

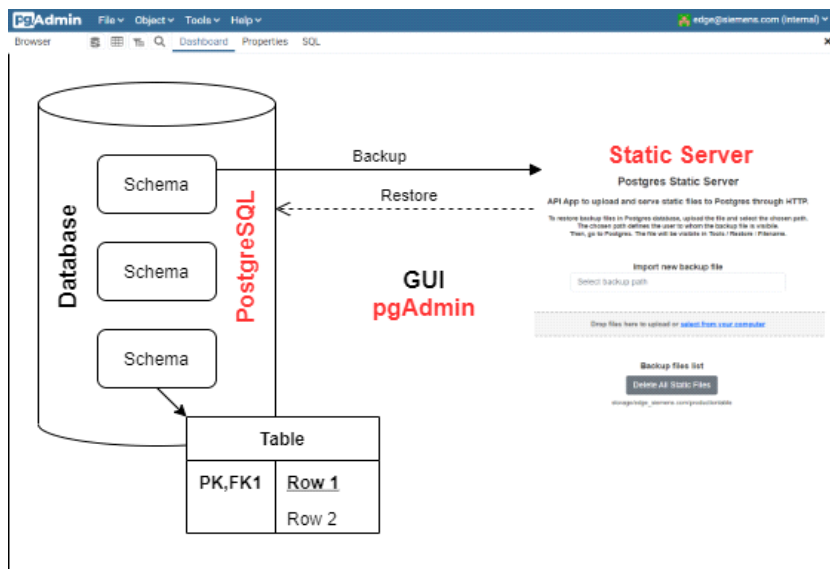
1 - Introduzione

giovedì 5 agosto 2021 16:42

L'applicazione **edge-postgresql** consente di gestire, direttamente su Edge Device, database relazionali ad oggetti, non solo in termini di **costruzione e gestione di basi di dati**, ma anche in termini di **backup e restore di dati** precedentemente archiviati localmente.

Come evidenziato dalla figura sottostante, l'applicazione è costituita da tre diversi microservizi, descritti più in dettaglio nei prossimi paragrafi:

- edge-postgresql
- edge-pgadmin
- postgresql-static-server



Prima di iniziare

Questa guida descrive come installare ed utilizzare l'applicazione **edge-postgresql**.

Prima di procedere all'installazione dell'applicazione, consultare i **requisiti** necessari al paragrafo: [2 - Requisiti App](#).

I dettagli relativi alla procedura da seguire per l'**installazione** dell'applicazione, sono disponibili al seguente paragrafo: [3 - Installazione App](#).

Per tutti i dettagli sull'**utilizzo** delle applicazioni pgAdmin e Postgres Static Server consultare invece il capitolo [4 - Utilizzo App](#).

L'applicazione viene fornita insieme ad un **esempio applicativo**, descritto al paragrafo [5 - Esempio Applicativo](#), che mostra come poter gestire un flusso di dati con database PostgreSQL tramite l'applicazione SIMATIC Flow Creator (Node-RED) ed il nodo dedicato **node-red-contrib-postgres-variable**. Verrà inoltre mostrato come poter visualizzare i dati letti tramite dashboard di SIMATIC Flow Creator e tramite una dashboard dedicata all'interno dell'applicazione **Grafana**.

Per maggiori informazioni sull'installazione del nodo Node-RED dedicato a PostgreSQL consultare il paragrafo [A - Installazione nodo NodeRED PostgreSQL](#).

2 - Requisiti App

giovedì 5 agosto 2021 16:43

Requisiti Hardware

- L'applicazione **edge-postgresql** è compatibile solo con dispositivi **SIEMENS** dotati di funzionalità **Industrial Edge** abilitata.

Requisiti Software

- L'applicazione **edge-postgresql** necessita di **1430 MB** di **RAM** per il funzionamento, così divisa tra i vari microservizi:

Nome servizio	Limite Memoria
<i>edge-postgresql</i>	1024 MB
<i>edge-pgadmin</i>	256 MB
<i>postgresql-static-server</i>	150 MB

Nota: Questi limiti sono stati fissati per un volume di dati di media intensità nel database PostgreSQL, ma possono essere modificati in base alle proprie esigenze agendo sul file docker-compose (vedi paragrafo [6 - Costruzione App](#)) e quindi sulla configurazione dell'app tramite software Edge App Publisher, creando una versione personalizzata di questa applicazione.

3 - Installazione App

giovedì 5 agosto 2021 16:43

L'applicazione **edge-postgresql** viene fornita con il pacchetto app precostruito **edge-postgresql_x.x.x.app** (in base alla versione fornita x.x.x), pacchetto che può essere installato specificamente sugli Edge Device utilizzando **SIMATIC Edge**.

Prerequisiti

- Un **Edge Device** correttamente configurato e con onboarding effettuato su un sistema **Industrial Edge Management (IEM)** accessibile.
- Connessione ad un **Docker Engine** nel caso in cui si utilizzi il software **Industrial Edge App Publisher** per modificare/pubblicare l'applicazione.

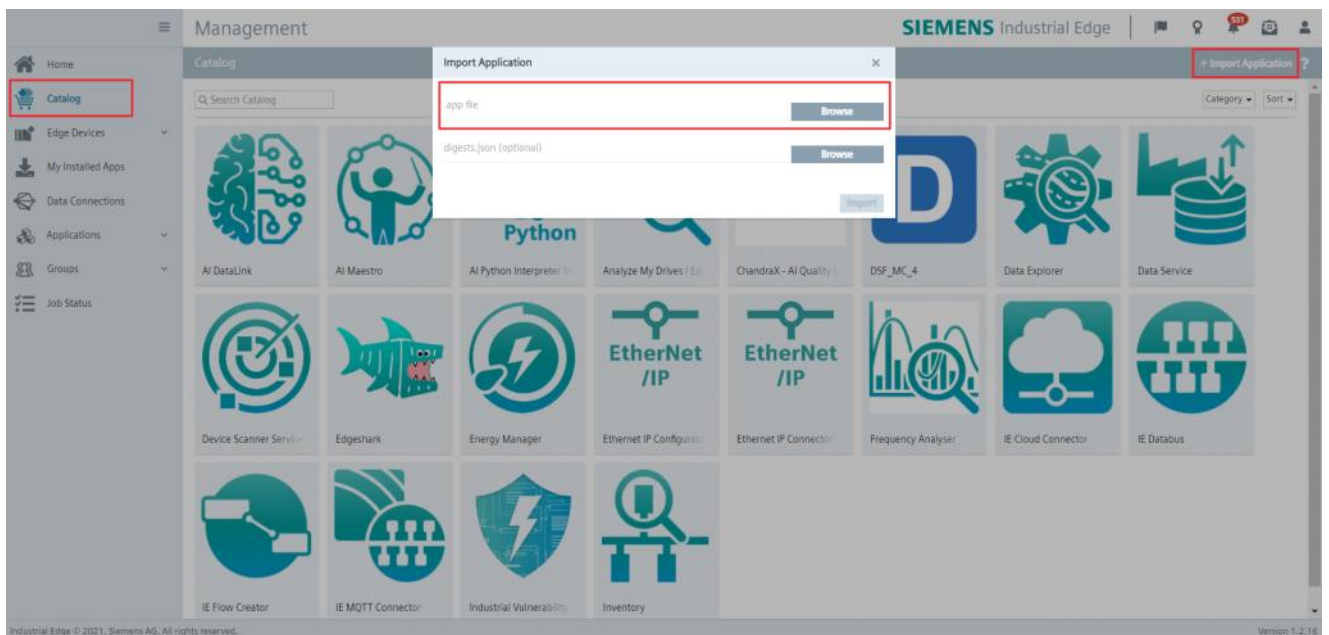
Caricamento App su IEM

Per caricare l'applicazione su IEM, è possibile utilizzare due diversi metodi:

- Import del file .app direttamente su portale IEM;
- Import del file .app su Industrial Edge Publisher e successivo caricamento dell'applicazione da Industrial Edge Publisher al portale IEM.

A - Import file app in IEM Portal

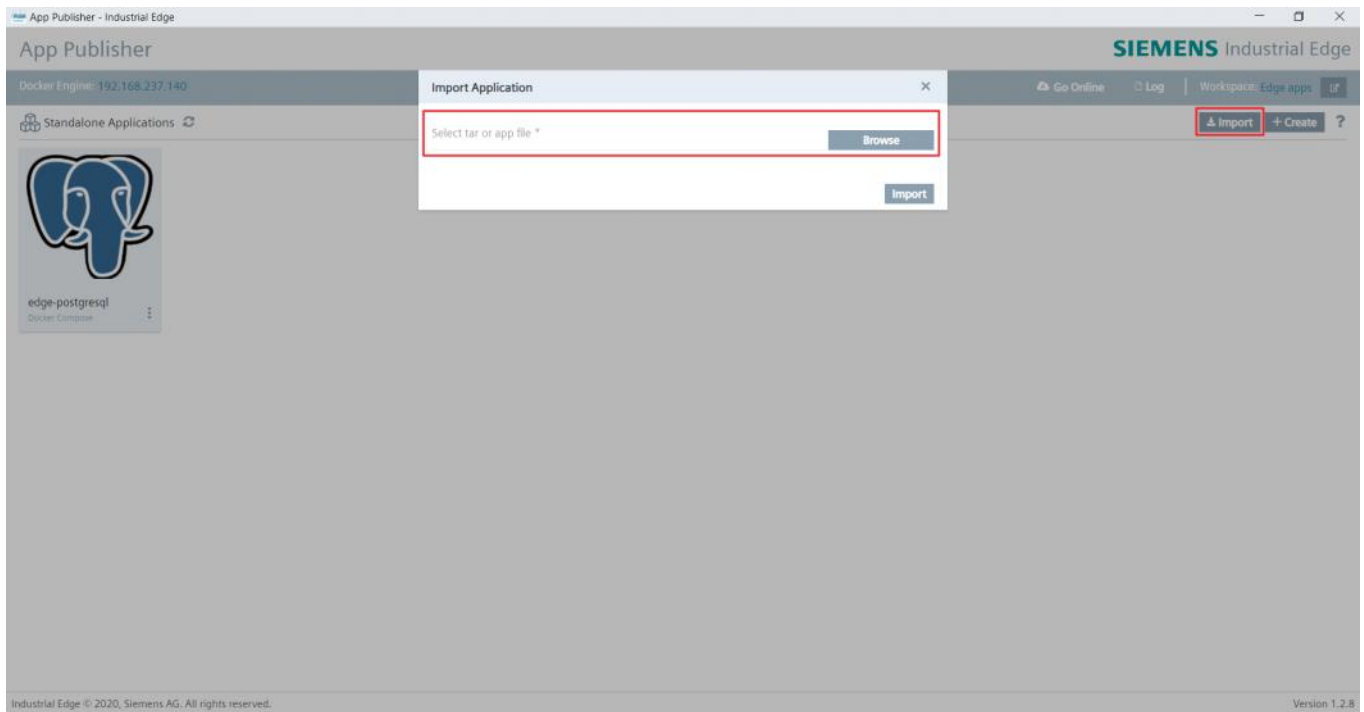
1. Copiare il file scaricato di **edge-postgresql_x.x.x.app** (in base alla versione x.x.x) sul proprio PC.
2. Aprire la pagina web **IEM Portal** del sistema **Industrial Edge Management** che controlla gli Edge Devices su cui si intende installare l'applicazione.
3. Nella sezione **Catalog** premere **+ Import Application** e scegliere il file da caricare tramite il tasto **Browse**.



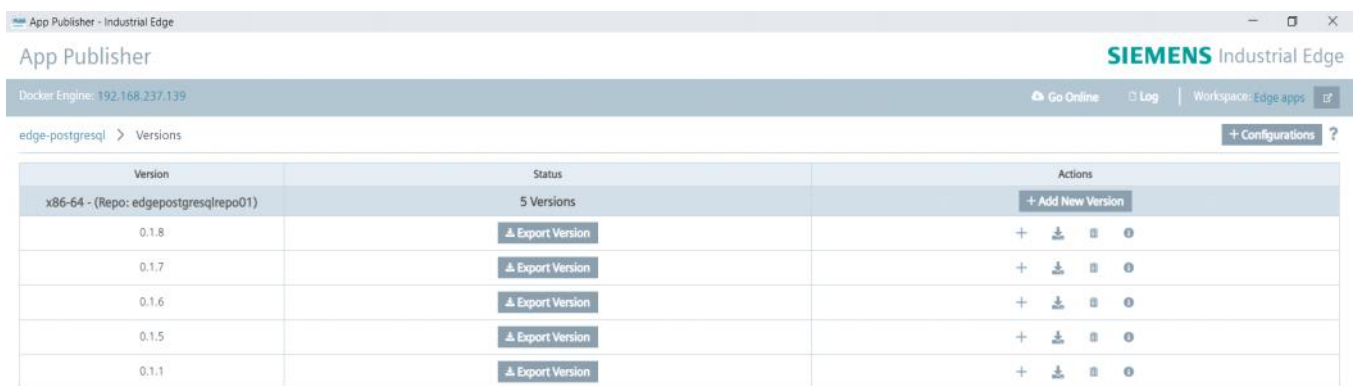
4. Attendere fino a completo caricamento del file relativo all'applicazione. E' possibile verificare lo stato tramite il tasto **Tasks** in alto a destra.

B - Import file app in Edge App Publisher

1. Copiare il file scaricato di **edge-postgresql_x.x.x.app** (in base alla versione x.x.x) sul proprio PC.
2. Aprire il software **Industrial Edge App Publisher**.
3. Premere il tasto **Import** e selezionare il file .app da importare tramite il tasto **Browse**.



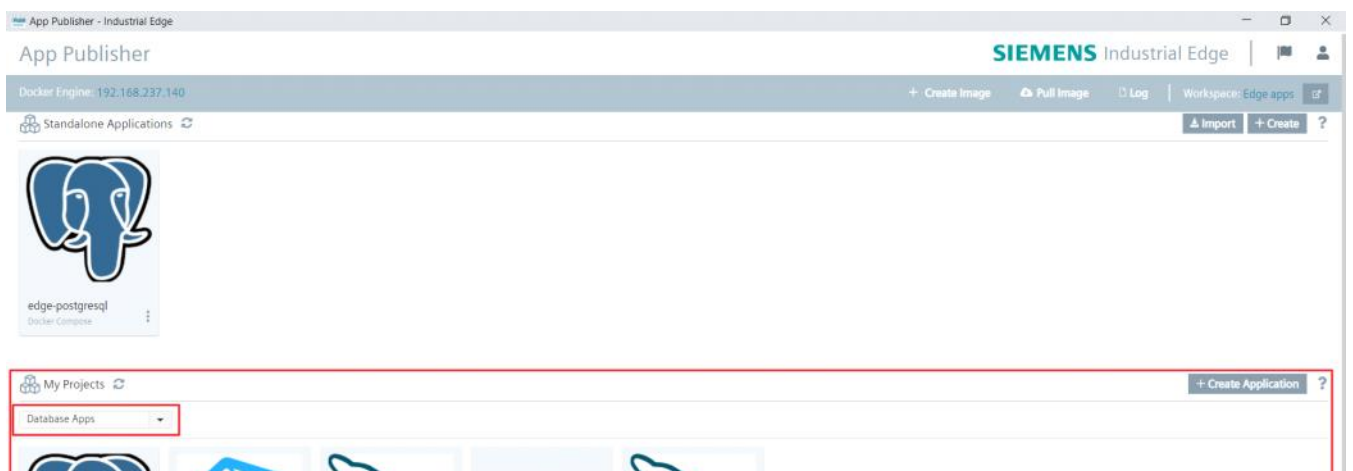
4. Attendere fino al completo caricamento dell'applicazione.
5. Al termine del caricamento, l'applicazione sarà visibile nella sezione **Standalone Applications** del software Industrial Edge App Publisher. E' possibile anche creare una nuova versione dell'applicazione nel menu repository dedicato, cliccando sull'icona dell'applicazione.

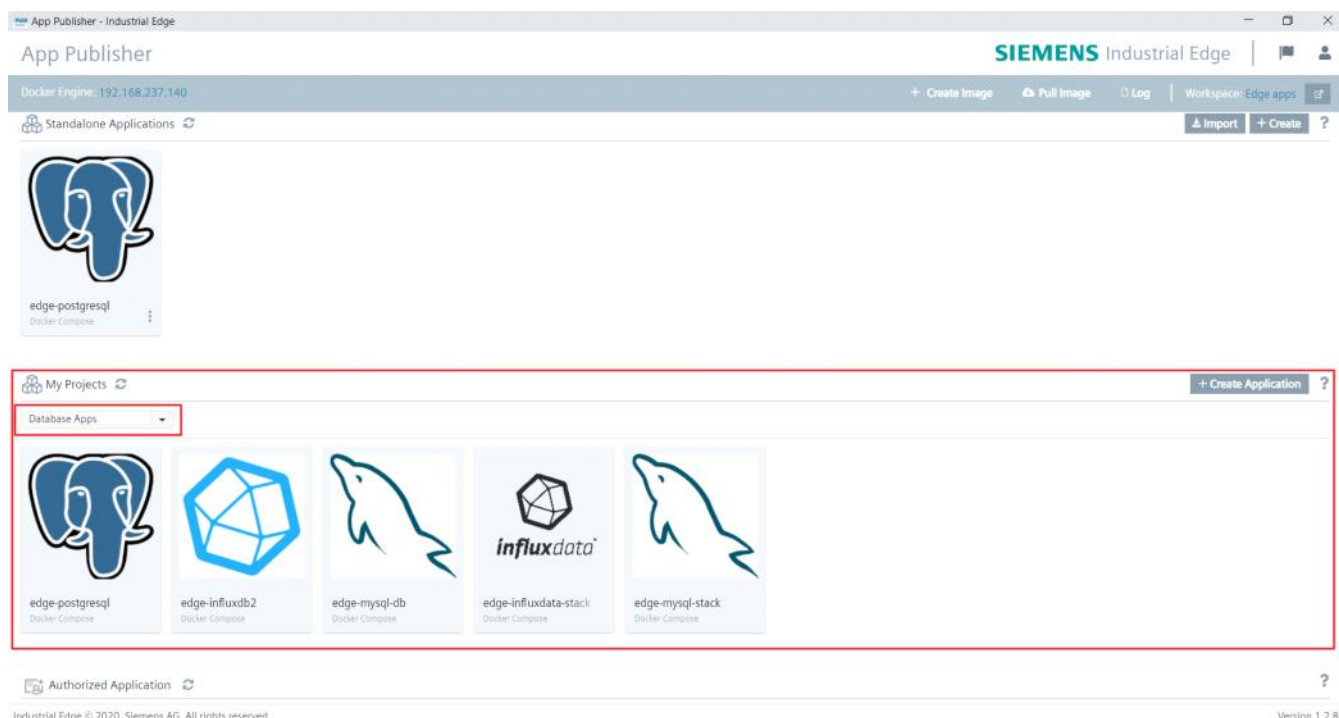


6. Connettersi al proprio IEM tramite il tasto **Go Online**, inserendo l'indirizzo *https* del proprio IEM e premendo su **Connect**. Verrà richiesto il login da parte dell'utente.



7. Una volta connessi, appariranno i progetti (**My Projects**) e le applicazioni (**Authorized Application**) attualmente su IEM. Scegliere un progetto esistente, oppure creare un nuovo progetto dove poter inserire l'applicazione fornita.





8. Aprire il menu della repository dell'app edge-postgresql accedendo alla sezione **Standalone Application** e cliccando l'icona dell'applicazione.
9. Premere il tasto **Import Version to Edge Management** per importare l'applicazione su IEM.



