IPCIでWebクローリングを実装する

2025/10/1

群馬大学医学部附属病院システム統合センター
防衛医科大学校 情報戦略官 デジタル化推進本部推進補佐官
鳥飼　幸太

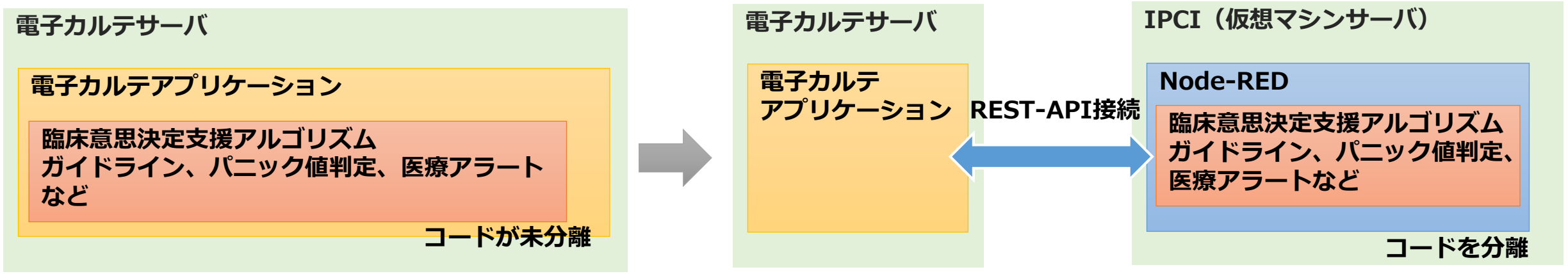
IPCI概要

- In-Process Clinical Intelligenceの略
- 提唱・開発者：鳥飼　幸太（群馬大学医学部附属病院）
- 臨床意思決定支援(Clinical Decision Support)機能の実装モデルを含む
- 臨床現場での1次利用を主眼としている
- 医療で生じる情報判断-伝達手段を併せて提供する

- Vmware/Ubuntuベースの仮想マシン、主にオープンソースで構成（データ）
- 2022年より日本Mテクノロジー学会から継続的にリソースを提供、講習会を実施している
- ライセンス形態：2025年現在学術機関および企業での利用において無償（運用時に届出をお願い）

IPCIが目指すもの

- 医療現場における「臨床に有用なアルゴリズム・ガイドライン・コード」の運用永続化
- 医療機関サイドでの「求めるサービスの院内開発」の促進
- ePathの標準実装モデル化（医療情報学会・日本クリニカルパス学会合同委員会で検討中）



ePathプロジェクトで提唱されたOATユニットモデル

NEWS

2021.8.30	「ePathのデータ要素と構造に関する仕様書」における「標準パスコード」を設定しました。 → 標準クリニカルパスコードについて（PDF）
2021.6.10	成果物「電子クリニカルパス作成・運用マニュアル」（PDF）を掲載しました。
2021.3.30	公開シンポジウムの映像を公開いたしました。
2021.3.13	公開シンポジウムを開催致します。 時間：13:00 – 16:00

課題

現在、日本国内の病院の1/3以上に電子カルテシステムが導入されており、電子カルテを導入すれば日本全国の患者データの比較が容易にできるかと思われたが、電子カルテシステムベンダー間でデータ交換できるシステムは存在していないため、データの後利用は困難な状況にある。一方、クリニカルパス(以下、パス）は、入院診療計画書として保険収載もされており、広く電子カルテ上でも普及しているので、パスを病院間で共通化することにより、複数医療施設間での患者データを比較できる可能性があるが、ベンダー間でパスの項目、構造とも大きく異なっており、実際には比較が困難な状況にある。

用語解説

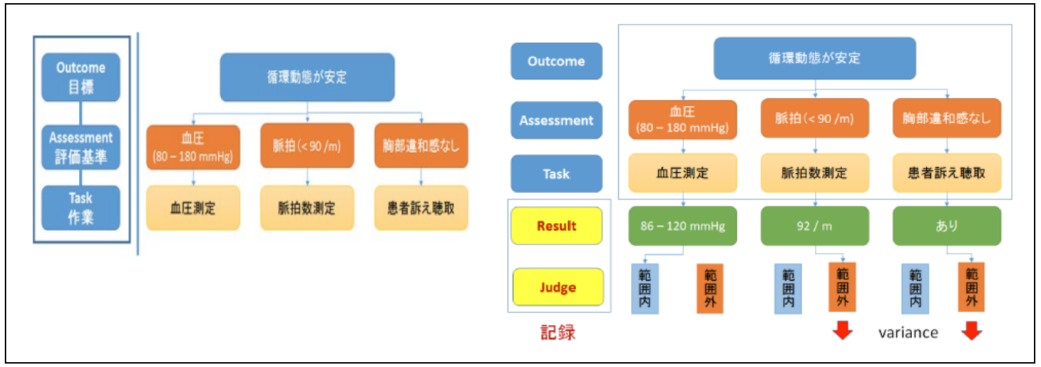
<入院診療計画書>

入院時に医師、看護師およびその他の医療従事者が共同して患者の診療計画を作成し、患者に対し交付される文書。病名、主治医、症状、治療計画、手術内容および日程、推定される入院期間等についての記載が必要である。
保険点数上は、入院診療計画が策定され，説明が行われていない場合は，入院基本料等より減額となる。

<クリニカルパス>

「患者状態と診療行為の目標、および評価・記録を含む標準診療計画であり、標準からの偏位(ずれ)を分析することで医療の質を改善する手法」と定義される。「アウトカム志向型パス」は、アウトカムユニット(アウトカムーアセスメントータスク)を医療行為の最小単位と定め、この最小単位の組み合わせにより、診療プロセスを構築するものであり、日本クリニカルパス学会が提唱している。下図にアウトカムユニットと、その記録の例を示す。

アウトカムユニットの構造と記録の例



ePath実装モデルとしての目的：

ePathプロジェクトで提唱されているOATユニットのIPCI内Node-REDでの実装を試行する機能を達するmsgオブジェクトの構造を検討する

OATを実装する上で求められる機能：
・テンプレート（xlsx、手入力で作成）からのO/A読み込みが容易に行えること
・Aについては単一のアセスメントについて記述したノードを作成し、再利用しやすいこと
・Oに付随する複数のAを取り扱えること
・複数のOによって構成される抽象度の高いOが記述できること

