Faculté des Sciences Jean Perrin
Licence 3 Informatique
TEC : Technologies Emergentes

Année 2017/2018

UNIVERSITÉ D'ARTOIS

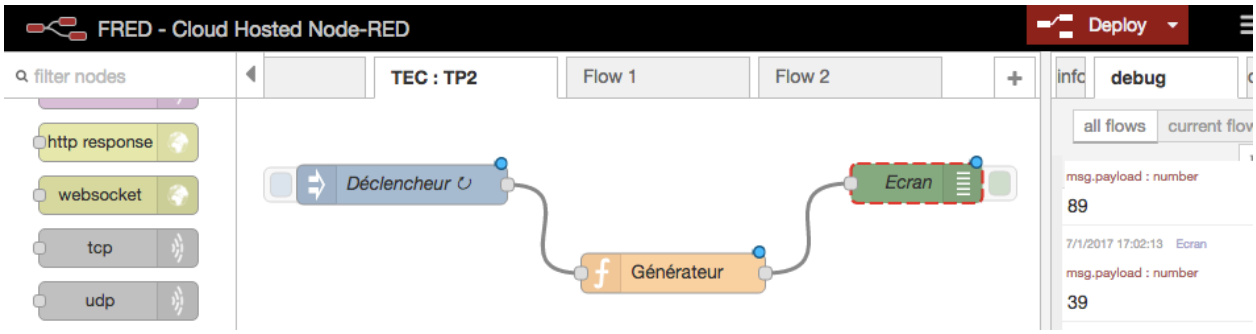
TP 2

Objectifs du TP2

- Découverte du nœud dblite (SQLite) et création d'une table SQLite.
- Génération de données aléatoires et stockage dans la base de données SQLite
- Lecture de la base de données SQLite.
- Représentation graphique des données à l'aide de nœuds standard Node-RED.

Exercice 1: Création d'un flux générant des données aléatoirement

Dans cet exercice, il est demandé de démarrer un nouveau flux nommé TEC:TP2. Comme le montre la capture écran ci-dessous, ce flux comporte trois nœuds: Déclencheur (de type **Inject**), Générateur (de type **Function**) et Ecran (de type **Debug**).

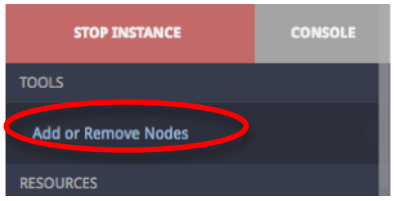


Indication : Voir la doc JavaScript http://www.w3schools.com/js/default.asp pour la génération aléatoire de données.

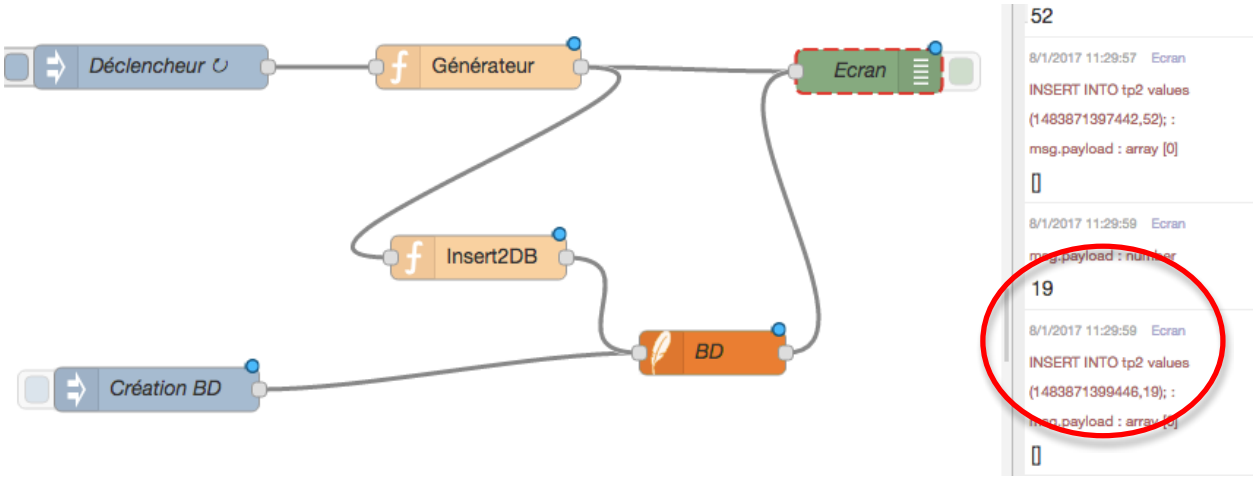
Exercice 2: Stockage des données dans une BD SQLite

