



# IPCIでWebクロールリングを実装する

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

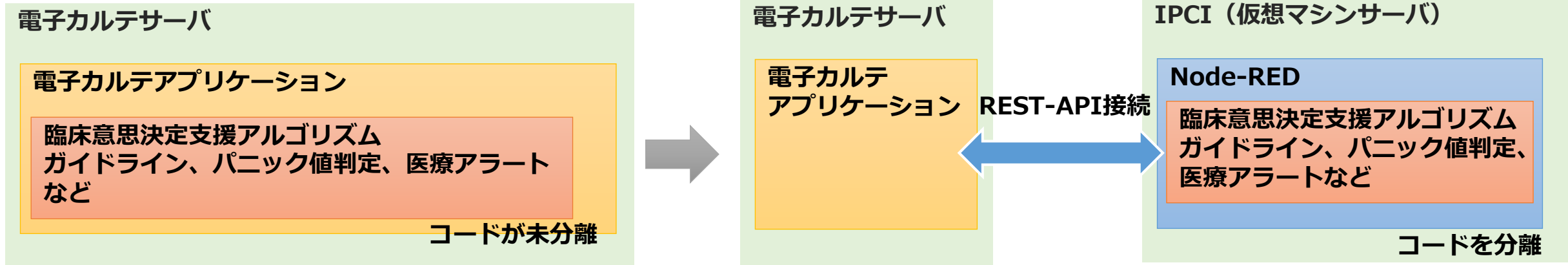
- In-Process Clinical Intelligenceの略
  - 提唱・開発者：鳥飼 幸太（群馬大学医学部附属病院）
  - 臨床意思決定支援(Clinical Decision Support)機能の実装モデルを含む
  - 臨床現場での1次利用を主眼としている
  - 医療で生じる情報判断-伝達手段を併せて提供する
- 
- Vmware/Ubuntuベースの仮想マシン、主にオープンソースで構成（データ）
  - 2022年より日本Mテクノロジー学会から継続的にリソースを提供、講習会を実施している
  - ライセンス形態：2025年現在学術機関および企業での利用において無償（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



▶ トップページ ▶ サイトマップ ▶ リンク

クリニカルパス標準データモデルの開発および利活用  
(略称: ePathプロジェクト)



ePathProjectについて



組織図



セミナーのご案内



ひな型バスダウンロード



お問い合わせ



メンバーサイト

## 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ため、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>

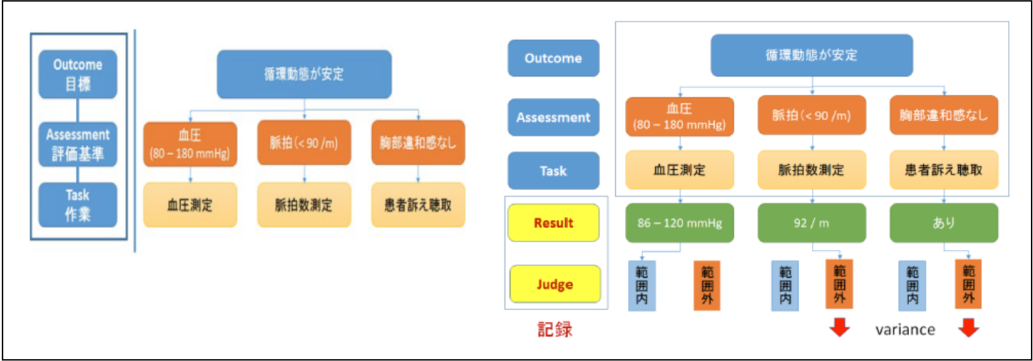
入院時に医師、看護師およびその他の医療従事者が共同して患者の診療計画を作成し、患者に対し交付される文書。病名、主治医、症状、治療計画、手術内容および日程、推定される入院期間等についての記載が必要である。

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＝アセスメント＝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