IPCI解説と
本セミナーで扱う技術範囲

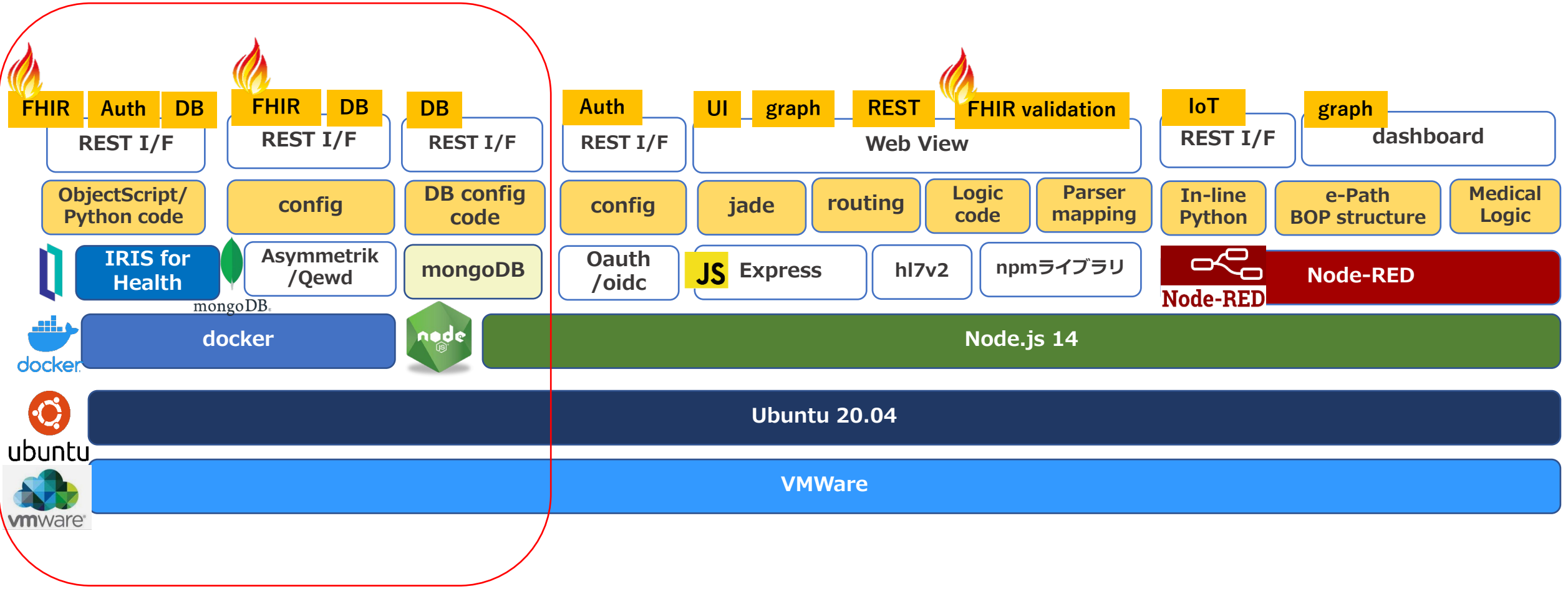
IPCI機能概要図

IPCIについて詳しくは日本Mテクノロジー学会までお問い合わせください

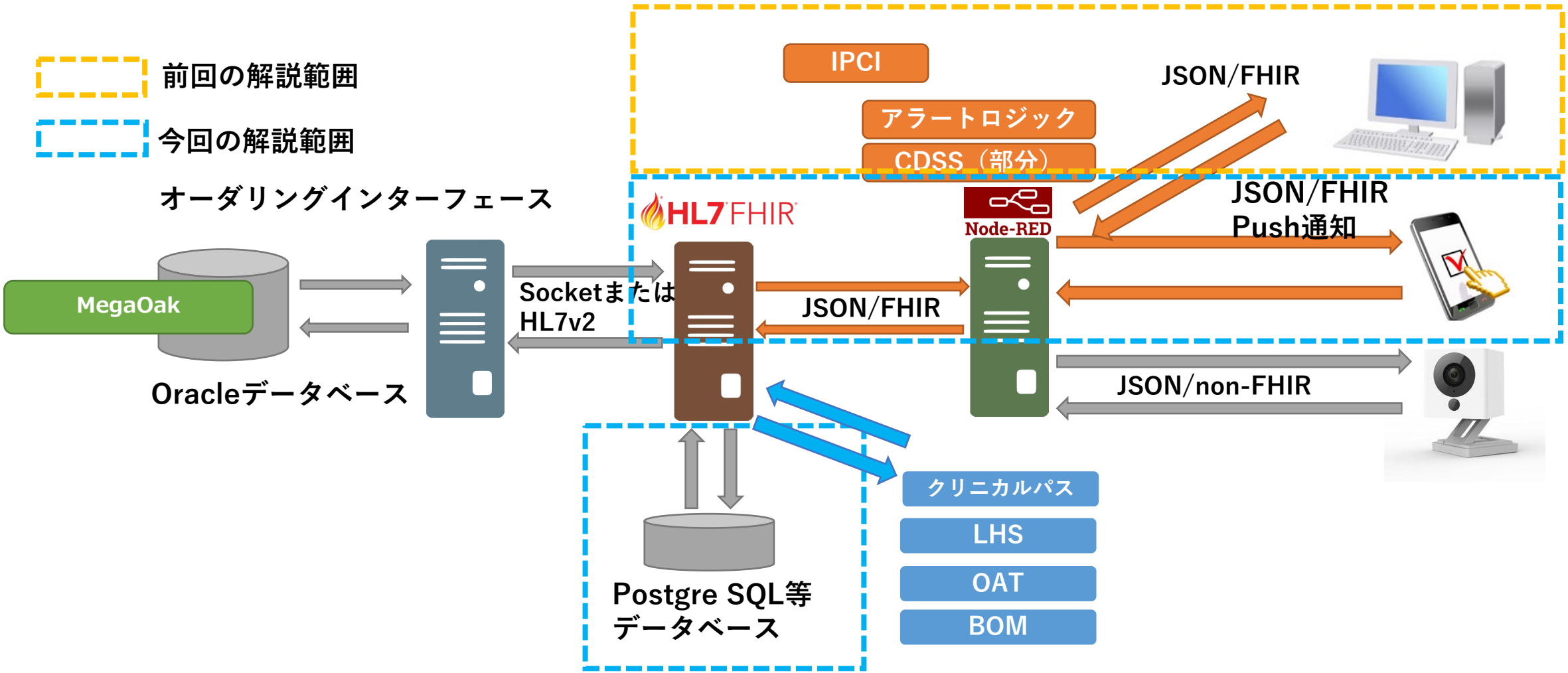
mta-office@mta.gr.jp

会員はIPCIインストール等技術解説動画を参照できます

https://www.mta.gr.jp/members.html

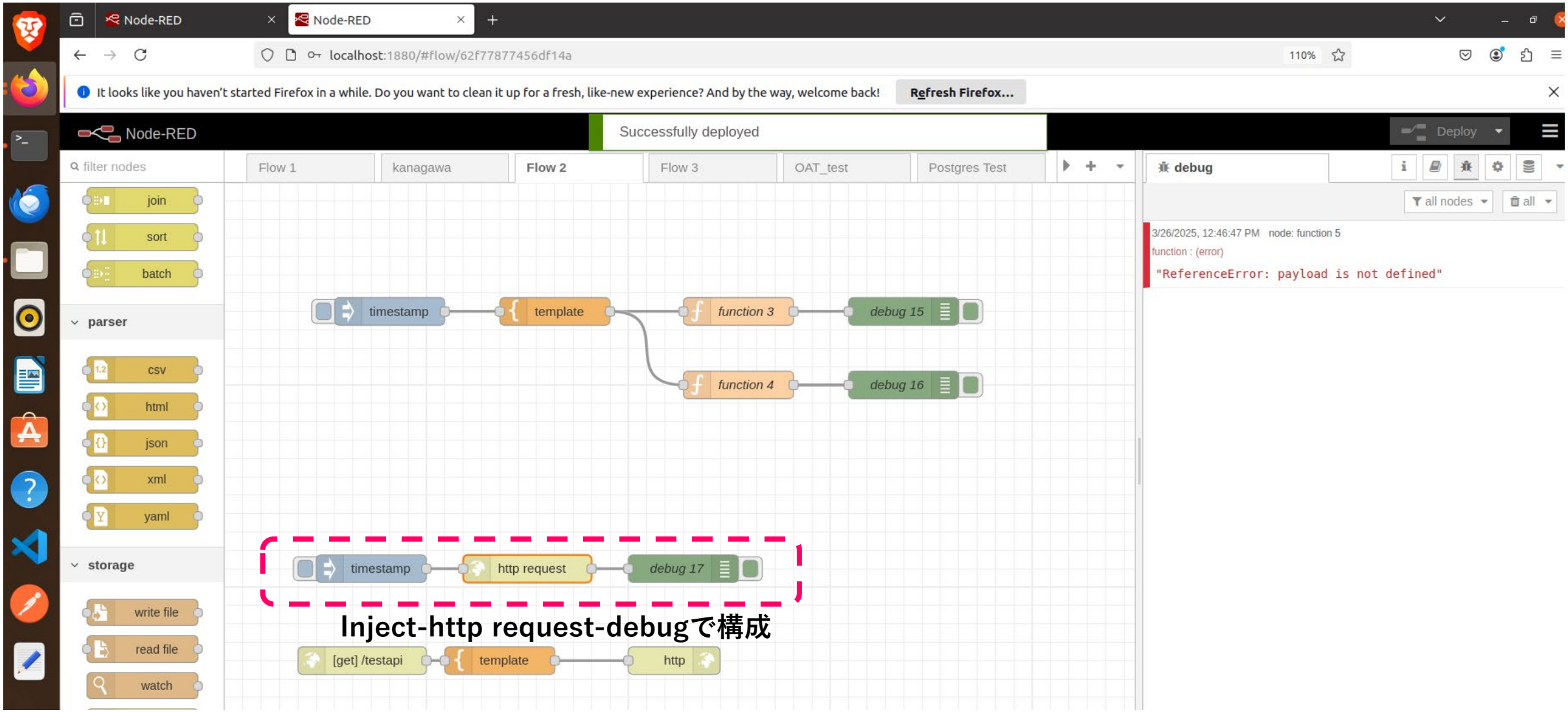


電子カルテとIPCIの接続関係設定と機能割付け（アーキテクチャ）

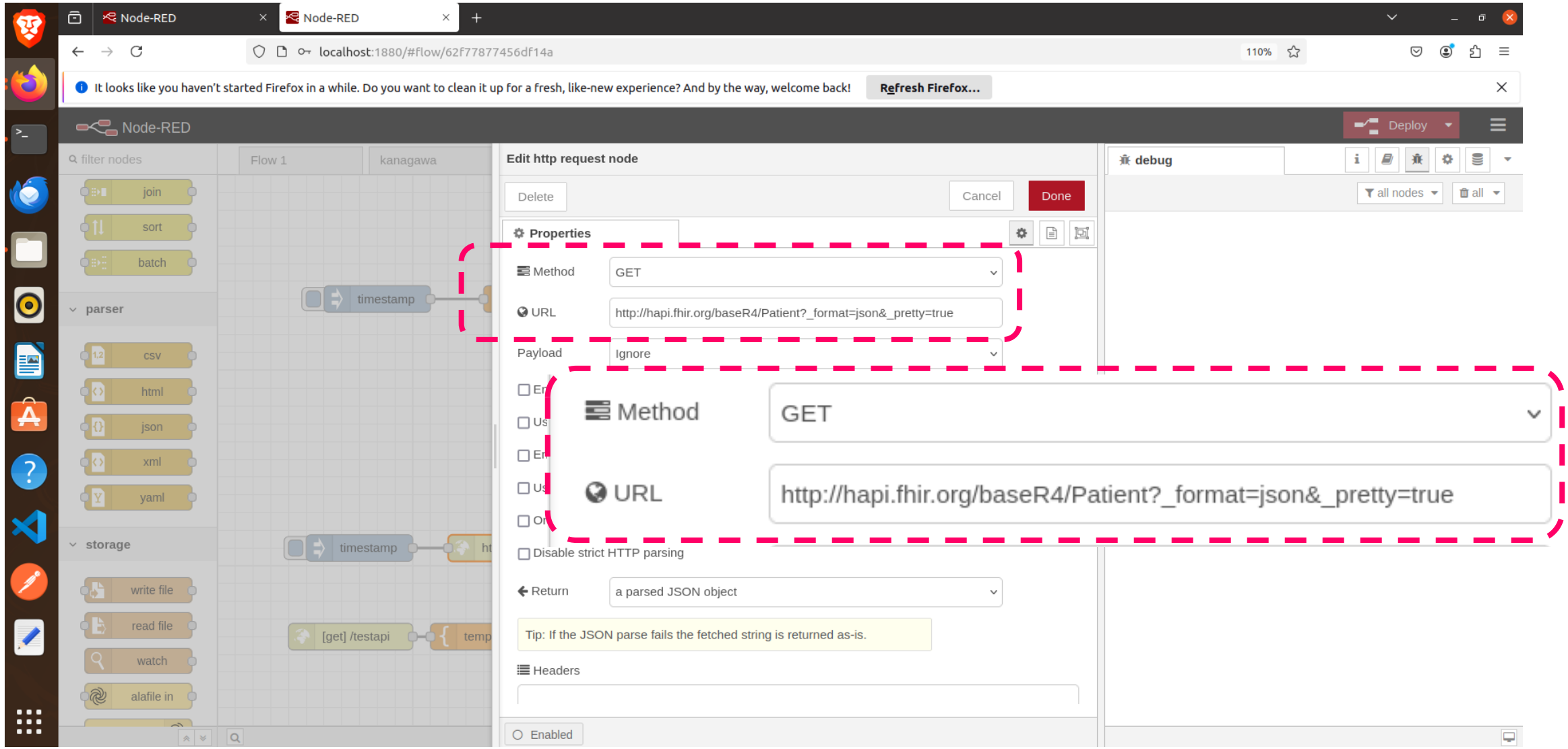


IPCI(Ubuntu)から
FHIRサーバにアクセスする

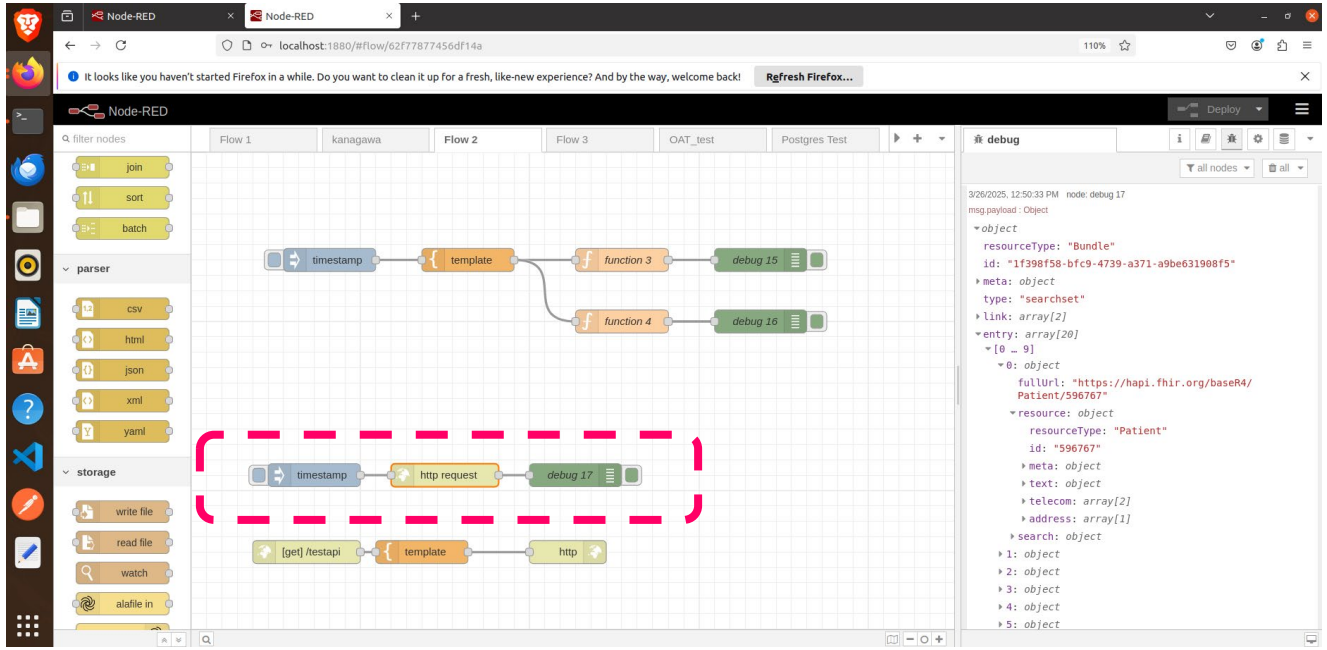
Node-RED構成



Node-RED構成

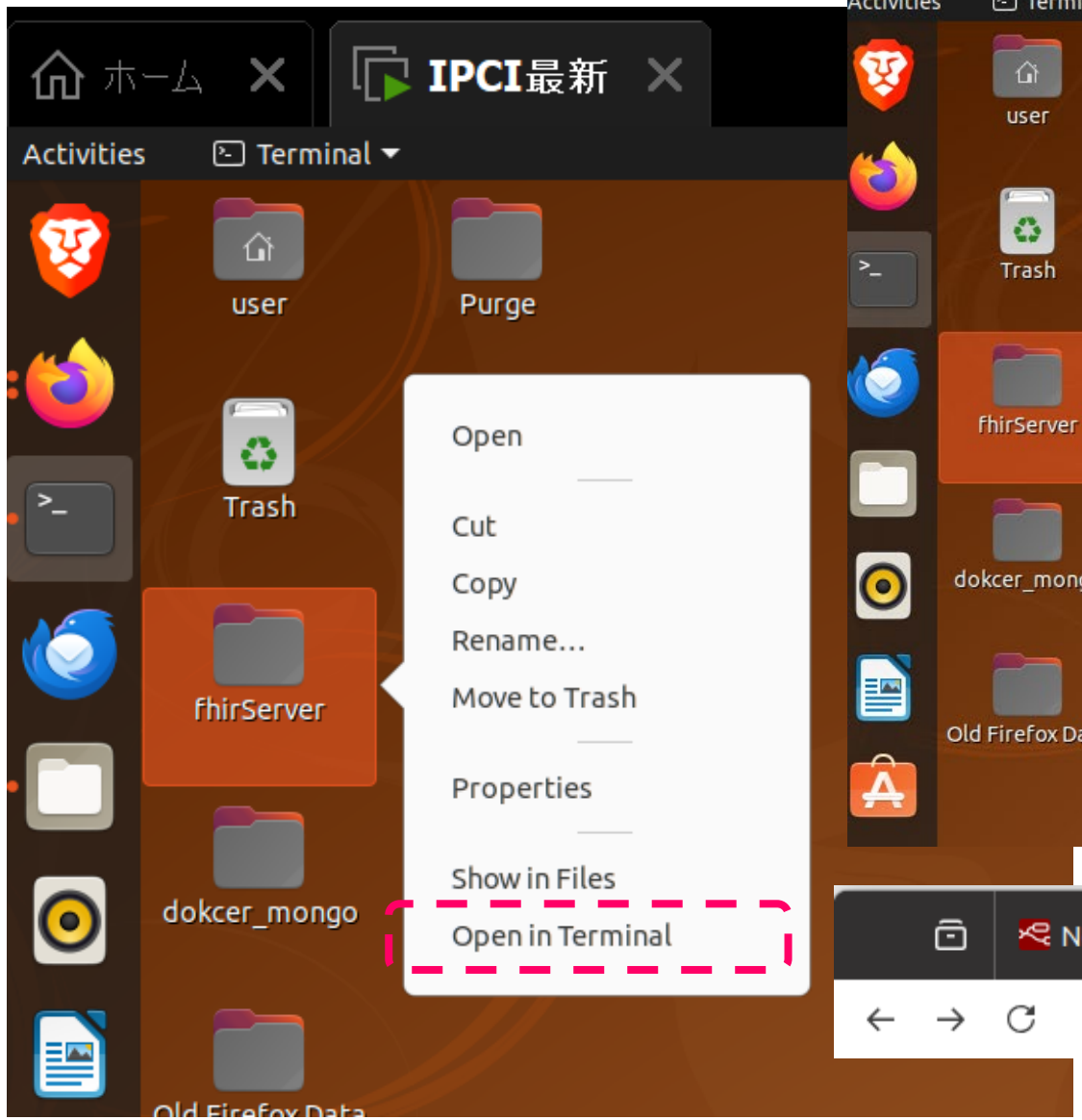


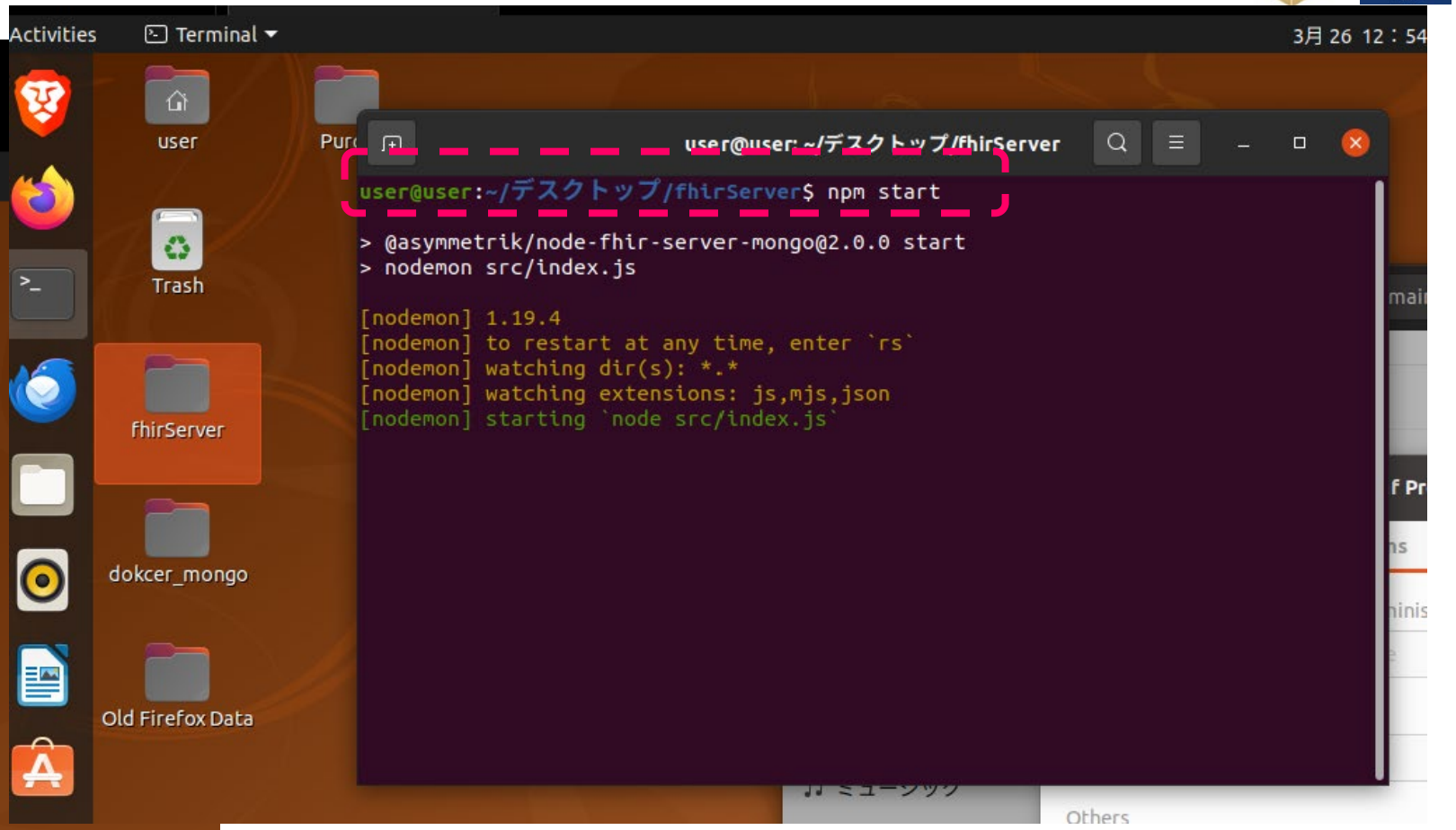
Node-RED構成

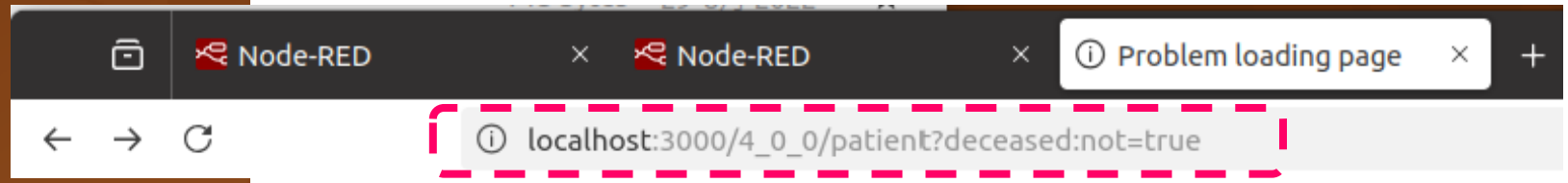


FHIRデータを取得できた

IPCI内のFHIRサーバ起動







http request URIを上記に変更すればよい

IPCI(Ubuntu)でのWebクローリング導入

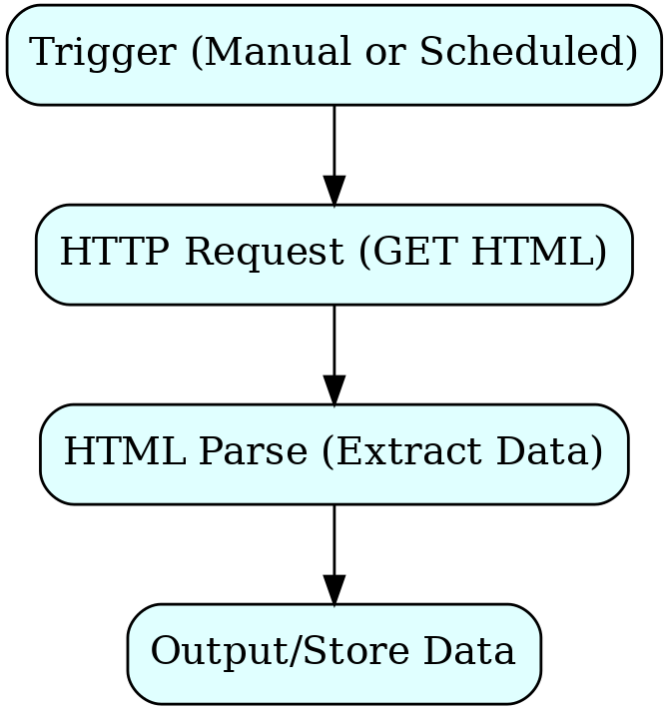
本資料の目的と概要

- 静的HTMLサイトからのデータ取得（Webクローリング）の必要性と用途
  - （例：競合調査、情報収集など）

- Node-REDによるローコード開発でWebクローリングを行う利点
  - （視覚的フロー設計、他システム連携容易など）

- このプレゼンで紹介する手順の全体像
  - （HTTPリクエストでHTML取得→HTML解析→データ抽出→結果利用）

静的HTMLと動的コンテンツの違い