10. Selezionare un progetto su cui caricare l'applicazione e premere **Create** per eseguire il task di import della applicazione su IEM. Attendere il completamento del task. Lo stato del task è visibile nella sezione **Tasks**.

Import Version

Select projects

Database Apps

Select Category

Other

Website

www.siemens.it

☐ Use Edge Core Auth Service (optional)

Do not use Edge Core Auth Service.

☐ External Configurator

No external configurator.

Version

Major

0

Minor

1

Maintenance

8

Release Notes (optional)

Endpoint from internal pgAdmin --> [core-name]:5433

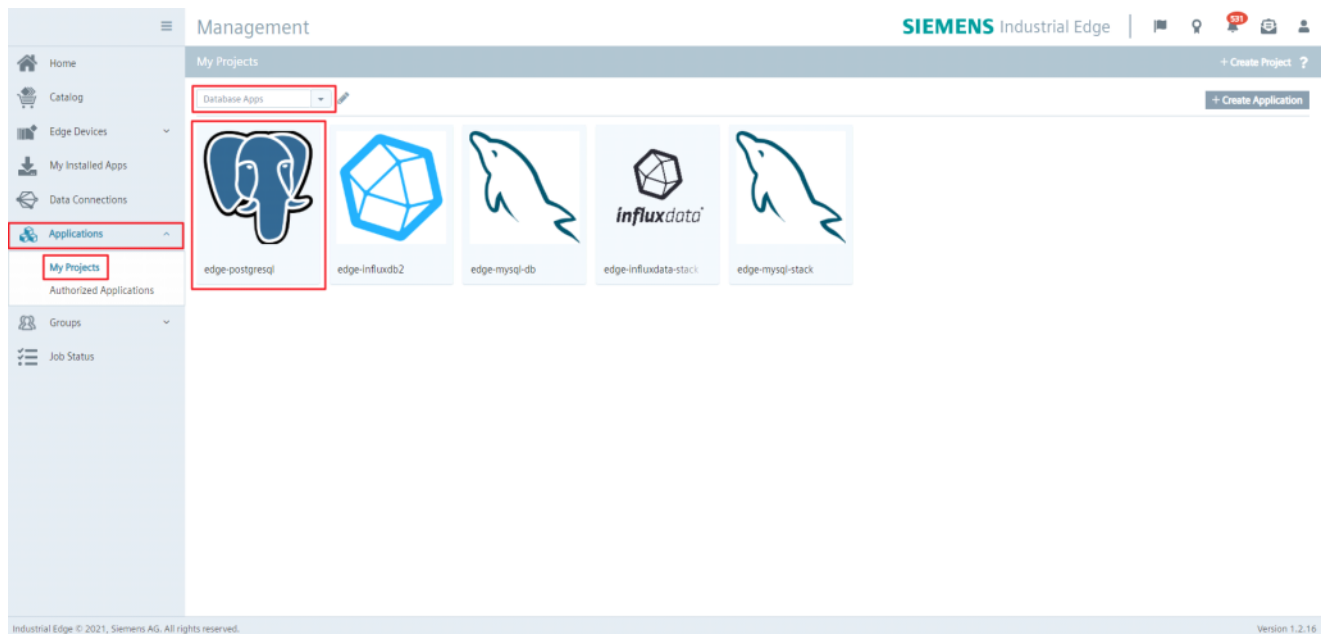
Endpoint from external pgAdmin --> [core-name]:35433

Endpoint from internal static-server --> [core-name]:5434

Endpoint from external static-server --> [core-name]:35434

Create

11. Recarsi ora nella sezione **My Projects** del software Edge App Publisher, selezionare il progetto in cui è localizzata l'applicazione edge-postgresql ed entrare nel menu della repository.
12. Premere **Start Upload** per iniziare il trasferimento della versione desiderata verso il catalogo applicazioni locale dell'istanza IEM. Attendere che l'app sia caricata. E' possibile verificare lo stato tramite il tasto **Tasks** in alto a destra.
13. Al termine dell'upload sarà possibile verificare la presenza dell'applicazione nel menu **My Projects** della sezione **Application** del portale web IEM.

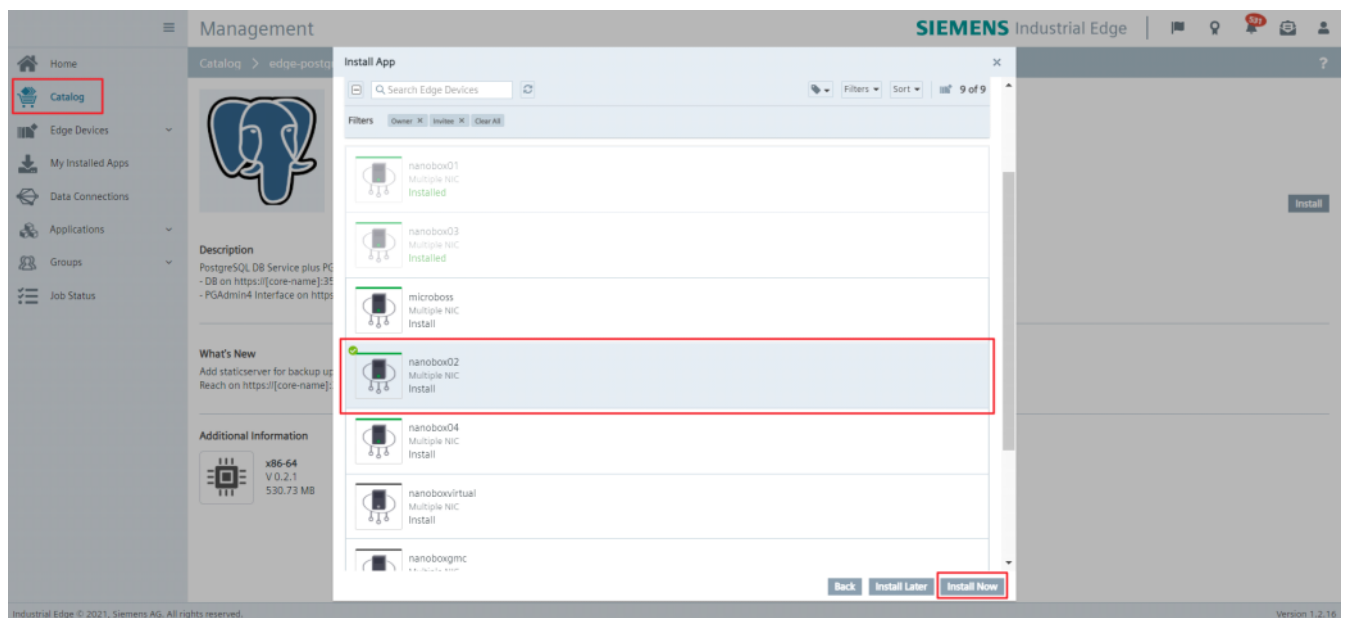


14. Selezionare l'applicazione e premere il tasto **Publish** in alto a destra per scegliere la versione da rendere pubblica nel **Catalog** IEM. Pubblicare quindi l'applicazione.
15. Verificare che la versione desiderata dell'applicazione sia ora nello stato **Live** e che nella sezione **Catalog** dello IEM Portal sia visibile l'applicazione creata con la versione corretta.

Installazione App su Edge Device

Una volta importata l'app all'interno del proprio catalogo di applicazioni su IEM, seguendo le istruzioni riportate nel paragrafo precedente, sarà possibile installarla su uno degli Edge Devices gestiti dallo IEM.

1. Dalla sezione **Catalog** selezionare l'app **edge-postgresql** per aprire la finestra dedicata.
2. Premere **Install**, selezionare l'Edge Device su cui si vuole installare l'applicazione e selezionare il tasto **Install Now**.



3. Dopo aver cliccato **Install Now**, verrà creato un nuovo **task** sull'Edge Device, dedicato all'installazione. Attendere quindi che l'app venga correttamente installata. E' possibile verificare lo stato di installazione all'interno della sezione **Job Status** o nel menu **Tasks** dell'Edge Device.
4. Una volta completata l'installazione, sarà possibile utilizzare l'applicazione all'interno dell'Edge Device configurato.

4 - Utilizzo App

giovedì 5 agosto 2021 16:43

Come anticipato al paragrafo [1 - Introduzione](#), l'applicazione edge-postgresql è costituita da tre diversi servizi:

- **edge-postgresql**
- **edge-pgadmin**
- **postgresql-static-server**

edge-postgresql

PostgreSQL è un **Database Management System (DBMS)** ad oggetti che supporta la gran parte dello standard SQL, con funzionalità quali:

- Query complesse;
- Foreign keys;
- Funzioni e procedure;
- Viste;
- Trigger;
- Integrità delle transizioni.

PostgreSQL utilizza un **modello client/server**, gestendo quindi una sessione tramite due processi cooperativi:

- Un processo server (postgres), che gestisce i file del database e accetta connessioni simultanee al database da una o più applicazioni client, eseguendo azioni sul database per conto dei client.
- L'applicazione client (front-end) dell'utente che desidera eseguire operazioni su database.

Client e server possono trovarsi su host diversi e comunicano, in questo caso, tramite una connessione di rete TCP/IP.

Grazie alla sua licenza, PostgreSQL può essere utilizzato, modificato e distribuito gratuitamente da chiunque e per qualsiasi scopo, sia esso privato, commerciale o accademico.

Per maggiori informazioni su PostgreSQL visitare: <https://www.postgresql.org/>

edge-pgAdmin

pgAdmin è un **front-end** opensource, compatibile con Linux, Windows, Solaris e Mac OSX, caratterizzato da una interfaccia grafica che consente di **amministrare in modo semplificato database di PostgreSQL**.

L'applicazione supporta tutte le funzionalità tipiche del database PostgreSQL, dalla scrittura di query SQL semplici allo sviluppo di database complessi. E' indirizzata sia agli amministratori che agli utenti del database, integrando la gestione dei permessi prelevati dal database PostgreSQL.

pgAdmin, in particolare, permette di creare un nuovo database da zero e di accedere a funzionalità di inserimento dati per la popolazione del database, quali query, funzioni o procedure, utilizzando il linguaggio SQL.

Per maggiori informazioni su pgAdmin: <https://www.pgadmin.org/>

Interfaccia grafica

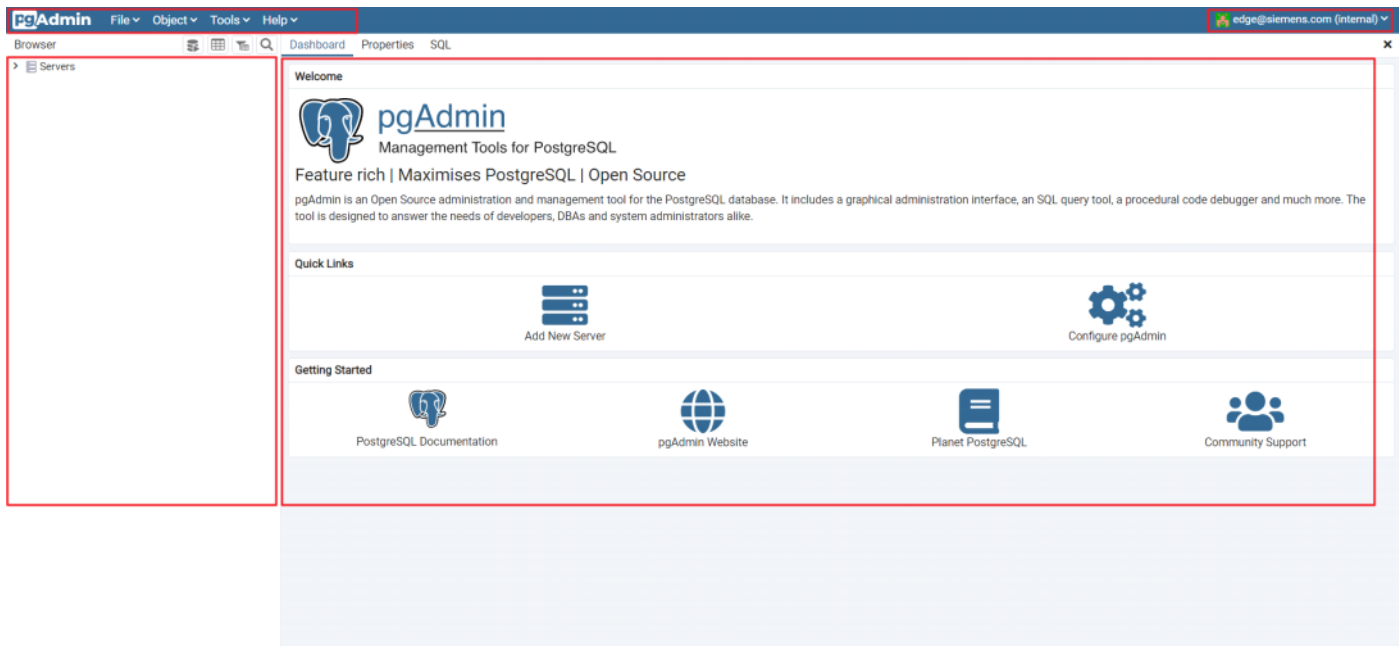
All'apertura, l'interfaccia grafica di pgAdmin presenta una barra in alto ed una finestra divisa in due pannelli distinti.

La barra in alto permette di accedere a diversi menu operativi, e di visualizzare lo user loggato all'interno dell'applicazione.

Il pannello a sinistra mostra un albero di navigazione all'interno delle connessioni e dei database disponibili.

Il pannello a destra mostra invece delle tab che permettono di:

- Configurare nuove connessioni a database server;
- Configurare e customizzare l'interfaccia pgAdmin;
- Accedere a documentazione online, sito web di PostgreSQL e community di supporto.

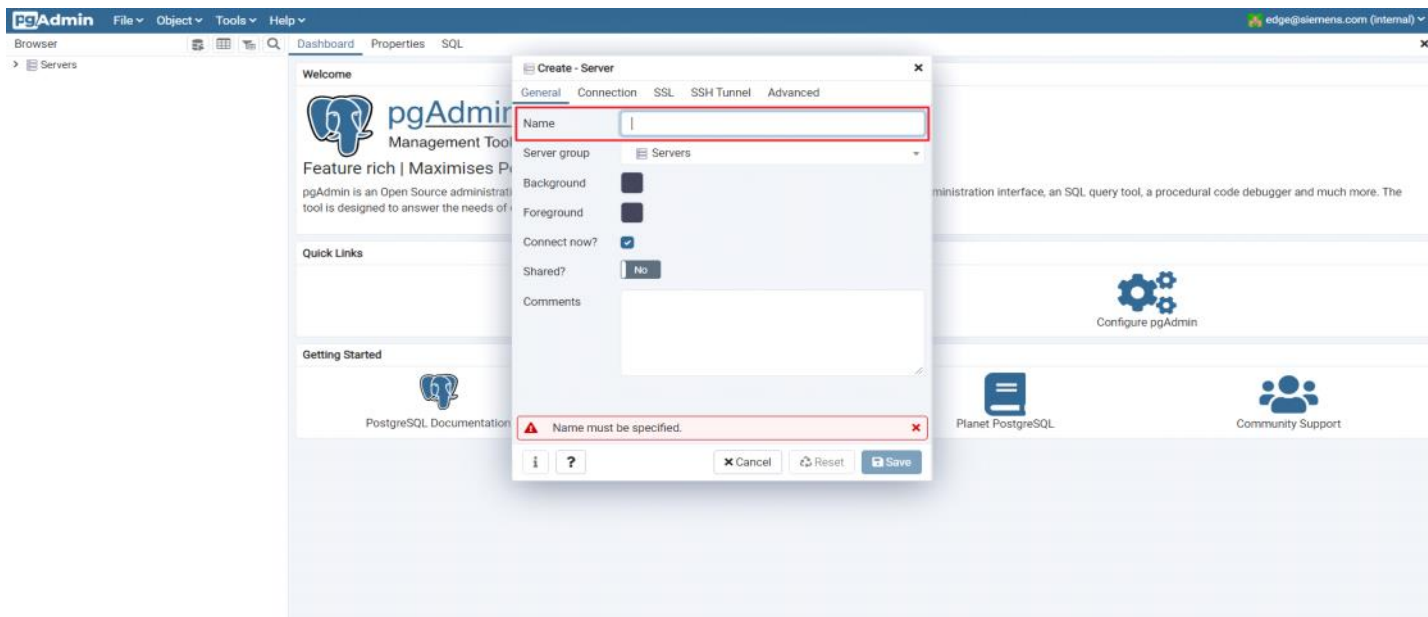


Creazione di una nuova connessione verso un database server

Per configurare una nuova connessione all'interno di pgAdmin, è necessario innanzitutto specificare il **nome** o il **descrittivo della connessione**. Questo verrà poi visualizzato nell'albero di navigazione delle connessioni, sulla porzione a sinistra della pagina web.

Oltre al nome della connessione, è possibile specificare:

- Gruppo di appartenenza del server configurato: il server configurato sarà visibile nell'albero di navigazione sotto il nodo padre o gruppo di appartenenza specificato;
- Colori di background e foreground del server;
- Eventuale tentativo di connessione al server al completamento e alla chiusura della finestra di dialogo;
- Possibilità di condivisione del server con altri utenti;
- Eventuali commenti.

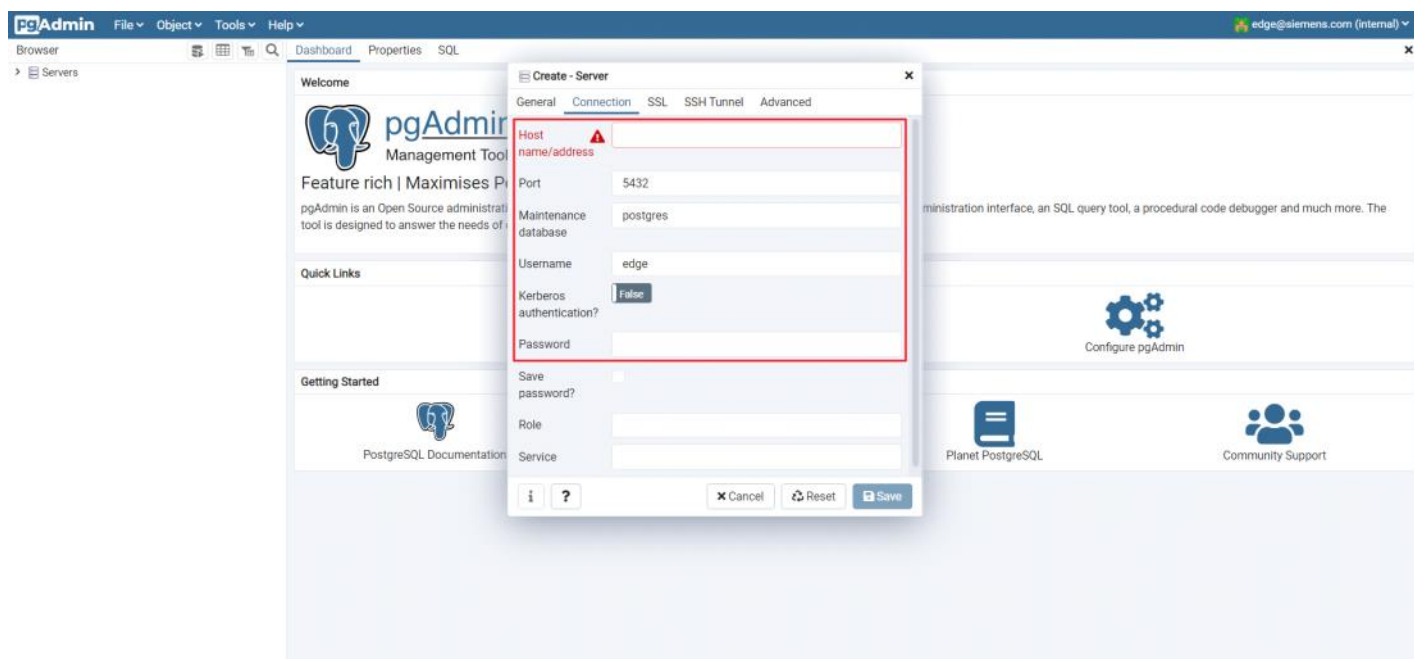


Una volta specificate queste informazioni è poi necessario indicare, all'interno del campo **Host name/address**, l'**indirizzo IP dell'host del server** o il **nome di dominio completo del server**. La porta del listener dell'host del server viene invece definita nel campo **Port**, e il suo valore predefinito è 5432.

