Sviluppi Snap4City

Andrea Spitaleri

Indice

1 Setup iniziale

La macchina host sulla quale girano le VM è un sistema Linux Kubuntu 20.04.

Ho proceduto col download, configurazione, e messa in funzione delle ultime versioni allora disponibili delle VM necessarie per instaurare una piccola infrastruttura Snap4City. Le VM sono disponibili alla pagina Snpa4City relativa alle Installations, raggiungibile all'indirizzo https://www.snap4city.org/drupal/node/471

Iniziamo con la VM Dashboard. I passi eseguiti sono ispirati alla relativa pagina della guida https://www.snap4city.org/drupal/node/487

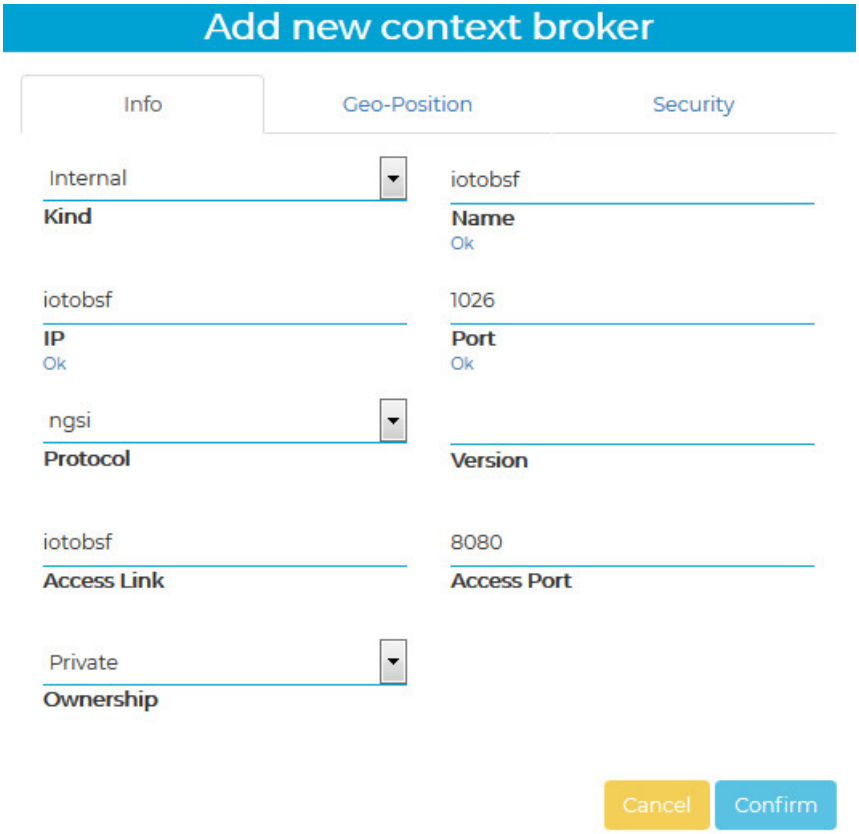
- Ho scaricato la VM Main v1.4 (Dashboard)
- Invece di usare VMWare come indicato ho preferito utilizzare VirtualBox. Per poter fare ciò ho dovuto creare una nuova VM (4 GB Ram, 4 CPUs) e assegnare il disco virtuale appena scaricato. È di fondamentale importanza impostare la scheda di rete guest come "di tipo Bridge".
- Ho avviato il sistema guest, ho annotato l'IP associato alla scheda di rete virtuale e l'ho associato all'hostname "dashboard" (editando il file /etc/-hosts). Il file andrà aggiornato di nuovo al momento della configurazione delle altre VM.
- Ho associato l'hostname "dashboard" a tale IP del guest anche nel file hosts della macchina host. Il file andrà aggiornato di nuovo alla configurazione delle altre VM.
- Ho editato il file /home/debian/nr-run.sh aggiornandolo con l'IP della macchina "dashboard". Il file andrà aggiornato di nuovo alla configurazione della VM IOTOBSF.

A questo punto si può accedere correttamente alla Snap4City Dashboard da browser all'indirizzo http://dashboard, loggando con user userrootadmin e password Sl9.wrE@k .

Dopodiché, proseguiamo con la VM IOTOBSF, ossia il Context Broker interno Snap4City. Pagina guida: https://www.snap4city.org/drupal/node/534

- Ho scaricato la VM IOTOBSF v1.1 (Context Broker Snap4City)
- Ho usato VirtualBox con 2 GB RAM e 4 CPUs. Scheda di rete di tipo Bridge.
- In questo caso ho dovuto prima rinominare la scheda di rete guest come "enp0s3" editando il file /etc/network/interfaces (a causa del fatto che al boot il s.o. rinominava la scheda di rete), e poi procedere con l'associazione dell'hostname "iotobsf" su tutte le VM precedenti e sulla macchina host.
- Ho aggiornato il file /home/debian/nr-run.sh della macchina Main con l'IP di questa VM.

Adesso, accedendo ad http://dashboard ho potuto registrare il Broker nella IOT Directory come di seguito:



A questo punto il Broker interno è configurato correttamente, ed espone il server Tomcat all'indirizzo http://iotobsf:8080.
Adesso per procedere è necessario scaricare e configurare anche la VM KBSSM, ossia la macchina sulla quale gira la Knowledge Base. Pagina guida: https://www.snap4city.org/drupal/node/

- Ho scaricato la VM KBSSM v1.1 (Knowledge Base)
- Ho usato VirtualBox con 4 GB di RAM e 4 CPUs, scheda di rete sempre Bridge.
- Come per le altre VM, ho associato l'IP della macchina guest all'hostname "kbssm" sia sulla medesima VM che su tutte le precedenti e infine anche sulla macchina host, editando su tutte il file hosts

Adesso la VM con la Knowledge Base è attiva e configurata correttamente.

Per iniziare a fare delle prove sul corretto funzionamento del sistema finora configurato, ho dovuto preventivamente documentarmi sul protocollo NGSI e sul funzionamento dei Context Broker. A tal fine rimando alla documantazione ufficiale https://fiware-orion.readthedocs.io/en/master/, e all'elaborato "Big Data Architectures" dello studente Antonino Mauro Liuzzo.