Dans cet exercice, nous allons en plus de l'affichage des données sur l'Ecran, enregistrer ces données dans une simple table SQLite.

1. Commencez par ajouter le nœud **litedb** permettant de créer une base de données SQLite. En effet, le nœud **litedb** doit d'abord être ajouté à la liste des nœuds.
   NB. Notez qu'il faut redémarrer votre instance pour voir le nœud **litedb** apparaître dans la liste des nœuds.



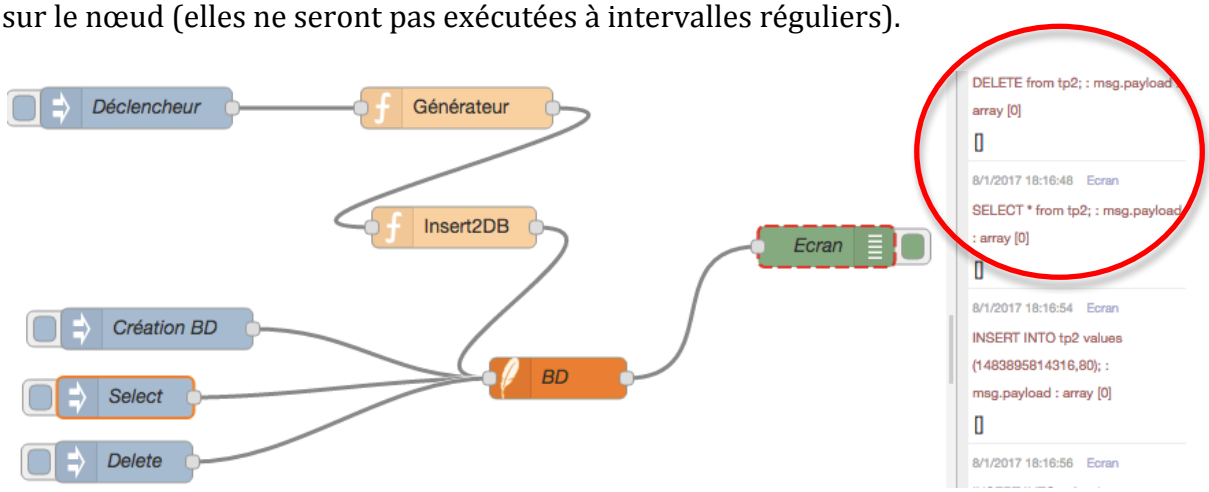
2. Après avoir ajouté un nœud **litedb** nommé BD, il va falloir créer la base et la remplir. Pour cela vous ajouterez un nœud **Function** nommé Insert2DB et un nœud **Inject** nommé Création BD comme sur la figure suivante :



- Dans le nœud **Création BD**, il suffit d'écrire la commande SQL permettant de créer une nouvelle table. Pour simplifier, la table sera nommé **tp2** et elle contiendra uniquement deux colonnes de type entier : *timestamp* et *valeur*. Notez que cette commande SQL est à insérer dans le champ «Topic» du nœud Inject. Bien sûr, nul besoin de configurer ce nœud pour se répéter à intervalles réguliers, on le déclenchera une seule fois manuellement.
  Si la création réussit, on verra le code SQL de création de la table apparaitre dans le Debug.

