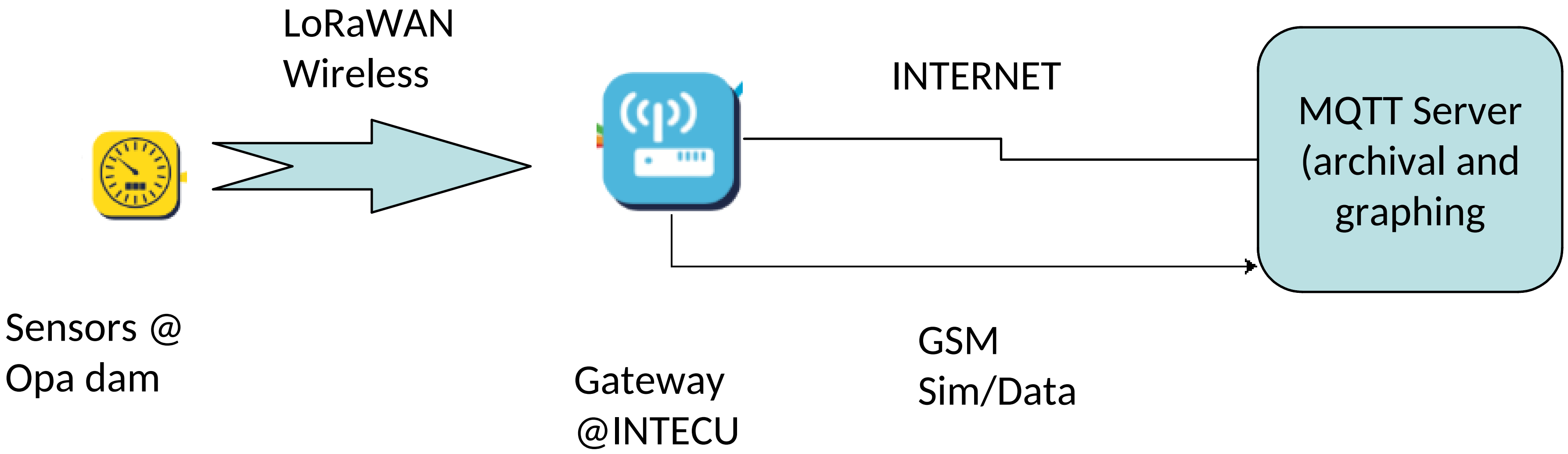
Opensource IoT Platform

Advanced hands-on with Thingsboard

IoT Pipeline: From sensor to dashboard/archive



LoRaWAN Wireless

INTERNET

MQTT Server (archival and graphing

Sensors @ Opa dam

Gateway @INTECU

GSM Sim/Data

Open-source solution(s)

Some ideas

- Mosquito+MySQL+grafana *(implemented)*
- InfluxDB for long-term archival of readings
- ClickhouseDB+EMQX
  - No clear idea for analysis+visualization

Final (operating since Feb 2024).

- Thingsboard IoT cloud platform
  - **Json** Data ingest: MQTT, HTTP, CoAP..
  - Database: PostgreSQL + Cassandra
  - Analysis: inbuilt rules engines
  - Visualization: Dashboard/widgets
  - Multi layer Cloud architecture/profiles
    - Cloud Provider: install, upgrade + setup of platform, Creates cloud tenants
    - Cloud Tenant: Creates Devices, Dashboard, entities, access controls, create customers and end users
  - Comprehensive documentation:
  - https://thingsboard.io/docs/

mqtt.sti.ictp.it

- Powered by Thingsboard CE
  - Opensource
  - Written in Java
  - Limitations of CE
    - Branding (change logo, etc)
    - Widgets (limits)
      - API based work-around recommended

- High reliability/availability cluster
  - Hardware Load-balancer
    - 140.105.33.222
    - Automatic switching (< 1 min)
  - 2 independent “cloud” server instances
    - Running on 2 different clouds/subnets
    - 4 core, 8GB ram
    - OS: Ubuntu 20.04 and Ubuntu 22.04
  - PostgresDB clustering
  - Service/ICTS monitoring
  - Workarounds for TB-CE limitations

