Procedure guidate per snap4city

Un manuale che raccoglie una serie di procedure guidate per poter usare snap4city. In Giallo le sezioni incomplete, in rosso quelle da fare

Indice:

A. Capire la terminologia di Snap4City:

Descrizione degli elementi fondamentali di snap4city:

- **Dashboard**: la dashboard è una interfaccia che un utente può usare per accedere ed elaborare una determinata tipologia di dati; ogni dashboard ha vari elementi (mappe, grafici, menu etc…) che si chiamano widget: con snap4city si possono sviluppare varie tipologie di dashboard e definire una serie di widget che la compongono
- **Widget:** il widget è il singolo elemento che mostra i dati in una dashboard; esistono tante tipologie di widget: mappe, grafici, indicatori di valori etc… Ci sono dei widget semplici e dei widget composti (costituiti da un gruppo di widget che comunica tra di loro); Ogni widget è collegato ad una sorgente dati ed è caratterizzato da una icona;
- **Nature and SubNature (snap4city ontology):** ogni dato di snap4city deve essere descritto secondo una classificazione che prevede una “natura” ed una “sottonatura”; questa classificazione permette a snap4city se sta gestendo dati della stessa natura o di natura differente
- **High Level Type**: ogni tipologia di dato in snap4City deve far parte di un DataType; ci possono essere varie tipologie di Data Type queste sono le principali:
  - **myPOI**: un punto sul territorio associato ad una Natura e SottoNatura (es: un bar, un ristorante, una fermata del pullman);
  - **myKPI**: acronimo di Key Performance Indicator è un punto sul territorio associato ad una Natura e SottoNatura (es: un parcheggio) con la possibilità di associare un valore nel tempo
  - **Sensor**: Un punto sul territorio che acquisisce in automatico dei dati nel tempo; (es. una stazione meteo, un sensore di inquinamento, un sensore del traffico)

La differenza tra myKPI e Sensor è che al MyKPI posso associare un unico valore (es. contatore auto) mentre al Sensor posso associare una serie di valori (tmin, tmax, psum, …)

B. Collegare un sensore a snap4city

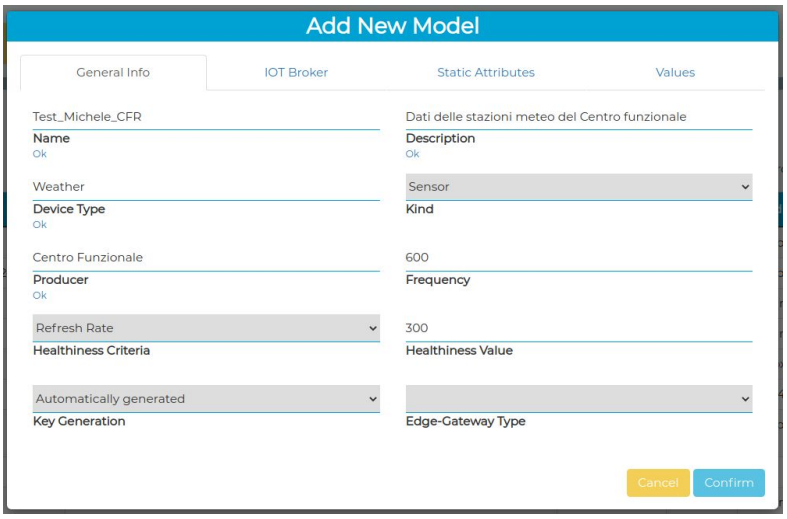
Ecco i passaggi con la definizione degli elementi che servono per descrivere una infrastruttura di sensori con l'approccio Internet of things (IoT). Dopo questa introduzione verranno definiti i passaggi da seguire, ma è importante chiarire la terminologia adottata.

1. L'organizzazione deve avere almeno 1 **IoT Broker**: un broker è un web service accessibile ad un determinato indirizzo che permette di gestire i dati secondo un determinato protocollo predefinito; l'esempio attuale considera che esista già un ContextBroker installato che si chiama orionUNIFI;
2. occorre creare un **IoT Device Models**: il Model definisce un formato di dati che sarà connesso ad 1 o più sensori: per la precisione definisce tutte le variabili che poi saranno registrate ad uno o più device **(X.1)**
3. occorre registrare in snap4city (almeno) 1 **IoTDevice**: il device è un dispositivo posizionato in una determinato luogo connesso con 1 o più sensori; il dispositivo è connesso ad 1 **ContextBroker; (X.2)**
4. creare una **ioTApp**: la iotApp è uno strumento che definisce con un diagramma a blocchi una serie di passaggi per poter inserire i dati nel device **(X.3)**
5. Aggiungere ad una dashboard **(X.4)**

