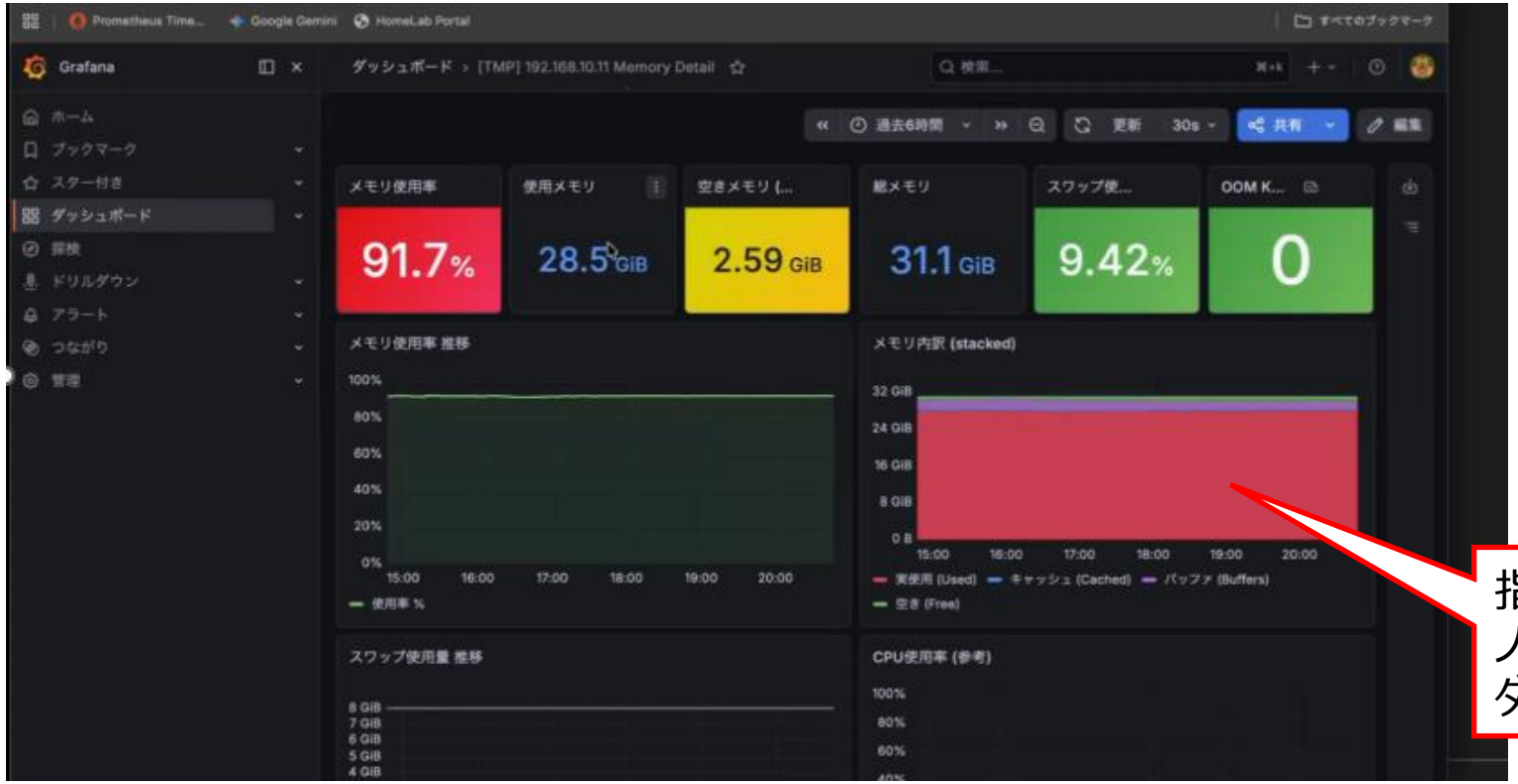
一時的なダッシュボードは
どのような構成で作成
したかの説明を添えてくれている





本当はやりたかった・・いや、今後やるんだ・・MCP連携（デモ）

- あたらしく作成された一時ダッシュボードを開いて確認
  ⇒指示通りに、メモリ使用率の高いノード情報のみピックアップされたダッシュボードが作成されていることが確認できた



指示通りに、メモリ使用率の高いノード情報のみピックアップされたダッシュボードが作成されている

MCP連携 まとめ

- 確認できたこと
  - MCPを用いることで、AIがオブザーバビリティ情報を横断的に参照
  - 調査支援、ダッシュボードの作成支援ができた
- 期待される効果
  - 調査時における観点の抜け漏れ防止
  - Grafanaに不慣れな人の支援
- 課題
  - オンプレでの実現には、いくつかハードルがある（セキュリティ等）
    - クラウドLLMの利用可否。ローカルLLMを検討すべきか？
    - メトリクスやログデータをAIに渡せるか

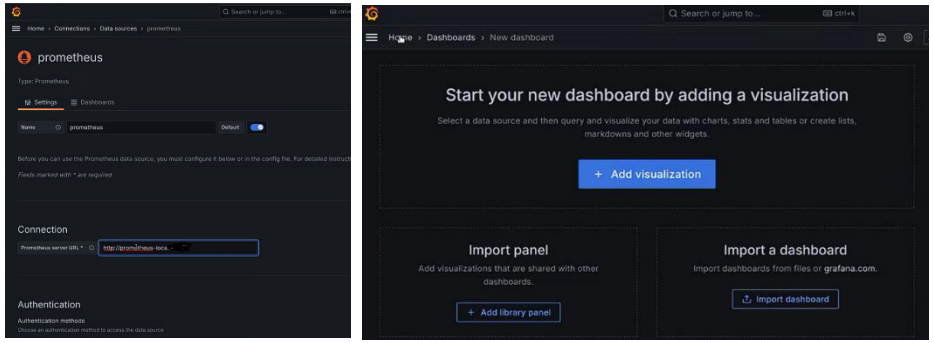
まとめ

- PoCで期待する動作を確認できた
  - 今後はオンプレ環境での運用やローカルLLMの活用を検討したい
  - 同じように試している方がいればぜひ情報交換させてください

(参考)Grafana Foundation SDKとは？

- Grafanaダッシュボードの開発・修正・検証では、変更履歴の管理やレビューをしやすくするため、GUI操作による手作業ではなく、Grafana Foundation SDKを用いたコード管理方式を検討している。

【(A)GUI操作でGrafanaダッシュボードを設定構築・修正】



GUI操作でダッシュボードの設定

【(B)Grafana Foundation SDKを用いた設定構築・修正】

Go言語等でダッシュボードの設計内容をコードに書く → SDKでGrafana用ダッシュボード(のJSON)を生成 → Grafanaへ反映

	メリット	デメリット
(A)	・直感的に操作できる ・すぐに作成、修正できる ・画面を見ながら調整しやすい ・プログラミング知識がなくても扱いやすい	・手作業のため設定漏れや入力ミスが起きやすい ・同じ修正を複数箇所へ反映するのが大変 ・変更履歴やレビューがしづらい ・環境間で設定差分が出やすい ・ダッシュボード数が増えると保守が大変
(B)	・設定をコードとして管理できる ・Git等のバージョン管理ツールで変更履歴を追いやすい ・レビューしやすい ・同じ設定を複数環境へ展開しやすい ・共通部品化、テンプレート化しやすい ・CI/CDによる自動化がしやすい	・SDKやプログラミングの学習が必要 ・初期導入に手間がかかる ・小規模・単発用途では過剰になる場合がある ・画面上で細かく見た目を調整する作業はGUIの方が楽な場合がある