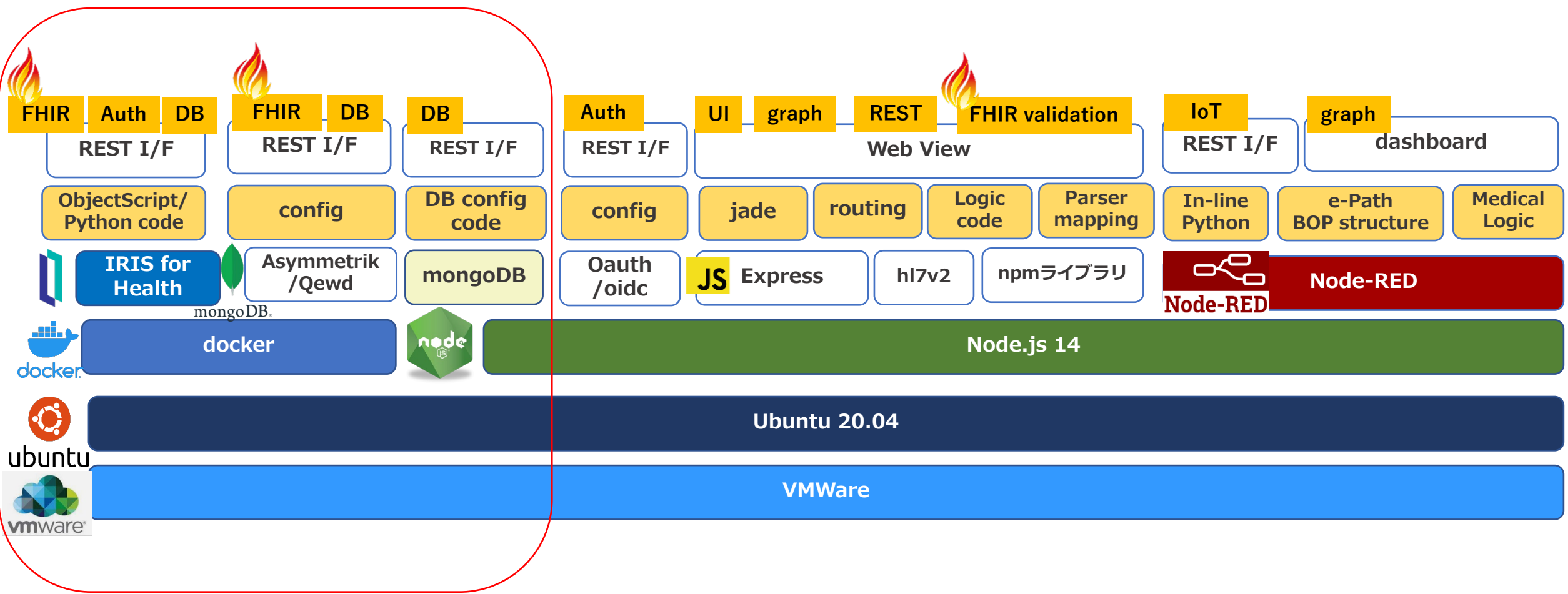
# IPCI機能概要図

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

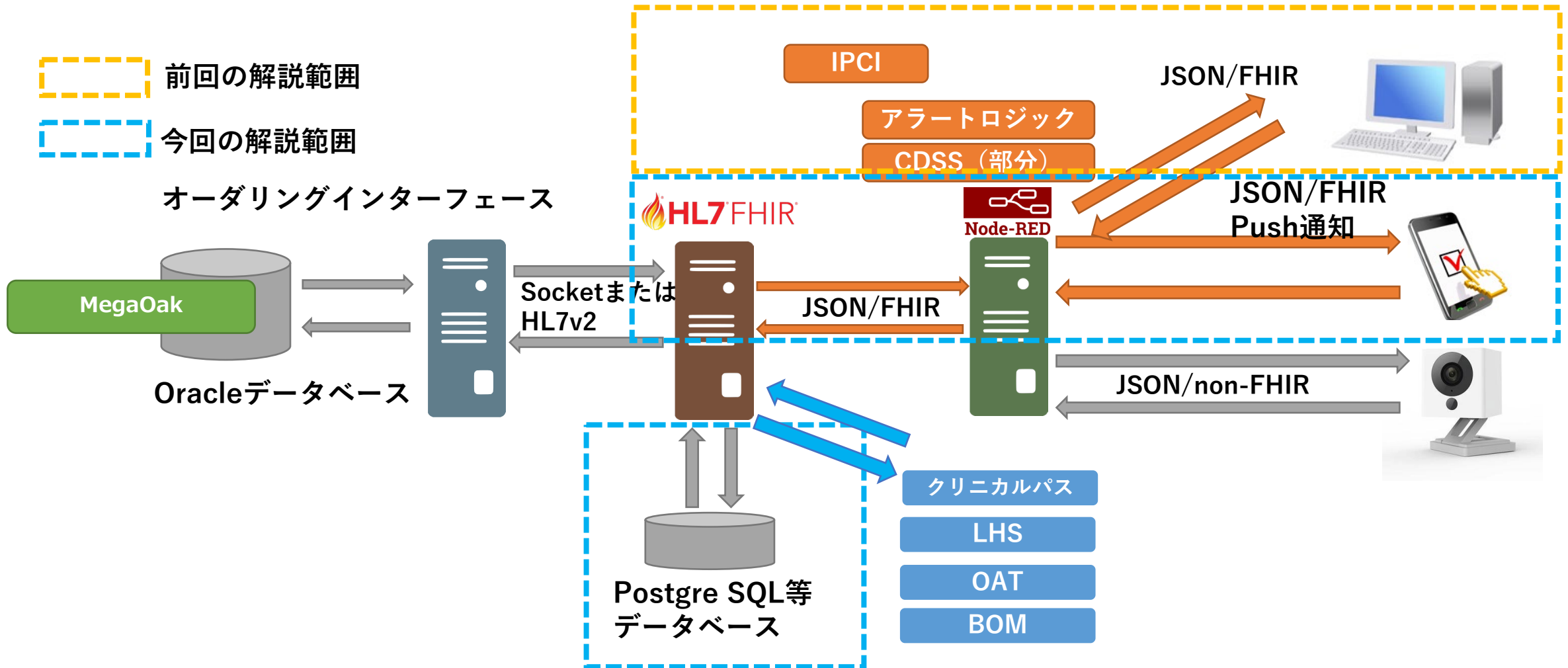
[mta-office@mta.gr.jp](mailto:mta-office@mta.gr.jp)