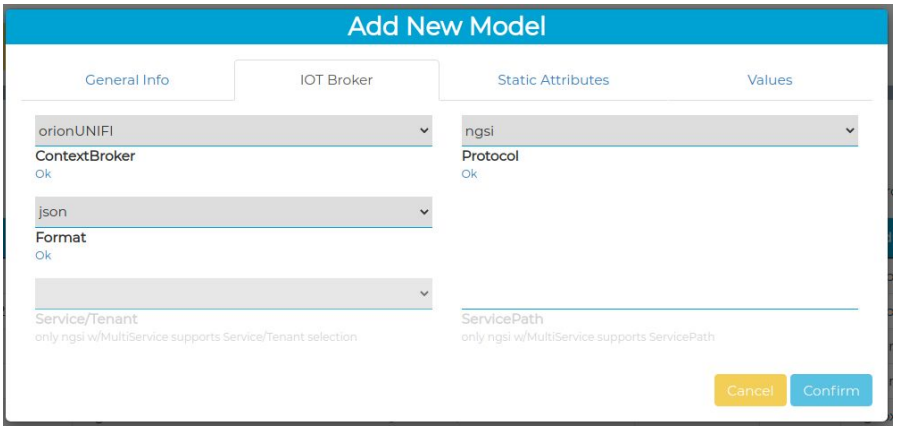
B.1 Creazione di un IoT Device Model

Nel seguente link una descrizione completa in inglese di come si inserisce e configura un https://www.snap4city.org/drupal/node/591)

- va definito il nome del device, la descrizione ed il tipo; in questo caso si inserisce il dato di una stazione meteo e quindi si usa Weather come tipologia



- cliccare su “IOT Broker” e scegliere il broker dalla tendina

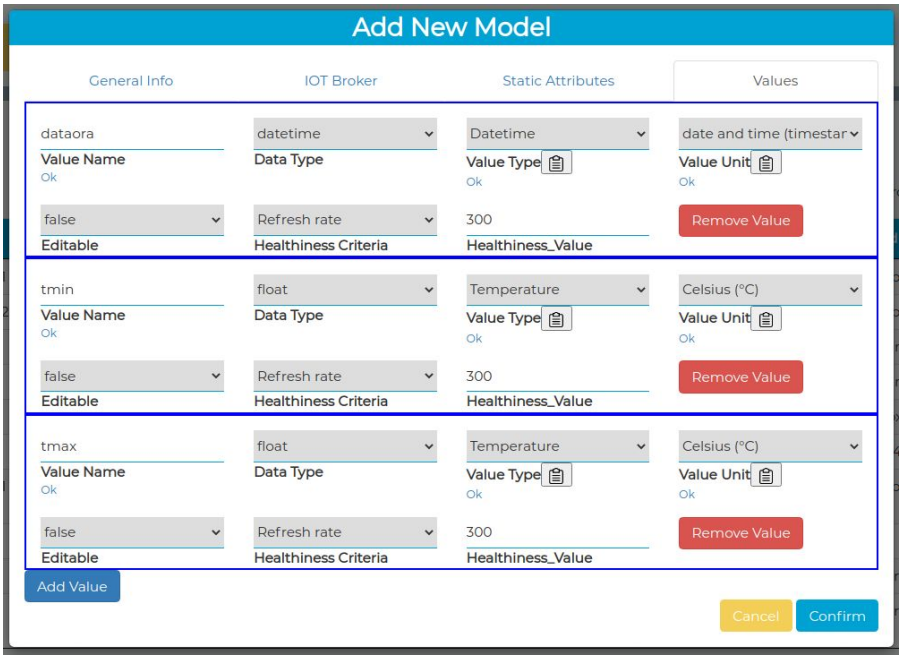


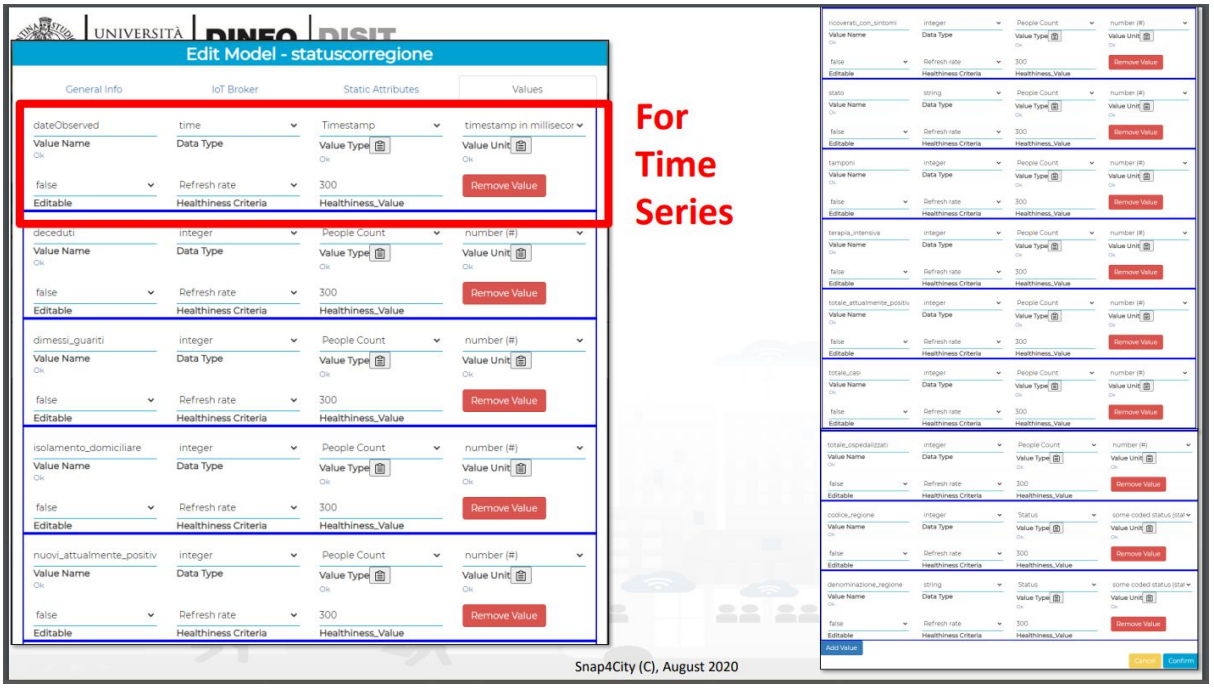
- nel tab Values sì deve andare a definire la struttura dei dati che verranno acquisiti dal modello
    - per ogni variabile aggiungere una “riga” con Add Value
    - selezionare il data type (data, stringa, float, ...)
    - selezionare dalla tendina il value type
    - selezionare il value unitbnn0