- Les colonnes de la table ci-dessus sont sensées stocker les valeurs générées par le générateur aléatoire de l'exercice 2. Pour chaque valeur, on insère aussi son *timestamp*. Pour ce faire, on utilisera le nœud **Insert2DB** qui va récupérer la valeur générée par le générateur aléatoire et appelle une fonction JavaScript pour obtenir le *timestamp*. Avec ces deux informations, on forme la requête SQL d'insertion et on la retourne pour l'envoyer s'exécuter sur la base de données.

3. Contrôler le contenu de la base de données : Pour pouvoir consulter et vider la base de données, on peut le faire à travers deux nœuds Inject qui vont envoyer des commandes à la base de données. Pour illustrer cela, on ajoutera un nœud **Select** qui va envoyer une commande permettant de voir tout le contenu de la table tp2 et un nœud Inject nommé **Delete** permettant de supprimer le contenu de la base. Ces nœuds seront configurés pour envoyer ces commander lorsque l'utilisateur clique sur le nœud (elles ne seront pas exécutées à intervalles réguliers).



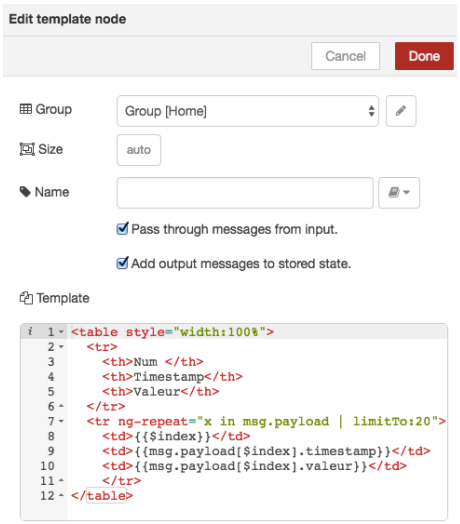
Une commande SQL envoyée et exécutée avec succès par la base de données apparaitra dans le Debug.

Exercice 3: Visualisation des données

Dans cet exercice, on s'intéressera à la visualisation des données enregistrées dans la base de données.

1. Commençons par visualiser le contenu de la table dans une page html. On affichera les données dans une table html comme sur la figure ci-dessous.

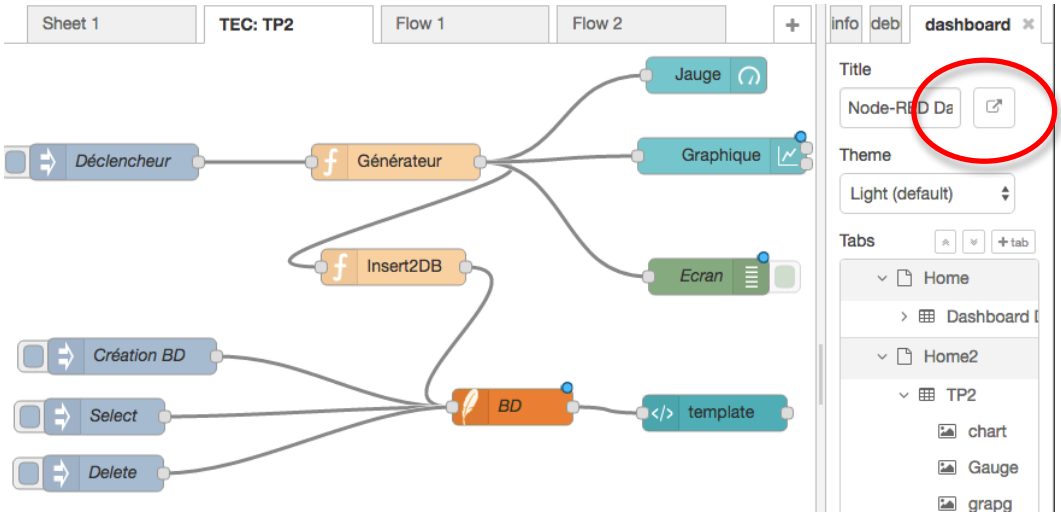
Num	Timestamp	Valeur
0	1483895814316	80
1	1483895816081	1
2	1483896423787	43
3	1483896425008	28
4	1483896425886	39
5	1483896426709	11
6	1483896426988	80
7	1483896427273	46
8	1483896427337	30
9	1483896427751	60
10	1483896428116	60
11	1483896428245	33
12	1483896428509	86
13	1483896428764	65
14	1483896429092	85
15	1483896429227	48
16	1483896429499	69
17	1483896429658	86