静的HTMLページ: リクエストするとサーバから直接HTMLが返され、内容表示に追加のJavaScript実行を必要としないページwebscraping.blog。HTML内にデータが含まれるため、そのまま解析（パース）が可能

動的ページ: 表示内容を生成するためにクライアント側でJavaScriptを実行する必要があるページ。静的手法ではデータ取得が困難で、ヘッドレスブラウザ（例：Puppeteer）によるレンダリングが必要になる本資料の範囲外）

Node-REDのHTTPノードは静的HTMLの取得に有効だが、動的コンテンツには標準機能だけでは対応困難である

クローリング処理の全体像

フロー概要: 手動またはスケジュールによるトリガー
→HTTPリクエストノードでページ取得
→HTMLノードでコンテンツ解析→結果を出力・保存する

ポイント: 一連の処理をNode-RED上のノードの流れに対応付けて説明する。例えば「Injectノードで定期実行→HTTPリクエストでHTML取得→HTMLノードでデータ抽出→DebugノードやDBノードで結果確認/保存」

使用ノードの紹介

Injectノード（トリガー）：手動またはスケジュール実行のきっかけを作るノード。一定間隔でのクローリングもこのノードで設定可能。

HTTPリクエストノード: 指定したURLに対してHTTP GETリクエストを送り、Webページ（HTML）を取得する
静的ページからHTMLテキストを取得する役割で、本フローの中核となる。

HTMLノード: 取得したHTMLテキストから、特定のタグやクラスの要素を抽出するノード
CSSセレクタを指定して対象要素をテキストや属性値として取り出せる（HTMLパーサとして機能）

デバッグノード: フローの出力をデバッグタブに表示する補助ノード。取得した生HTMLやパース結果のデータを確認するのに使用

（必要に応じて）関数ノード/Changeノード: 抽出したテキストを加工・整形したり、データ構造を整える場合に使用。JavaScriptで自由な処理を書ける
（必要に応じて）保存・通知系ノード: 抽出データを蓄積・活用するためのノード。例：ファイルノードでCSV保存、データベースノードでDBに記録、あるいはダッシュボードやメール送信ノードで通知など