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 interface showing a flow diagram and a debug console.

The flow diagram consists of two main parts:

- Top Flow:** A sequence of nodes: `timestamp` → `template` → `function 3` → `debug 15`. A branch from `template` connects to `function 4` → `debug 16`.
- Bottom Flow (highlighted with a red dashed box):** A sequence of nodes: `[get] /testapi` → `template` → `http`. This flow is connected to the top flow via an `Inject-http request-debug` node (labeled `debug 17`).

The debug console on the right shows an error message:

```
3/26/2025, 12:46:47 PM node: function 5  
function : (error)  
"ReferenceError: payload is not defined"
```

# Node-RED構成



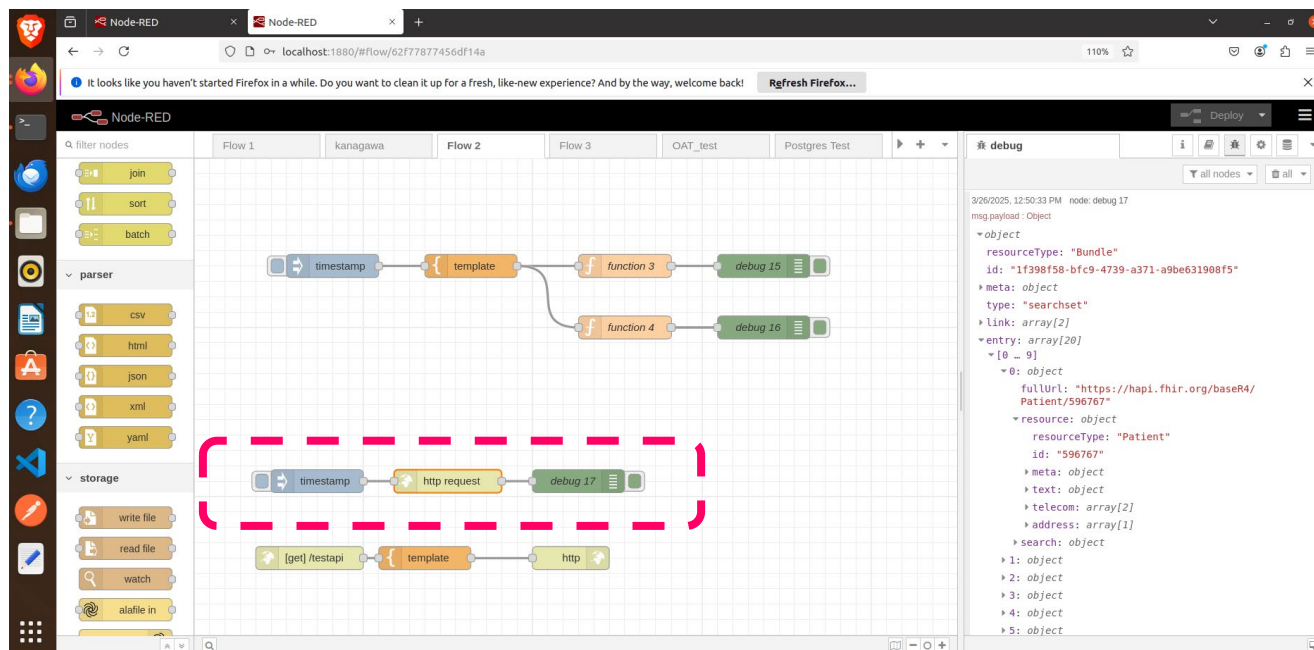
Node-RED interface showing a flow editor and a debug console. The flow editor displays a sequence of nodes: join, sort, batch, parser (csv, html, json, xml, yaml), and storage (write file, read file, watch, alafile in). The debug console shows the output of the flow, including a timestamp and a JSON object.