In ultimo, è necessario specificare lo **user** e la **password** (ed eventualmente i privilegi dello user nel campo **Role**) che verranno utilizzati durante l'autenticazione al server.

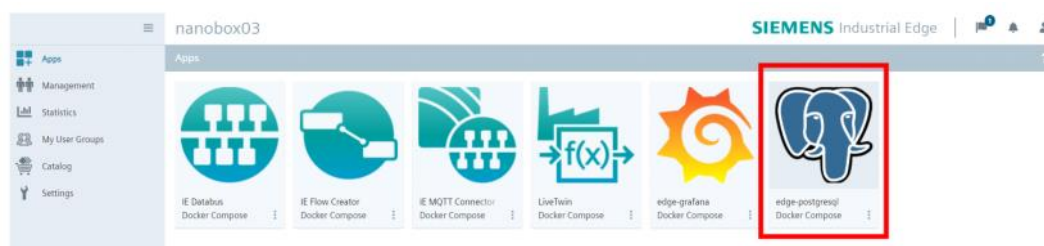
Nel paragrafo seguente verrà indicata la configurazione server adottata nell'esempio applicativo fornito.

Per informazioni più approfondite sui campi non citati, visitare: https://www.pgadmin.org/docs/pgadmin4/5.6/server_dialog.html

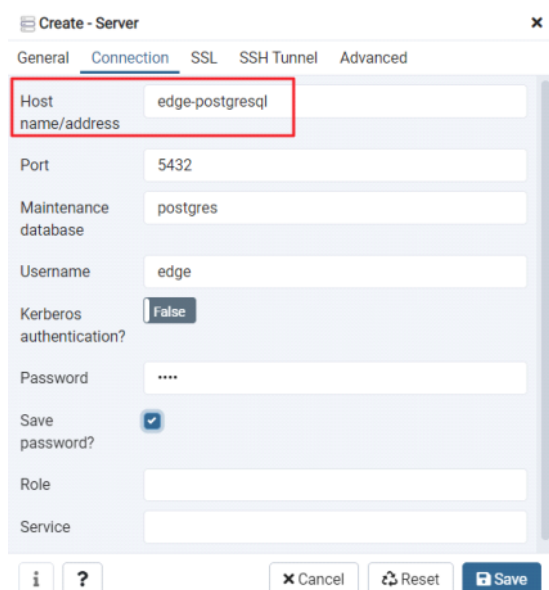


pgAdmin su Edge Device

La pagina web relativa a pgAdmin è accessibile all'indirizzo [http://\[core-name\]:35433](http://[core-name]:35433), o semplicemente cliccando sull'icona edge-postgresql all'interno dell'Edge Device su cui l'applicazione è installata. Per accedere all'applicazione, è necessario utilizzare lo user associato all'Edge Device (ad esempio edge@siemens.com) e password edge.



L'applicazione edge-postgresql viene fornita con un database pre-caricato chiamato **edge**. Le configurazioni di connessione al database server **edge** sono le seguenti:



postgresql-static-server

Postgres Static Server è un'applicazione API caratterizzata da un'interfaccia grafica che consente l'**upload di file di backup salvati in locale nel proprio PC**, con lo scopo di utilizzare questi file per il **restore di un database o di specifici oggetti del database**.

L'interfaccia grafica dell'applicazione si compone di tre sezioni principali:

1. **Path:** sezione per la selezione del *path* in cui verrà salvato il file di backup del database. Il path scelto definisce l'utente a cui il file di backup

caricato sarà visibile in pgAdmin. Il menu a tendina relativo a questa sezione viene compilato in automatico con gli utenti che hanno effettuato almeno un accesso in pgAdmin.

2. **File:** sezione per il caricamento file tramite *drag & drop* o tramite selezione del file dall'esplorazione risorse del proprio PC. Non c'è limite alla dimensione del file caricabile.
3. **Backup files list e Delete Static files:** lista di tutti i file caricati e funzione di cancellazione di tutti i file salvati all'interno del *path* indicato nella prima sezione. Una volta cancellati i file, questi non saranno più visibili o utilizzabili in pgAdmin.

Postgres Static Server

API App to upload and serve static files to Postgres through HTTP.

To restore backup files in Postgres database, upload the file and select the chosen path.
The chosen path defines the user to whom the backup file is visible.
Then, go to Postgres. The file will be visible in Tools / Restore / Filename.

Import new backup file

Select backup path ▼ 1

Drop files here to upload or [select from your computer](#) 2

Uploading edge_backup canceled

Backup files list

Delete All Static Files 3

storage/edge_siemens.com/edge_backup

La pagina web di Postgres Static Server è accessibile all'indirizzo [http://\[core-name\]:35434](http://[core-name]:35434).

5 - Esempio Applicativo

giovedì 5 agosto 2021 16:44

L'utilizzo di database consente di creare un archivio dati strutturato all'interno del quale salvare dati a medio-lungo termine, agendo poi a posteriori sui dati raccolti per attività quali la visualizzazione o l'analisi dei dati di storico, con lo scopo di ottenere informazioni rilevanti. Se progettato correttamente, un database può infatti consentire di strutturare dati eterogenei tra loro, assicurando l'accuratezza e l'integrità delle informazioni salvate e facilitando la ricerca o la successiva elaborazione dei dati stessi.

In questo esempio applicativo verrà mostrato come:

- Utilizzare il **database PostgreSQL** per implementare una raccolta dati efficace;
- **Graficare i dati raccolti** tramite molteplici modalità di visualizzazione;
- Effettuare operazioni di **backup/restore di database** PostgreSQL o di singoli oggetti all'interno del database (tabelle, procedure, funzioni, ecc.), con lo scopo di utilizzare il backup come copia di restore su diversi Edge Devices, favorendo così la replicabilità e scalabilità del database relativo ad uno use case su più dispositivi, a seconda delle esigenze.

Scopo dell'applicazione

Utilizzando le funzionalità offerte dall'applicazione fornita e dalle applicazioni elencate nel paragrafo seguente, sarà possibile implementare diverse funzionalità:

- Configurazione scambio dati con SIMATIC S7 Connector App e IE Databus App;
- Scrittura dati provenienti da IE Databus con protocollo MQTT in database PostgreSQL dedicato;
- Data retention e cancellazione automatizzata dei dati dal database PostgreSQL;
- Dashboard per visualizzazione dati con Grafana;
- Lettura dati da database PostgreSQL dedicato con filtro temporale e filtro condizionale;
- Visualizzazione risultati con SIMATIC Flow Creator;
- Backup/restore di database PostgreSQL tramite Postgres Static Server.

Prerequisiti

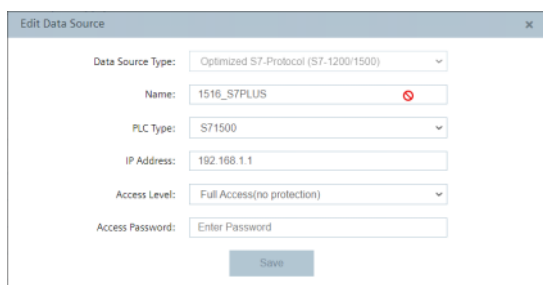
- Per consentire la comunicazione con una sorgente dati S7, un PLC della famiglia **SIMATIC** (S7-300, S7-1200, S7-1500,...).
- L'applicazione **SIMATIC S7 Connector** deve essere **installata** e **configurata** sull'Edge Device utilizzato.
- L'applicazione **SIMATIC IE Databus** deve essere **installata** e **configurata** sull'Edge Device utilizzato.
- L'applicazione **SIMATIC Flow Creator** deve essere **installata** sull'Edge Device utilizzato.
- Il nodo **node-red-contrib-postgres-variable** deve essere installato nella libreria nodi di SIMATIC Flow Creator. Per maggiori dettagli consultare il capitolo [A - Installazione nodo NodeRED PostgreSQL](#).
- Il file **Flow_AppExample_S7toPostgreSQL.json** deve essere importato all'interno dell'applicazione SIMATIC Flow Creator tramite la funzionalità **Import** dal menù dedicato.
- L'applicazione **edge-postgresql** deve essere **installata** sull'Edge Device utilizzato. Per maggiori dettagli seguire il capitolo [3 - Installazione App](#).
- L'applicazione **edge-postgresql** viene fornita con un database pre-caricato chiamato **edge**. Questo database deve necessariamente esistere per permettere al flusso NodeRED creato di funzionare correttamente.

Configurazione scambio dati con SIMATIC S7 Connector App e IE Databus App

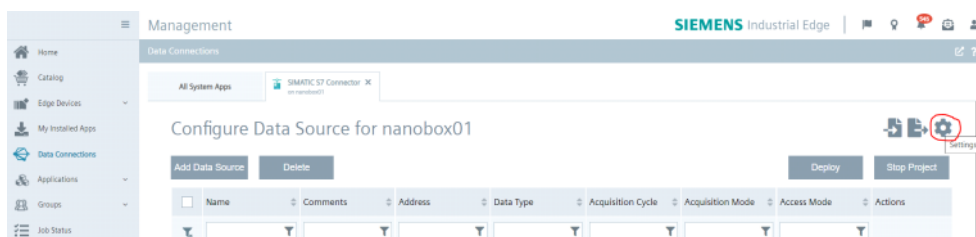
Per poter utilizzare il database **PostgreSQL** per il salvataggio di informazioni, occorre prima effettuare uno **scambio dati** con una sorgente dati in grado di generare ciclicamente nuovi valori.

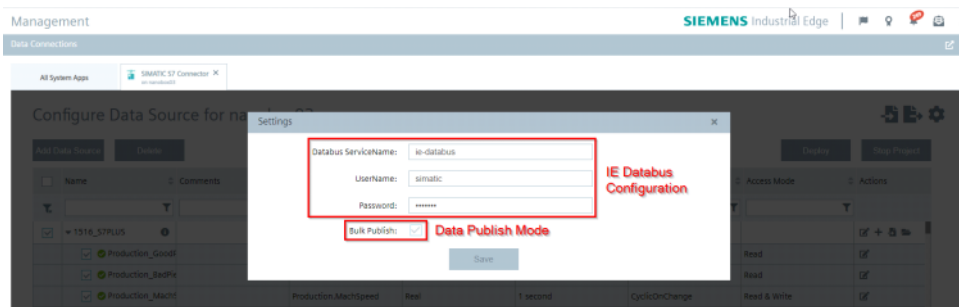
In questo esempio applicativo viene considerato l'utilizzo di una sorgente dati **PLC SIMATIC S7**, configurata all'interno dell'applicazione **SIMATIC S7 Connector** inserendo le proprietà necessarie alla comunicazione e la lista delle variabili da monitorare.

All'interno dell'applicazione S7 Connector, in questo caso, una CPU S7-1500 è stata configurata come **Datasource** con protocollo **S7+** e con modalità **Bulk Publish** di pubblicazione dei dati:



La modalità **Bulk Publish** può essere impostata tramite il tasto **Settings** presente nell'interfaccia di configurazione di S7 Connector, insieme all'utente e alla password utilizzati per connettersi al broker MQTT dell'applicazione **SIMATIC IE Databus** (in questo esempio useremo utente **edge** e password **edge**):

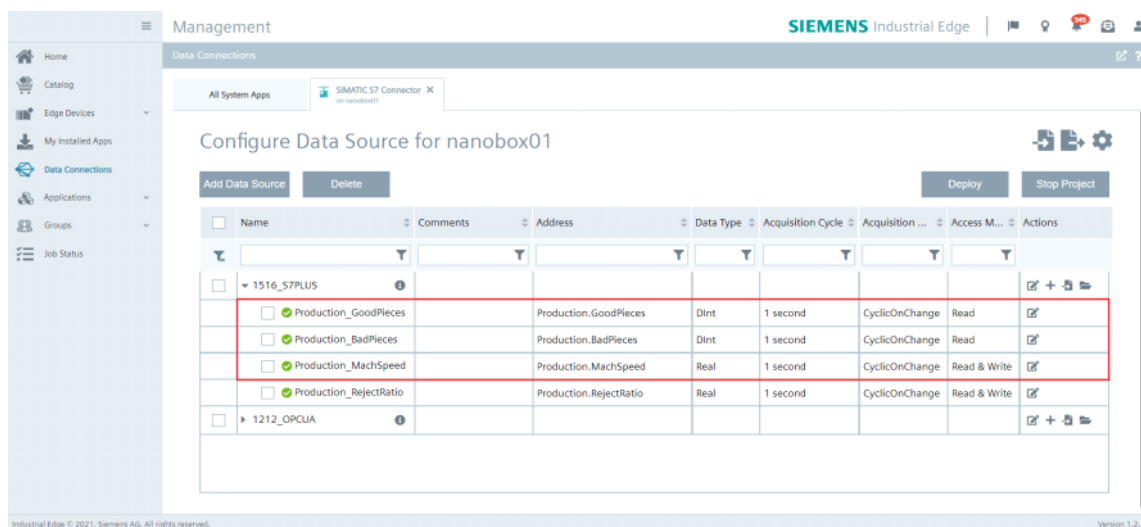




Per lo scopo verranno considerate **3 variabili**, schematizzate nella tabella seguente:

Id Datapoint	Descrizione	S7+ Address	Type	Access Mode
<i>Production_GoodPieces</i>	Contatore Pezzi Prodotti	Production.GoodPieces	Dint	Read
<i>Production_BadPieces</i>	Contatore Pezzi Scartati	Production.BadPieces	Dint	Read
<i>Production_MachSpeed</i>	Velocità di produzione in pezzi/min.	Production.MachSpeed	Real	Read&Write

Di seguito è visibile il risultato della configurazione della sorgente dati con S7 Connector all'interno della sezione **Data Connections** del sistema **Industrial Edge Management**:



Nella modalità **Bulk Publish**, quando S7 Connector effettua una lettura dei dati, viene utilizzato un solo topic MQTT dove sono pubblicate tutte le variabili che hanno subito un cambiamento di valore nel tempo di ciclo configurato (modalità *CyclicOnChange*). I dati letti dalle variabili configurate saranno quindi disponibili attraverso l'applicazione **SIMATIC IE Databus** utilizzando il topic MQTT dedicato alla datasource configurata (in questo esempio "1516_S7PLUS"), che emetterà ciclicamente un messaggio JSON contenente la proprietà **vals**, ovvero un array con tutte le proprietà delle variabili lette.

Di seguito un esempio di **messaggio JSON** ottenuto da S7 Connector tramite MQTT in fase di lettura delle variabili:

```
{
  "topic": "ie/d/j/simatic/v1/s7c1/dp/r/1516_S7PLUS/default",
  "payload": {
    "seq": 84631,
    "vals": [
      {
        "id": "101",
        "qc": 3,
        "ts": "2021-03-10T22:23:04.146Z",
        "val": 80
      },
      {
        "id": "102",
        "qc": 3,
        "ts": "2021-03-10T22:23:04.146Z",
        "val": 20
      },
      {
        "id": "103",
        "qc": 3,
        "ts": "2021-03-10T22:23:04.146Z",
        "val": 120.5
      }
    ]
  }
}
```

Nella tabella seguente è specificato il significato delle tipiche **proprietà** del messaggio ottenuto da S7 Connector tramite MQTT in fase di lettura delle variabili, con riferimento al messaggio di esempio sopra riportato:

Proprietà	Descrizione	Valore Esempio
-----------	-------------	----------------

<i>topic</i>	Indica il topic MQTT di provenienza del messaggio ricevuto	<i>ie/d/j/simatic/v1/s7c1/dp/r/1516_S7PLUS/default</i>
<i>payload</i>	E' il corpo del messaggio, contenente tutte le proprietà popolate da S7 Connector.	{ "seq": 84631, "vals": [...] }
<i>seq</i>	Numero progressivo della sequenza di lettura. Ogni nuova lettura incrementa questo numero di 1.	84631
<i>vals</i>	Array che contiene tutte le variabili lette in un ciclo e le loro proprietà.	[{ "id": "103", "qc": 3, "ts": "2021-03-10T22:23:04.146Z", "val": 120.5 },]
<i>id</i>	Id corrispondente al nome della variabile configurato all'interno dell'app S7 Connector.	103
<i>qc</i>	Quality Code della lettura.	3
<i>ts</i>	Timestamp in formato ISO86901 (YYYY-MM-DDTHH:MM:SS).	"2021-03-10T22:23:04.146Z"
<i>val</i>	Valore della variabile letta.	120.5

Per maggiori informazioni sulle applicazioni S7 Connector, IE Databus e sulla lettura variabili tramite protocollo MQTT, consultare i manuali dedicati:

- SIMATIC S7 Connector Operating Manual - <https://support.industry.siemens.com/cs/us/en/view/109783783>
- SIMATIC IE Databus Operating Manual - <https://support.industry.siemens.com/cs/us/en/view/109783784>
- Edge Management Operation Manual - <https://support.industry.siemens.com/cs/us/en/view/109793845>

Creazione ConnectionMap per Mapping Nome Tag - Id Tag

Come descritto al paragrafo precedente, tutti i messaggi ricevuti sul topic data sono caratterizzati dalla proprietà *id*, ovvero da un numero univoco che viene assegnato da S7 Connector Configurator ad ogni tag configurata. La corrispondenza tra *id* e nome delle varie tag è visibile all'interno del messaggio MQTT chiamato *metadata*, che S7 Connector Configurator invia ad ogni client MQTT connesso al topic specificato e che viene aggiornato ogni qualvolta la configurazione delle Data Source in S7 Connector venga modificata.

Di seguito un esempio di messaggio MQTT *metadata* ricevuto:

```
{
  "topic": "ie/d/j/simatic/v1/s7c1/dp",
  "payload": {
    "seq": 1,
    "connections": [
      {
        "name": "1516_S7PLUS",
        "type": "S7+",
        "dataPoints": [
          {
            "name": "default",
            "topic": "ie/d/j/simatic/v1/s7c1/dp/r/1516_S7PLUS/default",
            "publishType": "bulk",
            "dataPointDefinitions": [
              {
                "name": "Production_GoodPieces",
                "id": "101",
                "dataType": "DInt"
              },
              {
                "name": "Production_BadPieces",
                "id": "102",
                "dataType": "DInt"
              },
              {
                "name": "Production_MachSpeed",
                "id": "103",
                "dataType": "Real"
              }
            ]
          }
        ]
      }
    ]
  },
  "qos": 2,
  "retain": true,
  "_msgid": "6d13e819.8185e8"
}
```

Come visibile nel messaggio sopra riportato, ogni tag del messaggio MQTT *metadata* è caratterizzata da tre proprietà: *name* (descrittivo della tag), *id* (numero univoco identificativo della tag) e *dataType* (tipo di dato della tag).