- Creato un Device Model nell'IOT Directory di un ipotetico sensore di temperatura (che quindi raccoglie valori di tipo float) associato al cb iotobsf.
- Creato un nuovo Device, sempre tramite IOT Directory con tale sensore, e associato al Context Broker interno iotobsf. Nota: senza la VM KBSSM attiva ciò non si sarebbe potuto portare a compimento.
- Creato una IOT App Node-Red con annessa Dashboard, che tramite i blocchetti Fiware Orion configurati con i parametri del Device appena creato, manda ogni tanto un valore della temperatura e lo rilegge indietro a scopo di test. Così facendo, il broker crede che sia proprio tale device a mandare i dati. La rilettura dei dati funziona bene, ma il blocchetto Orion Listener che dovrebbe notificare aggiornamenti della temperatura sulla Dashboard non funziona. Ad ogni modo questo piccolo particolare non è essenziale, e comunque ne parleremo nel capitolo 6.
- Con il programma Postman faccio delle richieste GET all'indirizzo http://iotobsf:1026/v2/, e vedo effettivamente l'albero JSON con il Device e l'ultima temperatura registrata.

A questo punto si tratta di creare una VM con un Context Broker sempre di tipo Fiware Orion, ma esterno alla piattaforma Snap4City. Allo scopo userò infatti l'implementazione del CB Fiware Orion pubblicata da Telefónica I+D alla pagina https://github.com/telefonicaid/fiware-orion. Ecco come ho proceduto:

- Ho creato tramite VirtualBox una nuova VM con 4 GB di RAM, 4 CPUs, scheda di rete Bridge, e s.o. Xubuntu con username xubuntu. Per una prossima implementazione potrei anche utilizzare un s.o. Debian senza Desktop Environment così come sulle altre VM, al fine di rendere la VM più leggera sulle risorse della macchina host.
- Anche qui ho associato l'IP della VM all'hostname "extcb" sia sulla stessa macchina che su tutte le precedenti che sulla macchina host, editando il file hosts.
- Come indicato alla pagina https://stackoverflow.com/questions/48504976/installing-orion-context-broker ho eseguito i seguenti comandi sulla VM guest:
  - sudo apt-get install emacs aptitude git
  - cd
  - mkdir git
  - cd git
  - git clone https://github.com/telefonicaid/fiware-orion.git
  - cd fiware-orion
  - sudo aptitude install cmake g++ libcurl4-openssl-dev uuid-dev libgnutls-dev libgcrypt-dev scons libboost-dev libboost-regex-dev libboost-thread-dev libboost-filesystem-dev
  - sudo mkdir /opt/libmicrohttpd
  - sudo chown xubuntu /opt/libmicrohttpd

  - cd /opt/libmicrohttpd
  - wget http://ftp.gnu.org/gnu/libmicrohttpd/libmicrohttpd-0.9.48.tar.gz
  - tar xvf libmicrohttpd-0.9.48.tar.gz
  - cd libmicrohttpd-0.9.48
  - ./configure –disable-messages –disable-postprocessor –disable-dauth make
  - sudo make install
  - sudo ldconfig
  - sudo mkdir /opt/mongodb
  - sudo chown xubuntu /opt/mongodb
  - cd /opt/mongodb
  - wget https://github.com/mongodb/mongo-cxx-driver/archive/legacy-1.1.2.tar.gz
  - tar xfvz legacy-1.1.2.tar.gz
  - cd mongo-cxx-driver-legacy-1.1.2
  - sudo scons install –prefix=/usr/local –disable-warnings-as-errors
  - sudo mkdir /opt/rapidjson
  - sudo chown < xubuntu /opt/rapidjson
  - cd /opt/rapidjson
  - wget https://github.com/miloyip/rapidjson/archive/v1.0.2.tar.gz
  - tar xfvz v1.0.2.tar.gz
  - sudo mv rapidjson-1.0.2/include/rapidjson/ /usr/local/include
  - sudo touch /usr/bin/contextBroker
  - sudo touch /etc/default/contextBroker
  - sudo mkdir /etc/init.d/contextBroker
  - sudo chown xubuntu /usr/bin/contextBroker
  - sudo chown xubuntu /etc/default/contextBroker
  - sudo chown xubuntu /etc/init.d/contextBroker
  - cd /git/fiware-orion
  - make di

- A questo punto possiamo avviare il Context Broker esterno lanciando su VM extcb il comando “contextBroker -fg”.
- Per vedere se il CB è effettivamente in funzione possiamo collegarci (da qualunque delle precedenti macchine) all’indirizzo http://extcb:1026/version e dovremmo ricevere una risposta di tipo:

```
{
    {
        "orion" : {
                "version" : "2.4.0-next",
          "uptime" : "0 d, 0 h, 0 m, 23 s",
          "git_hash" : "4
             ce87519e5b5141c1e77abfa832d7346a62ec750",
          "compile_time" : "nodate",
          "compiled_by" : "xubuntu",
          "compiled_in" : "extcb",
          "release_date" : "nodate",
          "doc" : "https://fiware-orion.rtfd.io/"
                }
        }
}
```

A questo punto il Context Broker esterno Fiware Orion è perfettamente up and running.

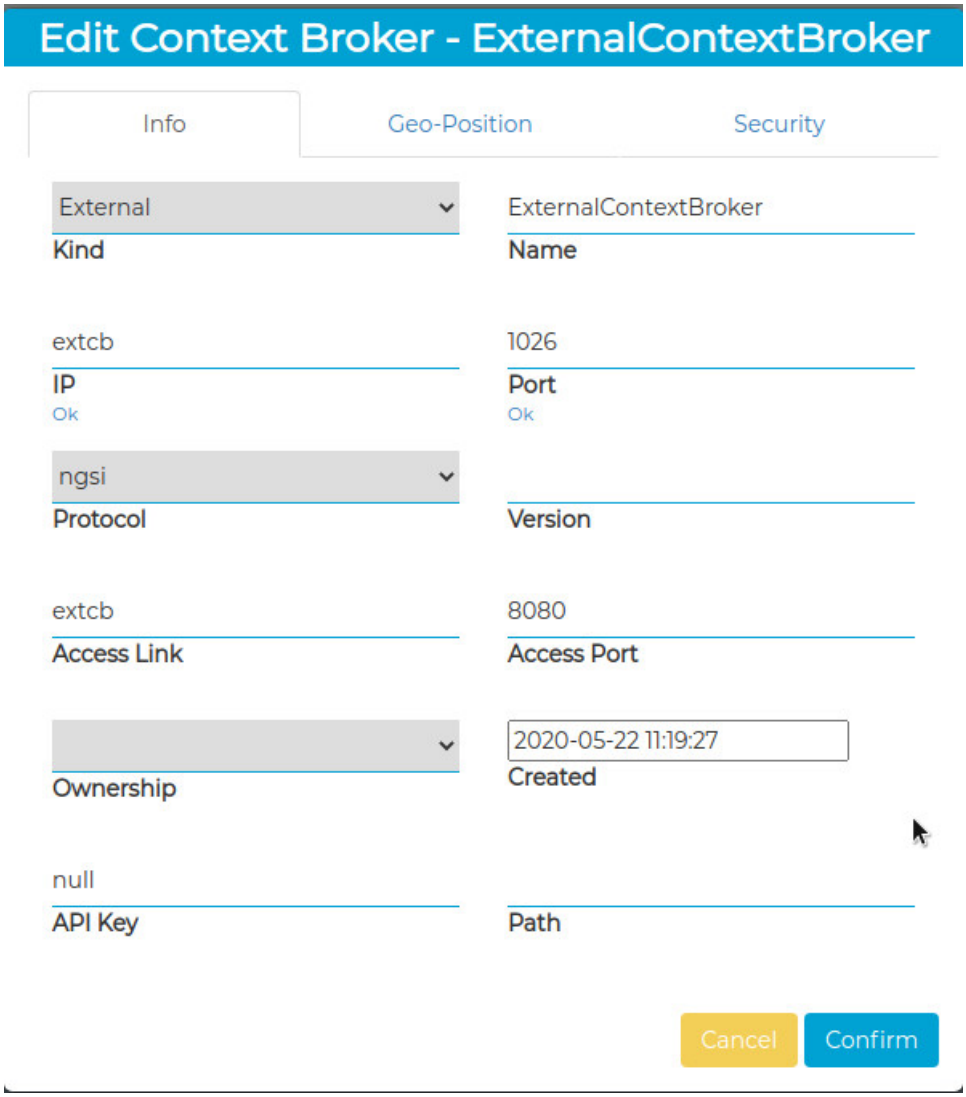
2 Configurazioni, debug e primi tentativi