Tenant Admin interface

Tenant Admins

- The following STI staff can create devices, entities, customers, dashboard, etc
  - mzennaro@ictp.it
  - epietros@ictp.it
  - mrainone@ictp.it
  - rpaskaus@ictp.it
  - icts@ictp.it

- Note: passwords are **NOT** the ICTP (email) credentials.
- Supports direct password recovery

Tasks

- Create/Manage Entities
  - Devices
  - Assets:
    - Locations (Towns, countries)
  - Rules Chains
    - for Analysis, etc.
  - Dashboards
  - Customers
    - IEEES, Ile-Ife
    - WACREN, Accra

Managing Devices

- Creation requires Name, Label (description), Profile, is-gateway, must be assigned to a customer
  - Once complete, provide a unique “Access token”
    - Used in place of username for mqtt authentication, no password required.
    - Example: MQTT

      -
        ```
        mosquitto_pub -d -q 1 -h mqtt.sti.ictp.it -p 1883 -t /v1/devices/me/telemetry -u “PLACE_ACCESS_TOKEN_HERE” -m “{temperature:25}”
        ```
      - One-way MQTTS (SSL) example:

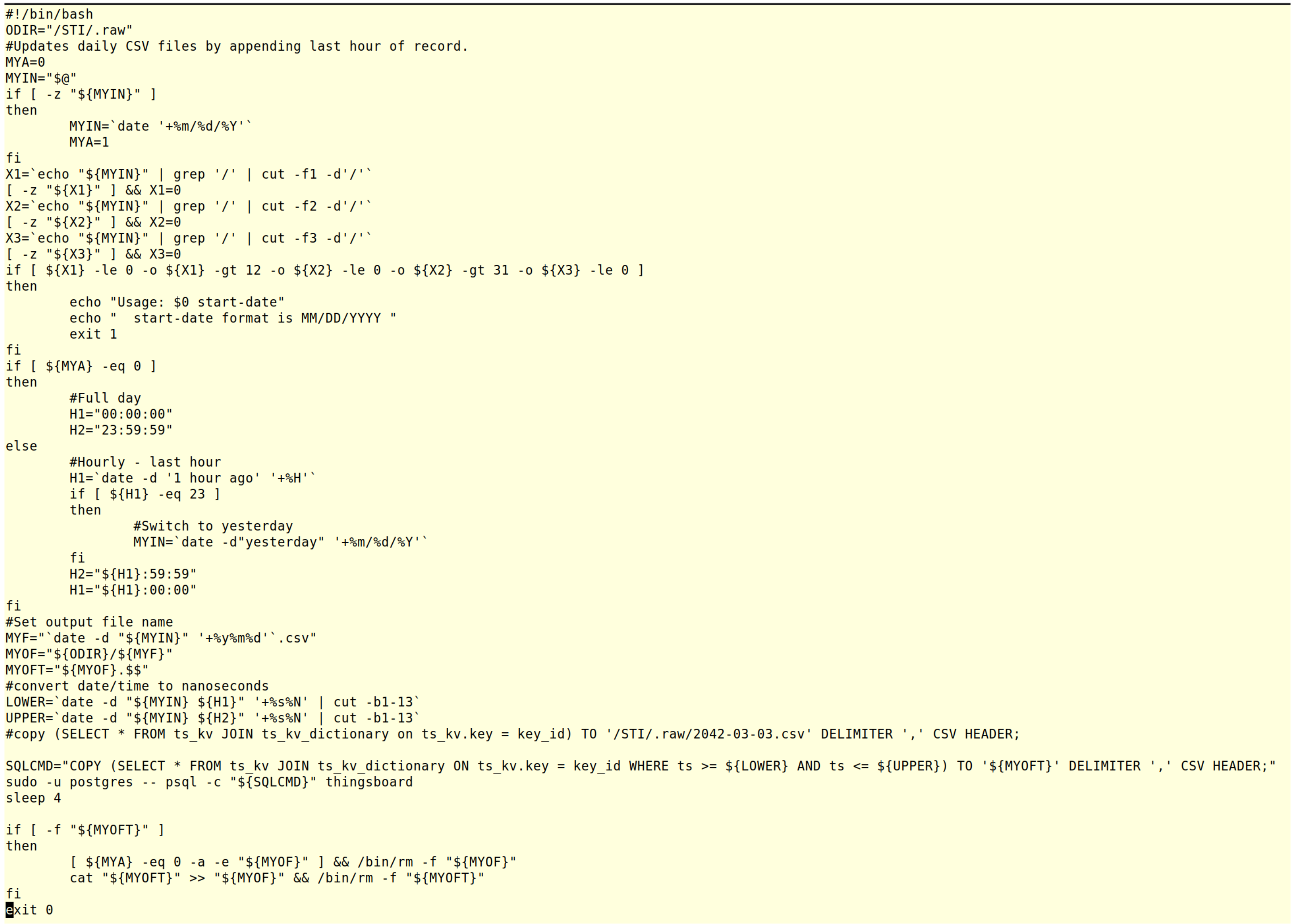
      -
        ```
        mosquitto_pub --capath /etc/ssl/certs -d -q 1 -h mqtt.sti.ictp.it -p 8883 -t /v1/devices/me/telemetry -u “PLACE_ACCESS_TOKEN_HERE” -m “{temperature:25}”
        ```
- *NOTE: A device can be accessed using any supported protocol: MQTT, MQTTS, HTTPS, CoAP, CoAPS, L2M2M..*