Un esempio di lettura dei valori delle tag in un messaggio MQTT *data* (descritto in dettaglio successivamente), è invece il seguente:

```
{
  "topic": "ie/d/j/simatic/v1/s7c1/dp/r/1516_S7PLUS/default",
  "payload": {
    "seq": 167553,
    "vals": [
      {
        "id": "101",
```

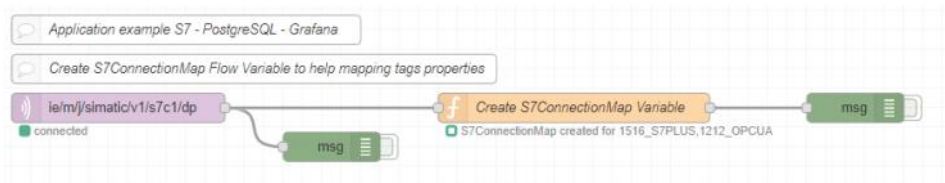
```

      "qc": 3,
      "ts": "2021-08-30T15:24:05.870Z",
      "val": 115403
    },
    {
      "id": "102",
      "qc": 3,
      "ts": "2021-08-30T15:24:05.870Z",
      "val": 478
    },
    {
      "id": "103",
      "qc": 3,
      "ts": "2021-08-30T15:24:05.870Z",
      "val": 120.5
    }
  ],
  "qos": 0,
  "retain": false,
  "_msgid": "151f43ab.77230c"
}

```

Come è possibile vedere nel messaggio *data* sopra riportato, ogni tag letta viene identificata esclusivamente tramite un *id* univoco, senza nessuna indicazione relativa al nome della tag configurato all'interno dell'applicazione S7 Connector Configurator. Questo può rendere difficoltoso, in fase di lettura dati, il riconoscimento delle tag configurate.

Per facilitare quindi la lettura ed il riconoscimento delle variabili configurate, nel file **Flow_AppExample_S7toPostgreSQL.json** dell'esempio applicativo fornito è presente un flusso di SIMATIC Flow Creator che, a partire dal messaggio MQTT metadata (nodo *mqtt-in*) effettua il mapping tra *id* e *name* delle tag configurate.



Per farlo, il flusso utilizza una funzione (nodo *function*) che salva all'interno della variabile globale *S7ConnectionMap* degli oggetti di tipo *Map*, ovvero delle coppie chiave-valore (*id-name* oppure *name-id* delle tag) ordinate a seconda dell'ordine di inserimento delle varie coppie all'interno della variabile. Il contenuto del nodo *function* è il seguente:

```

//Create an object containing each S7 Connector connection property with different Map Objects to create
correspondence between Tags IDs, Names and Types. Initialize the connections Mapping Object.
let S7ConnectionMap = {
  "nameList":[], // array of available S7 Connections names. Order is the same in Map objects below.
  "typeList":[], // array of available S7 Connections types. Order is the same of nameList.
  "nameIDMaps":[], // array of Tags Names-IDs object. Order is the same of nameList.
  "IDNameMaps":[], // array of Tags IDs-Names Map object. Order is the same of nameList.
  "IDTypeMaps":[] // array of Tags IDs-Type Map object. Order is the same of nameList.
}

//Check Payload
let m = msg.payload;
if (m.seq == undefined) {
  // update global maps
  flow.set("S7ConnectionMap", null);
  // update function node status
  node.status({fill:"red",shape:"ring",text:"S7Map cannot be created"});

  return null;
}

//Iterate through connections
for (let i = 0; i < m.connections.length; i++)
{
  let connection = m.connections[i];
  // push connection name and type in global object
  S7ConnectionMap.nameList.push(connection.name);
  S7ConnectionMap.typeList.push(connection.type);
  // init maps
  let nameIDMap = new Map();
  let IDNameMap = new Map();
  let IDTypeMap = new Map();

  // Iterate through dataPoints
  let dataPoints = connection.dataPoints;
  for (let j = 0; j < dataPoints.length; j++)
  {
    let dataPoint = dataPoints[j];
    // Iterate through dataPointDefinitions
    let dataPointDefinitions = dataPoint.dataPointDefinitions;
    for (let k = 0; k < dataPointDefinitions.length; k++)
    {
      let dataPointDefinition = dataPointDefinitions[k];
      // push in maps the datapoint property
      nameIDMap.set(dataPointDefinition.name, dataPointDefinition.id);
      IDNameMap.set(dataPointDefinition.id, dataPointDefinition.name);
      IDTypeMap.set(dataPointDefinition.id, dataPointDefinition.dataType);
    }
  }
  // push mappings in global object
}

```

```

    S7ConnectionMap.nameIDMaps.push(nameIDMap);
    S7ConnectionMap.IDnameMaps.push(IDNameMap);
    S7ConnectionMap.IDTypeMaps.push(IDTypeMap);
}

// Update global maps
flow.set("S7ConnectionMap", S7ConnectionMap);

// Set S7Map as output payload
msg.payload = S7ConnectionMap;

// Update function node status
node.status({fill:"green",shape:"ring",text:"S7ConnectionMap created for " + S7ConnectionMap.nameList.join()});

return msg;

```

Per utilizzare la variabile globale **S7ConnectionMap** per successive elaborazioni all'interno di un nuovo nodo function, è necessario per prima cosa estrarre la variabile globale S7ConnectionMap e l'indice relativo al datasource di interesse (nel caso specifico il PLC 1516_S7PLUS).

```

let S7ConnectionMap = flow.get("S7ConnectionMap");
let connectionIndex = S7ConnectionMap.nameList.indexOf("1516_S7PLUS");

```

Una volta a disposizione la variabile globale S7ConnectionMap, è possibile ottenere la mappa *name-id* delle tag configurate utilizzando la seguente riga di codice:

```

let nameIDMap = S7ConnectionMap.nameIDMaps[connectionIndex];

```

Viceversa, per ottenere la mappa *id-name* delle tag configurate è possibile utilizzare la seguente:

```

let IDNameMap = S7ConnectionMap.IDnameMaps[connectionIndex];

```

Per ottenere l'id di una specifica tag a partire dal suo nome basterà quindi estrarla dalla mappa nameIDMap, utilizzando la chiave (*name*) di interesse, ad esempio "Production_GoodPieces". L'output della riga di codice sotto riportata sarà 101, ovvero l'*id* associato alla tag con *name* "Production_GoodPieces".

```

let tagId = nameIDMap.get("Production_GoodPieces");

```

Per ottenere invece il nome di una specifica tag a partire dal suo id sarà sufficiente estrarla dalla mappa IDNameMap, utilizzando la chiave (*id*) di interesse, ad esempio 101. L'output della riga di codice sotto riportata sarà "Production_GoodPieces", ovvero il *name* associato alla tag con *id* 101.

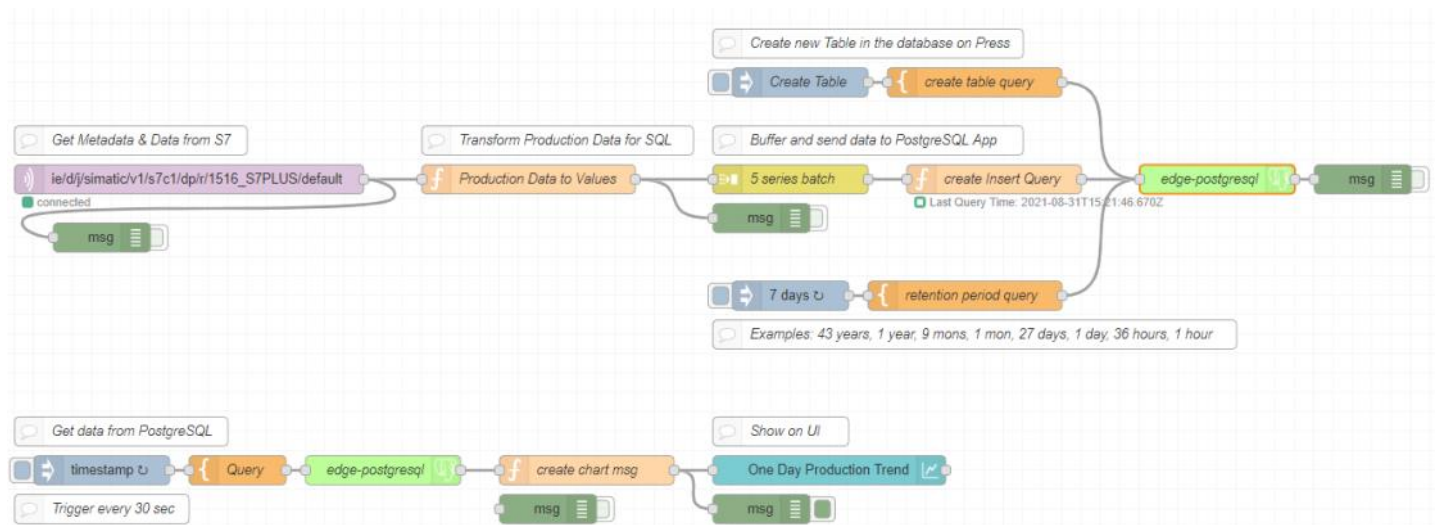
```

let tagName = IDNameMap.get("101");

```

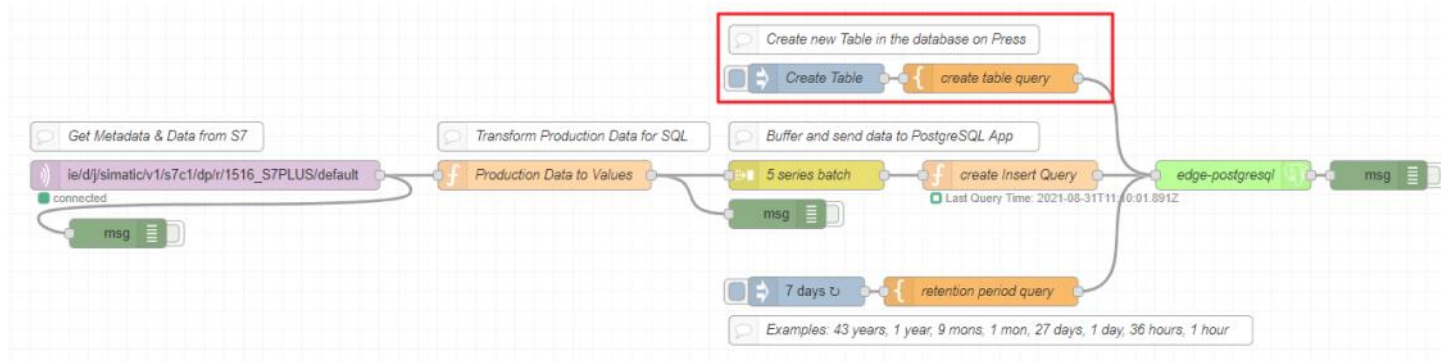
Scrittura dati provenienti da IE Databus con protocollo MQTT in database PostgreSQL

Una volta configurate all'interno dell'applicazione **S7 Connector** le variabili da scambiare con la sorgente dati PLC, è possibile utilizzare l'applicazione **SIMATIC Flow Creator** per raccogliere i dati letti, processarli ed inviarli al database PostgreSQL pre-configurato dell'applicazione **edge-postgresql**, tramite il flusso visibile nell'immagine sottostante e fornito insieme all'esempio applicativo.



Prima di proseguire con i dettagli sul caricamento dati all'interno del database **edge** in PostgreSQL, è necessario creare all'interno del database una tabella che conterrà i dati letti.

Nell'esempio applicativo fornito, i dati letti dalla sorgente dati PLC vengono caricati all'interno di una tabella chiamata **production** nel database PostgreSQL. La tabella **production** può essere creata manualmente tramite l'interfaccia utente di pgAdmin, oppure utilizzando il flusso evidenziato in rosso nella figura sottostante.



Alla ricezione di un trigger manuale da parte dell'utente sul nodo *inject* (*Create Table*), viene eseguito il corpo del nodo *function* (*create table query*), che va ad eliminare tramite istruzione DROP TABLE la tabella **production**, se esistente, e a ricrearla all'interno del database tramite istruzione CREATE TABLE.

```

DROP TABLE IF EXISTS production;

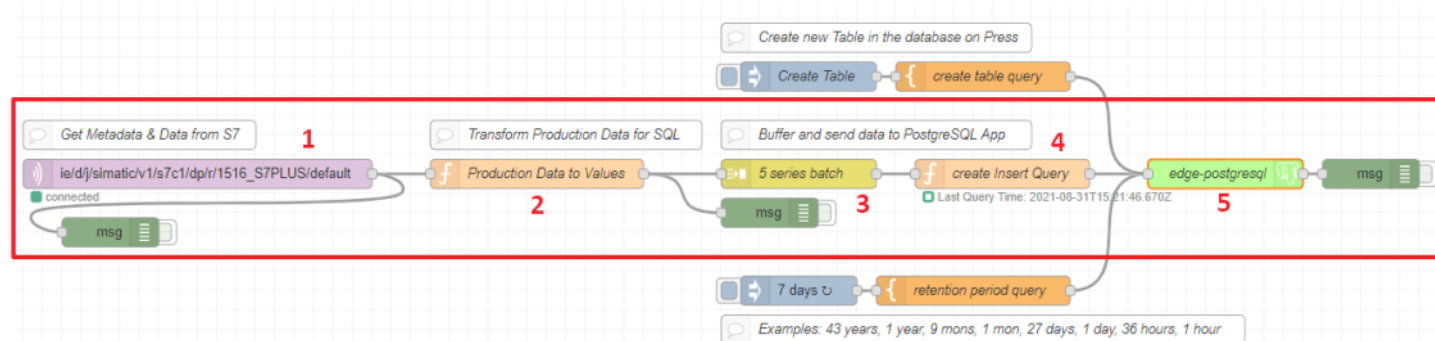
CREATE TABLE production (
  id SERIAL PRIMARY KEY,
  timestamp TIMESTAMP(3) NOT NULL,
  GoodPieces INTEGER,
  BadPieces INTEGER,
  MachSpeed REAL,
  MachineName VARCHAR(255) NOT NULL
);

```

Come visibile dal codice SQL sopra riportato, la tabella **production** è costituita da 6 colonne:

- 1 - *id*: identificativo univoco del record in tabella. Questa colonna è la chiave primaria della tabella e viene incrementata di 1 in maniera automatica all'inserimento di un nuovo record.
- 2 - *timestamp*: data e ora della lettura.
- 3 - *GoodPieces*: numero intero rappresentante il numero di pezzi prodotti.
- 4 - *BadPieces*: numero intero rappresentante il numero di pezzi scartati.
- 5 - *MachSpeed*: numero reale rappresentante la velocità della macchina.
- 6 - *MachineName*: stringa rappresentante il nome della macchina associata alla produzione.

All'interno del flusso di SIMATIC Flow Creator dell'esempio applicativo fornito, la lettura dei dati provenienti da S7 Connector viene effettuata tramite un nodo *mqtt-in* (1), che successivamente invia il contenuto del messaggio ad un nodo *function* (2) programmato per processare i dati e renderli compatibili con la standard richiesto dal nodo *postgres* (4) che si occuperà di inviare la richiesta al database PostgreSQL pre-configurato e denominato **edge**. Per un'ottimizzazione del carico di scrittura, un nodo *join* (3) crea un batch di 5 serie di dati prima dell'invio degli stessi al database PostgreSQL.



1 - Ricezione messaggio bulk da MQTT

Attraverso il nodo *mqtt-in* (1), Flow Creator si connette al Broker MQTT IE Databus, utilizzando il topic dedicato al PLC configurato per lo scopo, in questo esempio "1516_S7PLUS":

Edit mqtt in node

Delete

Cancel

Done

Properties

Server

ie-databus:1883

Topic

ie/d/j/simatic/v1/s7c1/dp/r/1516_S7PLUS/default

QoS

0

Output

a parsed JSON object

Name

Name

Edit mqtt in node > Edit mqtt-broker node

Delete

Cancel

Update

Properties

Name

Name

Connection

Security

Messages

Server

ie-databus

Port

1883

Enable secure (SSL/TLS) connection

Client ID

Leave blank for auto generated

Keep alive time (s)

60

Use clean session

☒

Use legacy MQTT 3.1 support

☐

Sul topic indicato in figura, ad ogni ciclo di lettura, riceveremo messaggi da MQTT IE Databus, contenente tutti i dati letti dal PLC S7-1500 configurato all'interno dell'applicazione S7 Connector. Il nome del PLC configurato è parte del topic da sottoscrivere.

Di seguito un esempio di output per il nodo *mqtt-in*:

```
{
  "topic": "ie/d/j/simatic/v1/s7c1/dp/r/1516_S7PLUS/default",
  "payload": {
    "seq": 84631,
    "vals": [
      {
        "id": "101",
        "qc": 3,
        "ts": "2021-03-10T22:23:04.146Z",
        "val": 80
      },
      {
        "id": "102",
        "qc": 3,
        "ts": "2021-03-10T22:23:04.146Z",
        "val": 20
      },
      {
        "id": "103",
        "qc": 3,
        "ts": "2021-03-10T22:23:04.146Z",
        "val": 120.5
      }
    ]
  }
}
```

Per la connessione al broker MQTT **SIMATIC IE Databus** è necessaria la configurazione di un utente abilitato allo scambio dati. In questo caso è stato configurato utilizzando i seguenti parametri:

IE Databus Address	IE Databus Port	IE Databus User	IE Databus Password
ie-databus	1883	edge	edge

2 - Data pre-processing per PostgreSQL

Prima di poter salvare i dati ricevuti nel database PostgreSQL sarà necessario **formattare** il messaggio ricevuto dal nodo *mqtt-in* attraverso dei nodi di tipo *function*, secondo le specifiche richieste dal nodo *postgres* dedicato.

In particolare, all'interno dell'esempio applicativo fornito, i nodi function utilizzati per la formattazione dei messaggi sono due, intervallati da un nodo *batcher*, descritto in dettaglio nel paragrafo seguente.

Il primo nodo *function* (2) si occupa di processare un messaggio in input tramite uno script in linguaggio Javascript e creare uno o più messaggi di uscita. Un esempio di output del nodo *function* (2) è il seguente:

```
{
  "payload": "('2021-09-01 19:05:02.339', 'Demo_Line', 74222, 92082, 262)",
  "_msgid": "1dc24460.34b41c"
}
```

Come si può vedere nel payload del messaggio sopra riportato, le informazioni in output sono: timestamp della lettura, nome della macchina, unite alla lettura delle tre tags "Production_GoodPieces", "Production_BadPieces" e "Production_MachSpeed".

Il payload del messaggio viene formattato all'interno del nodo *function* (2) come una stringa, in cui le variabili menzionate sopra sono divise da "," e racchiuse da "()". Questo tipo di formattazione faciliterà la scrittura dati tramite query di INSERT all'interno della tabella **production** nel database **edge** PostgreSQL.