Attenzione!!!. Nel dataModel se si ha a che fare con una serie temporale, occorre creare un campo codificato in questo modo:

- il nome deve essere dateObserved (da controllare)
- il Data Type deve essere Time
- il Value Type deve essere **Timestamp**
- il valued deve essere timestamp in millisecond

Se non si codifica bene poi il sistema non riconosce la data di misurazione.

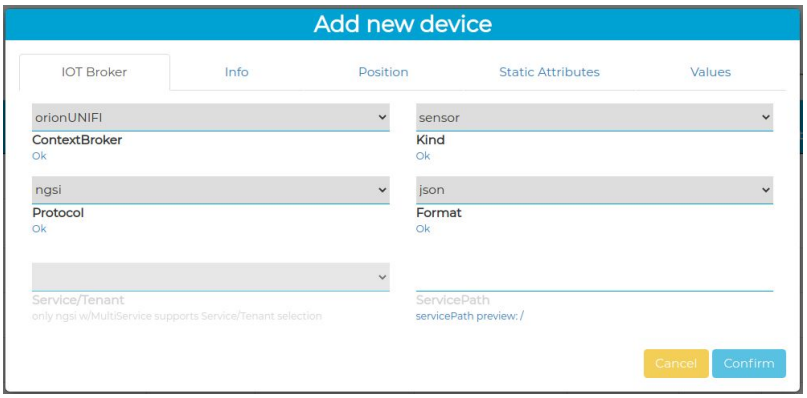




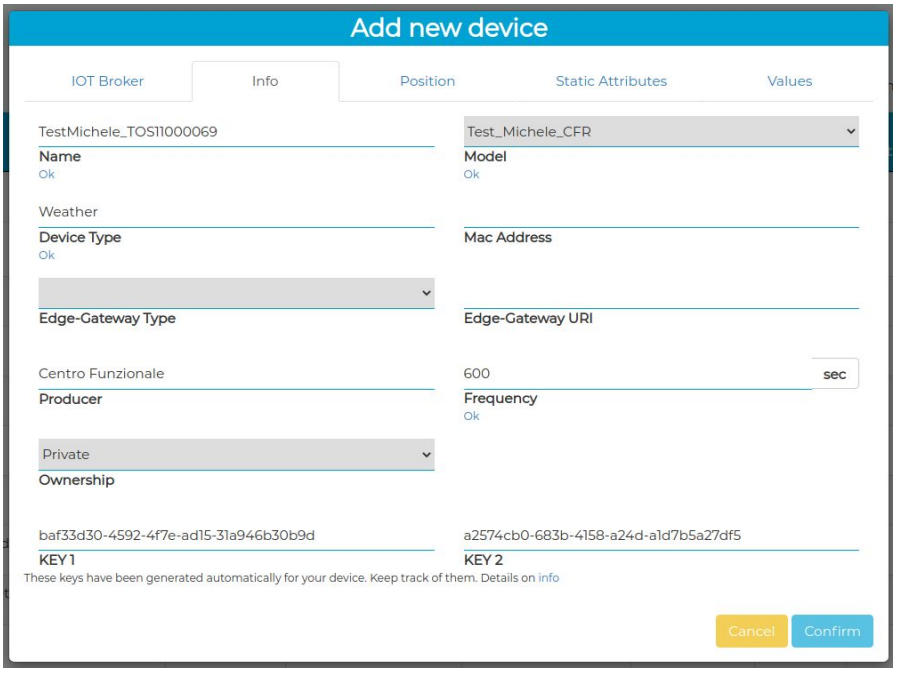
B.2 Registrazione Manuale di un IoTDevice