Adesso registro un Device chiamato Ext_temp_sensor sul nuovo CB. Per fare ciò, tramite Postman posso mandare delle richieste POST all'indirizzo http://extcb:1026/v2/entities/ con un body del tipo:

```
{
    "id": "Ext_temp_sensor",
        "type": "Test",
        "latitude": {
                "type": "float",
                "value": "43.86176",
                "metadata": {}
        },
        "longitude": {
                "type": "float",
                "value": "11.03165",
                "metadata": {}
        },
        "model": {
                "type": "string",
                "value": "Temperature sensor",
                "metadata": {}
        },
        "temperature": {
                "type": "float",
                "value": 30.901039076,
                "metadata": {}
        }
}
```

Facendo una richiesta GET al medesimo indirizzo possiamo notare che il Device è stato correttamente registrato.

Dopodiché, visto che il Context Broker extcb funziona bene, lo registro nella IOT Directory, così come avevo fatto con il cb iotobsf, ma stavolta con i seguenti parametri:



Il passo successivo è quello di importare il Device (precedentemente registrato sul CB extcb tramite Postman) nella IOT Directory. Prima di fare ciò devo creare un nuovo Device Model associato stavolta al cb extcb, sempre con temperatura di tipo float.

Il primo problema riscontrato è stato che nel momento in cui registro il device esterno nella IOT Directory, alla pressione del tasto "Check" (che dovrebbe controllare se effettivamente nel cb extcb esiste un device col nome indicato –

cosa comunque non estremamente esplicita e chiara nella UI) non accade nulla, o meglio la pagina rimane in attesa infinita.

A una prima ispezione tramite il Google Chrome Dev Tool noto che la console presenta un errore di accesso negato per "user@localhost". Ci dev'essere una errata configurazione per i parametri di configurazione al DB!
Allora, sulla VM Main ho modificato il file /home/debian/iot-directory/web/conf/database.ini in modo tale che l'username di tipo "prod" sia "root" invece che semplicemente "user":

```
fileDesc = "Database"
customForm = "false"
fileDeletable = "false"
username[desc] = "Database username"
username[dev] = "user"
username[test] = "user"
username[prod] = "root"
password[desc] = "Database password"
password[dev] = "pwd"
password[test] = "pwd"
password[prod] = "password"
dbname[desc] = "Schema name"
dbname[dev] = "iotdb"
dbname[test] = "iotdb"
dbname[prod] = "iotdb"
```

Adesso la connessione al DB avviene, ma il risultato finale non cambia... La pagina web rimane in attesa infinita.

Analizzo allora sempre tramite Google Chrome Dev Tool, usando breakpoint per capire il flusso d'esecuzione del file /home/debian/iot-directory/web/management/js/devices.js .
Quando viene chiamata la funzione "activateStub", da lì viene fatta una richiesta POST contenente i dati del device appena inseriti nel form, e poi non succede più nulla, o meglio il sistema rimane in uno stato di attesa infinita.

Analizzando la cosa lato server, vedo, tramite il comando "ps -aux | grep node" che dal momento in cui la VM Main viene avviata c'è un server NodeJS in esecuzione:

```
root 1554 0.0 0.9 897844 39944 ? Sl 17:06 0:00 nodejs
   snapIoTDirectory_rw.js
```

cioè un processo nodejs con parametro (in questo caso solo il file js) snapIoTDirectory_rw.js

Nel momento in cui faccio l'aggiunta del device external e il Check va in attesa infinita, allora, con lo stesso comando, noto che si è creato un altro processo:

```
root 4153 5.1 1.0 1002580 44440 ? Sl 22:04 0:00 node
   ./snap4cityBroker/externalBroker.js
   ExternalContextBroker Ext_temp_sensor  extcb:1026
   userrootadmin  extcb 8080  custom  null  null
   Organization  external null
```

ossia un processo node con vari parametri: il primo è un file js in esecuzione e il resto sono proprio i parametri inseriti nel form! Questo processo rimane aperto per un tempo infinito.

Al fine di fare debug interrompo entrambi i processi, mi sposto nella cartella /home/debian/iot-directory/web/snap4cityServer/ e riavvio il processo nodejs snapIoTDirectory_rw.js.

Adesso posso vedere a video tutte le stampe dei console.log (non conoscevo ancora metodi più pratici per fare debug di server NodeJS).

Infatti a quel punto, dopo "api running on port 3001" ottengo la stampa "invoking extract". Se vado nel codice del file snapIoTDirectory_rw.js a vedere dove viene fatta questa stampa, noto che subito dopo viene creato un processo figlio: si tratta proprio di ./snap4cityBroker/externalBroker.js. Allora è lui ad avere problemi! Devo perciò debuggare lui!

A questo punto però mi viene un dubbio: perché tra i parametri inviati c'è anche extcb e 8080? Sulla extcb:8080 non c'è nessun server, a differenza del Tomcat sulla iotobsf! Allora modifico il CB external dalla IOT Directory, togliendo i parametri Access Link e Access Port.