Di seguito il contenuto del nodo *function* (2):

```

// Insert here the ordered output tag list
let selectedTags = ["Production_GoodPieces", "Production_BadPieces", "Production_MachSpeed"];

// Get S7Map variable
let S7ConnectionMap = flow.get("S7ConnectionMap");

// Find index of fps716 connection
let connectionIndex = S7ConnectionMap.nameList.indexOf("1516_S7PLUS");

// Use the index to get the right map
let nameIDMap = S7ConnectionMap.nameIDMaps[connectionIndex];

// Get IDs of selected Tags
let selectedIDs = selectedTags.map(name => nameIDMap.get(name));

// Create an empty values array
let selectedValues = new Array(selectedTags.length).fill("NULL");

// Initialize timestamp
let timestamp = null;

// Iterate through readed tags
for (let i=0; i < msg.payload.vals.length; i++){

    // Iterate through selected ids
    for(let j=0; j < selectedIDs.length; j++){

        // Search for tag
        if(msg.payload.vals[i].id == selectedIDs[j])
        {
            // Set timestamp
            timestamp = new Date(Date.parse(msg.payload.vals[i].ts)+7200000).toISOString().replace("T", " ").replace("Z", "")
            // Assign message to selected vals
            selectedValues[j] = Number(msg.payload.vals[i].val.toFixed(2));
            // stop on first match, first for will continue on next tag
            break;
        }
    }
}

// Create out message
let outMsg = {};

// If payload contains some values send it
if (timestamp)
{
    // Create values string
    outMsg.payload = "(" + timestamp + ", '" + selectedValues.join(",") + "'";

    return outMsg;
}

```

Nel codice Javascript sopra riportato, si può vedere come il nodo *function* (2) vada innanzitutto a definire le tag di interesse (*selectedTags*), e ad utilizzare poi le istruzioni descritte al paragrafo [Creazione ConnectionMap per Mapping Nome Tag - Id Tag](#) per effettuare il mapping tra *id* e *name* delle tag selezionate. Nel codice, in particolare, la proprietà *id* di ogni variabile letta viene confrontata con l'id delle variabili selezionate e recuperate tramite la variabile globale *S7ConnectionMap*: in caso di match, la funzione proseguirà con le successive istruzioni. Il messaggio di output della funzione conterrà il timestamp (*epoch*) in ms ricavato a partire dalla proprietà *ts* di ogni variabile letta. Conterrà inoltre, i valori delle variabili di interesse, identificati a partire dalla proprietà *val* dei messaggi di lettura, approssimati a 2 cifre decimali. La stringa "Demo_Line" è invece un parametro fissato che identifica il nome della macchina a cui le tag lette sono associate. Questa stringa corrisponderà alla colonna **machineName** nella tabella **production**, e verrà compilata con "Demo_Line" per tutti i dati inseriti all'interno della tabella del database di PostgreSQL.

Il nodo *join* (3) successivo al nodo *function* (2), si occuperà di creare l'array di serie che identifica il batch di dati da inserire successivamente nella tabella **production** del database PostgreSQL. Le caratteristiche di questo nodo sono spiegate più in dettaglio nel paragrafo seguente ([Buffer serie di dati in Batch](#)).

Per quanto riguarda il nodo *function* (4), invece, questo ha lo scopo di comporre una query di INSERT dei dati estratti dal *payload* del messaggio in input al nodo stesso.

```

// Create out message
let outMsg = {}

// Compose query
let query = 'INSERT INTO production ';
query += '(timestamp, machineName, goodPieces, badPieces, machineSpeed) VALUES';
query += msg.payload;

outMsg.payload = query;

return outMsg;

```

Un esempio di payload in ingresso al nodo *function* (4) è il seguente:

```

("2021-09-02 15:36:37.295", 'Demo_Line', 11303, NULL, 220), ('2021-09-02 15:36:38.299', 'Demo_Line', 11307, NULL, 222),
('2021-09-02 15:36:39.303', 'Demo_Line', 11309, 13876, 224), ('2021-09-02 15:36:40.303', 'Demo_Line', NULL, 13880, 226),
('2021-09-02 15:36:41.303', 'Demo_Line', 11311, 13882, 228)"

```

Ogni payload conterrà una serie di 5 letture in cui, per ogni lettura, sono presenti i seguenti parametri: *timestamp*, *machineName*, *goodPieces*, *badPieces* e *machineSpeed*.

Il nodo *function* (4) formatterà il payload in ingresso nella query seguente:

```
"INSERT INTO production (timestamp, machineName, goodPieces, badPieces, machineSpeed) VALUES
('2021-09-02 15:36:37.295', 'Demo_Line',11303,NULL,220),('2021-09-02 15:36:38.299', 'Demo_Line',11307,NULL,222),
('2021-09-02 15:36:39.303', 'Demo_Line',11309,13876,224),('2021-09-02 15:36:40.303', 'Demo_Line',NULL,13880,226),
('2021-09-02 15:36:41.303', 'Demo_Line',11311,13882,228)"
```

3 - Buffer serie di dati in Batch

Il database PostgreSQL è in grado di gestire carichi di query elevati, ma è buona abitudine cercare di ottimizzare il carico di scrittura su database raccogliendo qualche dato prima di scriverlo nel database.

Questa funzione di buffer dei dati può essere semplicemente creata in Flow Creator utilizzando un nodo di tipo *join*, che si occupa di unire una serie di messaggi in un unico messaggio secondo diverse regole e configurazioni.

In questo esempio applicativo, il nodo *join* utilizzato crea un batch di 5 serie di dati sotto forma di stringa, come mostrato al paragrafo precedente. Una volta che il contatore dei messaggi raggiunge il limite identificato dalla proprietà *count* configurata, il nodo *join* procederà all'invio del batch dati al database PostgreSQL.

Di seguito la configurazione del nodo *join* nel flusso di Flow Creator dell'esempio applicativo fornito.

4 - Invio dati a PostgreSQL

Per la comunicazione con il database PostgreSQL è possibile sfruttare le PostgreSQL API esposte da PostgreSQL per lo scambio dati con il database. Queste PostgreSQL API sono alla base del nodo di Node-RED *node-red-contrib-postgres-variable*, creato per scrivere e interrogare i dati da un database di PostgreSQL (<https://flows.nodered.org/node/node-red-contrib-postgres-variable>).

L'applicazione *edge-postgresql* fornita comprende il servizio container **edge-postgresql** basato sulla versione 13.3-alpine di PostgreSQL e, come accennato in precedenza, espone un database pre-definito denominato **edge** con utente e password pre-configurate "*edge*" / "*edge*". Questa applicazione può essere contattata da altre applicazioni presenti sia sull'Edge Device utilizzato sia da reti esterne connesse e alle porte fisiche dell'Edge Device grazie alla funzionalità di *port forwarding* offerta dal servizio Docker.

Questi parametri di connessione possono essere inseriti direttamente nella configurazione del parametro "**Server**" dei nodi della libreria *node-red-contrib-postgres-variable*:

Service Name	Hostname	Internal Port	External Host	External Port	Database Name	User	Password
edge-postgresql	edge-postgresql	5432	[ied-ip-address] oppure [ied-dns-name]	35432	edge	edge	edge

Di seguito un'immagine della tabella **production** menzionata in precedenza, all'interno della quale vengono caricati i valori relativi alle tag lette.

The screenshot shows the PgAdmin interface with a SQL query editor and a results table. The query is:

```
1 SELECT *
2 FROM production
3 ORDER BY timestamp DESC
```

The results table has the following columns: id (PK integer), timestamp (timestamp without time zone), goodpieces (integer), badpieces (integer), machinespeed (real), and machinename (character varying (255)).

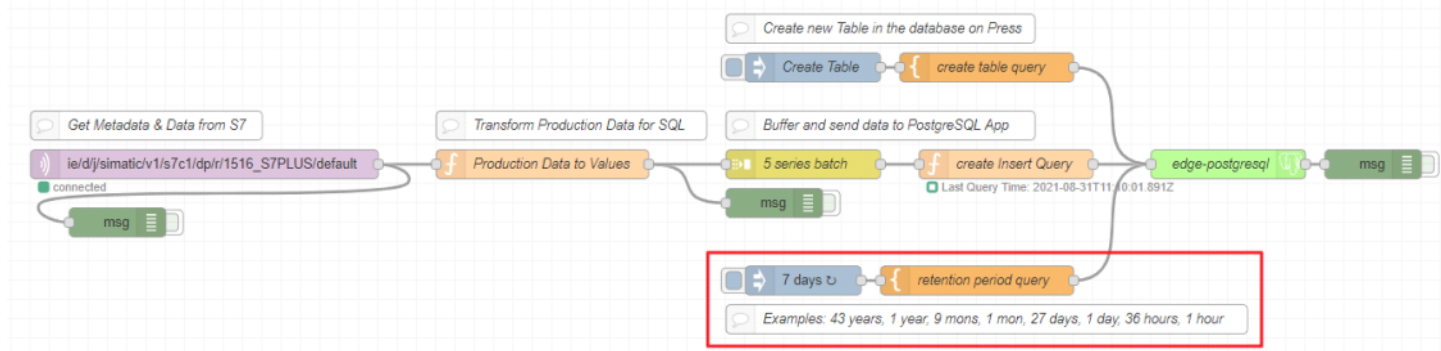
id	timestamp	goodpieces	badpieces	machinespeed	machinename
1	2248310	2021-09-09 14:13:07.871	67069	82466	338 Demo_Line
2	2248309	2021-09-09 14:13:06.875	67066	82463	336 Demo_Line
3	2248308	2021-09-09 14:13:05.869	[null]	82460	334 Demo_Line
4	2248307	2021-09-09 14:13:04.873	67063	82454	332 Demo_Line
5	2248306	2021-09-09 14:13:03.866	67060	[null]	330 Demo_Line
6	2248305	2021-09-09 14:13:02.871	67054	82451	328 Demo_Line
7	2248304	2021-09-09 14:13:01.872	[null]	82448	326 Demo_Line
8	2248303	2021-09-09 14:13:00.877	[null]	82442	324 Demo_Line
9	2248302	2021-09-09 14:12:59.874	[null]	82436	322 Demo_Line
10	2248301	2021-09-09 14:12:58.87	67051	[null]	320 Demo_Line
11	2248300	2021-09-09 14:12:57.872	67045	82430	318 Demo_Line
12	2248299	2021-09-09 14:12:56.872	67042	82427	316 Demo_Line
13	2248298	2021-09-09 14:12:55.868	67039	82424	314 Demo_Line
14	2248297	2021-09-09 14:12:54.865	67036	82421	312 Demo_Line
15	2248296	2021-09-09 14:12:53.871	67033	82418	310 Demo_Line
16	2248295	2021-09-09 14:12:52.872	[null]	82415	308 Demo_Line
17	2248294	2021-09-09 14:12:51.865	[null]	82409	306 Demo_Line
18	2248293	2021-09-09 14:12:50.874	67030	82403	304 Demo_Line

Data retention e cancellazione automatizzata dei dati dal database

E' fondamentale, durante la progettazione di un database, definire la cosiddetta *data retention*, ovvero il periodo di permanenza e conservazione dei dati caricati all'interno del database. Infatti a seconda delle necessità, i dati caricati possono essere cancellati dopo un periodo di permanenza nel database di breve, medio o lungo termine, previo eventuale backup manuale degli stessi.

Il periodo di conservazione dei dati, è solitamente dettato anche dalla capacità di memorizzazione dell'hardware utilizzato. Per hardware con capacità di memorizzazione limitate, sarà necessario prevedere un periodo di latenza dati all'interno del database di breve o medio termine. Viceversa, nel caso di hardware con capacità di memorizzazione estese, sarà possibile aumentare il periodo di latenza dati.

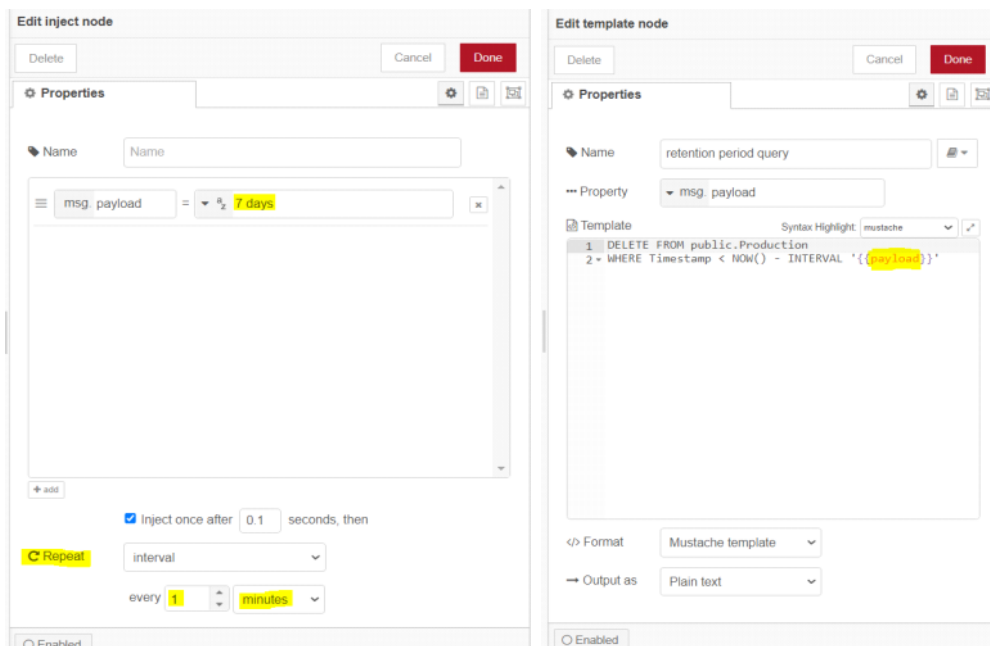
Nell'esempio applicativo fornito è presente un flusso di SIMATIC Flow Creator che prevede la cancellazione dei dati caricati all'interno della tabella **production** del database tramite una semplice query di tipo DELETE, eseguita ciclicamente.



Il tempo che intercorre tra l'esecuzione di una query di DELETE e la successiva, viene definito all'interno di un nodo *inject*. Nel caso specifico di questo esempio, l'esecuzione della query di DELETE viene triggerata ogni 1 minuto.

All'interno del nodo *inject* è inoltre definita la proprietà di payload "7 days", che indica la latenza temporale dei dati all'interno del database. Questo dato viene passato al nodo *function*, che lo utilizza per definire il filtro (WHERE nella query) di cancellazione dei dati: all'esecuzione della query di DELETE in PostgreSQL, verranno eliminati tutti i dati più vecchi di 7 giorni (se esistenti).

Nelle figure sotto è possibile vedere la configurazione del nodo *inject* (a sinistra) e il contenuto del nodo *function* (a destra).



Chiaramente, le tempistiche definite in questo flusso sono modificabili secondo necessità.

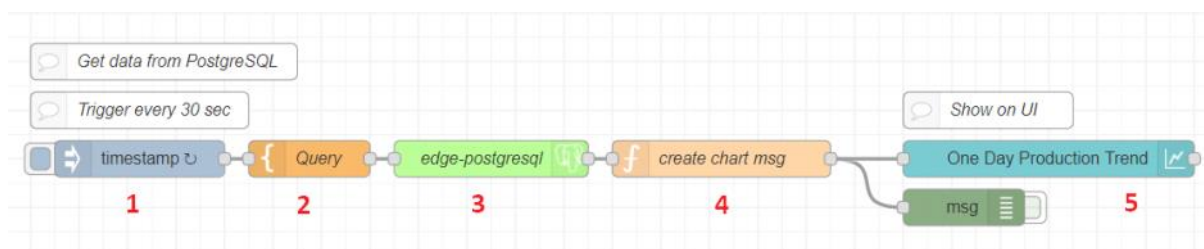
Lettura dati da database PostgreSQL dedicato con filtro temporale e filtro condizionale

Oltre all'inserimento di nuovi dati all'interno di una tabella del database PostgreSQL, potrebbe essere necessario interrogare il database per processare e richiedere dati di storico, per scopi di visualizzazione, analisi o reportistica.

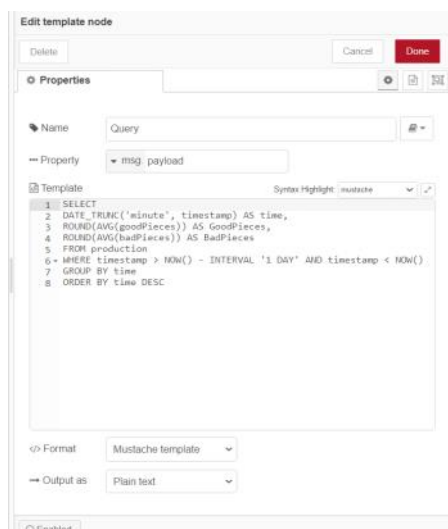
La libreria di nodi `node-red-contrib-postgres-variable` offre la possibilità di effettuare query personalizzate per interrogare il database PostgreSQL. La query viene specificata nella configurazione del nodo ed il risultato della query viene restituito nel messaggio in output da l nodo tramite la proprietà `msg.payload`.

All'interno del flusso di SIMATIC Flow Creator dell'esempio applicativo fornito, la lettura dei dati dal database PostgreSQL viene azionata da un messaggio di trigger dal nodo *inject* (1) ogni 30 secondi, dopo il quale il nodo *function* (2) formatta una query da inviare al nodo *postgres* (3), che si occupa di inviare la query configurata nel nodo *function* (2) al database PostgreSQL. Una volta ottenuta la risposta, questa viene processata e formattata da un altro nodo *function* (4) per la visualizzazione di un grafico su Dashboard tramite il nodo *ui_chart* (5).

Di seguito sono indicati i dettagli del flusso NodeRED coinvolti:



In questo esempio applicativo utilizzeremo il nodo *postgres* connesso al servizio database *edge-postgresql* e una query dedicata per leggere i dati salvati "Production_GoodPieces" e "Production_BadPieces" nelle ultime 24 ore ed aggregarli in modo tale da ridurre il numero di campioni da visualizzare.



La sintassi utilizzata per la query è SQL.

```
SELECT
    DATE_TRUNC('minute', timestamp) AS time,
    ROUND(AVG(goodPieces)) AS GoodPieces,
    ROUND(AVG(badPieces)) AS BadPieces
FROM production
WHERE timestamp > NOW() - INTERVAL '1 DAY' AND timestamp < NOW()
GROUP BY time
ORDER BY time DESC
```

La query utilizzata andrà a selezionare dai dati delle ultime 24 ore la media dei valori "goodPieces" e "badPieces" per ogni minuto. Questo consente di sottocampionare i dati grezzi, raccolti ogni secondo, con un fattore 1/60.

La risposta del database viene inviata quindi in uscita al nodo *postgres* (3) tramite la proprietà *msg.payload*. Di seguito un estratto di esempio del messaggio ricevuto in risposta alla query:

```
{
  "payload": [
    {
      "time": "2021-03-27T20:47:00.000Z",
      "goodpieces": 131096,
      "badpieces": 57514
    },
    {
      "time": "2021-03-27T20:48:00.000Z",
      "goodpieces": 131327,
      "badpieces": 57597
    },
    ...
    {
      "time": "2021-03-28T20:47:00.000Z",
      "goodpieces": 131802,
      "badpieces": 59069
    },
    {
      "time": "2021-03-28T20:48:00.000Z",
      "goodpieces": 131886,
      "badpieces": 59106
    }
  ],
  "topic": ""
}
```

A questo punto sarà possibile processare il contenuto della proprietà *msg.payload* formattandolo per la visualizzazione su un grafico a linee.

Dashboard per visualizzazione risultati con SIMATIC Flow Creator

Per poter visualizzare i dati ottenuti dalla query precedentemente descritta è possibile utilizzare la funzionalità di Web Dashboard di SIMATIC Flow Creator con i nodi della libreria *node-red-dashboard*, ovvero una serie di nodi dedicati a diversi oggetti grafici come manometri, campi di testo, grafici e tanto altro.

Per maggiori informazioni sulla libreria *node-red-dashboard* visitare la documentazione ufficiale <https://flows.nodered.org/node/node-red-dashboard>.

In particolare, il nodo *chart-ui* consente di visualizzare grafici di diverso tipo (linee, barre, torta) sulla Web Dashboard di SIMATIC Flow Creator, sia inviando in real-time nuovi dati sia inviando l'intera struttura dati.