Aggiungi un nuovo IoTDevice

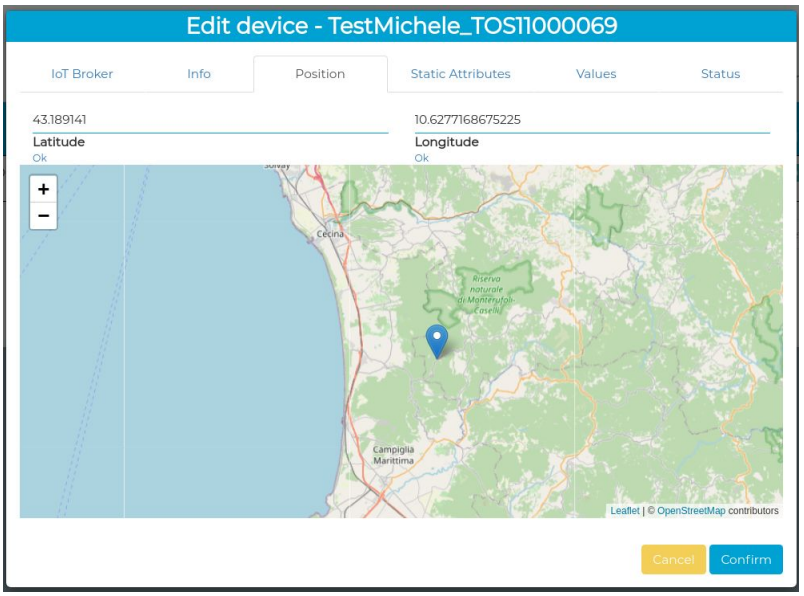
- selezionare il contextBroker



- Nel tab Info:
    - assegnare un nome
    - selezionare il modello (creato al punto precedente)
    - salvarsi separatamente le chiavi KEY1 e KEY2



- Nel tab Position inserire le coordinate

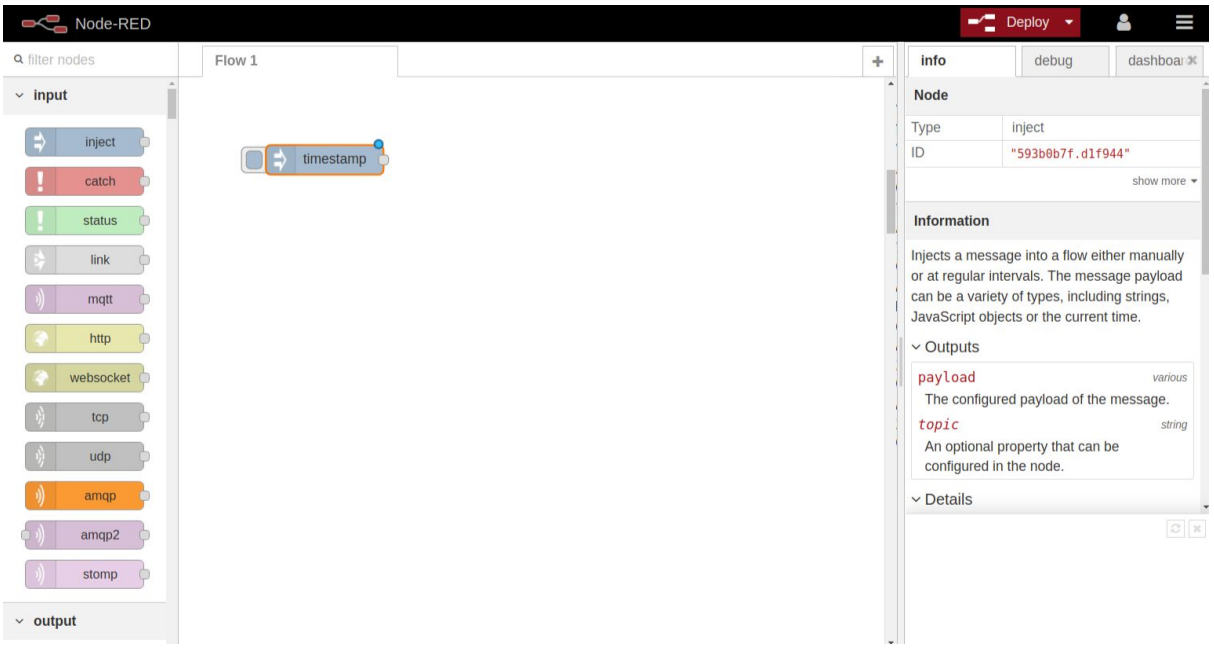


Salvare il device creato

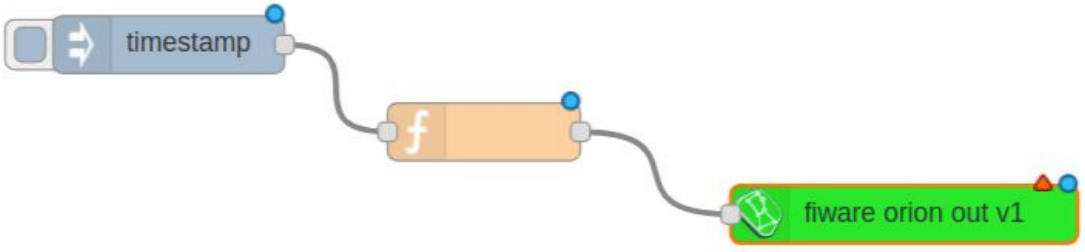
Suggerimento: Nel caso di definizione di più sensori dello stesso tipo (in generale potremmo allargare a tutti i sensori del progetto Pisa), possiamo impostare gli stessi k1 e k2 per tutti i sensori.