Il risultato adesso, da server NodeJS è la stampa "stderr: not found – returning stderr: not found". Nel browser stavolta l'attesa non è infinita, e compare un messaggio che comunica che un device così descritto non esiste sul CB. Allora rimetto i parametri del CB nell'IOT Directory come prima.

Analizzando di nuovo il processo ./snap4cityBroker/externalBroker.js mi rendo conto che il tutto rimane sospeso probabilmente alla chiamata xhttp.send(); in quanto il link alla quale farebbe la richiesta GET mi pare di capire che sia extcb:8080, al quale non c'è niente!

Infatti abbiamo sempre e comunque una stampa "Connection established", quindi la connessione al database avviene correttamente. Poi si arriva alla chiamata xhttp.send() e tutto riamane sospeso...

Mi accorgo che ho davvero bisogno di uno strumento di debug migliore dei semplici console.log. Ecco allora che scopro, leggendo l'articolo all'indirizzo https://medium.com/@paul_irish/debugging-node-js-nightlies-with-chrome-devtools-7c4a1b95ae27, che lanciando node –inspect-brk [nome del processo NodeJS], il Google Chrome Dev Tools presenta una funzione per debuggare i server NodeJS con possibilità di usare breakpoint e fare lo step delle istruzioni. L'unica cosa è che non riesce a connettersi alla VM Main. Allora prendo il file externalBroker.js e una volta installato nodejs sulla mia macchina host, lancio il comando node -–inspect-brk externalBroker.js.

Essendo questa volta eseguito in locale, il Google Chrome Dev Tool riesce a connettersi al debugger e riesco a vedere la causa del problema: alla linea xhttp.send() c'è una stampa di errore del tipo "xmlhttprequest protocol not supported".

Allora capisco che forse il problema potrebbe essere che il sistema non antepone automaticamente il prefisso "http://" prima di "extcb:1026/eccetera...". Allora vado nella IOT Directory e modifico l'Access Link da "extcb:1026" a "http://extcb:1026". Mi accorgo inoltre che il sistema non postpone il suffis-

so “/v2/entities” al link, come da API NGSI, quindi provvedo anche a inserire “/v2/entities” nella casella Path (da non confondere con Service Path).

È cambiato qualcosa! Adesso il browser da un messaggio dove cita che il database è sprovvisto di regole di estrazione dei dati! La cosa importante è che se al posto del corretto nome del device metto un nome errato, il browser dice che il dispositivo non esiste sul Broker esterno!

Perfetto! Questo significa che adesso il sistema adesso vede effettivamente il device su extcb, deve solo imparare come estrarne i dati! Il check va a buon fine!

Allora ho creato una extraction rule nel DB seguendo le istruzioni del documento “Tutorial on the bulk load”, istruendo il sistema di prelevare in questo caso i dati della temperatura.

Adesso riesco finalmente a registrare il device nella IOT Directory! Nota: La VM Knowledge Base deve essere up and running.

Perfetto! Dopodiché provvedo a registrare device diversi sull’extcb STAVOLTA IN MULTI-TENANCY (MT) e con i loro SERVICE PATH (SP), sempre usando Postman.

Per fare ciò il context broker sull’extcb va lanciato col comando “contextBroker -fg -multiservice”.

3 Compatibilità Multi-tenancy e Service Path

Nella VM Main 1.4 non c’è compatibilità con i concetti di Multi-tenancy e ServicePath, perciò tutta la parte web IOT Directory sulla macchina dashboard va aggiornata. Oltre alla parte web ci sono da fare anche alcune modifiche al DB per renderlo compatibile con la versione della parte web aggiornata.

Per fare ciò scarico l’ultima versione (risalente al 16 Luglio 2020) della parte web e il ddl del db aggiornato da GitHub all’indirizzo https://github.com/disit/iot-directory.

La suddetta versione è però compatibile con i concetti di Multitenancy e Service Path solo per quanto riguarda devices su Context Broker interno IOTOBSF.

Ciò che dovrò fare è rendere la IOT Directory compatibile anche con devices su Context Broker esterni.

Faccio backup della parte web originale, e metto nella VM Main la versione aggiornata, stando attento alla cartella conf. In particolare, per far continuare a funzionare il tutto (ossia per non far comparire errori):

nel file general.ini mantengo
pathCertificate[prod] = "../../certificate"
al posto della nuova
pathCertificate[prod] = "../../../certificate"
e mantengo
pathLog[prod] = "../../log/log.txt"
al posto della nuova
pathLog[prod] = "../../../log/log.txt"

nel file ldap.ini mantengo
ldapBaseName[prod] = "dc=ldap,dc=organization,dc=com"
al posto della nuova
ldapBaseName[prod] = "dc=ldap,dc=example,dc=org"
mantengo
ldapAdminName[prod] = "cn=admin,dc=ldap,dc=organization,dc=com"
al posto della nuova
ldapAdminName[prod] = "cn=admin,dc=ldap,dc=example,dc=org"
e mantengo
ldapAdminPwd[prod] = "FFjIt2u"
al posto della nuova
ldapAdminPwd[prod] = "admin"

nel file sso.ini mantengo
redirectUri[prod] = "http://dashboard/iot2"
al posto della nuova
redirectUri[prod] = "http://dashboard/iotdirectory"

Una volta aggiornato tutto però, mi si sono presentati 3 problemi:

1. nel nuovo file createtables.sql, contenente il nuovo ddl del db, non c'è la tabella value_types, la quale serve, nella IOT Directory, alla parte di "IOT Orion Broker Mapping Rules", infatti, appena ci si reca alla pagina compare un alert che cita: iotdb.value_types' doesn't exist. Rimettendo di nuovo la tabella nel DB l'errore sparisce.

2. Nella pagina "My IOT Devices" compare l'errore "Problem contacting the Snap4City server (Dictionary). Please try later".

3. Nel momento in cui aggiungo un context broker di tipo Multi tenancy, inserendo i vari Tenants (che nel mio caso sono "palazzo_01" e "palazzo_02") compare una scritta rossa che riporta le regole di formattazione del tenant name. Da documentazione Fiware Orion, alla pagina https://fiware-orion.readthedocs.io/en/master/user/multitenancy/index.html possiamo leggere che i numeri sono ammessi, tra le regole mostrate su Snap4City giustamente non c'è scritto che non si possono inserire numeri, ma il sistema comunque non accetta neanche una cifra. Forse la regex di controllo è errata.