Database

- PostgreSQL version 15
  - Handles both configuration and time-Series data.
    - Cassandra is only needed when dealing with big installations (> 1000 devices).
  - Multiple master-master table level clustering technology
    - Bi-directional replication by bucardo
  - Database schema:
    - time-series data is saved as key-value pairs in a partitioned table named ts_kv
      - ts_kv_2024_02, ts_kv_2024_03, ts_kv_2024_04, ts_kv_indefinite, ts_kv_latest
      - time field (ts) is stored in nanoseconds
  - Database backups to 2nd disk daily, then 1 copy per week for 6 weeks
    - autopostgresqlbackup
    - All time series data dumped to disk (NFS server) every hour, daily script can create CSV file for a particular device..

script 1: dump_tb_ts_to_csv

- Uses database access
- Depends on PostgreSQL to export to CSV file.
- Raw data is saved to /STI (an external NFS disk)
  - Every hour.
- 2nd script filters for a specific device
  - sorted into separate directories CSV files can be accessed by sftp or http.



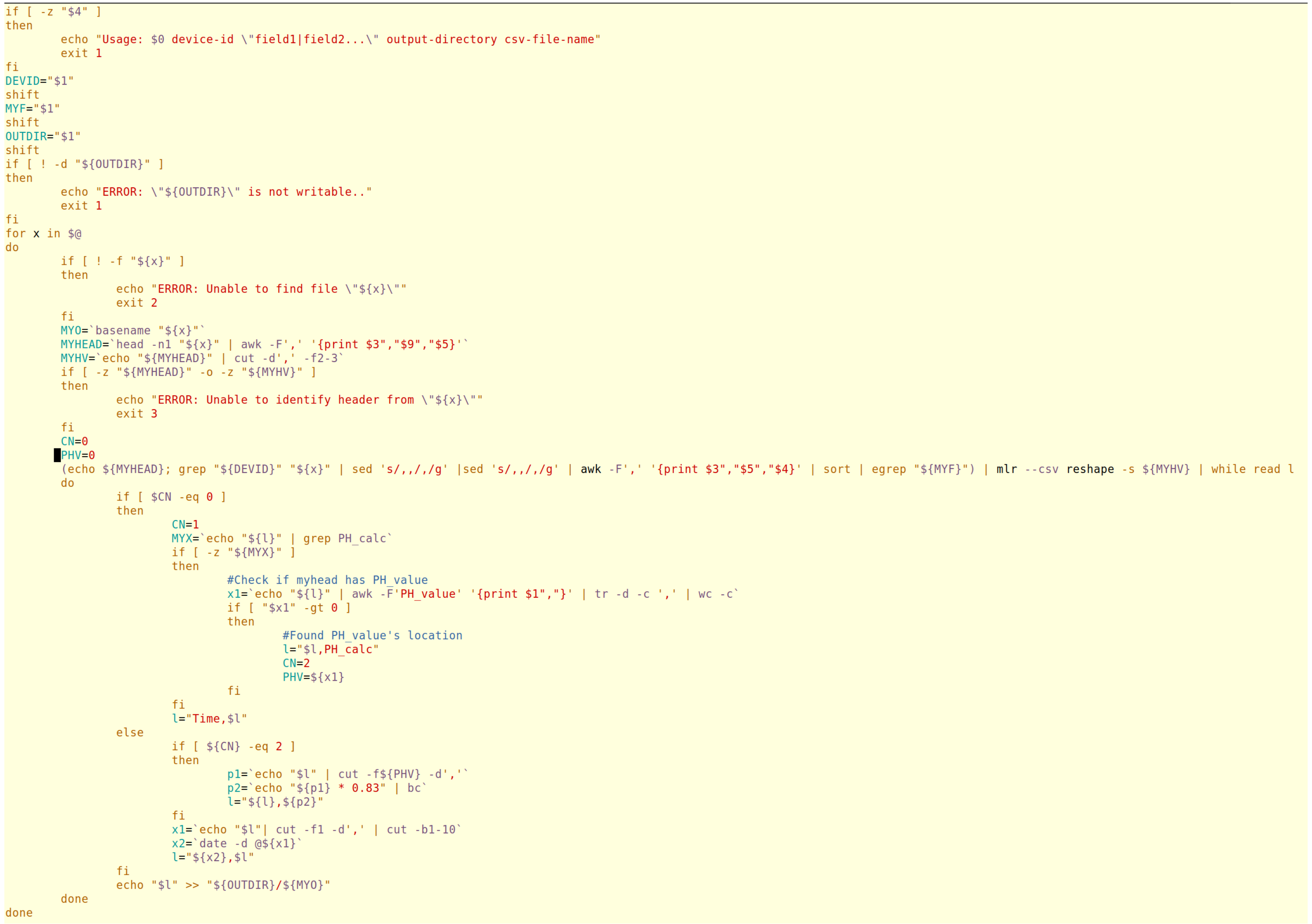
```
#!/bin/bash
ODIR="/STI/.raw"
#Updates daily CSV files by appending last hour of record.
MYA=0
MYIN="$@"
if [ -z "${MYIN}" ]
then
        MYIN=`date '+%m/%d/%Y'`
        MYA=1
fi
X1=`echo "${MYIN}" | grep '/' | cut -f1 -d'/'`
[ -z "${X1}" ] && X1=0
X2=`echo "${MYIN}" | grep '/' | cut -f2 -d'/'`