B.3 inserire un dato IoT con una IoTApp

- andare su IOT Applications e creare una nuova IOTApp; l'application Type deve essere basic. E' possibile anche ri-usare una IoT Application che avete già creato; si consiglia di inserire in 1 iotApp più di 1 procedura per poterle gestire in maniera più semplici e per non occupare troppe risorse del sistema
- l'iotAPP ha dei blocchi nella colonna di sinistra; selezionare il block "inject" e trascinarlo nella board; Il blocco serve per poter avere un comando per far iniziare una serie di operazioni a cascata; dopo averlo trascinato cliccare su "deploy" per "pubblicare" l'app. A questo punto se cliccate sul "pulsante" alla sinistra del blocco il sistema "inietta" (inserisce) una variabile nel sistema, in questo caso inserisce la data corrente (timestamp) (compare in messaggio "Successfully injected: timestamp)"



- per poter inserire un dato servono almeno tre blocchi; inserire i blocchi e collegarli tra di loro nel modo che si vede nell'immagine successiva
    - inject: già visto per dare il via alla procedura
    - function: è un blocco che permette di aggiungere codice javascript
    - fiware_orion_out_v1: questo blocco permette di inserire i dati in un contextBroker di tipo "Orion Context Broker" alla versione v1; questo è il device attualmente più diffuso, nella libreria ci sono dei blocchi dedicati a contextBroker di altre versioni



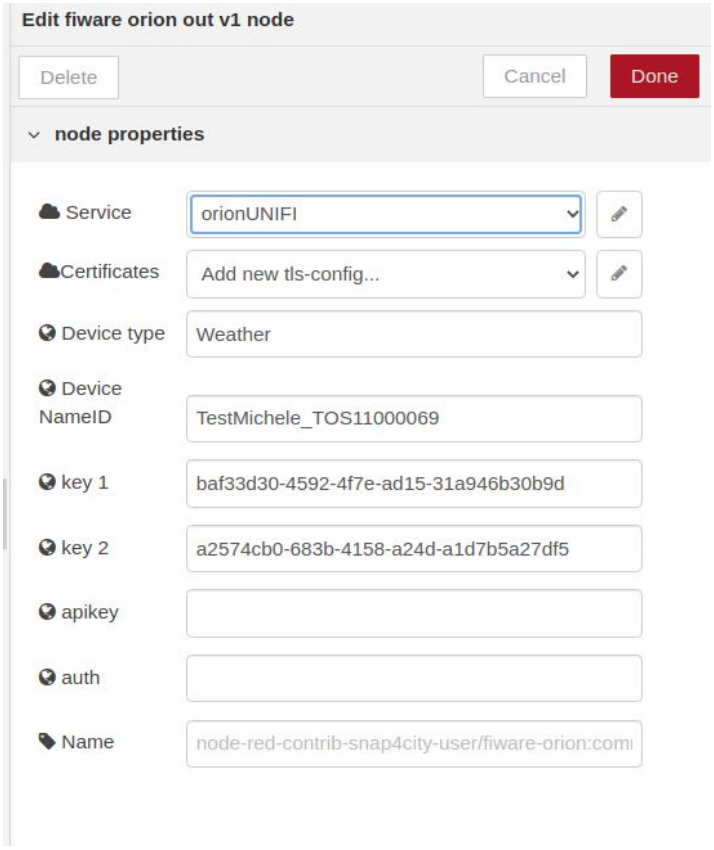
- cliccare sul blocco funzione (quello al centro) e scrivere la seguente funzione di inserimento dei dati; nell'esempio che mostriamo inseriamo ogni volta che si clicca su inject un dato con la data corrente e con dei valori fissi
    - id: il codice esatto del device che avete creato
    - type: il tipo che avete definito nel iotModel e dell'iotDevice

- name: i nomi esatti degli attributi definiti nell'iotModel
- value: i valori che vogliamo inserire; new Date() inserisce la data corrente

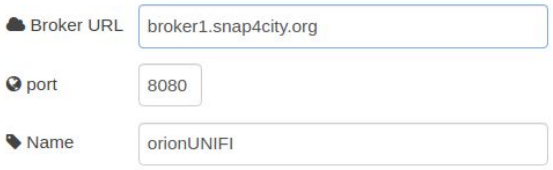
```
var data={
 "id": "test_diego_pisa",
 "type": "misura",
 "attributes":[
    {"name": "dateObserved", "value": new Date().toISOString(),
"type": "time"},
    {"name": "temperature", "value": 32.1 , "type": "float"},
    {"name": "humidity", "value": 44.1 , "type": "float"}
  ]
}
var msg={payload: data}
return msg;
```