Da comunicazione mail con Angelo risulta che i primi due errori derivano dal fatto che è in via di implementazione una funzione per cui i value type verranno scaricati da un servizio web, o quando questo non è raggiungibile verrà letto un file json salvato localmente sulla vm, quindi la tabella non servirà più (una volta terminata l'implementazione).

L'errore del punto 3 è stato da me risolto correggendo la regex di controllo.

Indagando ho scoperto, inerentemente al problema 2, che dall'aggiornamento, il file /conf/serviceURI.ini contiene anche le linee:

processLoaderURI[desc] = "Process Loader Api URI for get info on value type

and value unit (remove completly if not available)"
processLoaderURI[dev] = "http://localhost/test_processloader/api/"
processLoaderURI[test] = "http://localhost/test_processloader/api/"
processLoaderURI[prod] = "http://localhost/processloader/api/"

A quanto pare il servizio web a cui queste linee fanno riferimento non è attivo, quindi mi accingo ad escluderle, commentandole, dal file ini, in modo che il sistema, come fallback, legga il contenuto dai file json.

Quindi adesso mi rimane da:

- risolvere il problema alla pagina IOT Orion Broker Mapping Rules andando a fargli prendere i parametri da web/json invece che da DB. Questo comportamento è già attivo per la pagina "My IOT Devices".
- implementare la compatibilità MT e SP per CB esterni, così come sono già implementati quelli interni.

Ho subito risolto il problema alla pagina IOT Orion Broker Mapping Rules (per fargli andare a fargli prendere i parametri da web/json invece che da DB). Inoltre adesso vengono visualizzati nella UI soltanto i value units attinenti al value type di volta in volta selezionato.

Per fare ciò, essenzialmente ho effettuato modifiche per rendere il sistema non più dipendente dalle tabelle "value_types" e "value_units".

In particolare, ho commentato le funzioni generatelabels e generateunits da /api/common.php e sostituito tutte le chiamate a tali funzioni, contenute in molti file della directory /api , con chiamate a retrieveFromDictionary (sempre in /api/common.php). Le chiamate a generatedatatypes sono invece intatte, dato che i data types sono attualmente ancora su db.

Benissimo, adesso sembra tutto pronto per la parte vera e propria di implementazione della compatibilità MT/SP per devices su CB esterni, ma così non è, infatti mi accorgo che la VM Knowledge Base non è (ancora) compatibile MT/SP, infatti non riesco a registrare device MT/SP su CB interno IOTOBSF (il quale invece è già -parzialmente- compatibile MT/SP). Ciò si evince dal fatto che la KB non mi rilascia link di registrazione su Knowledge Base (in quanto forse non sa interpretare la richiesta).

Ne parlo con Angelo, e mi conferma che la VM KBSSM 1.1 non è compatibile con i devices MT/SP. Per sistemare ciò ho ricevuto da Angelo una versione aggiornata del file /var/lib/tomcat8/webapps/iot.war (su VM KBSSM). Adesso, alle richieste di registrazione di devices di tipo ngsi w/MultiService, la VM KBSSM risponde con un link corretto e non più con una stringa vuota.

Quindi adesso riesco ad aggiungere alla IOT Directory anche devices MT/SP su CB interno. Il device viene correttamente gestito (inserito o rimosso) sia su DB che su CB.

Perfetto. Possiamo finalmente implementare la compatibilità MT/SP per devices su CB esterni.

4 IOT Directory e devices MT/SP su CB esterni

Rispetto alla versione della iot-directory presente su GitHub al commit del 16 luglio 2020, sono riuscito a implementare la possibilità di aggiungere alla IOT Directory devices MT/SP che sono su CB esterni. Per ottenere questa compatibilità le modifiche che ho fatto sono:

In /management/js/devices.js , alla funzione che risponde alla pressione del tasto addNewDeviceCheckExternalBtn ho aggiunto il fatto di prelevare anche i dati di service e servicePath dalla UI, e ho modificato la signature della funzione activateStub in modo da ricevere anche questi parametri.

Sempre nella funzione activateStub ho arricchito la stringa "data" appendendo i service e servicePath appena ottenuti.

Questa stringa non è altro che il body della richiesta fatta a stubs/api/extract che viene gestita dal server nella funzione router.route('/extract') del file /snap4cityServer/snapIoTDirectory_rw.js.

Dentro questa funzione adesso abbiamo questi due parametri in più, i quali vengono passati al processo figlio /snap4cityServer/snap4cityBroker/externalBroker.js che li riceve proprio come argomenti e li usa nella funzione retrieveData.

Piccola digressione: anche externalBroker.js si affidava al DB per il fetch dei value_types e value_units, perciò ho dovuto sostituire tali chiamate al DB con una XMLHttpRequest con action = get_param_values, la quale è gestita da /api/device.php , il quale poi chiama retrieveFromDictionary da /api/common.php come fa negli altri casi.

Nella funzione retrieveData di externalBroker.js ho semplicemente aggiunto gli header "Fiware-Service" e "Fiware-ServicePath", come da specifica Fiware, associando i parametri appena ricevuti, alla richiesta GET fatta al Context Broker esterno.

Così facendo, il device e tutti i suoi attributi vengono correttamente raggiunti ed elaborati, e il dispositivo viene correttamente inserito nella IOT Directory (quindi su DB).

Perfetto! Adesso quindi posso aggiungere alla IOT Directory devices sia su CB interno che esterno, sia di tipo classico che di tipo MT/SP.

5 IOT Broker periodic update settings

Sempre all'interno della IOT Directory si ha la pagina IOT Broker periodic update settings la quale consente di controllare periodicamente se su Context Broker esterni ci sono nuovi device da registrare, e qualora ce ne fossero, consente di aggiungerli alla IOT Directory. Tuttavia tutto ciò non è completamente funzionante in tutti i suoi aspetti.

La maggior parte dei problemi si trovano nella schermata di "edit" del device, la quale consente di modificare i parametri del device trovato, o di integrare informazioni mancanti, prima dell'aggiunta all'IOT Directory.

In ordine crescente di importanza, i problemi finora riscontrati sono:

- Nella scheda IOT Broker, la casella Format non preseleziona l'opzione attinente tra quelle proposte.
- Nella scheda Info, in Model non si ha una menu a tendina, ma una casella di testo inagibile. Ci dovrebbe essere un menu a tendina con i device model inerenti a CB esterni, e presentare la preselezione del Model attinente.
- C'è una scheda Position che dovrebbe presentare una mappa, ma questa non appare, in quanto non riceve coordinate Lat/Long corrette a causa di un problema nel file ngsi2IotDirectory_rw.js. Tuttavia c'è un'inconsistenza nel programma, infatti secondo il DB lat e long non sono parametri obbligatori (mandatory params), ma secondo la UI sì. Infine, come faccio ad estrarre lat e long da un device se non ho le relative extraction rules? Se metto le extraction rules per lat e long, questi parametri andranno nella scheda Values, come se fossero valori da estrarre dal device, mentre invece servono alla mappa. Questa parte qui è un po' da definire meglio nella UI così da eliminare inconsistenze.