In questo esempio applicativo, a partire dai dati ricevuti dal nodo *postgres* come risultato della query, verrà creata la struttura dati corretta per la visualizzazione di un grafico a linee contenente i dati delle ultime 24 ore. Utilizzando il nodo *chart-ui* è possibile definire i vari parametri di configurazione come la Tab ("S7 - PostgreSQL App Example") e il Gruppo ("Control and Monitor") Dashboard di appartenenza:

Prima di procedere alla visualizzazione sarà però necessario formattare correttamente i dati secondo lo standard del nodo *chart-ui*, utilizzando in questo caso un nodo *function* con una funzione dedicata al processamento dell'array di serie temporali ricevuto.

Per maggiori informazioni su come formattare correttamente una o più serie di dati per la visualizzazione tramite nodo *chart-ui* visitare la

documentazione ufficiale del nodo *node-red-dashboard*, dedicata specificatamente al nodo: <https://github.com/node-red/node-red-dashboard/blob/master/Charts.md>

Di seguito la funzione utilizzata per la formattazione delle serie temporali ricevute:

```
// Define influxdb fields names
let outFields = ["goodpieces", "badpieces"]

// Create base out message
let outMsg = {payload:[]};

// Create chart properties
outMsg.payload[0].series = [outFields[0], outFields[1]];
outMsg.payload[0].data = [[],[]];
outMsg.payload[0].labels = [""];

// Scan input data series and create chart data series
for (let i = 0; i < msg.payload.length; i++)
{
    let ts = new Date(msg.payload[i]["time"]).getTime();
    // If value of selected fields exists, push data to array together with timestamp
    if (msg.payload[i][outFields[0]] !== null)
    {
        outMsg.payload[0].data[0].push({"x": ts, "y": msg.payload[i][outFields[0]]});
    }
    if (msg.payload[i][outFields[1]] !== null)
    {
        outMsg.payload[0].data[1].push({"x": ts, "y": msg.payload[i][outFields[1]]});
    }
}

return outMsg;
```

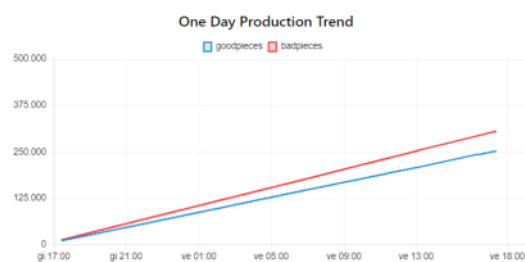
Questa funzione crea la struttura dati per un grafico a linee con due serie temporali denominate "GoodPieces" e "BadPieces", contenente diversi campioni ed il relativo timestamp. Di seguito un estratto di esempio del messaggio in uscita dal nodo *function*:

```
{
  "payload": [
    {
      "series": ["goodpieces", "badpieces"],
      "data": [
        [
          {
            "x": 1616884320000,
            "y": 166138
          },
          ...
          ...
          {
            "x": 1616944260000,
            "y": 92754
          }
        ],
        [
          {
            "x": 1616884320000,
            "y": 72844
          },
          ...
          ...
          {
            "x": 1616944260000,
            "y": 41251
          }
        ]
      ],
      "labels": [""]
    }
  ]
}
```

I dati formattati sono visualizzabili direttamente come due serie su grafico a linee collegandosi alla web dashboard di SIMAT IC Flow Creator utilizzando il link [https://\[device-ip-address\]/ui/](https://[device-ip-address]/ui/) :

≡ S7 - PostgreSQL App Example

Control and Monitor



Dashboard di visualizzazione dati con Grafana

La visualizzazione dei dati caricati all'interno del database PostgreSQL può avvenire non solo tramite l'applicazione SIMATIC Flow Creator, ma anche tramite altri applicativi o modalità.

Un esempio di applicativo utilizzabile è Grafana, una soluzione open-source che consente di interrogare, visualizzare, inviare avvisi ed esplorare le metriche di interesse, indipendentemente da dove sono archiviate. Fornisce quindi gli strumenti per trasformare dati e serie temporali provenienti da diverse sorgenti (PostgreSQL, InfluxDB, MySQL, ecc.) in diversi tipi di grafici.

Per maggiori informazioni su Grafana consultare: <https://grafana.com/docs/grafana/latest/>.

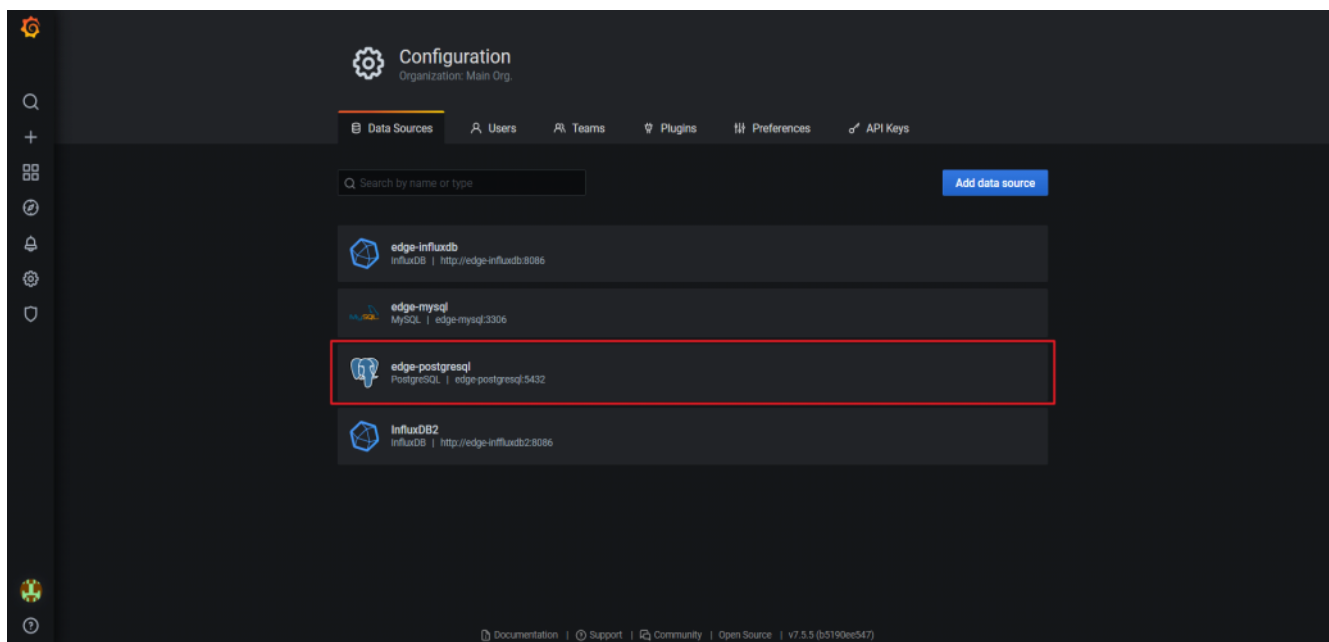
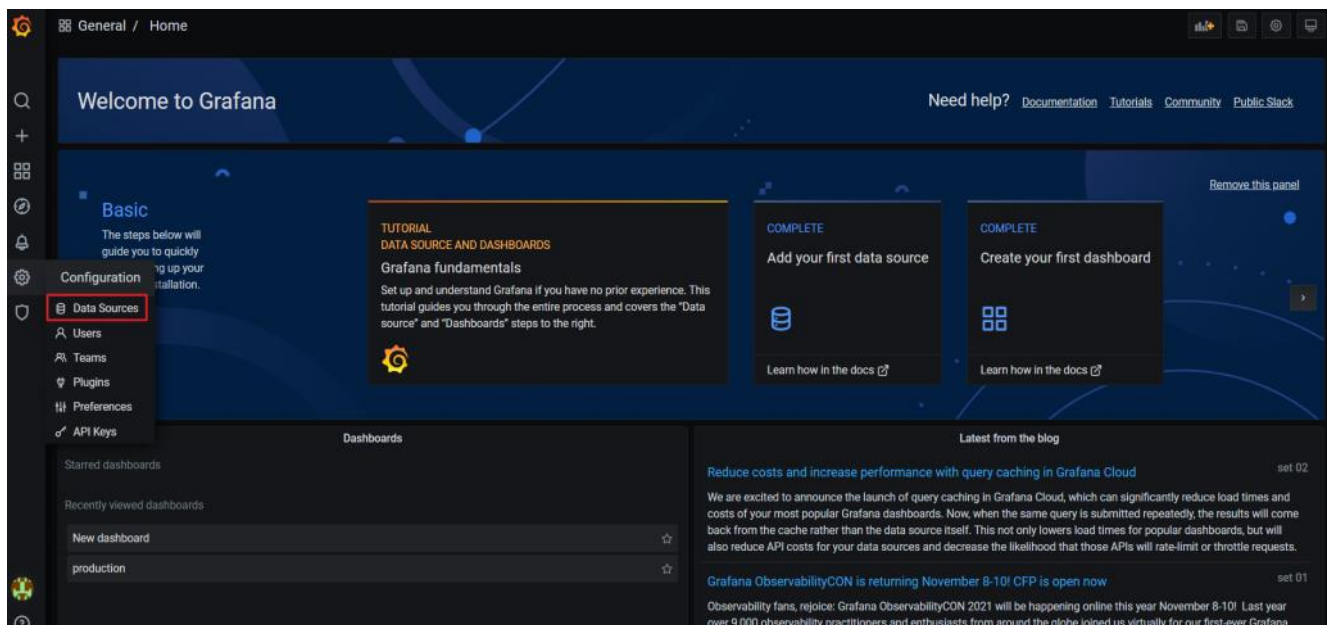
L'applicazione Grafana viene fornita con il pacchetto app precostruito **edge-grafana_x.x.x.app** (in base alla versione fornita x.x.x), pacchetto che può essere installato specificamente sugli Edge Device utilizzando **SIMATIC Edge**.

Per installare l'applicazione, è possibile seguire le stesse indicazioni descritte al paragrafo [3 - Installazione App](#) per l'applicazione edge-postgresql.

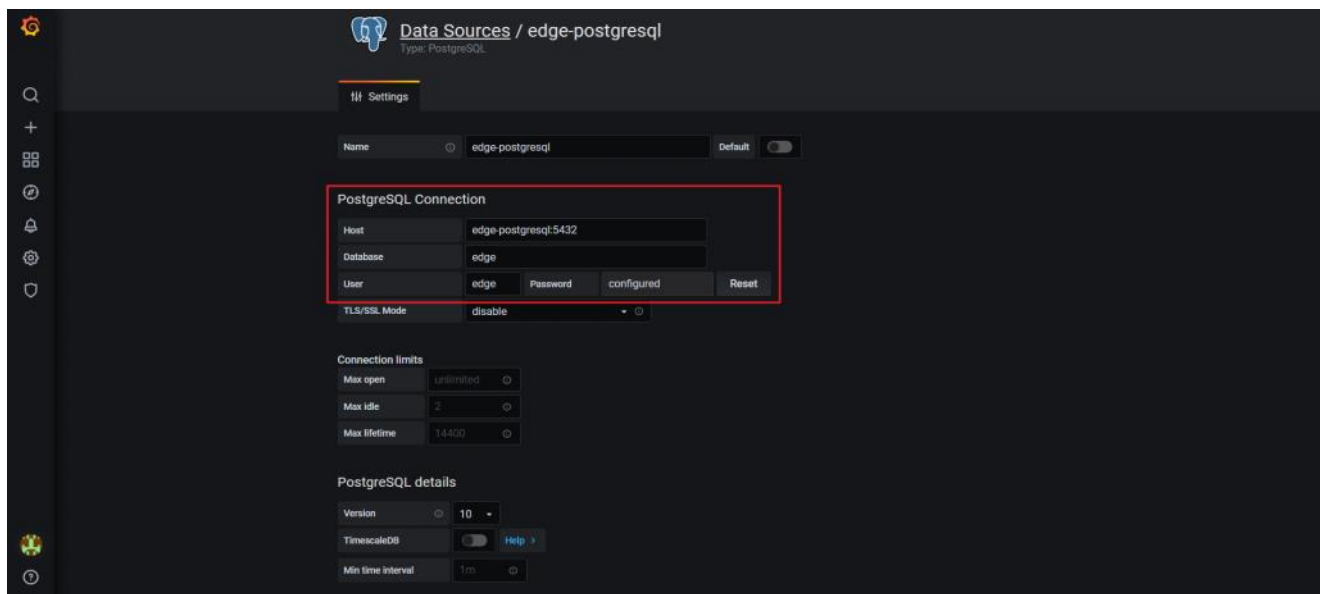
Una volta installata l'applicazione, è possibile accedervi tramite il seguente indirizzo e credenziali:

Service Name	Hostname	Internal Port	External Host	External Port	User	Password
edge-grafana	edge-grafana	3000	[ied-ip-address] oppure [ied-dns-name]	33000	edge	edge

Dalla home page di Grafana, accedendo alla sezione *Configuration / Data sources*, è possibile visualizzare le sorgenti dati già pre-configurate nel pacchetto app preconstituito edge-grafana. Le sorgenti dati configurate sono InfluxDB, InfluxDB2, MySQL e, quella di interesse in questo esempio applicativo, PostgreSQL.



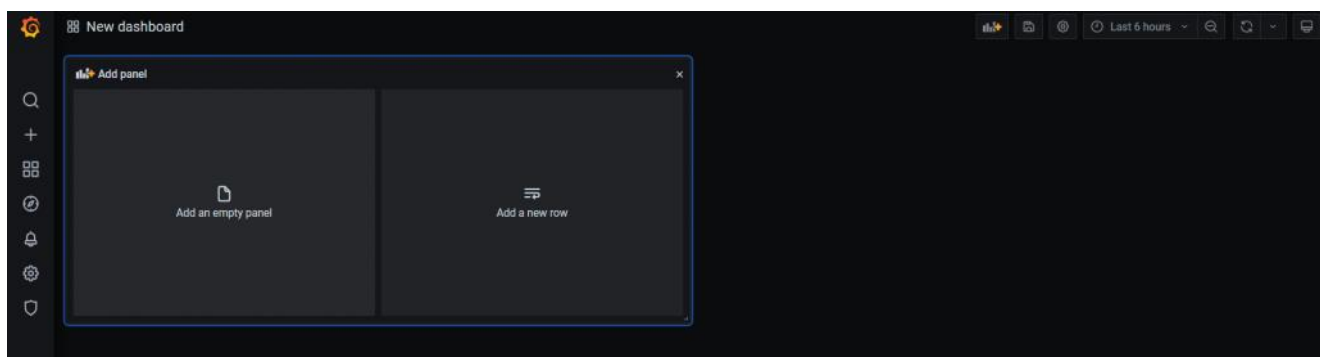
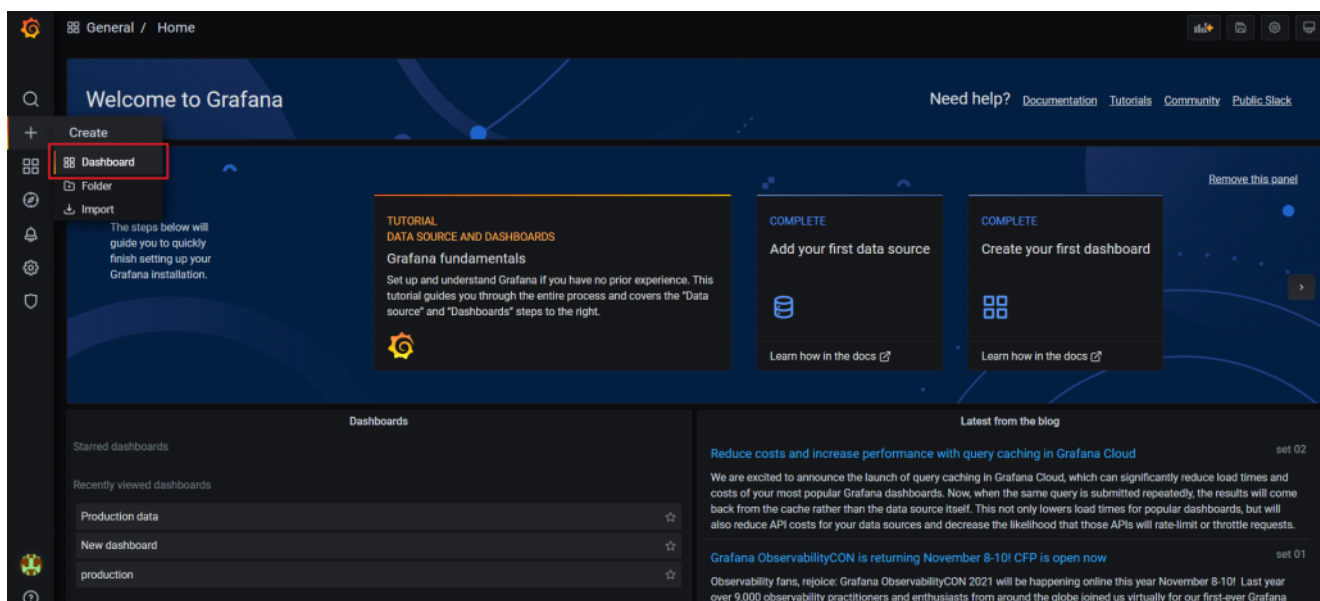
Cliccando su PostgreSQL, è possibile visualizzare i parametri di connessione, coincidenti con quelli descritti al paragrafo [Invio dati a PostgreSQL](#).



Come accennato precedentemente, Grafana consente di visualizzare graficamente dati provenienti da diverse sorgenti, tramite degli appositi pannelli, chiamati **dashboards**.

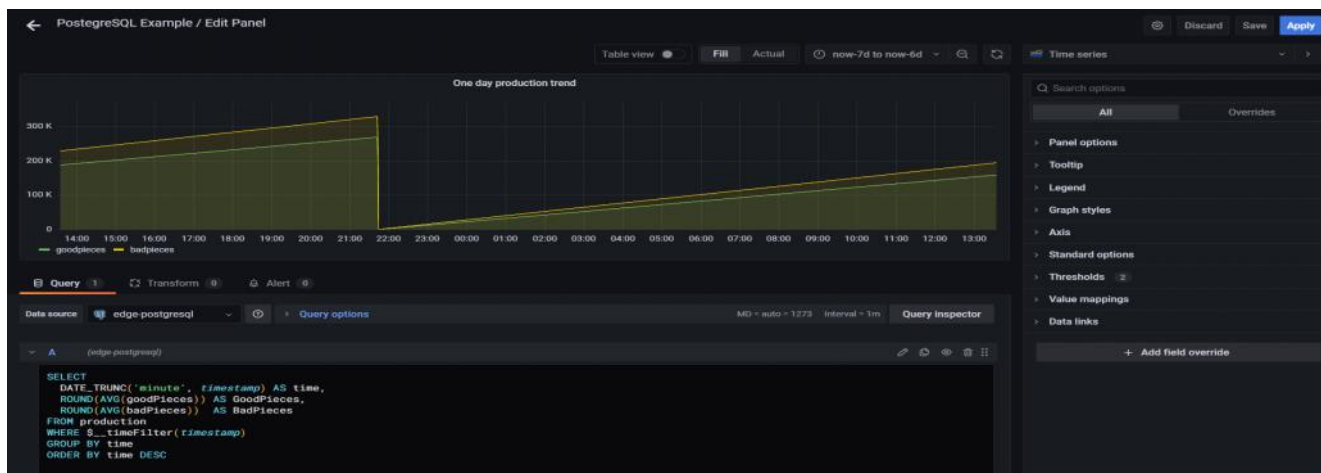
La creazione di questi pannelli può essere fatta mediante query builder interattivi, che facilitano l'integrazione dell'applicazione con sorgenti dati quali MySQL, InfluxDB e PostgreSQL.