Attenzione il campo temporale (dateOserved) deve essere in formato testuale ISO: es: "2020-08-07T12:51:50.548Z"; IN JS da un campo data si ottiene la stringa ISO con la funzione toISOString();

- Configurare il **Fiware orion out v1**
  - fare doppio click
  - inserire orionUNIFI
  - inserire k1 e k2 (se non li avete li trovate nella scheda del device)



Dove il service va modificato con la matita inserendo **orionUNIFI** come nella seguente figura



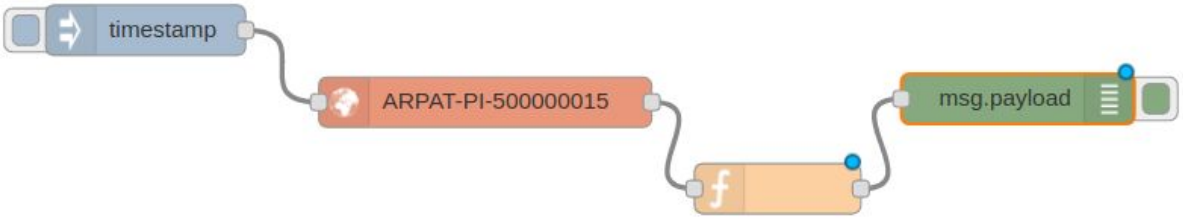
Importante: E' possibile passare l'autenticazione del sensore nel messaggio che viene inviato al Fiware orion out v1; per farlo è sufficiente preparere l'oggetto nel seguente modo:

```
var k1 = "f6efd060-375b-4f40-af09-41e0814f7039";
var k2 = "108bfe91-8140-41fd-92d8-c2b317c3cef8";
...
var auth={"k1": k1,"k2": k2,"apikey":"apikey","basicAuth": "basicAuthKey"};
return {"auth": auth, "payload":ret}
```

B.4 Interrogare un sensore

Per interrogare un sensore utilizzando nodeRed è necessario utilizzare il blocchetto **service info** che deve essere configurato con il ServiceUri del sensore.

Il blocchetti da utilizzare sono i seguenti:

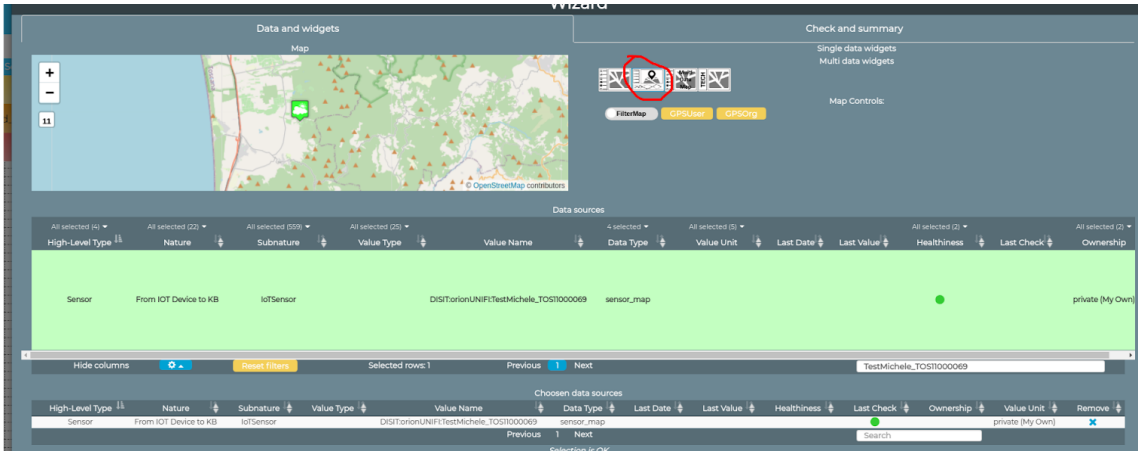


La funzione deve essere fatta in questo modo per prendere il valore dell'ultima data inserita:

```
var input=msg.payload;
var lastRec = input.realtime.results.bindings[0];
var lastDate = lastRec.dateObserved.value;
```

B.5 Creare una dashboard / aggiungere il widget ad una dashboard esistente

- Aprire la Wizard
- Cercare il nome del device che è stato creato
- Selezionare la prima riga (quella con il pallino verde)
- Selezionare il tipo di widget evidenziato nella figura sottostante.