Ad ogni modo, mi sono limitato ad applicare alcune correzioni alla UI. Per ora le modifiche fatte al codice rispetto alla versione presente su GitHub al 16 Luglio 2020 sono:

- In management/assotiationRules.php ho modificato le query così da mostrare solo CB esterni e device models inerenti solo a CB esterni. Inoltre ho modificato alcuni segmenti di testo così da spiegare meglio il funzionamento della UI.
- In management/js/assotiationRules.js ho migliorato il modo in cui si visualizzano alcuni dettagli relativi a value_type e value_unit.

Tali modifiche sono disponibili sulla mia repository al link https://github.com/spita90/iot-directory

6 MT e SP nelle IOT App di Node-Red

Come affermato precedentemente, abbiamo raggiunto la piena compatibilità Multi-tenancy e Service Path nella gestione dei devices, sia su Context Broker interni che, a questo punto, anche esterni.

Ciò che serve adesso, è rendere fruibili queste nuove funzionalità anche all'interno delle IOT App, ovvero nell'editor Node-Red disponibile alla pagina IOT Applications.

Per fare ciò si dovrà mettere mano al codice JavaScript nei file dei blocchetti Orion della libreria Snap4City.

Per prima cosa ho installato Node-Red tramite npm su macchina host, così da essere più agile nelle modifiche e via via nei test. Node-Red, una volta avviato col comando "node-red" rende disponibile l'editor grafico alla pagina http://localhost:1880/.

A quel punto ho installato, attraverso l'editor in "Manage palette", le librerie "node-red-contrib-snap4city-developer" e "node-red-contrib-snap4city-user".

Dopodiché ho dovuto modificare il file ~/.node-red/settings.js per far interfacciare la mia installazione locale di Node-Red con la Dashboard:

- Ho aggiunto in cima le seguenti righe:

```
var keycloak_base_uri='http://dashboard:8088/auth/
    admin/master'
var keycloak_clientid='nodered-iotedge'
var keycloak_clientsecret='ec4f3896-8fee-4f37
    -8703-eee13e7bd609'

var iotdirectory_uri='https://www.snap4city.org/
    iotdirectorytest/'
var ownership_url='http://dashboard/ownership-api
    /'
```

- Ho associato le variabili, così da averle subito nella funzione "module.exports":

```
keycloakBaseUri: keycloak_base_uri,
keycloakClientid: keycloak_clientid,
keycloakClientsecret: keycloak_clientsecret,

iotDirectoryUrl: iotdirectory_uri,
ownershipUrl:ownership_url,
```

Dopodiché ho dovuto creare su KeyCloak un nuovo Client "nodered-iotedge":

- Vado (ovviamente con VM Main accesa) all'indirizzo http://dashboard:8088/auth/. Credenziali: admin admin.
- In Clients ho creato un nuovo client di nome "nodered-iotedge"

Adesso, riavvio node-red, accedo a http://localhost:1880/ e posso usare i blocchetti Node-Red di Snap4City della mia installazione locale di Node-Red con i devices nella IOT Directory della macchina Dashboard! Nota: per tutte le future operazioni la VM KBSSM deve essere attiva.

Voglio per prima cosa verificare il corretto funzionamento dello scenario non MT/SP.

Per fare dei test metto un blocchetto Inject, un Random, un Function, un Fiware Orion Out v1, e un Debug. Vediamo come sono disposti:

Il blocchetto Orion, per collegarlo al mio device non MT/SP "Temp_sensor" su CB interno, è cosi configurato:

- Ad Authentication inserisco le credenziali per accedere alla Dashboard: User userrootadmin e Password Sl9.wrE@k
- A Service metto "iotobsf" con porta 8443. Ovviamente si capisce che servirà la macchina IOTOBSF attiva.
- Il Device Type è lo stesso del mio device: "Test"
- Il Device NameID è il nome del mio device: "Temp_sensor".

- Le K1 e K2 sono le due Key del mio device, così come visualizzabili in "IOT Devices" nella IOT Directory.

In ingresso a tale blocchetto metto un blocchetto Function così definito:

```
var temp = msg.payload;
msg.headers = {};
msg.headers['Content-type'] = 'text/plain';
msg.payload = [{ "name": "temperature", "value": temp,
    "type": "float" }];
return msg;
```

A sua volta, questo blocchetto ha in ingresso un blocchetto Random, che ci da un numero float casuale (che sarà una fittizia temperatura rilevata dal device).

Lui a sua volta ha in ingresso un Inject.

In aggiunta a tutto questo, metto, a parte, un blocchetto Fiware Orion Subscribe v1 con annesso Debug per avere da parte del CB tutte le notifiche relative a variazioni di temperatura del device. Al blocchetto idealmente si può attaccare un Widget di una Dashboard per la visualizzazione grafica delle temperature rilevate via via.

Ora però ricordo che nell'istanza Node-Red su Dashboard, cioè alla pagina IOT Applications c'era un problema sul listening dei cambiamenti, come annotato nel Capitolo 1.

Il problema sussisteva nel fatto che nonostante la subscription venisse correttamente registrata dal CB iotobsf, l'indirizzo a cui il CB deve inviare le notifiche, ossia nel mio caso "http://dashboard:1880/iotapp/938266349ea1e8" risultava a lui irraggiungibile.

Molto strano, dato che la VM IOTOBSF si sa collegare correttamente a "dashboard", in quanto il suo ip è in /etc/hosts.

Chiamiamo questo problema "Problema 1", problema che come sottolineato, si presentava su Node-Red da Dashboard.

Un problema incontrato invece adesso, da Node-Red avviato da macchina host, è che il CB iotobsf sembra non registrare la subscription. Lo chiamo, da qui in avanti, "Problema 2".

Decido di voler risolvere subito il Problema 1, in quanto il corretto funzionamento del listening farebbe molto comodo per il debug.

Quindi, per quanto riguarda il Problema 1, ho notato che il CB Fiware Orion all'interno della VM IOTOBSF è in esecuzione tramite un Docker Container.

Infatti digitando "docker ps" vedo come processi Dockerizzati "fiware_orion_1" e "fiware_mongo_1". Mi annoto l'id del container fiware_orion_1 (nel mio caso 9b2fdd9a4301, ma potrebbe cambiare dopo un riavvio).