[ -z "${X2}" ] && X2=0
X3=`echo "${MYIN}" | grep '/' | cut -f3 -d'/'`
[ -z "${X3}" ] && X3=0
if [ ${X1} -le 0 -o ${X1} -gt 12 -o ${X2} -le 0 -o ${X2} -gt 31 -o ${X3} -le 0 ]
then
        echo "Usage: $0 start-date"
        echo "  start-date format is MM/DD/YYYY "
        exit 1
fi
if [ ${MYA} -eq 0 ]
then
        #Full day
        H1="00:00:00"
        H2="23:59:59"
else
        #Hourly - last hour
        H1=`date -d '1 hour ago' '+%H'`
        if [ ${H1} -eq 23 ]
        then
                #Switch to yesterday
                MYIN=`date -d"yesterday" '+%m/%d/%Y'`
        fi
        H2="${H1}:59:59"
        H1="${H1}:00:00"
fi
#Set output file name
MYF="`date -d "${MYIN}" '+%y%m%d'`.csv"
MYOF="${ODIR}/${MYF}"
MYOFT="${MYOF}.$$"
#convert date/time to nanoseconds
LOWER=`date -d "${MYIN} ${H1}" '+%s%N' | cut -b1-13`
UPPER=`date -d "${MYIN} ${H2}" '+%s%N' | cut -b1-13`
#copy (SELECT * FROM ts_kv JOIN ts_kv_dictionary on ts_kv.key = key_id) TO '/STI/.raw/2042-03-03.csv' DELIMITER ',' CSV HEADER;

SQLCMD="COPY (SELECT * FROM ts_kv JOIN ts_kv_dictionary ON ts_kv.key = key_id WHERE ts >= ${LOWER} AND ts <= ${UPPER}) TO '${MYOFT}' DELIMITER ',' CSV HEADER;"
sudo -u postgres -- psql -c "${SQLCMD}" thingsboard
sleep 4

if [ -f "${MYOFT}" ]
then
        [ ${MYA} -eq 0 -a -e "${MYOF}" ] && /bin/rm -f "${MYOF}"
        cat "${MYOFT}" >> "${MYOF}" && /bin/rm -f "${MYOFT}"
fi
exit 0
```