B.6: Aggiungere device con iotApp

C. Aggiungere o modificare un POI

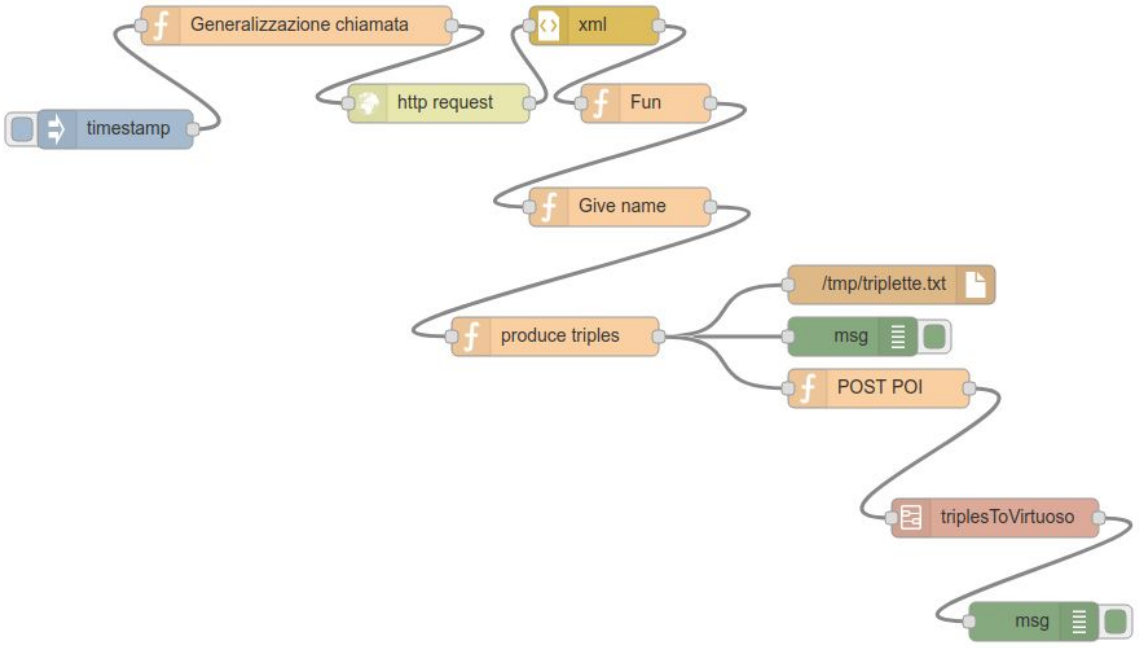
Il POI (Point of Interest) è un sistema che gestisce il punto di interesse.

C.1 formato RDF: come deve essere fatto

<9243476> a km4c:Public_hospital ;
  schema:name "Ospedale Cisanello" ;
  geo:lat "43.7060647"^^xsd:float ;
  geo:long "10.4428542"^^xsd:float ;
  geo:geometry "POINT(10.4428542 43.7060647)"^^virtrdf:Geometry ;
  schema:addressLocality "Pisa" ;
  schema:addressRegion "PI" .

C.2 iotAPP che carica i POI da OSM a snap4citu

Prima tramite http_request, ricevo tutti i dati da OSM, produco le triple e le mando a virtuoso



C.3 fare query su virtuoso

1- Query semplice

SPARQL
```
PREFIX schema:  <http://schema.org/>
SELECT ?soggetto, ?nome
WHERE {
   ?soggetto schema:name ?nome .
}
order by ?soggetto
```

- i prefissi servono per non scivere ogni volta http://schema.org/name -> schema:name
- nel WHERE lui filta tutte le triplette che hanno verbo= http://schema.org/name
    - mette in ?soggetto il soggetto
    - e in ?nome il predicato

2. mostra tutte le triple di un oggetto

SPARQL
```
PREFIX schema:  <http://schema.org/>
SELECT ?soggetto, ?verbo, ?oggetto
WHERE {
   ?soggetto ?verbo ?oggetto .
   FILTER ( ?soggetto =  <http://www.disit.org/km4city/resource/POI_00005>        )

}
order by ?soggetto
```

3 campi multipli

C4 Manuale per Elastic Search

```
GET _xpack/sql?format=txt
{
  "query": """
     SELECT sensorID, max(date_time) FROM "iotdata-organization"
     GROUP BY sensorID LIMIT 5
  """
}

#Cancellare tutti i dati di un sensore
POST /iotdata-organization/_delete_by_query
{
  "query": {
    "match": {
      "deviceName": "test_diego_01"
    }
  }
}
```

HLT	**Ingestion**	**Formato**	**Store**	**API**
POI	caricati con iotAPI (fa un POST su virtuoso) in formato RDF	RDF	Virtuoso (RDF)	API dashboard -> superservicemap -> servicemap -> Virtuoso
MyPOI	(UX) + ??	SQL	MySQL (dashWiz)	
Sensor	UX + IoTAPI	RDF?	Virtuoso + MySQL (dashWiz)	
SensorData	IotAPI	JSON	elasticSearch?	API dashboard -> superservicemap -> servicemap -> Virtuoso
KPI				
MyKPI				

ExternalService	Query su DB	URL		API dashboard -> external_service

#togliere il porvider di dati di OSM
DELETE FROM `servicemaps` WHERE `servicemaps`.`id` = \'sm-201\'"