Adesso digito "docker exec -it 9b2fdd9a4301 /bin/bash" e ottengo il terminale come se fossi dentro il Container di Orion. A questo punto se mando ping verso dashboard noto che effettivamente non vanno a buon fine.

A quanto pare, i processi eseguiti tramite Docker non vedono gli hostname indicati in /etc/hosts, ma fortunatamente si possono indicare in altro modo al Docker. Vediamo come:

- Ho modificato il file ~/fiware/docker-compose.yml come di seguito:

```
version: "3.7"
services:
   orion:
      image: fiware/orion
      depends_on:
        - mongo
      extra_hosts:
        - "dashboard:192.168.1.203"
      ports:
        - "1026:1026"
      command: -dbhost mongo
      restart: always
   mongo:
      image: mongo:3.6
      restart: always
      volumes:
        - /home/debian/fiware/db:/data/db
```

  ossia ho aggiunto le due righe riguardante l'hostname dashboard e il suo indirizzo di rete.

- spostandomi in ~/fiware ho eseguito:

```
sudo docker-compose down
sudo docker-compose up -d
```

Così facendo il Docker di Fiware Orion è ripartito, con in più l'indicazione dell'hostname dashboard e adesso Fiware Orion riesce a raggiungere dashboard per mandargli le notifiche.

Così il Problema 1 è stato risolto. Perfetto!

Passo al Problema 2.

Noto che in realtà la subscription viene sempre registrata correttamente nel CB, ma se node-red è attivo e con blocchetto Listener attivo e in listening sullo stesso url di notifica della subscription, per qualche motivo, il blocchetto stesso fa immediatamente unsubscribe sull'id di quella subscription.

Il problema è dato da un bug nel codice contenuto nel file orion.js.

Ho notato che c'era una cattiva gestione degli id delle subscription associati agli id dei blocchetti Node-Red, per cui all'aggiornamento dei parametri di un blocchetto di listening veniva giustamente fatta una richiesta per una nuova subscription, ma subito dopo non veniva cancellata solo quella vecchia, ma anche quella nuova. Inoltre non era ben gestito il caso di nuova subscription, infatti il programma cercava inutilmente di cancellare subscription con id "undefined". È stato risolto. Infine, l'url al quale fare la notifica veniva formato male se invece di un hostname c'era un indirizzo ip. Anche questo è stato risolto.

Per tenere traccia di tutte queste modifiche ho creato una repository, raggiungibile all'indirizzo https://github.com/spita90/s4c-node-modules. Le modifiche di cui sopra sono visibili alla pagina del commit "Subscription notification fix" del 3 Agosto 2020.

I due problemi appena risolti erano però inerenti al funzionamento generale del sistema, e non hanno nulla a che vedere col mondo MT/SP. Mi accingo quindi alla vera e propria implementazione dei paradigmi MT/SP nei blocchetti.

L'idea è quella di implementare la compatibilità MT/SP su Node-Red per devices su CB interno, e controllare che tutto funzioni bene.

Allora tramite IOT Directory creo un device "Stanza1" su tenant "palazzo_01" nel path "/piano_01" del CB interno iotobsf. Il nome che ho dato al Broker è "MT_SP_iotobsf" per non confonderlo con "iotobsf".

Modifico l'orion.html così da avere nei blocchetti le caselle per inserire i parametri Service e ServicePath. Modifico poi l'orion.js per inviare Service e ServicePath come header delle richieste, come da specifica NGSI, oltre ad anteporli al nome del device (per avere quindi tenant./path.deviceName).

Fatto ciò ottengo errori.

Nel caso non MT/SP alla richiesta:

```
queryContext Temp_sensor
options:{"hostname":"iotobsf","port":"8443","path":"/
    v1/queryContext/?limit=20&elementid=Temp_sensor
```

ottengo dall'Orion Filter (il firewall su iotobsf) risposta OK, con codice 200. Tutto perfetto.

Nel caso MT/SP invece alla richiesta:

```
queryContext palazzo_01./piano_01.Stanza1
options:{"hostname":"iotobsf","port":"8443","path":"/
    v1/queryContext/?limit=20&elementid=palazzo_01./
    piano_01.Stanza1
```

ottengo dall'Orion Filter ottengo un rifiuto, con codice 401.

Contatto Angelo e mi dice che il problema è che avendo chiamato il CB "MT_SP_iotobsf" il firewall si rifiuta di procedere, in quanto è configurato per ricevere "iotobsf", mentre questo Broker, giustamente, è chiamato in modo diverso (nonostante si tratti della stessa VM).

Allora mi passa un nuovo file orionbrokerfilter.war da mettere in /opt/tomcat/webapps, e mi dice che con questo file non serve più anteporre tenant e servicepath al nome del device. Service e ServicePath vanno solo mandati come header, come da specifica NGSI.

Prima di essere messo in posizione, il file .war va aperto con un programma per archivi compressi, e, recandosi quindi al suo interno nella directory ./WEB-INF/classes/ troviamo il file application-deploy.properties, da modificare come segue:

```
spring.messages.basename=messages/messages
spring.messages.cache-seconds=-1
```

```
3  spring.messages.encoding=UTF-8
4
5  logging.config=classpath:/log4j2-spring-deploy.xml
6
7  #spring.jpa.show-sql=true
8  #spring.jpa.properties.hibernate.format_sql=true
9  #logging.level.org.springframework.web=DEBUG
10 #logging.level.org.hibernate=DEBUG
11
12 spring.openidconnect.clientid=orionbrokerfilter
13 spring.openidconnect.username=rootfilter
14 spring.openidconnect.password=ADf4A6tpEspBLPAV
15
16 #keyclock di produzione
17 spring.openidconnect.token_endpoint=http://dashboard
       :8088/auth/realms/master/protocol/openid-connect/
       token
18
19 spring.ownership_endpoint=http://dashboard/ownership-
       api/v1/list
20 spring.delegation_endpoint=http://dashboard:8080/
       datamanager/api/v1/apps
21 spring.servicemapkb_endpoint=http://kbssm:8080/
       ServiceMap/sparql
22 spring.orionbroker_endpoint=http://localhost:1026
23
24 spring.elapsingcache.minutes=3
25
26 cors.origins.accepted=http://dashboard
27
28 spring.prefixelementID=Organization:MT_SP_iotobsf
29
30 connection.timeout=10000
```

Si noti che alla riga spring.prefixelementID la Organization specificata è MT_SP_iotobsf, così per avere la compatibilità MT/SP.