The "Edit http request node" dialog is open, showing the following configuration:

- Method: GET
- URL: `http://hapi.fhir.org/baseR4/Patient?_format=json&_pretty=true`
- Payload: Ignore
- Return: a parsed JSON object
- Tip: If the JSON parse fails the fetched string is returned as-is.
- Headers: (empty)
- Enabled: ☒

The dialog is highlighted with a red dashed border.

# Node-RED構成



debug

3/26/2025, 12:50:33 PM node: debug 17

msg.payload : Object

```
▼ object
  resourceType: "Bundle"
  id: "1f398f58-bfc9-4739-a371-a9be631908f5"
  meta: object
    type: "searchset"
    link: array[2]
  entry: array[20]
    ▼ [0 ... 9]
      ▼ 0: object
        fullUrl: "https://hapi.fhir.org/baseR4/Patient/596767"
        resource: object
          resourceType: "Patient"
          id: "596767"
          meta: object
          text: object
          telecom: array[2]
          address: array[1]
          search: object
        1: object
        2: object
        3: object
        4: object
        5: object
```

**FHIRデータを取得できた**

# IPCI内のFHIRサーバ起動



Activities Terminal

ホーム × IPCI最新 ×

Activities Terminal

user Purge

Trash

fhirServer

doker\_mongo

Old Firefox Data

Open

Cut

Copy

Rename...

Move to Trash

Properties

Show in Files

Open in Terminal

user@user: ~/デスクトップ/fhirServer

```
user@user:~/デスクトップ/fhirServer$ npm start
> @asymmetrik/node-fhir-server-mongo@2.0.0 start
> nodemon src/index.js

[nodemon] 1.19.4
[nodemon] to restart at any time, enter `rs`
[nodemon] watching dir(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node src/index.js`
```

Node-RED Node-RED Problem loading page

localhost:3000/4\_0\_0/patient?deceased:not=true

http request URIを上記に変更すればよい

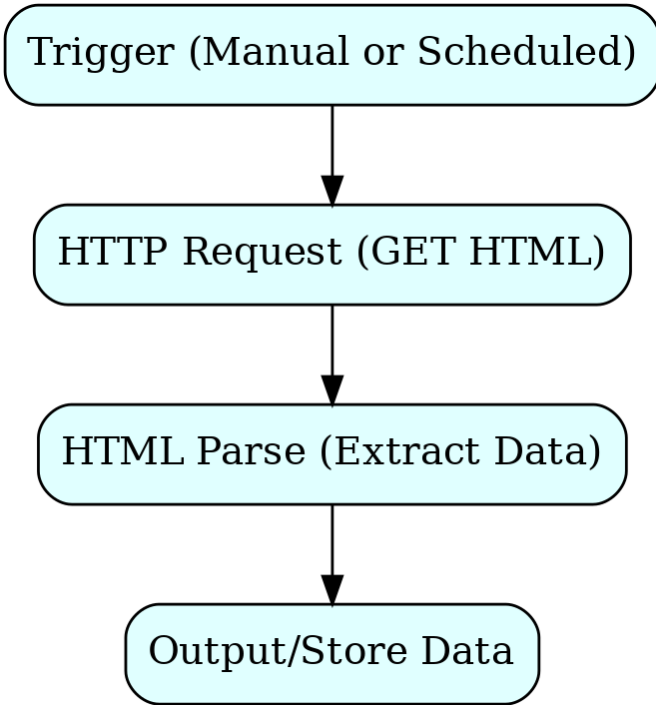


# IPCI(Ubuntu)でのWebクロール導入



- 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](https://www.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、`<span class="node-red-latest-version">`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・多ページクロール・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ルが可能。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ル: 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ルを構成する(再帰的クロール)。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: クロールで得たデータはそのままデバッグ表示だけでなく、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で自動化を進める。