フロー構築①: HTTPリクエストでページ取得

１　ノード配置: Injectノード、HTTPリクエストノード、デバッグノードをワークスペースに配置する。

２　HTTPノード設定: HTTPリクエストノードをダブルクリックして編集画面を開き、Method（メソッド）を「GET」に設定。URL欄にクローリング対象のページURLを入力する（例として練習用サイト https://www.scrapethissite.com/pages/simple/ などを使用可能）

３　ノード接続: Injectノードの出力をHTTPリクエストノードの入力に接続し、HTTPリクエストノードの出力をデバッグノードの入力に接続する。この一連の接続で、「手動トリガー → ページ取得 → 内容確認」の流れができる

４　フローのデプロイと実行: ノード配置と設定後、フローをデプロイ（配備）する。Injectノードのボタン（■→）をクリックしてフローを実行すると、HTTPリクエストノードが指定ページにアクセスし、取得したHTMLテキストをメッセージ（msg.payload）としてデバッグノードに渡す

５　結果確認: Node-REDエディタのデバッグタブを開き、メッセージを確認する。ページの生のHTMLソースがそのまま表示されていることを。「payload」にHTML文字列が入っていれば、ページ取得成功

フロー構築②: HTMLノードでデータ抽出

解析対象の特定: 事前に対象WebページのHTML構造を確認し、欲しいデータがどのHTML要素に含まれるかを調べる。ブラウザの開発者ツール（Chromeの「要素を検証」など）を使い、該当箇所のタグやクラス名を確認する例えば、Node-RED公式サイトからバージョン番号を取得したい場合、という要素にバージョン文字列があることがわかるcookbook.nodered.org。このように抽出したいテキストが入っている要素のCSSセレクタ（タグ名やクラス名）を特定する。

HTMLノード追加: ワークスペースにHTMLノードをドラッグ&ドロップで追加配置する。HTTPリクエストノードの出力を、このHTMLノードの入力につなぐ（Debugノードは一旦取り外し、HTMLノード経由の後ろにつなぎ直す）。

HTMLノード設定: HTMLノードを編集し、「Selector」（セレクタ）欄に先ほど調べたCSSセレクタ（例: .node-red-latest-version や目的の要素のパス）を入力する。【出力】は「要素のテキストのみ」(inner text) を選択し、その他の設定はデフォルトのままにするflowfuse.com。これにより、指定したセレクタにマッチするすべての要素のテキストが配列として出力される。

ノード再接続とデプロイ: HTTPリクエストノードの出力をHTMLノードの入力に、HTMLノードの出力をデバッグノードの入力に接続するflowfuse.com。フローを再デプロイし、再度Injectノードをクリックして実行する。

抽出結果の確認: デバッグタブに表示されたメッセージを確認する。設定が正しければ、msg.payloadには抽出されたテキストデータ（目的の情報）が格納されているはずである。例えば、サンプルサイトから国名一覧を抽出した場合は国名データの配列が、Node-REDサイトのバージョンならバージョン番号の文字列が得られる。

（必要に応じて）データ整形: 抽出結果がそのままでは扱いにくい場合、関数ノードやChangeノードでテキストのフォーマットを整えたり、JSONオブジェクトに組み立て直す処理を行う（例: カンマ区切りの文字列を分割しオブジェクト化する等）。この詳細は必要に応じ別途説明。

Node-REDフロー実装（画面デモ）

応用: 定期実行・多ページクローリング・結果利用

定期実行: Injectノードのプロパティで反復実行(interval)を設定することで、一定時間ごとに自動クローリングが可能。例えば毎日夜間にデータ取得するようなスケジュール設定もGUI上で簡単に行える。

複数ページのクローリング: 静的ページが複数存在し全体をクロールしたい場合、いくつかのアプローチが考えられる。

方法1: 複数のInject→HTTPノードを用意し、それぞれ別URLを設定して並列に処理する（ページ数が少ない場合に手軽）。

方法2: 関数ノード等でURLリスト（配列）を保持し、ループ処理でHTTPリクエストノードに渡す。もしくはSplitノードでURL配列を一括投入し順次リクエストを発行する方法もある。

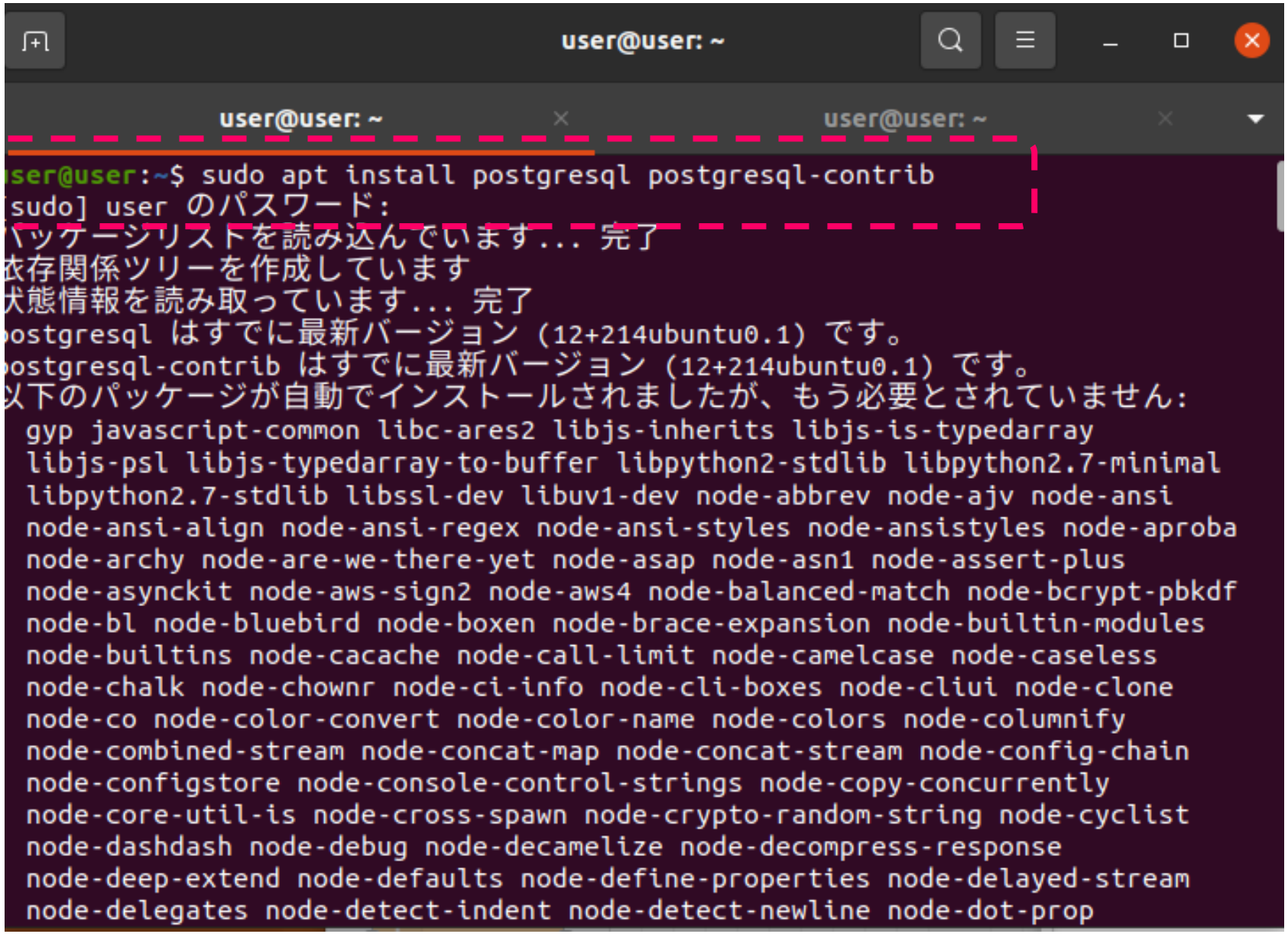
方法3: 1ページ内のリンクを解析して次のURLを取得し、メッセージをフィードバックループすることでクローラを構成する（再帰的クローリング）。こちらは高度な手法で、無限ループ防止など追加制御が必要になる。

取得データの保存/活用: クローリングで得たデータはそのままデバッグ表示するだけでなく、各種ノードを使って社内システムで活用できる形にする。例として：ファイルノードを使いCSVやJSONファイルに書き出し蓄積する。データベースノード（例：MySQL, MongoDB）でDBに保存し、後からクエリ・分析できるようにする。ダッシュボードノードでWeb UIに表示したり、メールやチャット通知ノードでアラートを飛ばすなど、Node-REDの他機能と組み合わせて自動化を進める。

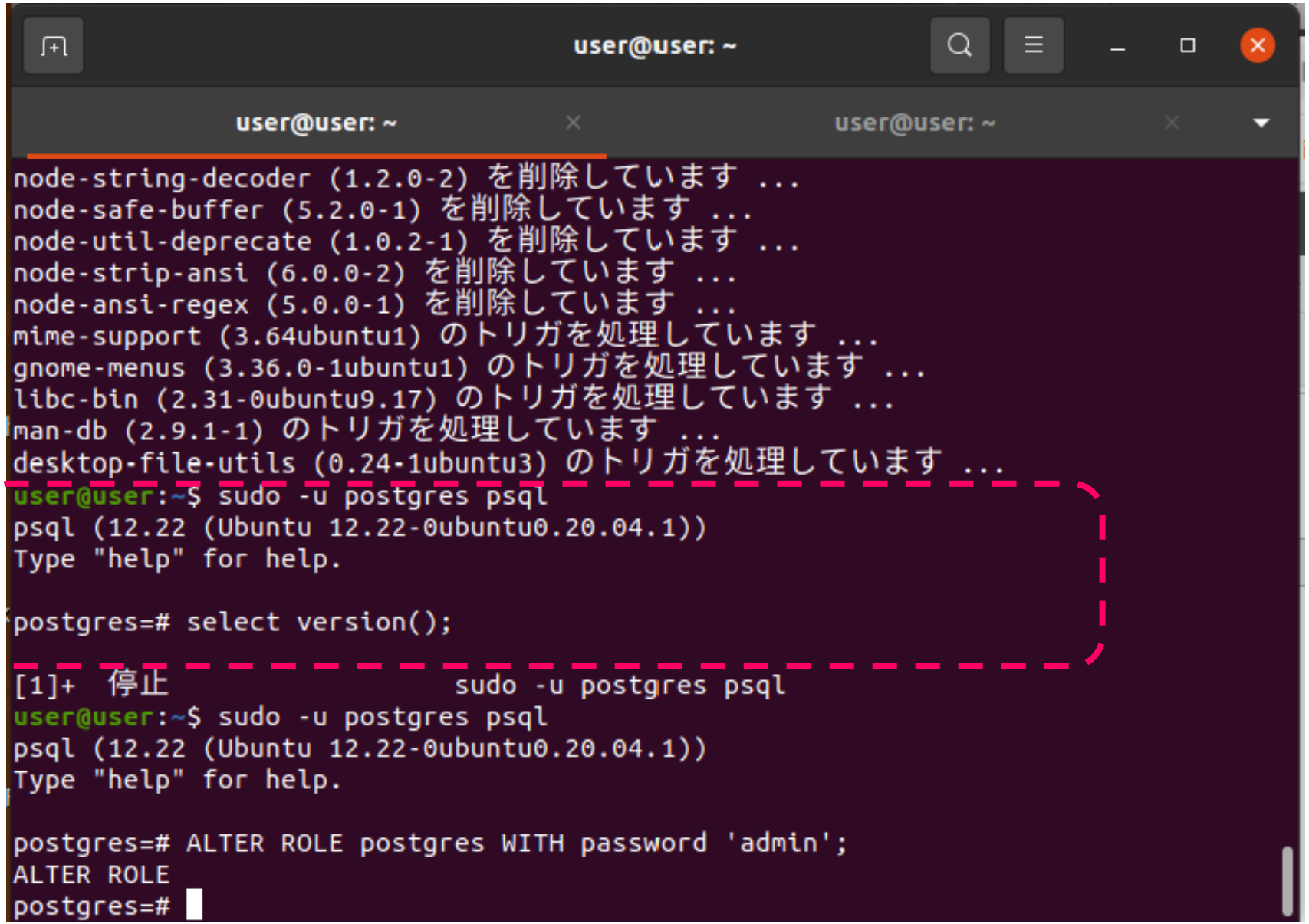
エラーハンドリング: HTTPノードのステータスコード（msg.statusCode）をチェックし、取得失敗や404エラーの場合の処理（再試行やスキップ）を組み込むことも信頼性向上に重要。