Dopo la modifica del file .properties e la ricompressione del file .war, provvedo con la sostituzione del file .war e con l'aggiornamento del codice in orion.js (per non anteporre più i nuovi header al nome del device). Perfetto! Adesso i blocchetti di query, update, e listening funzionano bene!

7 Device Discovery per CB esterni ngsi w/Multiservice

Col prof. Nesi parliamo del fatto che un altro aspetto di MT/SP che sarebbe bello avere su IOT Directory è il poter fare una discovery dei devices su CB esterni rappresentando i devices in un albero dove si vedono chiaramente i vari CB esterni con i vari tenant, i service path, i sotto path, e così via in una struttura gerarchica. Inoltre, il sistema rileverebbe tra i devices trovati, quali

tra loro non sono ancora stati aggiunti alla IOT Directory e dovrebbe dare la possibilità per questi devices di revisionare i loro parametri e aggiungerli alla IOT Directory.

Purtroppo Fiware Orion non prevede nelle API la possibilità di poter ottenere, tramite una richiesta, la gerarchia dei tenant e dei path del determinato Context Broker. Allora posso pensare di fare una richiesta getEntities globale per avere tutti i device di un determinato tenant, ma sempre da specifica Orion ciò che ottengo non include le informazioni del path dei vari devices.

Per questo motivo il Device Discovery che ho sviluppato è in un certo senso limitato.

Infatti, l'algoritmo effettua la scansione dei devices, oltre che alla root (il tenant predefinito quando questo non viene specificato), soltanto nei Tenant conosciuti (per come registrati nelle info del Context broker, nella IOT Directory) e nei Service Path conosciuti (ossia i path in cui si ha almeno un device aggiunto alla IOT Directory o quelli indicati nei Device Model). Questo perché non abbiamo modo di conoscere nuovi tenant e service path oltre a quelli che già conosciamo.

Per quanto riguarda la rappresentazione grafica ad albero userò la libreria D3 (https://d3js.org/), implementando, in particolare, una versione leggermente modificata del grafico gerarchico "Tidy Tree": https://observablehq.com/@d3/tidy-tree. Inizio a sviluppare il tutto. Vediamo come ho fatto:

Ho creato i file: /mamagement/deviceDiscovery.php, /management/js/deviceDiscovery.js, /api/deviceDiscoveryApi.php, /css/d3tree.css, /snap4cityServer/snap4cityBroker/discovery.js e modificato i file: /management/mainMenu.sql, /snap4cityServer/snapIoTDirectory_rw.js.

Per avere un collegamento alla pagina sotto il menu IOT Directory ho modificato le tabelle MainMenuSubmenus e MobMainMenuSubmenus del database Dashboard, inserendo le entry contenenti il link a deviceDiscovery.php.

La pagina contiene un tasto che se premuto avvia la discovery, che si sostanzia nelle seguenti azioni in successione:

- viene chiamato getCBServiceTrees sulla deviceDiscoveryApi.php
- tale funzione fa una query su DB (che si trova su VM Dashboard) di tutti i Context Broker che sono esterni e che hanno protocollo "ngsi w/Multiservice", ponendovi a fianco i relativi tenant conosciuti, ossia i tenant contenuti nelle info del Context broker (da IOT Directory), e ancora a fianco i relativi Service Path conosciuti (ossia quelli in cui si ha almeno un device aggiunto alla IOT Directory o quelli indicati nei Device Model).
- da questa tabella l'algoritmo crea un oggetto gerarchia che parte dalla lista di CB, e si sviluppa così che ogni cb contenga i suoi tenant, e ogni tenant contenga una lista di tutti i service path conosciuti.

  Attenzione. i tenant adesso non contengono i path come gerarchia ma bensì come lista sviluppata. Infatti, ad esempio, sotto "tenant1" ho: "path1", "path1/path1_1", "path1/path1_1/path1_1_1", ecc. Questo passo intermedio è necessario per poter svolgere il prossimo passo

- Faccio l'inferenza dei path impliciti mancanti. Mi spiego meglio: assumiamo di avere, in un certo tenant, i path "path1" e "path1/path1_1/path1_1_1".

  Vogliamo fare discovery, ovviamente, anche sul path "path1/path1_1", anche se questo non è elencato esplicitamente tra i path del tenant! Questa situazione accade se non abbiamo devices, deleted_devices, o device model con il path "path1/path1_1". L'algoritmo quindi rileva queste casistiche e aggiunge alle liste dei path quelli mancanti.

- su ogni path viene lanciato un activateStub. In pratica quindi, per ogni path viene chiamata la funzione discover su snapIoTDirectory_rw.js.

- tale funzione crea un processo nodeJS discovery.js figlio, il quale si occupa fattivamente di reperire i devices dal Context Broker esterno nel path specificato, e li restituisce al padre snapIoTDirectory_rw.js.

- snapIoTDirectory_rw.js prende i devices del determinato path e li restituisce in risposta all'activateStub

- l'activateStub salva i devices di volta in volta reperiti in un array globale.

- Quando tutte le chiamate sono quindi ritornate dal backend e tutti i devices trovati in tutti i path di tutti i tenant di tutti i context broker esterni sono memorizzati nell'array, viene chiamata la funzione getPreExistingExternalDevices il cui scopo è ottenere la lista dei devices relativi a CB esterni, già presenti su IOT Directory.

- per fare ciò la funzione getPreExistingExternalDevices chiama la funzione get_all_ext_devices_in_iot_dir su deviceDiscoveryApi.php.

- Quest'ultima fa una query al DB per scaricare la lista dei devices preesistenti, e li restituisce.

- Adesso avviene la costruzione della vera e propria gerarchia JSON dei CB con i tenant e i service Path, proprio ad albero, quindi in modo che i tenant non contengano (come prima) la lista dei path sviluppati, ma la gerarchia path/sotto-path vera e propria.

- a ogni path vengono infine aggiunti, come foglie, i devices trovati sui CB esterni. Al momento dell'aggiunta di un device all'albero, l'algoritmo controlla (confrontando con la lista di devices appena scaricata) se il device è già presente su IOT Directory o meno, segnando di colore verde i devices non ancora importati sulla IOT Directory.

Abbiamo quindi adesso una rappresentazione grafica della gerarchia di devices. Alla pressione del nome di un device "verde", ossia non ancora presente in IOT Directory esce fuori un dialog che permette di visionarne e modificarne i parametri, e infine di importare il suddetto device nella IOT Directory.

Per quanto riguarda i Values, i parametri vengono compilati cercando nelle extraction rules quelle relative al CB corrente. Qualora non vengano trovate extraction rules valide, i campi saranno comunque compilabili manualmente.

Anche in questo caso il codice è disponibile al link https://github.com/spita90/iot-directory.