bash script 2: tb_csv_to_user_csv

- Takes device-id, list of fields to extract
- uses raw CSV outputs/dumps from ts_kv (previous script).
- Scheduled for once a day
  - Ile-Ife.



```
if [ -z "$4" ]
then
        echo "Usage: $0 device-id \"field1|field2...\" output-directory csv-file-name"
        exit 1
fi
DEVID="$1"
shift
MYF="$1"
shift
OUTDIR="$1"
shift
if [ ! -d "${OUTDIR}" ]
then
        echo "ERROR: \"${OUTDIR}\" is not writable.."
        exit 1
fi
for x in $@
do
        if [ ! -f "${x}" ]
        then
                echo "ERROR: Unable to find file \"${x}\""
                exit 2
        fi
        MYO=`basename "${x}"`
        MYHEAD=`head -n1 "${x}" | awk -F',' '{print $3","$9","$5}'`
        MYHV=`echo "${MYHEAD}" | cut -d',' -f2-3`
        if [ -z "${MYHEAD}" -o -z "${MYHV}" ]
        then
                echo "ERROR: Unable to identify header from \"${x}\""
                exit 3
        fi
        CN=0
        PHV=0
        (echo ${MYHEAD}; grep "${DEVID}" "${x}" | sed 's/,,/,/g' |sed 's/,,/,/g' | awk -F',' '{print $3","$5","$4}' | sort | egrep "${MYF}") | mlr --csv reshape -s ${MYHV} | while read l
        do
                if [ $CN -eq 0 ]
                then
                        CN=1
                        MYX=`echo "${l}" | grep PH_calc`
                        if [ -z "${MYX}" ]
                        then
                                #Check if myhead has PH_value
                                x1=`echo "${l}" | awk -F'PH_value' '{print $1","}' | tr -d -c ',' | wc -c`
                                if [ "$x1" -gt 0 ]
                                then
                                        #Found PH_value's location
                                        l="$l,PH_calc"
                                        CN=2
                                        PHV=${x1}
                                fi
                        fi
                        l="Time,$l"
                else
                        if [ ${CN} -eq 2 ]
                        then
                                p1=`echo "$l" | cut -f${PHV} -d','`
                                p2=`echo "${p1} * 0.83" | bc`
                                l="${l},${p2}"
                        fi
                        x1=`echo "$l"| cut -f1 -d',' | cut -b1-10`
                        x2=`date -d @${x1}`
                        l="${x2},$l"
                fi
                echo "$l" >> "${OUTDIR}/${MYO}"
        done
done
```