IPCI(Ubuntu)へのPostgreSQL導入

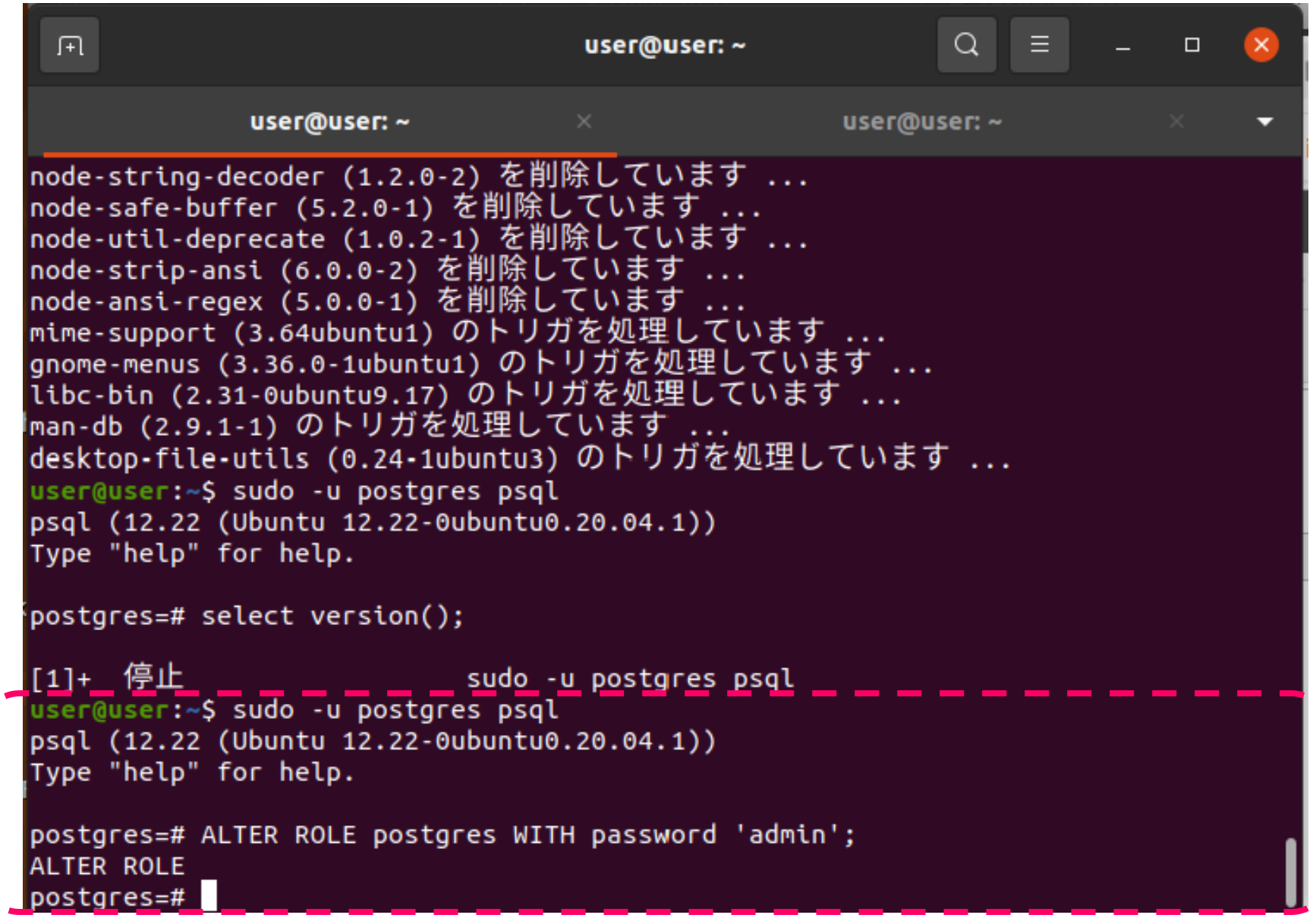
PostgreSQLデータベースソフトウェアのインストール



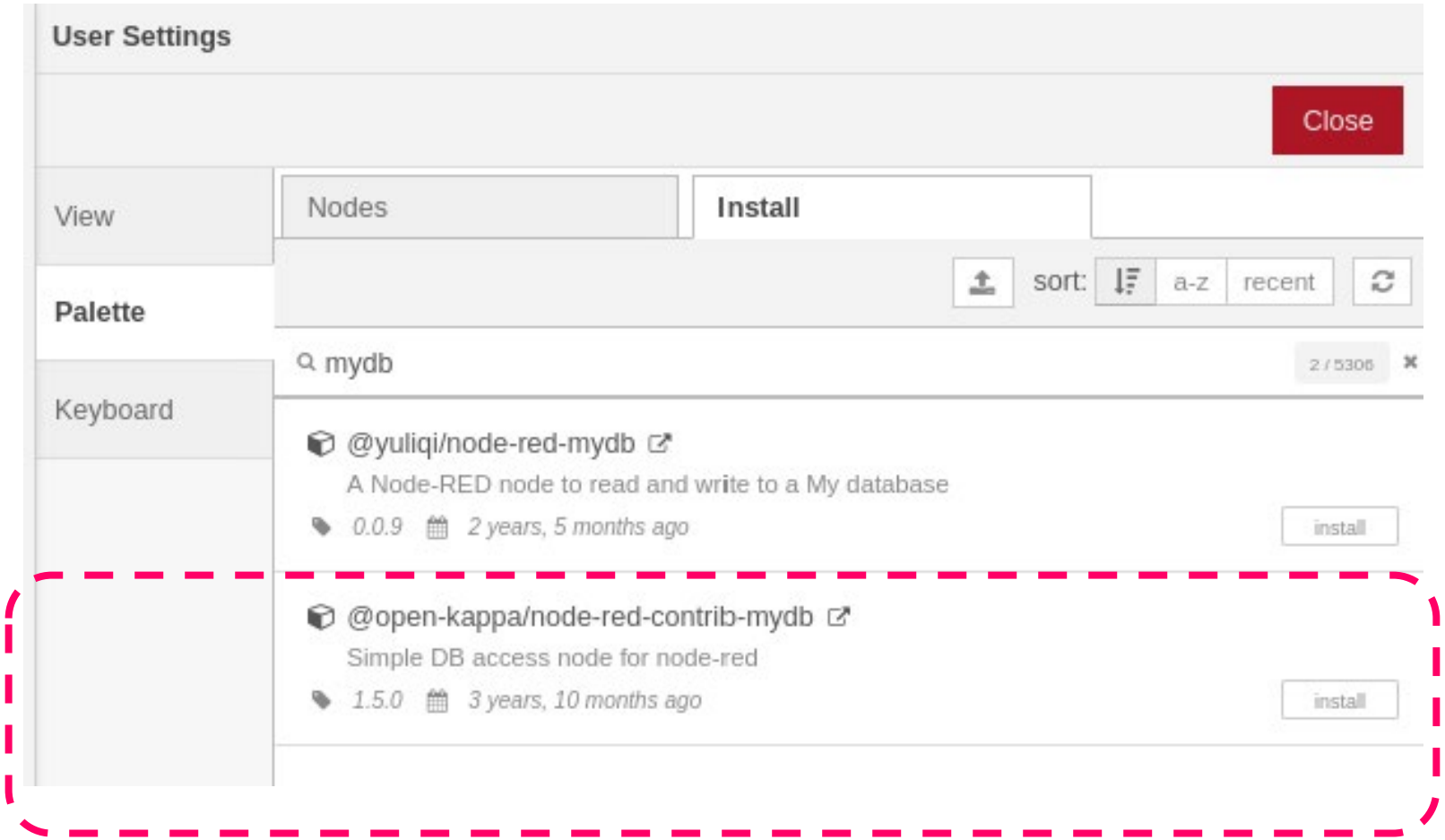
PostgreSQL 動作確認

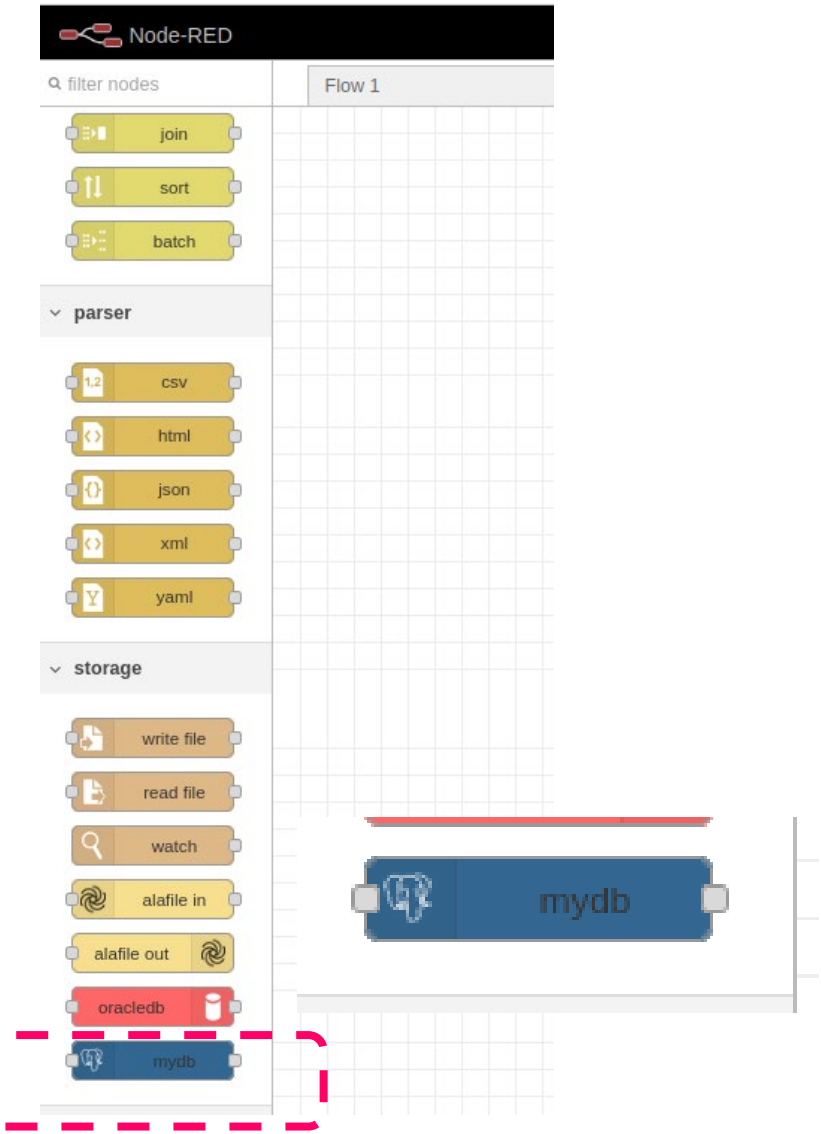


PostgreSQLパスワードの設定

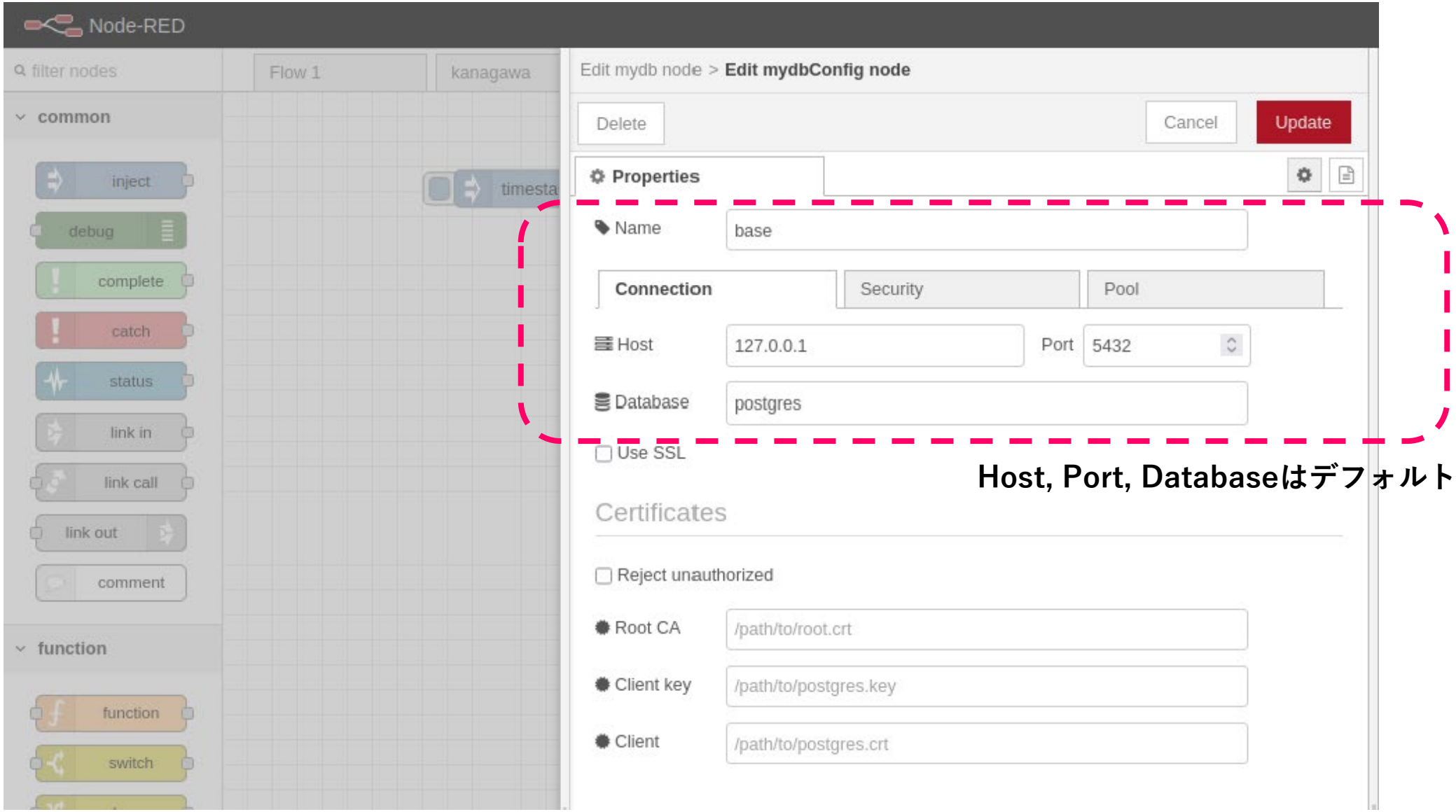


Node-RED側作業：PosgreSQLを呼べるツールのインストール