Per creare una nuova dashboard con sorgente dati PostgreSQL, basterà selezionare il menu *Create / Dashboard* ed aggiungere poi un nuovo pannello. Il pannello andrà configurato sulla base della visualizzazione desiderata.

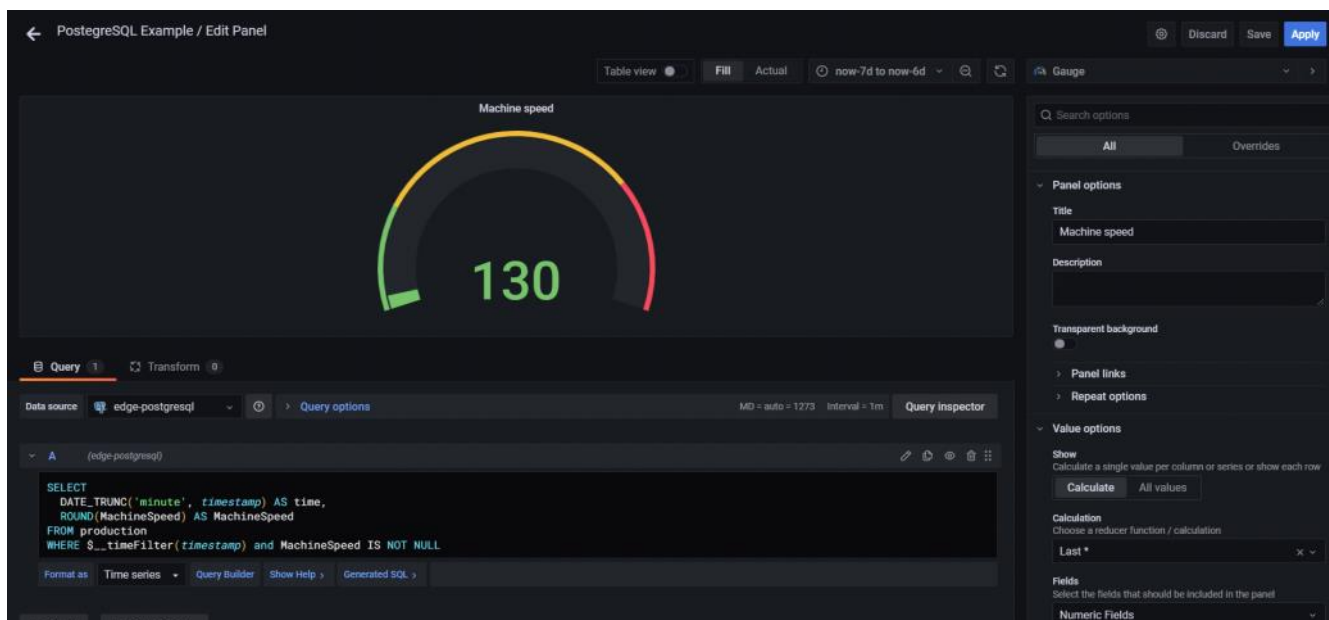


Di seguito un esempio di dashboard costruita per la visualizzazione del trend di produzione giornaliero dei dati letti da sorgente PLC e caricati all'interno del database PostgreSQL (vedi [Scrittura dati provenienti da IE Databus con protocollo MQTT in](#)).

Il grafico, costruito a partire dalla query in figura, permette di visualizzare sia il numero di pezzi conformi che non conformi prodotti nell'ultima giornata. Le caratteristiche del grafico possono essere modificate a piacere tramite il pannello sulla destra della pagina.



Un'altra tipologia di grafico configurabile all'interno di Grafana è il gauge, che permette di definire, per una specifica variabile, delle soglie di tolleranza e discriminare quindi facilmente, a livello visivo, se la variabile in questione è all'interno delle *thresholds* di sicurezza oppure no. Di seguito un esempio di gauge costruito per visualizzare graficamente il valore della variabile MachineSpeed.



La dashboard visualizzata quindi a partire dai pannelli sopra configurati, è la seguente:

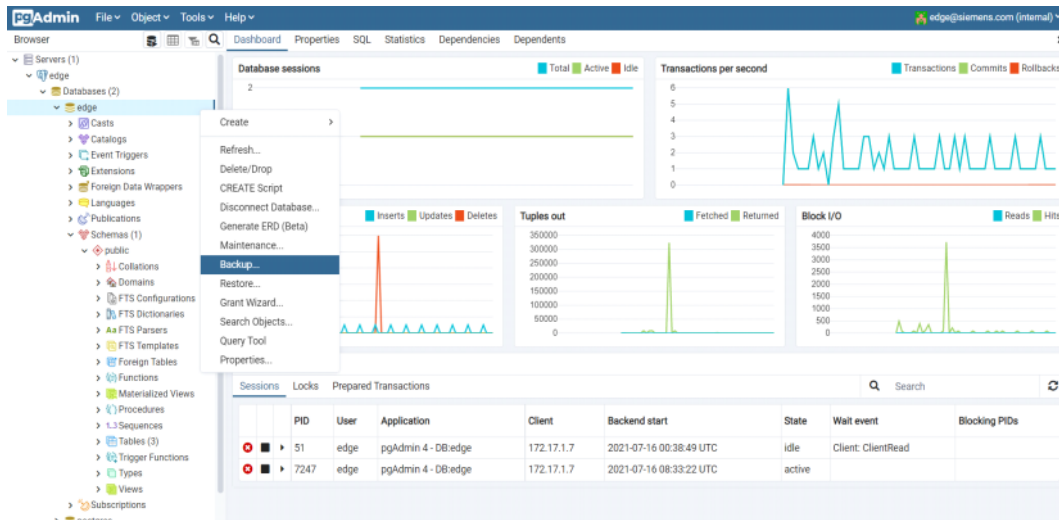


Backup/restore di database PostgreSQL tramite Postgres Static Server

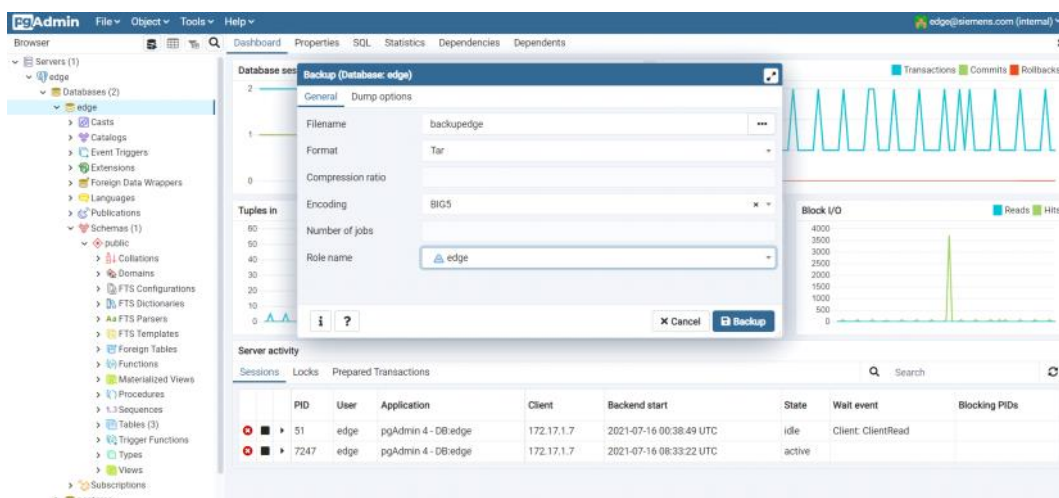
Le funzionalità di backup e ripristino di un database rappresentano uno strumento essenziale per garantire la sicurezza e la protezione di dati critici archiviati all'interno del database stesso. Lo scopo principale di queste funzionalità è infatti quello di ridurre al minimo il rischio di perdita irreversibile di dati, tramite il salvataggio di una copia di questi in locale, da poter ripristinare all'occorrenza. Inoltre, il backup dei file costituenti il database (script di creazione tabelle, viste, procedure o funzioni), permette di facilitare l'export di un database su diversi dispositivi, favorendo la scalabilità di un applicativo su più Edge Devices. In questo esempio applicativo verrà mostrato come poter effettuare il backup e ripristino di un database tramite il servizio **Postgres Static Server** e la funzionalità di **backup/restore integrata nel front-end di pgAdmin**. Nell'esempio, in particolare, il servizio Postgres Static Server verrà utilizzato per importare all'interno di pgAdmin dei file salvati in locale sul proprio PC.

1 - Backup

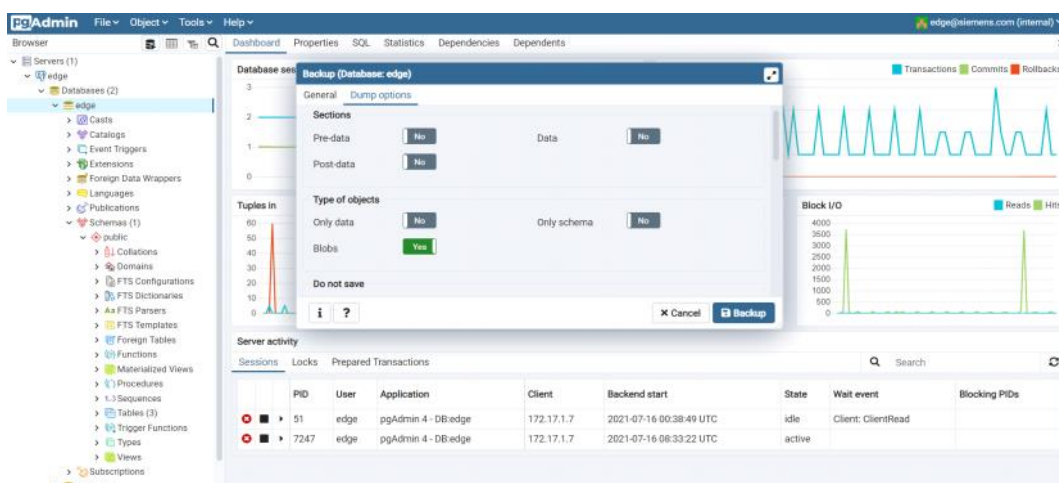
Per effettuare un backup in pgAdmin, occorre selezionare innanzitutto l'oggetto nel database di cui si vuole effettuare il backup. Nell'immagine sottostante il backup è relativo all'intero database "edge", ma è possibile effettuare ad esempio anche il backup di singole tabelle. Una volta selezionato l'oggetto, cliccare con il tasto destro e scegliere la voce *Backup...*



Inserire un nome da associare al file di backup, scegliere il formato Tar e selezionare il tipo di codifica caratteri che si vuole applicare al file caricato. Nell'esempio in figura è stata scelta la codifica BIG5, che utilizza due byte per carattere. In ultimo, specificare il ruolo proprietario del backup.



Quando si effettua un backup del database, pgAdmin dà la possibilità di scegliere tra diverse opzioni di filtraggio dei dati da salvare (*Dump options*). Per maggiori informazioni sulle *Dump options* di Backup, visitare: https://www.pgadmin.org/docs/pgadmin4/development/backup_dialog.html

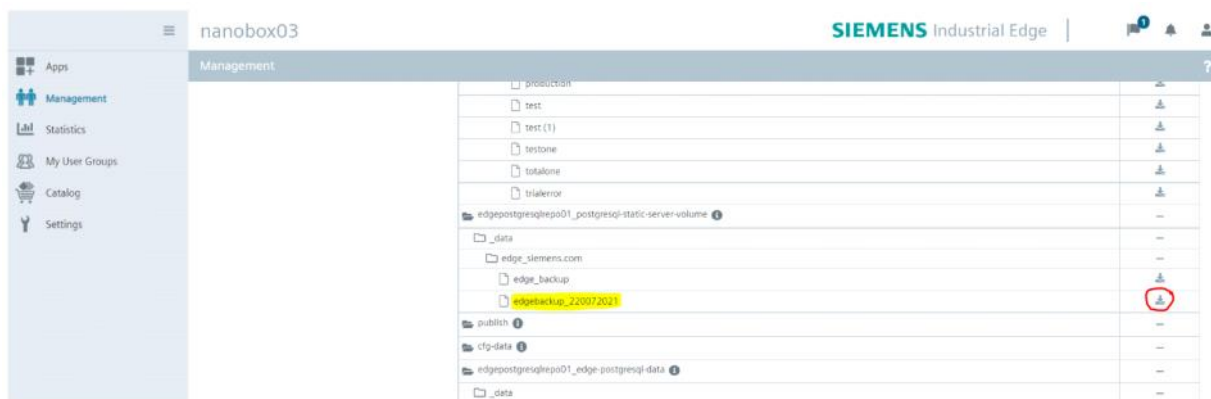


2 - Download del file di backup dagli Application Volumes

Una volta effettuato il backup del database, questo verrà salvato negli *Application Volumes* dell'applicazione edge-postgresql su Edge Device. Per scaricare il file in locale sul proprio PC, è necessario collegarsi alla pagina web dell'Edge Device su cui l'applicazione è installata, entrare nella sezione Management e selezionare l'applicazione edge-postgresql.



Nella sezione *Application Volumes* / *edgepostgresqlrepo01_postgresql-static-server-volume* / *_data* sarà visibile il file di backup creato, sotto la cartella relativa all'utente pgAdmin che ha effettuato il backup (*edge@siemens.com* in questo esempio).



Selezionare l'icona di download e scaricare il file sul proprio PC.

3 - Upload del file tramite Postgres Static Server

Per caricare un file salvato localmente sul proprio PC in pgAdmin, è necessario accedere a Postgres Static Server, tramite l'indirizzo: [https://\[core-name\]:35434](https://[core-name]:35434).

Nella pagina web, selezionare il path in cui si vuole caricare il file di backup. Il path selezionato coincide con l'utente pgAdmin a cui il file sarà poi disponibile per un successivo utilizzo.

Qualora al caricamento della pagina di Postgre Static Server, nel menu a tendina relativo al path non ci fosse nessun record, effettuare prima l'accesso in pgAdmin. Dopodiché, riaggiornare la pagina di Postgre Static Server: nel menu a tendina comparirà il nome dell'utente che ha effettuato l'accesso in pgAdmin.

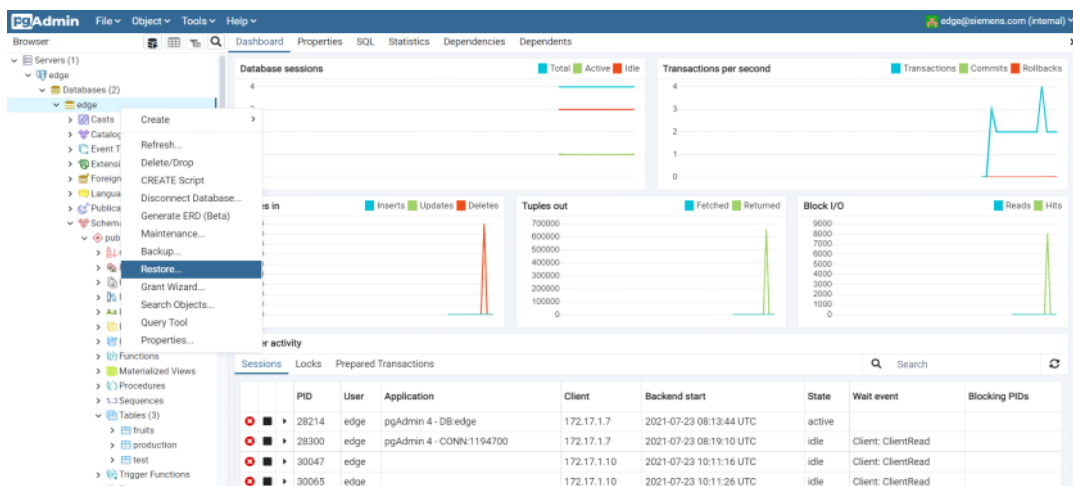
Dopo aver selezionato il path di backup, caricare il file desiderato. La barra di upload mostrerà il progresso nel caricamento del file.

Ad upload terminato, il file comparirà nella lista dei file di backup (sezione *Backup files list* della pagina).

4 - Restore

Per effettuare il restore del file di backup, accedere a pgAdmin e posizionarsi sull'oggetto di cui si vuole fare il restore.

Nel caso in figura, il restore è relativo all'intero database "edge".

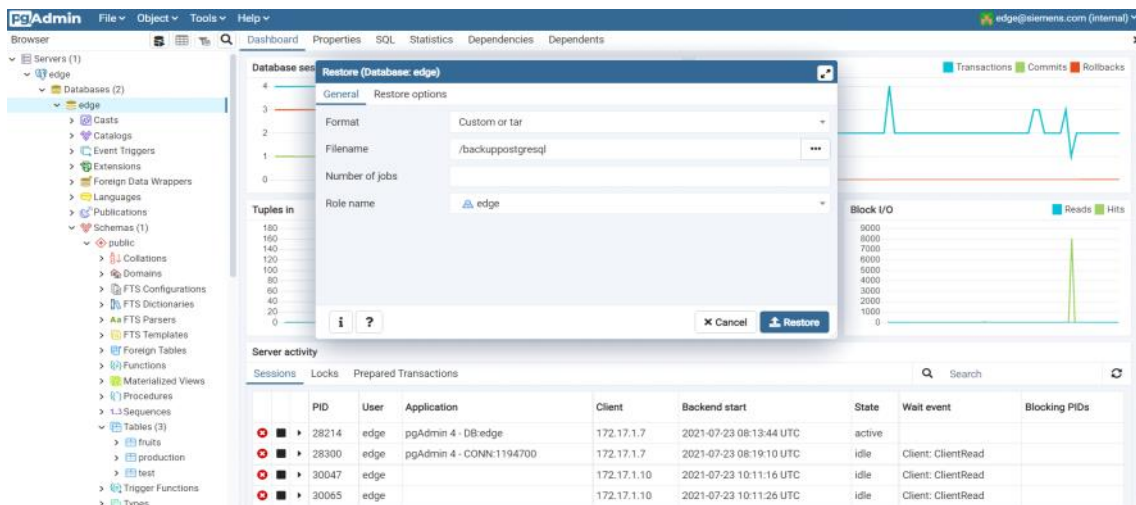
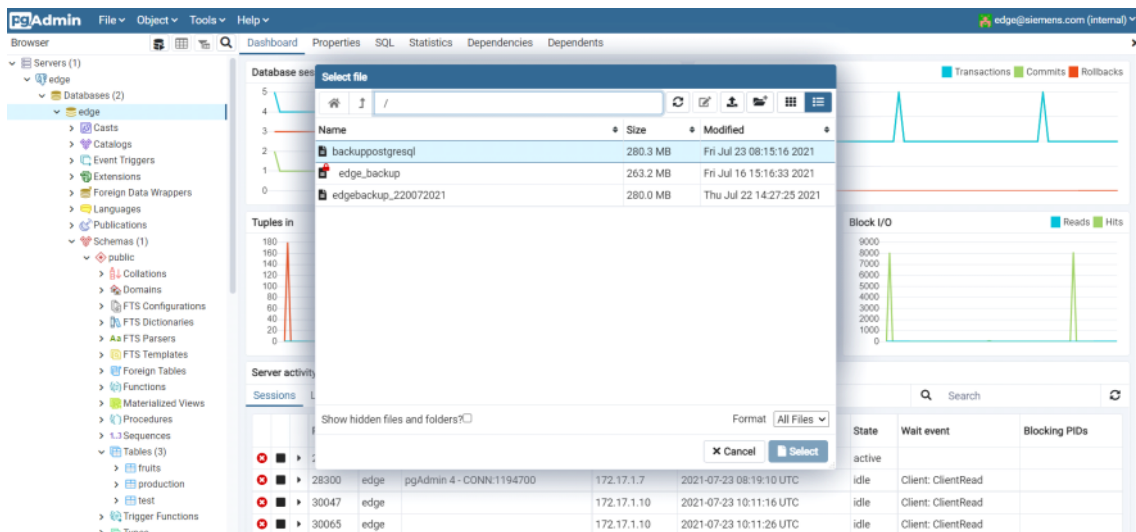
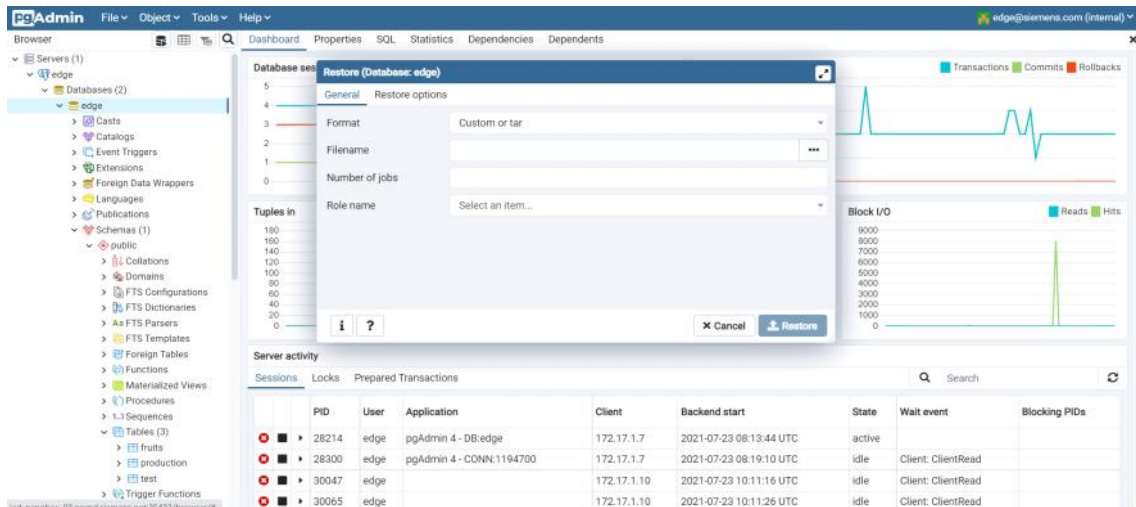


Selezionare la voce *Restore...*, scegliere il formato *Custom or tar* ed il nome del ruolo.