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



```
user@user: ~  
user@user:~$ sudo apt install postgresql postgresql-contrib  
[sudo] user のパスワード:  
パッケージリストを読み込んでいます... 完了  
依存関係ツリーを作成しています  
状態情報を読み取っています... 完了  
postgresql はすでに最新バージョン (12+214ubuntu0.1) です。  
postgresql-contrib はすでに最新バージョン (12+214ubuntu0.1) です。  
以下のパッケージが自動でインストールされましたが、もう必要とされていません:  
gyp javascript-common libc-ares2 libjs-inherits libjs-is-typedarray  
libjs-psl libjs-typedarray-to-buffer libpython2-stdlib libpython2.7-minimal  
libpython2.7-stdlib libssl-dev libuv1-dev node-abbrev node-ajv node-ansi  
node-ansi-align node-ansi-regex node-ansi-styles node-ansistyles node-aproba  
node-archy node-are-we-there-yet node-asap node-asn1 node-assert-plus  
node-asynckit node-aws-sign2 node-aws4 node-balanced-match node-bcrypt-pbkdf  
node-bl node-bluebird node-boxen node-brace-expansion node-builtin-modules  
node-builtins node-cacache node-call-limit node-camelcase node-caseless  
node-chalk node-chownr node-ci-info node-cli-boxes node-cliui node-clone  
node-co node-color-convert node-color-name node-colors node-columnify  
node-combined-stream node-concat-map node-concat-stream node-config-chain  
node-configstore node-console-control-strings node-copy-concurrently  
node-core-util-is node-cross-spawn node-crypto-random-string node-cyclist  
node-dashdash node-debug node-decamelize node-decompress-response  
node-deep-extend node-defaults node-define-properties node-delayed-stream  
node-delegates node-detect-indent node-detect-newline node-dot-prop
```



# PostgreSQL 動作確認



```
user@user: ~  
node-string-decoder (1.2.0-2) を削除しています ...  
node-safe-buffer (5.2.0-1) を削除しています ...  
node-util-deprecate (1.0.2-1) を削除しています ...  
node-strip-ansi (6.0.0-2) を削除しています ...  
node-ansi-regex (5.0.0-1) を削除しています ...  
mime-support (3.64ubuntu1) のトリガを処理しています ...  
gnome-menus (3.36.0-1ubuntu1) のトリガを処理しています ...  
libc-bin (2.31-0ubuntu9.17) のトリガを処理しています ...  
man-db (2.9.1-1) のトリガを処理しています ...  
desktop-file-utils (0.24-1ubuntu3) のトリガを処理しています ...  
user@user:~$ sudo -u postgres psql  
psql (12.22 (Ubuntu 12.22-0ubuntu0.20.04.1))  
Type "help" for help.  
  
postgres=# select version();  
  
[1]+  停止                  sudo -u postgres psql  
user@user:~$ sudo -u postgres psql  
psql (12.22 (Ubuntu 12.22-0ubuntu0.20.04.1))  
Type "help" for help.  
  
postgres=# ALTER ROLE postgres WITH password 'admin';  
ALTER ROLE  
postgres=#
```

# PostgreSQLパスワードの設定



```
user@user: ~  
node-string-decoder (1.2.0-2) を削除しています ...  
node-safe-buffer (5.2.0-1) を削除しています ...  
node-util-deprecate (1.0.2-1) を削除しています ...  
node-strip-ansi (6.0.0-2) を削除しています ...  
node-ansi-regex (5.0.0-1) を削除しています ...  
mime-support (3.64ubuntu1) のトリガを処理しています ...  
gnome-menus (3.36.0-1ubuntu1) のトリガを処理しています ...  
libc-bin (2.31-0ubuntu9.17) のトリガを処理しています ...  
man-db (2.9.1-1) のトリガを処理しています ...  
desktop-file-utils (0.24-1ubuntu3) のトリガを処理しています ...  
user@user:~$ sudo -u postgres psql  
psql (12.22 (Ubuntu 12.22-0ubuntu0.20.04.1))  
Type "help" for help.  
  
postgres=# select version();  
  
[1]+ 停止                  sudo -u postgres psql  
user@user:~$ sudo -u postgres psql  
psql (12.22 (Ubuntu 12.22-0ubuntu0.20.04.1))  
Type "help" for help.  
  
postgres=# ALTER ROLE postgres WITH password 'admin';  
ALTER ROLE  
postgres=#
```