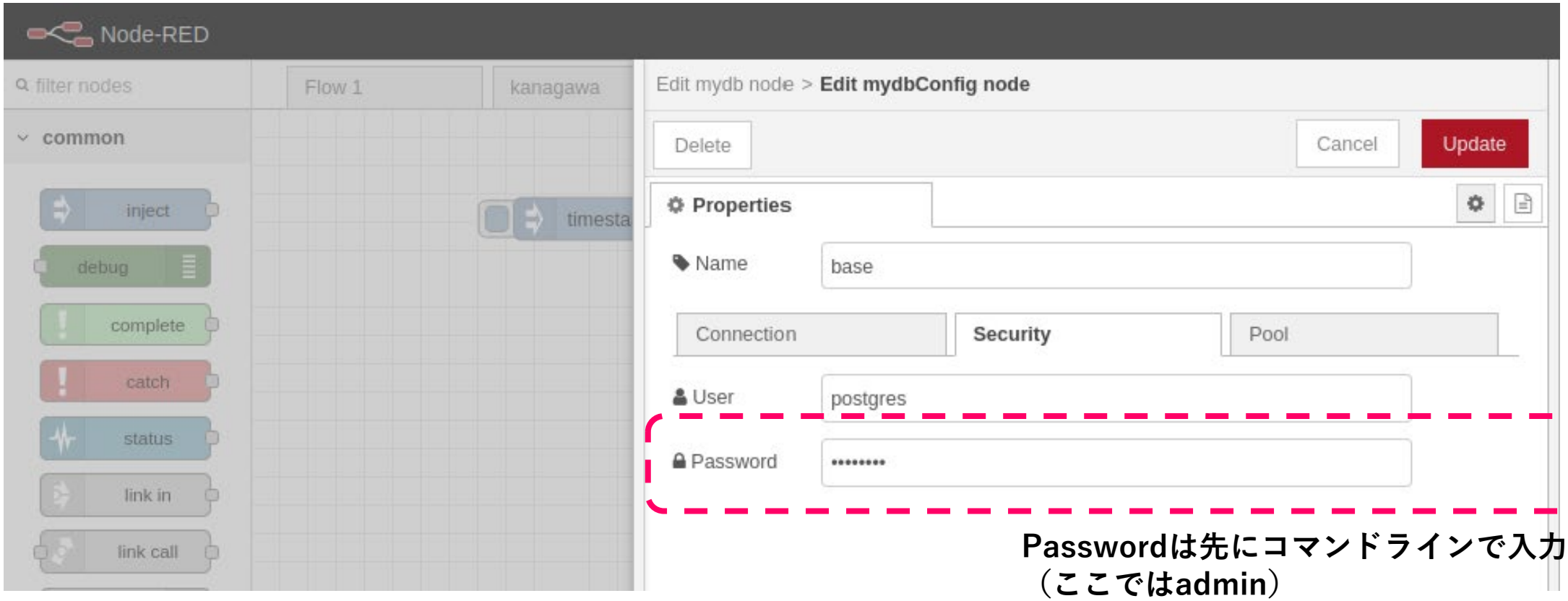


PostgreSQL接続設定



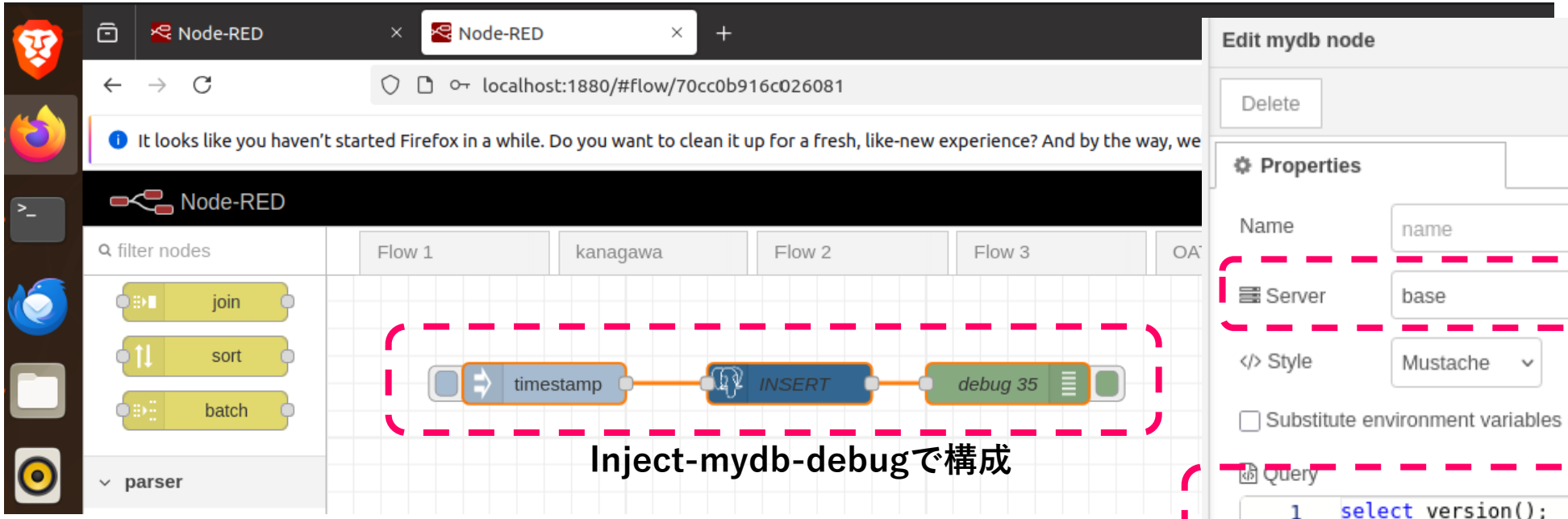
Host, Port, Databaseはデフォルト

PostgreSQL接続設定



Passwordは先にコマンドラインで入力したもの（ここではadmin）

PostgreSQL動作：最小構成

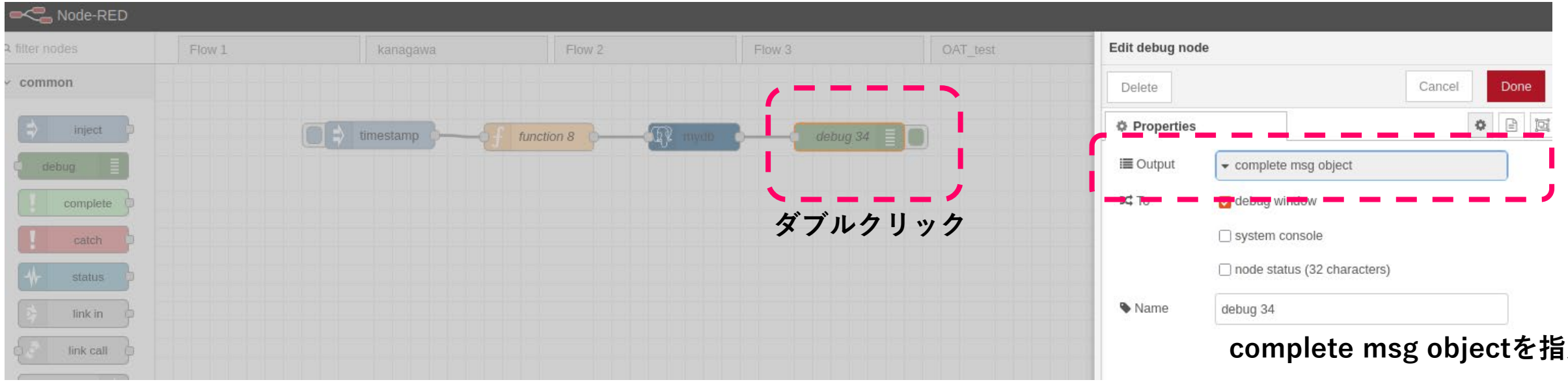


Inject-mydb-debugで構成