Aggiungere un `External Service`

```
INSERT INTO `DashboardWizard` (`id`, `nature`, `high_level_type`, `sub_nature`,
`low_level_type`, `unique_name_id`, `instance_uri`, `get_instances`, `last_date`,
`last_value`, `unit`,`metric`, `saved_direct`, `kb_based`, `sm_based`, `user`, `widgets`,
`parameters`, `healthiness`, `microAppExtServIcon`, `lastCheck`, `ownership`,
`organizations`, `latitude`,`longitude`, `value_unit`, `ownerHash`, `delegatedHash`,
`delegatedGroupHash`, `oldEntry`) VALUES (NULL, 'Social', 'External Service', 'Twitter
Vigilance', NULL, 'TestKibana','Firenze', 'get channel/search', NULL, NULL, 'webpagetv',
'no', 'direct', 'no', 'no', NULL, NULL, 'http://192.168.113.115/SMS/kibana_TEST.html',
'true', 'twitter-vigilance.png', '2018-04-16 02:35:00', 'public',
'[Organization,DISIT,Other]', '43.778055', '11.246921', NULL, NULL, NULL, NULL, NULL);
```

Debuggare i sensori

POI

Formato: RDF
Persistenza: Virtuoso
Ingestion: iotApi -> Virtuoso
API: Virtuoso -> servicemap -> superservicemap -> API dashboard

Sensor Observation
Formato: JSON
Persistenza: Mongo (ultima lettura); ElasticSearch (dato storicizzato)
Ingestion: iotApi -> Orion -> (OrionFilter per l'autenticazione) -> ??? -> elasticSearch
-> Mongo
API:

iotApi -> contextBroker ->

- con iotAPI si manda l’oggetto ad un contextBroker

```
var data={
 "id": "test_diego_01",
 "type": "Weather",
 "attributes":[
    {"name": "dateObserved", "value": new Date().toISOString(), "type": "time"},
    {"name": "tmin", "value": 132.1 , "type": "float"},
  ]
}
var msg={payload: data}
return msg;
```

- per verificare se il contextBroker ha aggirnato il dato fare una chiamata curl (la chiamata considera che si fa dal server)

```
curl localhost:1026/v2/entities/test_diego_01?type=Weather -s -S -H 'Accept:
application/json' | python -mjson.tool
```

Z. Installazione di snap4city

Z.1 Preparare la macchina

- la RAM deve essere di almeno 32 GB (altrimenti delle macchine possono fallire)
- Per elasticsearch è necessario configurare il max_map_count della macchina host; se non ha questo valore potrebbe avere dei problemi di memoria (il container docker viene subito fermato)
    - nano /etc/sysctl.conf
    - `#aggiungere la seguente riga`
    - `vm.max_map_count=262144`
    - `sysctl -p`

Z.2 Scaricare il docker-composer ed installare seguendo le istruzioni

```
git clone https://github.com/disit/snap4city-docker
cd snap4city-docker/DataCity-Small

# setup directories write permissions and sets vm.max_map_count=262144 for
elasticsearch
./setup.sh

#bring all services up
docker-compose up -d
#to bring all services up will takes some minutes
```

Z.3 Check delle macchine e post setup

Attenzione!!!! Occorre verificare che tutte le macchine siano partite. Nello specifico soprattutto le macchine con JAVA possono dare problemi di RAM e bloccarsi. Anche ElasticSearch può avere dei problemi se il server non è configurato bene.

```
#then setup virtuoso and elasticsearch (to be done only the first time but no
problem if you repeat)
./post-setup.sh
```

A questo punto seguire le istruzioni e dovrebbe funzionare in localhost

Z.4 Installazione in ambiente di esercizio

Per poterlo installare in esercizio occorre assegnare alla macchina un dominio e poi modificare in tutti gli script il nome del dominio stesso. Vanno modificati anche i riferimenti ai menu nel database mySQL. Abbiamo realizzato uno script **snap4city_onpremise_setup.sh** che effettua un backup dei soli file che verranno modificati con il nome del dominio che daremo all'ambiente di esercizio (nel nostro caso snap4city.aedit.it)

Dopo il run dello script vanno cambiati i redirectURI che sono in keycloak.

- keycloack:
  - accedere al sito con le credenziali admin / admin
  - andare nel clients
  - modificare i seguenti Clients
    - Js-grp-client
    - Js-kpi-client
    - Js-synoptic-client
    - Nodered
    - Php-dashboard-builder
    - Php-iot-directory
    - Php-notificator
    - Php-ownership-api
  - per ognuno aggiungere un Redirect URIs = <nome_del_dominio>

Z.5 Modifiche al docker-compose

- aggiungere un volume a keystore per non dover perdere ad ogni restart la configurazione

Nel docker-compose aggiungere il volume:

```
keycloak:
       ...
       volumes:
         - ./keycloak-db:/opt/jboss/keycloak/standalone/data
```

Va "salvata" la configurazione di snap4city nel volume che sì sta creando attraverso il comando da eseguire all'interno del container

```
> docker exec -it datacitysmall_keycloak_1 bash
> scp -r /opt/jboss/keycloak/standalone/data/*
aedit@192.168.213.68:/opt/snap4city-docker/DataCity-Small/aedit/keycloak-db
```

Z.99 cose non ancora sistemate

Come cambiare tutte le password

Problema con add device

- aggiungere in settings (altrimenti chiama desit)

iotDirectoryUrl: "http://snap4city.aedit.it/iot-directory/",

- attenti se il refresh_token contiene undefined (copiarlo dal tmp e attenzione metterelo accessibile)

--