# Node-RED側作業：PosgreSQLを呼べるツールのインストール



User Settings

View: Nodes Install

Palette

Keyboard

Search: mydb 2 / 5306

@yuliqi/node-red-mydb  
A Node-RED node to read and write to a My database  
0.0.9 2 years, 5 months ago install

@open-kappa/node-red-contrib-mydb  
Simple DB access node for node-red  
1.5.0 3 years, 10 months ago install

Node-RED

Flow 1

filter nodes

join sort batch

parser

csv html json xml yaml

storage

write file read file watch alafile in alafile out oracledb mydb

mydb

# PostgreSQL接続設定



The screenshot shows the Node-RED web interface. On the left, the 'common' and 'function' node palettes are visible. The main workspace shows a flow named 'kanagawa' with a 'timestamp' node. On the right, the 'Edit mydb node > Edit mydbConfig node' dialog is open. The 'Properties' tab is selected, showing the following configuration:

- Name: base
- Connection tab: Host (127.0.0.1), Port (5432), Database (postgres)
- Use SSL: ☐
- Certificates section:
  - Reject unauthorized: ☐
  - Root CA: /path/to/root.crt
  - Client key: /path/to/postgres.key
  - Client: /path/to/postgres.crt

A red dashed line highlights the 'Name', 'Host', 'Port', and 'Database' fields.

Host, Port, Databaseはデフォルト

# PostgreSQL接続設定



The screenshot shows the Node-RED web interface. On the left, the 'common' node palette is visible with nodes like inject, debug, complete, catch, status, link in, and link call. The main workspace shows a flow named 'Flow 1' with a 'timesta' node. On the right, the 'Edit mydb node > Edit mydbConfig node' dialog is open. It has tabs for 'Properties', 'Connection', 'Security', and 'Pool'. The 'Properties' tab is active, showing a 'Name' field with the value 'base'. Below this are three sub-tabs: 'Connection', 'Security', and 'Pool'. The 'Security' sub-tab is active, showing a 'User' field with the value 'postgres' and a 'Password' field with masked characters '.....'. A red dashed rectangle highlights the 'User' and 'Password' fields. At the bottom right of the dialog, there are buttons for 'Delete', 'Cancel', and 'Update'.

Node-RED

Flow 1 kanagawa

common

inject

debug

complete

catch

status

link in

link call

timesta

Edit mydb node > Edit mydbConfig node

Delete Cancel Update

Properties

Name base

Connection Security Pool

User postgres

Password .....

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



Node-RED interface showing a flow named 'kanagawa' with three nodes: 'timestamp', 'INSERT', and 'debug 35'. The flow is highlighted with a red dashed box.

Inject-mydb-debugで構成

Edit mydb node configuration panel. The 'Name' field is 'name', 'Server' is 'base', and 'Style' is 'Mustache'. The 'Query' field contains '1 select version();'. The configuration is highlighted with a red dashed box.

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



The screenshot shows the Node-RED web interface. A flow named 'kanagawa' is active, containing nodes: 'timestamp', 'function 8', 'mydb', and 'debug 34'. The 'debug 34' node is highlighted with a red dashed box and the text 'ダブルクリック' (Double Click) below it. To the right, the 'Edit debug node' panel is open, also with a red dashed box around it. In this panel, the 'Output' dropdown is set to 'complete msg object', and the 'debug window' checkbox is checked. The text 'complete msg objectを指定' (Specify complete msg object) is written below the panel.

Node-RED

filter nodes

Flow 1 kanagawa Flow 2 Flow 3 OAT\_test

common

- inject
- debug
- complete
- catch
- status
- link in
- link call

timestamp function 8 mydb debug 34

ダブルクリック

Edit debug node

Delete Cancel Done

Properties

Output complete msg object

debug window

system console

node status (32 characters)

Name debug 34

complete msg objectを指定



# PostgreSQL動作させた様子



Injectボタンを押す

Debug画面を選択

Versionが出力された



# PostgreSQL : テーブル作成(CREATE)



The screenshot shows the Node-RED web interface in a browser window. The address bar indicates the URL is `localhost:1880/#flow/70cc0b916c026081`. A notification bar at the top states: "It looks like you haven't started Firefox in a while. Do you want to clean it up for a fresh, like-new experience? And by the way, welcome back!" with a "Refresh Firefox..." button. The Node-RED interface shows a workspace with a flow named "Postgres Test" selected. The flow contains three nodes connected in sequence: a "timestamp" node, an "INSERT" node, and a "debug 35" node. These three nodes are enclosed in a dashed pink rectangular box. The left sidebar shows a "filter nodes" search bar and a list of nodes including "join", "sort", "batch", and a "parser" section. The top of the workspace has tabs for "Flow 1", "kanagawa", "Flow 2", "Flow 3", "OAT\_test", and "Postgres Test".