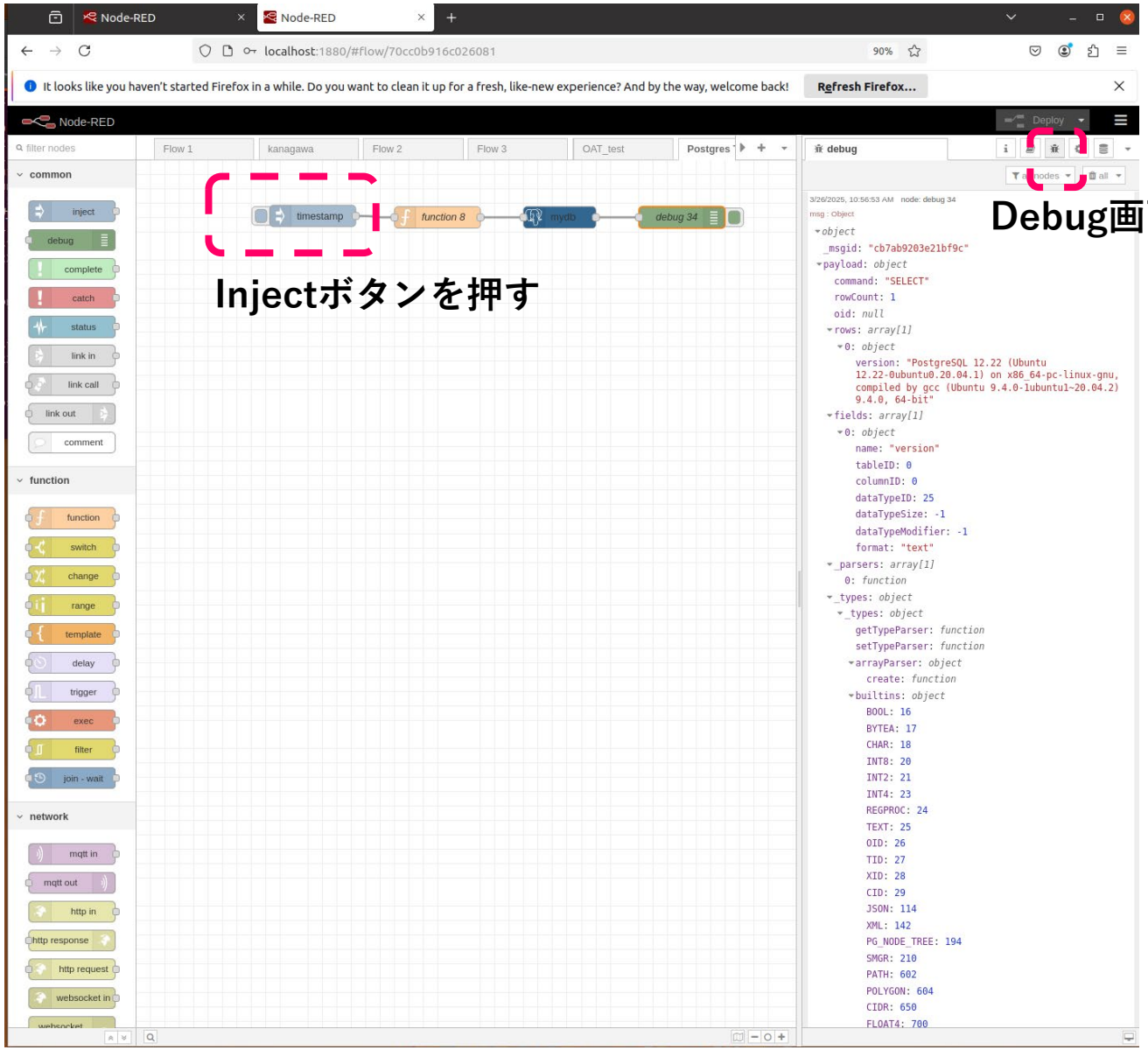
先の設定が引き継がれる

DBを作成していなくても応答するコマンド

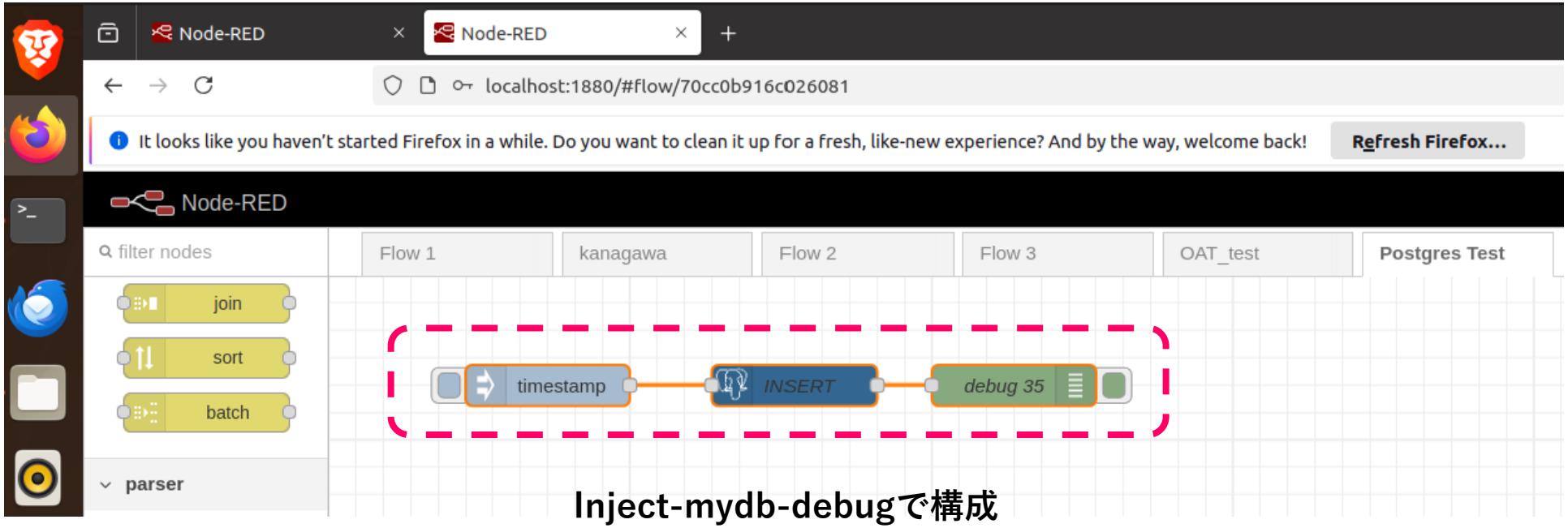
Debug出力をmsgオブジェクト全体にする



PostgreSQL動作させた様子

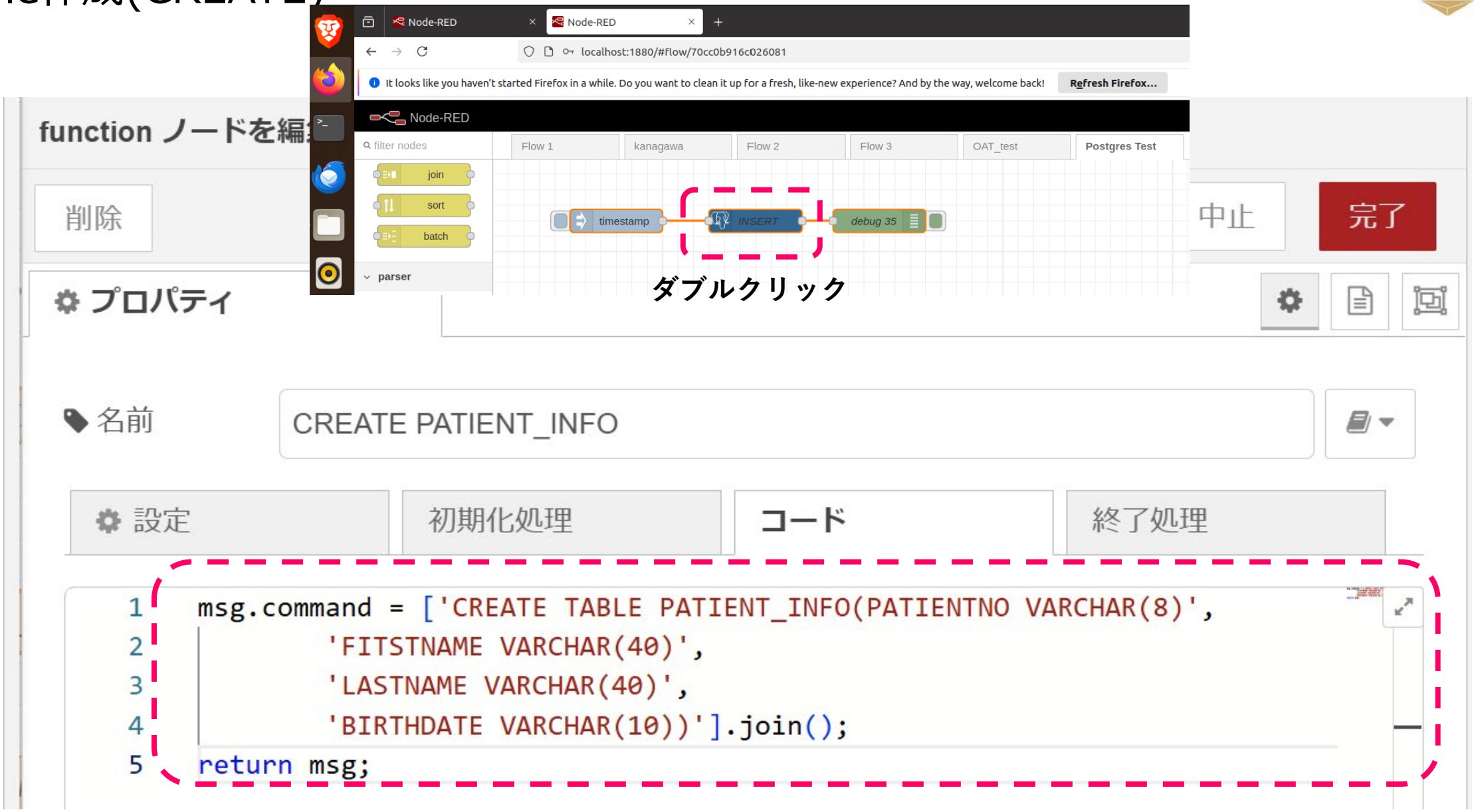


PostgreSQL：テーブル作成(CREATE)

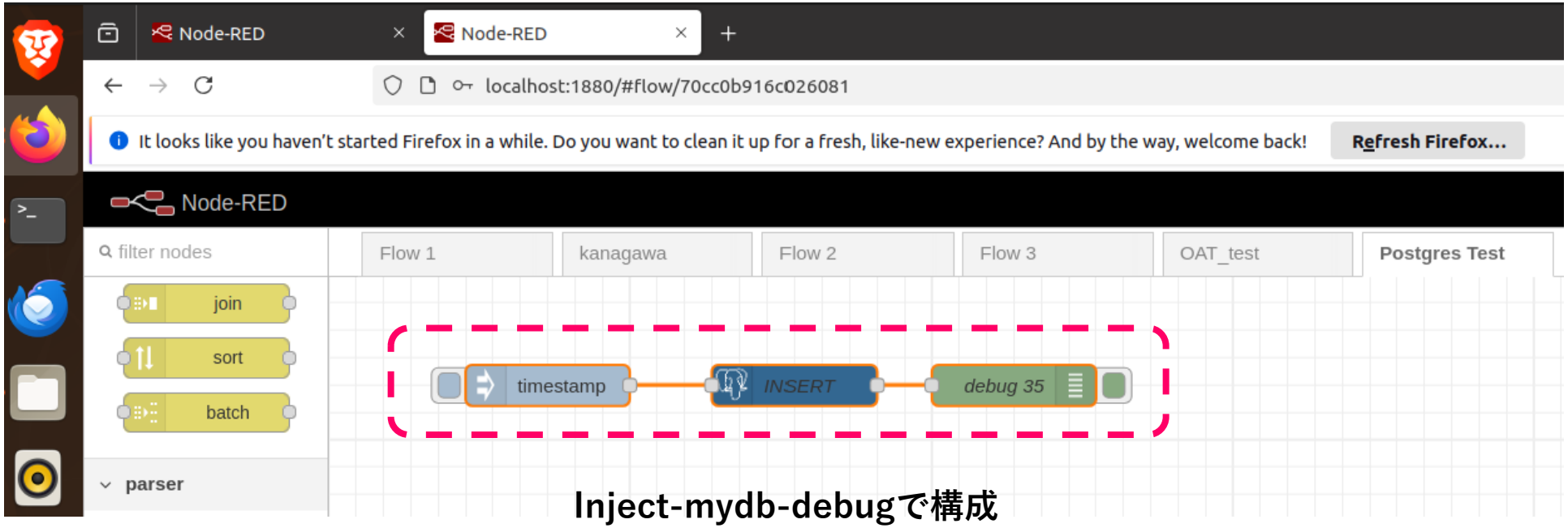


Inject-mydb-debugで構成

Table作成(CREATE)