Per selezionare il nome del file di backup da ripristinare, cliccare sui tre puntini accanto alla voce *Filename*.

Nella lista visualizzata compariranno tutti i file caricati tramite Postgres Static Server per l'utente pgAdmin loggato.

Una volta scelto il file, cliccare su **Select**.



Nella tab **Restore options** della maschera di Restore, pgAdmin permette di definire diversi tipi di opzioni di restore dei dati.
Per maggiori informazioni sulle **Dump options** di Restore, visitare: https://www.pgadmin.org/docs/pgadmin4/development/restore_dialog.html

Restore (Database: edge)

General Restore options

Sections

Pre-data ☐ No ☐ Data ☐ No ☐

Post-data ☐ No ☐

Type of objects

Only data ☐ No ☐ Only schema ☐ No ☐

Do not save

Owner ☐ No ☐ Privilege ☐ No ☐

Cancel Restore

Server activity

Sessions	Locks	Prepared Transactions					
PID	User	Application	Client	Backend start	State	Wait event	Blocking PIDs
28214	edge	pgAdmin 4 - DB:edge	172.17.1.7	2021-07-23 08:13:44 UTC	active		
28300	edge	pgAdmin 4 - CONN:1194700	172.17.1.7	2021-07-23 08:19:10 UTC	idle	Client: ClientRead	
30047	edge		172.17.1.10	2021-07-23 10:11:16 UTC	idle	Client: ClientRead	
30065	edge		172.17.1.10	2021-07-23 10:11:26 UTC	idle	Client: ClientRead	

Dopo aver selezionato il file e i parametri sopra indicati, cliccare sul tasto **Restore**.

La barra nella finestra in basso a destra indicherà lo stato di avanzamento nel restore dei dati.

A caricamento concluso, il database verrà aggiornato così da contenere tutti i dati presenti nel file di backup utilizzato per il restore.

Restore (Database: edge)

General Restore options

Sections

Pre-data ☐ No ☐ Data ☐ No ☐

Post-data ☐ No ☐

Type of objects

Only data ☐ No ☐ Only schema ☐ No ☐

Do not save

Owner ☐ No ☐ Privilege ☐ No ☐

Cancel Restore

Server activity

Sessions	Locks	Prepared Transactions					
PID	User	Application	Client	Backend start	State	Wait event	Blocking PIDs
28214	edge	pgAdmin 4 - DB:edge	172.17.1.7	2021-07-23 08:13:44 UTC	active		
28300	edge	pgAdmin 4 - CONN:1194700	172.17.1.7	2021-07-23 08:19:10 UTC	idle	Client: ClientRead	
30047	edge		172.17.1.10	2021-07-23 10:11:16 UTC	idle	Client: ClientRead	
30065	edge		172.17.1.10	2021-07-23 10:11:26 UTC	idle	Client: ClientRead	

Restoring backup on the server

Restoring backup on the server 'edge' (edge-postgresql:5432)

Fri Jul 23 2021 14:28:07 GMT+0200 (Ora legale dell'Europa centrale)

1.61 seconds

More details... Stop Process

Started

6 - Costruzione App

giovedì 5 agosto 2021 16:44

L'applicazione edge-postgresql è composta da diversi microservizi. I microservizi utilizzati in questa versione sono:

- **PostgreSQL**
- **pgAdmin**
- **Postgres Static Server**

Di seguito sono elencati i file principali per la costruzione dell'app:

[docker-compose.yml](#)

Le immagini base che compongono questa app sono state costruite usando lo strumento **docker-compose** (<https://docs.docker.com/compose/>) con il comando **docker-compose up -d --build** sul seguente file **docker-compose.yml**:

```
version: "2.4"

services:
  edge-postgresql:
    image: postgres:13.3-alpine
    restart: always
    environment:
      POSTGRES_DB: edge
      POSTGRES_USER: edge
      POSTGRES_PASSWORD: edge
      PGDATA: /var/lib/postgresql/data
    volumes:
      - edge-postgresql-data:/var/lib/postgresql/data
    ports:
      - "35432:5432"
    mem_limit: 1024m
    networks:
      - proxy-redirect

  edge-pgadmin:
    build: ./pgadmin
    image: edge-pgadmin:0.2.0
    restart: unless-stopped
    environment:
      PGADMIN_DEFAULT_EMAIL: edge@siemens.com
      PGADMIN_DEFAULT_PASSWORD: edge
      PGADMIN_LISTEN_PORT: 80
      PGADMIN_CONFIG_STORAGE_DIR: '"/usr/src/storage"'
    ports:
      - "35433:80"
    volumes:
      - edge-pgadmin-data:/var/lib/pgadmin
      - postgresql-static-server-volume:/usr/src/storage
    links:
      - "edge-postgresql:pgsql-server"
      - "postgresql-static-server"
    mem_limit: 256m
    networks:
      - proxy-redirect

  postgresql-static-server:
    build:
      context: ./static-server
    image: postgresql-static-server:0.2.0
    hostname: postgresql-static-server
    container_name: postgresql-static-server
    restart: always
    ports:
      - 35434:5434
    volumes:
      - postgresql-static-server-volume:/usr/src/app/public/storage
    mem_limit: 150mb
    networks:
      - proxy-redirect

volumes:
```

```

edge-postgresql-data:
edge-pgadmin-data:
postgresql-static-server-volume:

networks:
  proxy-redirect:
    external:
      name: proxy-redirect
    driver: bridge

```

Il file docker-compose sopra crea e porta in esecuzione i servizi **edge-postgresql**, **edge-pgadmin** e **postgresql-static-server**:

- **edge-postgresql:**
 - Usa l'immagine base postgres:13.3-alpine.
 - Mappa la porta 5432 necessaria all'utilizzo del database sulla porta 35432.
 - Crea il database predefinito "edge" e l'utente "edge" con password "edge" per il login al database PostgreSQL.
 - Mappa la cartella /var/lib/postgresql/data all'interno del container edge-postgresql al volume edge-postgresql-data nel sistema host. Qui saranno salvati tutti i dati relativi ai database PostgreSQL.
- **edge-pgadmin**
 - Costruisce l'immagine Docker personalizzata **edge-pgadmin:0.2.0** usando il file **pgAdmin/Dockerfile**.
 - Mappa la porta 80 che consente l'accesso all'interfaccia web di pgAdmin sulla porta 35433, tramite le credenziali definite dalle variabili d'ambiente dedicate (PGADMIN_DEFAULT_EMAIL, PGADMIN_DEFAULT_PASSWORD, PGADMIN_LISTEN_PORT, PGADMIN_CONFIG_STORAGE_DIR).
 - Imposta le connessioni di default verso i microservizi edge-postgresql e postgresql-static-server.
 - Mappa la cartella /usr/src/storage all'interno del container edge-pgadmin al volume edge-pgadmin-data e postgresql-static-server-volume nel sistema host. Qui saranno salvati tutte le customizzazioni fatte in pgAdmin.
- **postgresql-static-server**
 - Costruisce l'immagine Docker personalizzata **postgresql-static-server:0.2.0** usando il file **static-server/Dockerfile**.
 - Mappa la porta 5434 che consente l'accesso all'interfaccia web di Postgres Static Server sulla porta 35434.
 - Mappa la cartella /usr/src/app/public/storage all'interno del container postgresql-static-server al volume postgresql-static-server-volume nel sistema host. Qui saranno salvati tutti i file di backup creati tramite l'interfaccia web di pgAdmin, e verranno inoltre caricati i file di restore di database salvati in locale sul proprio PC.

pgAdmin/Dockerfile

Al fine di personalizzare l'immagine Docker di base per il microservizio edge-pgadmin di questa applicazione, è stato utilizzato il seguente Dockerfile:

```

FROM dpape/pgadmin4:5.3
USER root
RUN mkdir -p /usr/src/storage && chown -R 5050:5050 /usr/src/storage
USER pgadmin

```

Dal file docker-compose precedente, Docker avvierà il processo di costruzione del microservizio edge-pgadmin dall'immagine base pgadmin4:5.3 con il sistema Linux e pgAdmin preinstallati.

Le istruzioni sopra riportate, creeranno inoltre la directory /usr/src/storage e monteranno questa cartella in maniera permanente all'interno del container.

Infatti, considerando che per impostazioni predefinite di pgAdmin le cartelle vengono montate sotto lo user root e non con lo user loggato all'interfaccia web di pgAdmin (UID:GI 5050:5050), è necessario modificare i permessi di accesso alla cartella specifica, così da abilitare la lettura dei file nella cartella anche a user non root. Questo viene fatto, all'interno del Dockerfile, tramite il comando `chown -R 5050:5050 /usr/src/storage` e permetterà poi di visualizzare all'interno della cartella /storage tutti i file di backup del database caricati tramite l'applicazione Postgres Static Server.

Static-server/Dockerfile

Al fine di personalizzare l'immagine Docker di base per il microservizio postgresql-static-server di questa applicazione, è stato utilizzato il seguente Dockerfile:

```

#### Stage BASE #####
FROM node:14-alpine AS BASE

# Install base packages, create data dir
RUN set -ex && \
    apk add --no-cache \
        bash \
        tzdata \
        iputils \
        linux-headers \
        udev \
        openssl

# Copy app packages
WORKDIR /usr/src/app/
COPY ./app/ .

```



```

#### Stage BUILD #####
FROM BASE AS BUILD

RUN apk add --no-cache --virtual buildtools \
        ca-certificates \
        curl \
        openssh-client \
        build-base

# Install App Packages
WORKDIR /usr/src/app/
RUN npm install

# Generate HTTPS certs
WORKDIR /tmp/certs
RUN openssl req -new -newkey rsa:4096 -days 9999 -nodes -x509 \
    -subj "/C=IT/ST=Milan/L=Milan/O=Siemens Spa/CN=postgres-static-server" \
    -keyout server.key -out server.cert

#### Stage RELEASE #####
FROM BASE AS RELEASE

RUN export BUILD_DATE=$(date +"%Y-%m-%dT%H:%M:%SZ")
USER root

WORKDIR /usr/src/app/
# Copy builded apps from BUILD
COPY --from=BUILD /usr/src/app/ ./
# Copy generated certs from BUILD
COPY --from=BUILD /tmp/certs ./certs

CMD ["node", "/usr/src/app/main.js"]

```

Dal file docker-compose precedente, Docker avvierà il processo di costruzione del microservizio postgresql-static-server dall'immagine base node-14:alpine con sistema Linux Alpine preinstallato.

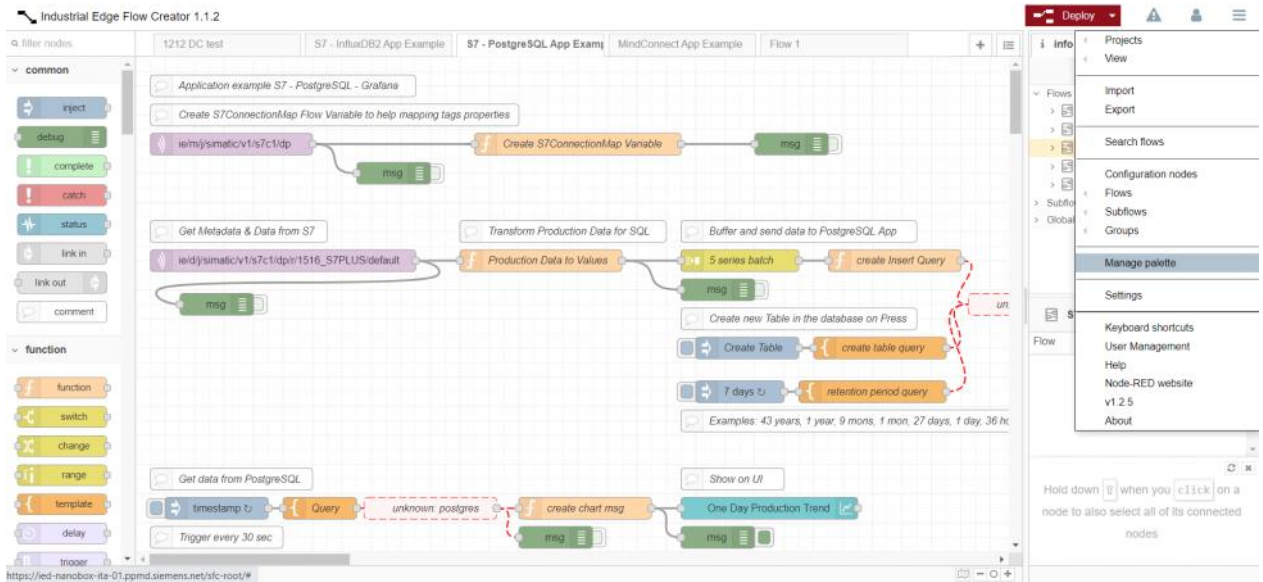
Il Dockerfile sopra riportato, inoltre, installerà i pacchetti di base necessari al funzionamento dell'applicazione, genererà i certificati HTTPS e definirà la working directory.

A - Installazione nodo NodeRED PostgreSQL

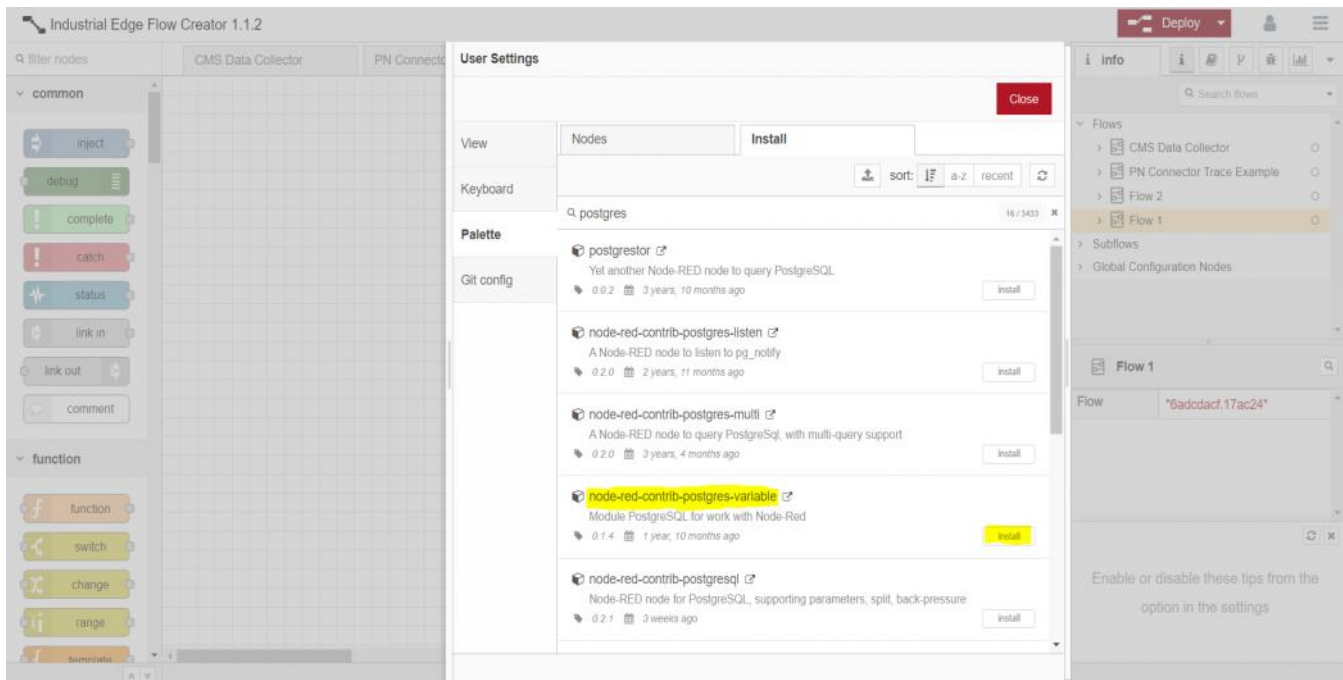
enerdì 6 agosto 2021 11:23

! N.B. - L'installazione di nodi aggiuntivi nell'applicazione SIMATIC Flow Creator (basata su Node-RED) è consentito, ma questi nodi non sono ufficialmente supportati da Siemens.

1. Aprire la pagina Editor dell'applicazione SIMATIC Flow Creator tramite click sull'icona presente sull'Edge Device utilizzato o utilizzando l'URL [https://\[corename\]/sfc-root/](https://[corename]/sfc-root/)
2. Dal menu principale navigare nel sottomenu **Manage Palette**.



3. Cliccare sulla tab **Install**, cercare il nodo con nome **node-red-contrib-postgres-variable** e premere il tasto **Install** per installare il nodo aggiuntivo.



4. Al termine dell'installazione, chiudere il sottomenu tramite il tasto **Close**.
5. Se richiesto, in base al nodo installato, potrebbe essere necessario un riavvio dell'applicazione SIMATIC Flow Creator.

Per maggiori informazioni consultare la documentazione ufficiale del nodo [node-red-contrib-postgres-variable](#).