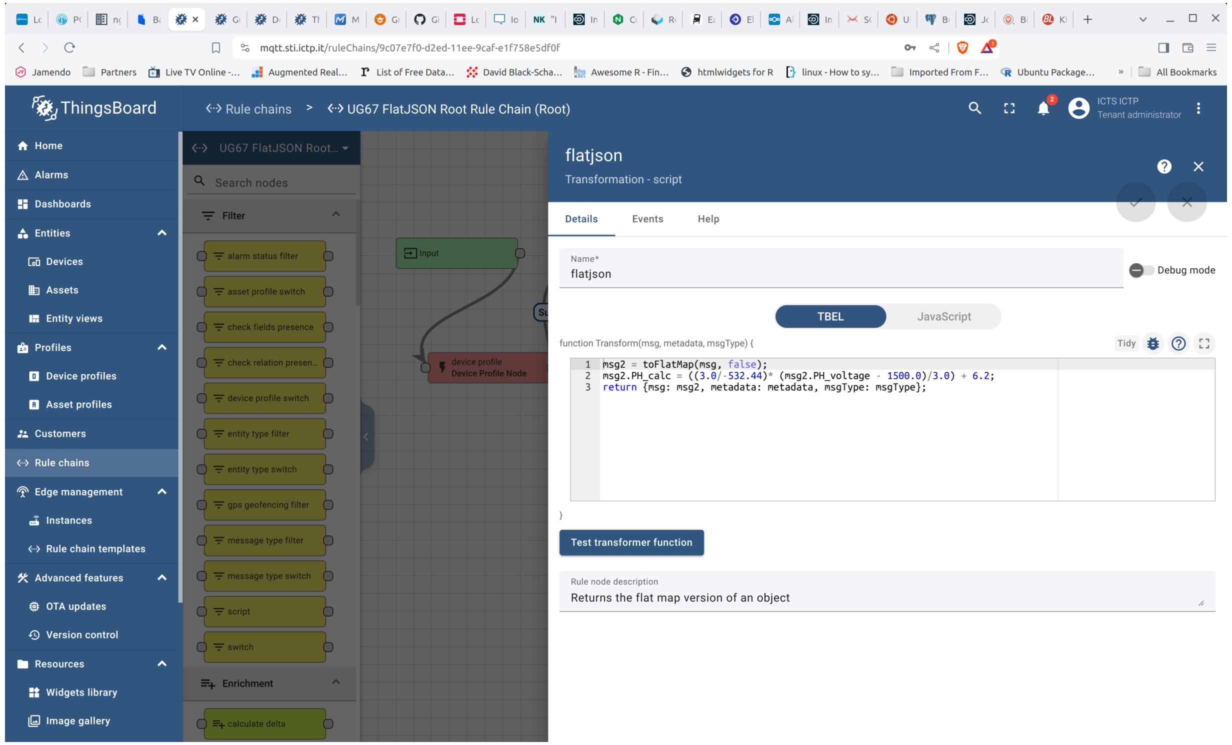
Using TB Rules Chain for Data analysis

- Graphical User Interface
    - Visual programming including tester/debugger.
  - TB Rules Chains
    - Message
      - data from device, device state, etc..
    - Node
      - Different types: filter, transform, even script, etc..
    - Chain
      - Interconnections of nodes to perform action
  - Default rule chain action is save incoming data to database.

- Analysis
  - Data validation, modification, substitution, etc
  - Trigger actions or alarms
  - Load additional data for processing
    - E.g Transfrom data use previous data/record
  - Calculate additional fields to be saved
  - Send email messages
  - Integrate with other external pipelines/tools

Locally created rule nodes

UG67 flatJson Rule Chain



flatjson

Transformation - script

Name*
flatjson

```
function Transform(msg, metadata, msgType) {
msg2 = toFlatMap(msg, false);
msg2.PH_calc = ((3.0/-532.44)* (msg2.PH_voltage - 1500.0)/3.0) + 6.2;
return {msg: msg2, metadata: metadata, msgType: msgType};
}
```