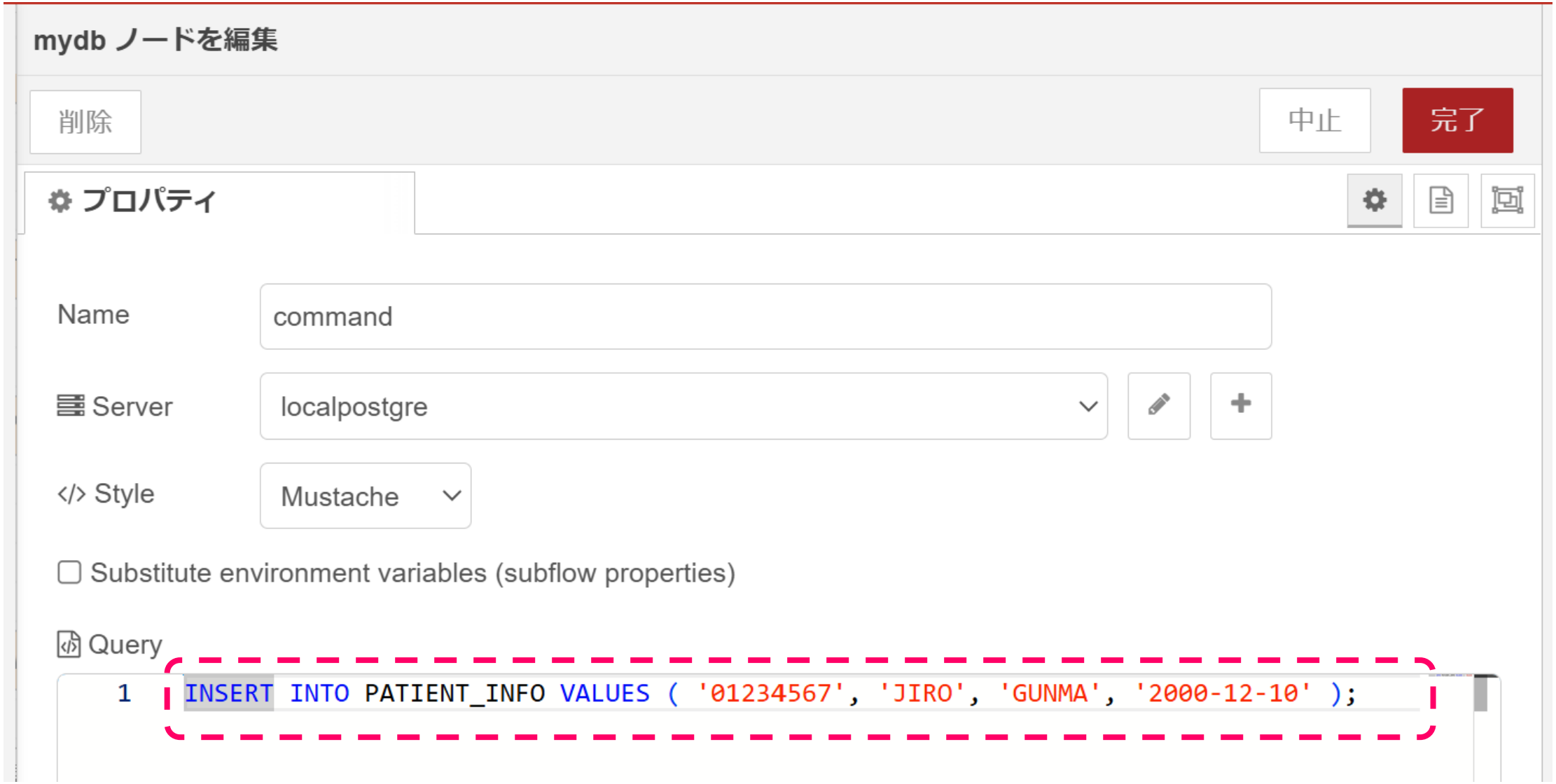


PostgreSQL：レコード作成(INSERT)

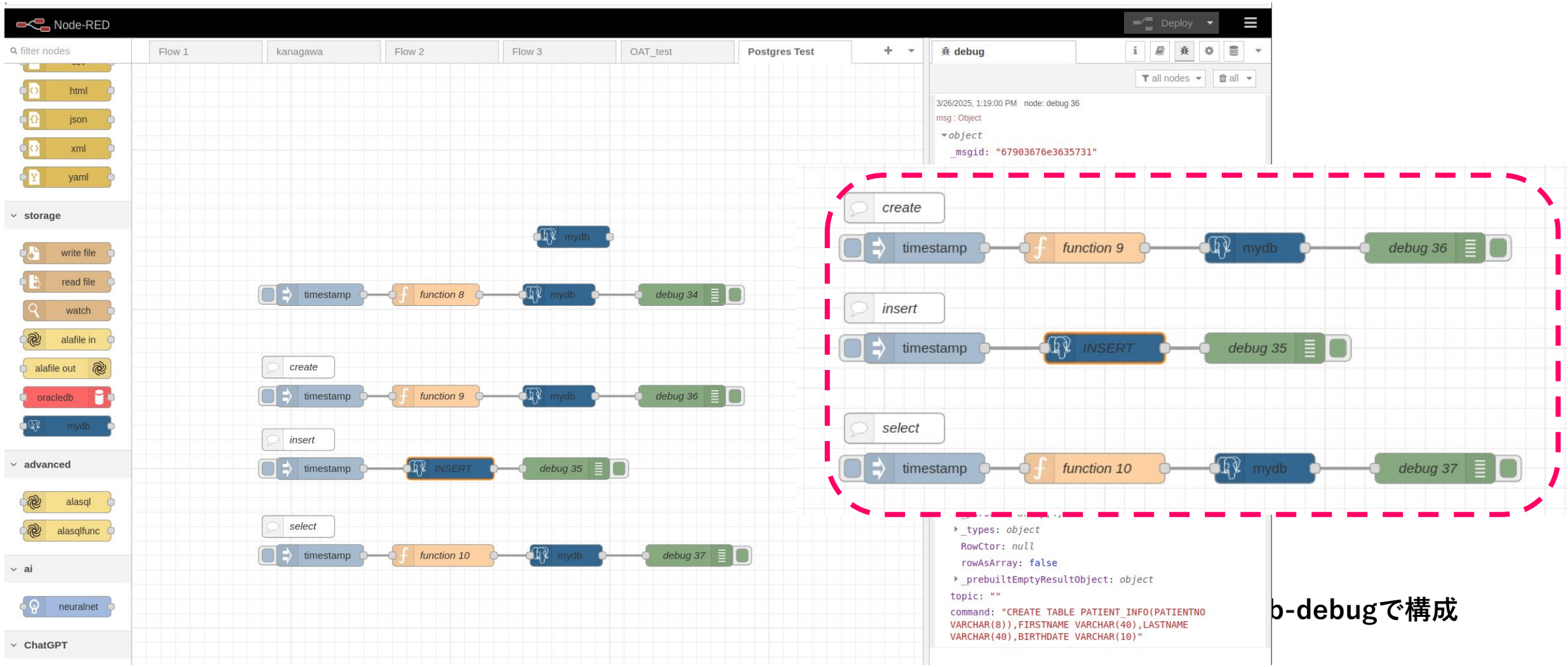


Inject-mydb-debugで構成

PostgreSQL：レコード作成（INSERT）

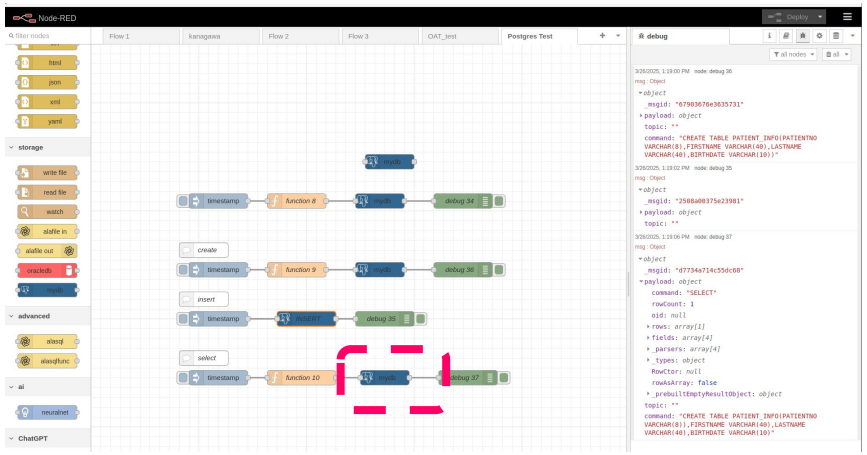


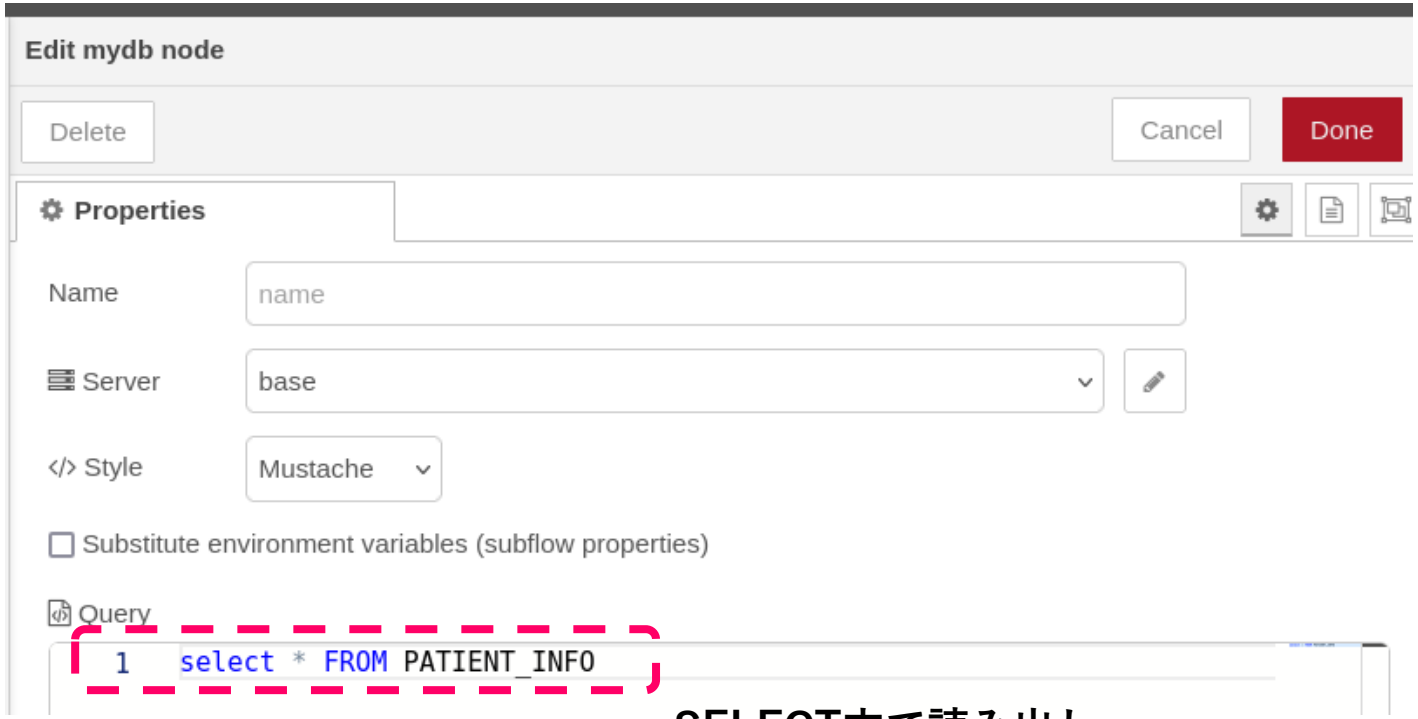
PostgreSQL：レコード作成（INSERT）



b-debugで構成

PostgreSQL：レコード参照（SELECT)





SELECT文で読み出し

出力はJSONオブジェクトになる

End of Document