Inject-mydb-debugで構成

# Table作成(CREATE)



function ノードを編集

削除

プロパティ

名前

CREATE PATIENT\_INFO

設定 初期化处理 コード 終了処理

```
1 msg.command = ['CREATE TABLE PATIENT_INFO(PATIENTNO VARCHAR(8)',  
2   'FITSTNAME VARCHAR(40)',  
3   'LASTNAME VARCHAR(40)',  
4   'BIRTHDATE VARCHAR(10))'].join();  
5 return msg;
```

Node-RED

localhost:1880/#flow/70ccb916c026081

It looks like you haven't started Firefox in a while. Do you want to clean it up for a fresh, like-new experience? And by the way, welcome back! Refresh Firefox...

Node-RED

filter nodes

Flow 1 kanagawa Flow 2 Flow 3 OAT\_test Postgres Test

join

sort

batch

timestamp

INSERT

debug 35

ダブルクリック

中止

完了

設定

初期化处理

コード

終了処理

# PostgreSQL : レコード作成(INSERT)



The screenshot shows the Node-RED web interface in a Firefox browser. The address bar shows 'localhost:1880/#flow/70cc0b916c026081'. A notification bar at the top says 'It looks like you haven't started Firefox in a while. Do you want to clean it up for a fresh, like-new experience? And by the way, welcome back! Refresh Firefox...'. The Node-RED interface has a sidebar on the left with a search bar 'filter nodes' and a list of nodes: 'join', 'sort', 'batch', and 'parser'. The main workspace shows a flow named 'Postgres Test' with three nodes connected in sequence: 'timestamp', 'INSERT', and 'debug 35'. The 'INSERT' node is highlighted with a red dashed box. Below the flow, the text 'Inject-mydb-debugで構成' is displayed.