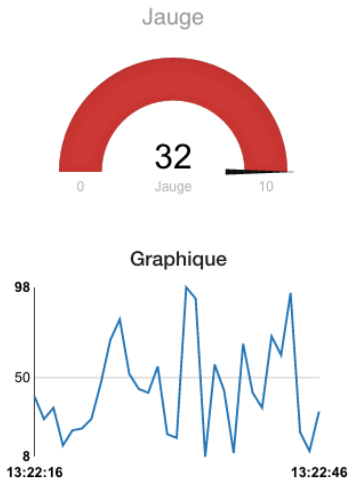
Pour réaliser cet affichage, on aura besoin d'un nœud **Template** (utiliser *ui_template*, dans la catégorie Dashboard). Pour avoir l'affichage ci-dessus, on insèrera le code html – angularJS montré sur la capture écran ci-contre.

Pour information, le code dans le Template est du angularJS (http://www.w3schools.com/angular/) permettant d'étendre html pour construire des pages dynamiquement.

2. Il est temps maintenant de découvrir deux nœuds permettant de visualiser graphiquement les données. Pour simplifier, vous allez juste représenter graphiquement les données fournies par le générateur aléatoire. Pour ce faire, vous allez connecter la sortie de votre générateur directement à une jauge (Gauje dans Node-Red) et un graphe comme sur la figure suivante. Les graphiques sont visibles sur une page Web séparée accessible depuis le lieu entouré en rouge sur la figure suivante.



Par défaut, pour votre jauge et graphique, vous aurez une page qui ressemble à la figure suivante :



Complément sur les messages dans Node-RED

Pour aller plus loin, il convient de préciser certains éléments concernant les messages (msg) échangés entre les nœuds. Un message msg est un objet JavaScript (http://www.w3schools.com/js/default.asp). Les propriétés d'un objet sont accessibles avec la syntaxe:

nomObjet . nomPropriété **ou** *nomObjet["nomPropriété"]*

Pour en créer une nouvelle propriété, il suffit de préciser son objet, son nom et sa valeur avec la syntaxe:

nomObjet . nomNouvellePropriété = valeur.

Dans Node-Red, un objet msg dispose en fonction des nœuds les ayant générés des principales propriétés suivantes:

- *msg.id*: Identifiant du message.
- *msg.payload*: Elle contient le corps principal du message. Par exemple, le texte d'un Tweet, le contenu d'une page Web ou une lecture de capteur.
- *msg.title*: C'est un titre donnant une description de ce que le message contient.
- *msg.description*: C'est une brève description des données contenues dans le message.
- *msg.data*: C'est une propriétés permettant de véhiculer une grande quantité d'informations.
- msg.location: Cette un objet contenant des propriétés représentant des informations de localisation. Plus précisément:
  - msg.location.lat - Latitude
  - msg.location.lon - Longitude
  - msg.location.name - le nom d'un emplacement
  - msg.location.city
  - msg.location.country
  - msg.time - un temps. L'heure doit être l'objet Date JavaScript ou un nombre de millisecondes depuis la référence.

De même, le payload peut-être structuré en fonction des besoins. Par exemple, pour les flux d'une application de fitness/activité physique, on pourra avoir besoin des champs suivants:
-msg.payload.id - ID de l'activité dans le système
-msg.payload.type - Le type de l'activité, par exemple: course, marche
-msg.payload.duration - Durée de l'activité en secondes
-msg.payload.distance - Distance de l'activité en mètres
-msg.payload.calories - Calories brûlées pendant l'activité en kilocalories
-msg.payload.starttime - Objet Date JavaScript représentant l'heure de début de l'activité.