Rule node description
Returns the flat map version of an object

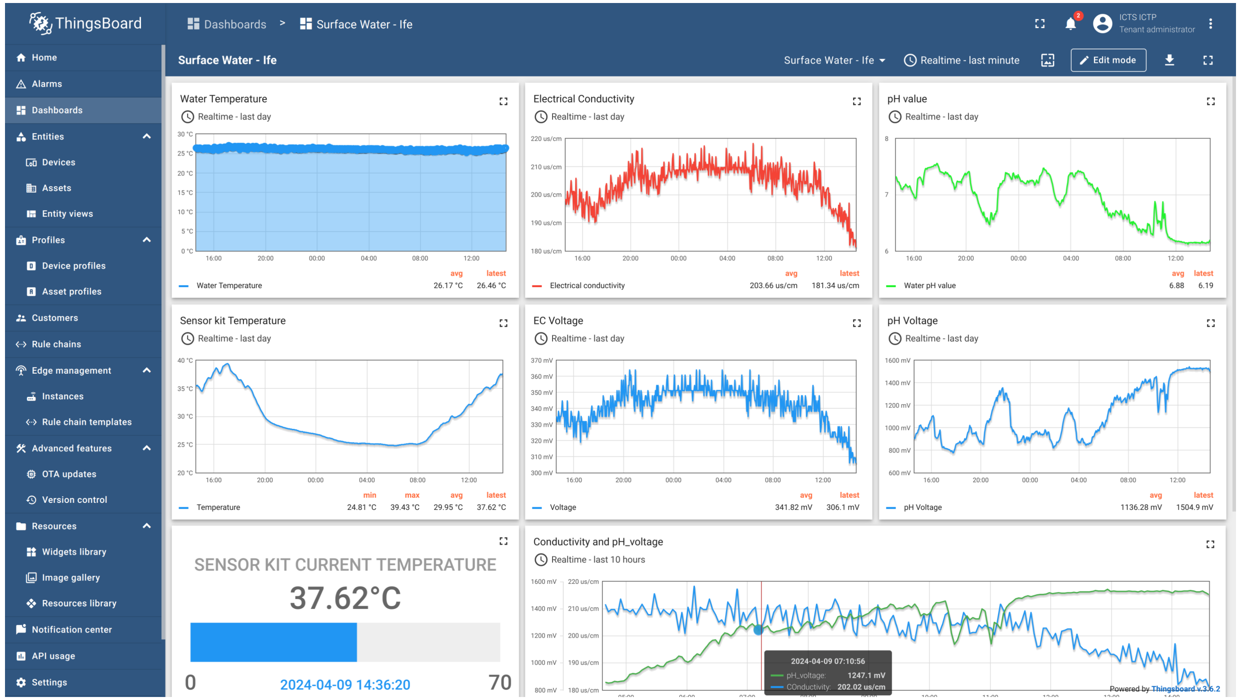
TB Expression Language or JavaScript

- Chain
  - Input --> Device Profile --> Message type switch --> ***flatjson*** --> Save Time series
- Flatjson (transformation script)
  - Expand nested Json is required to handle data from MileSight Gateway
  - PH re-calculation is used to perform software based re-calibration of the PH sensor
- Can also be used to decode LoRA packets?

Visualizations: dashboards

- Dashboards are a collection of widgets.
  - Can be re-arranged by customer or end-users
  - Can only be created by Tenant admin.
- Widgets: Interactive visualization elements
  - Plot various trypes of graphs, aggregate data.
  - Include buttons for alarms
  - Sending commands to devices
  - Display static data
  - Rich Library

Dashboard:Surface water



- GUI to select from Library and attach to data: Charts, cards, tables, alarms, counts, maps, analogue gauges, control, status, industrial, indoor, air quality, outdoor, liquid, input, gateway, edge, home page, navigation, HTML, GPIO, etc..
- Can also create new ones..