Inject-mydb-debugで構成

# PostgreSQL : レコード作成 (INSERT)



mydb ノードを編集

削除 中止 完了

プロパティ

Name command

Server localpostgre

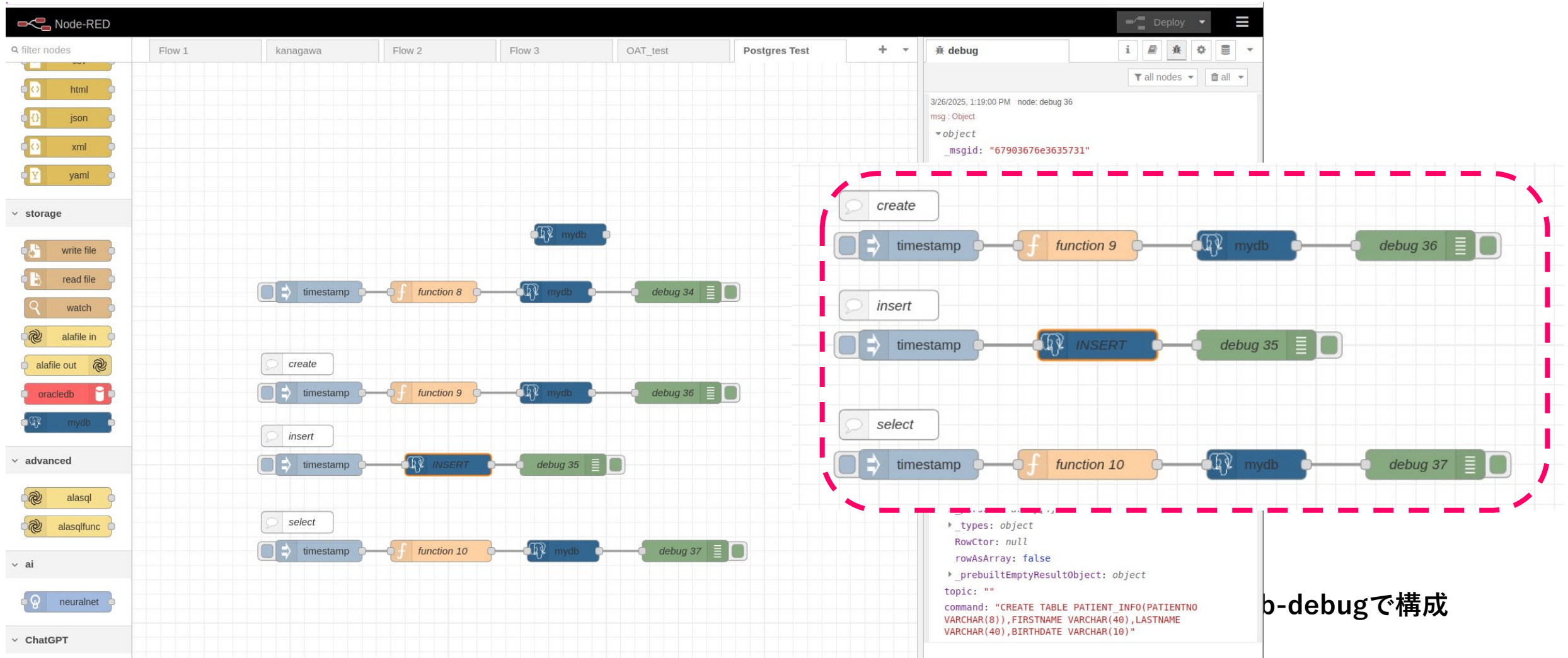
Style Mustache

☐ Substitute environment variables (subflow properties)

Query

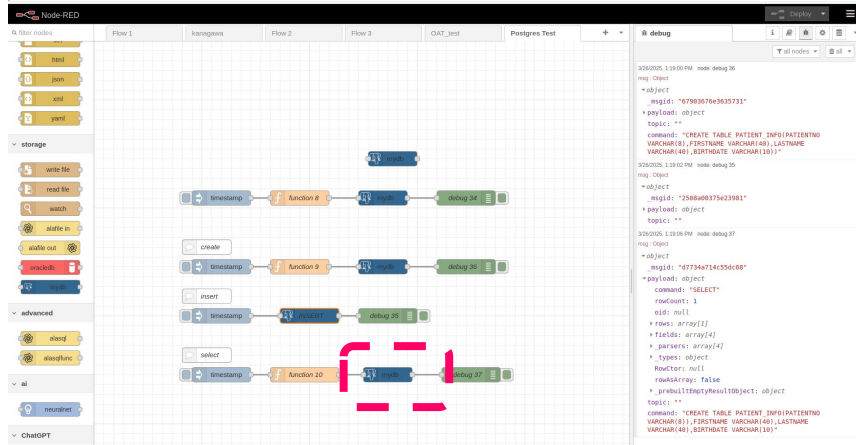
```
1 INSERT INTO PATIENT_INFO VALUES ( '01234567', 'JIRO', 'GUNMA', '2000-12-10' );
```

# PostgreSQL : レコード作成 (INSERT)



b-debugで構成

# PostgreSQL : レコード参照 (SELECT)



**Edit myddb node**

Delete Cancel Done

**Properties**

Name: name

Server: base

Style: Mustache

☐ Substitute environment variables (subflow properties)

**Query**

1 select \* FROM PATIENT\_INFO

SELECT文で読み出し

debug

3/26/2025, 1:19:00 PM node: debug 36

msg: Object

object

\_msgid: "67903676e3635731"

payload: object

topic: ""

command: "CREATE TABLE PATIENT\_INFO(PATIENTNO VARCHAR(8), FIRSTNAME VARCHAR(40), LASTNAME VARCHAR(40), BIRTHDATE VARCHAR(10))"

3/26/2025, 1:19:02 PM node: debug 35

msg: Object

object

\_msgid: "2508a00375e23981"

payload: object

topic: ""

3/26/2025, 1:19:06 PM node: debug 37

msg: Object

object

\_msgid: "d7734a714c55dc68"

payload: object

command: "SELECT"

rowCount: 1

oid: null

rows: array[1]

fields: array[4]

\_parsers: array[4]

\_types: object

RowCtor: null

rowAsArray: false

\_prebuiltEmptyResultObject: object

topic: ""

command: "CREATE TABLE PATIENT\_INFO(PATIENTNO VARCHAR(8), FIRSTNAME VARCHAR(40), LASTNAME VARCHAR(40), BIRTHDATE VARCHAR(10))"

出力はJSONオブジェクトになる



# End of Document

---