Laboratorio 4: Integración

Este laboratorio se propone integrar los distintos conceptos y desarrollos realizados durante los tres primeros laboratorios, y generar una maqueta completa desde la generación de datos hasta la visualización de los mismos.

Introducción

Existe una variada gama de tecnologías para procesar, almacenar y mostrar datos enviados por una red de nodos. Hasta hace unos años, una de las opciones más utilizadas para realizar esto con software libre y de codigo abierto era la dupla de InfluxDB y Grafana; el primero para almacenar datos y el segundo para visualizarlos en forma atractiva. Con el paso del tiempo, ambas comunidades de software tendieron a desarrollar ecosistemas más completos, integrando tanto el almacenamiento como el procesamiento y la visualización de datos. De todos modos cada desarrollo sigue teniendo su nicho de aplicación.

En este curso nos decantamos por usar la versión más reciente de InfluxDB, tanto para el almacenamiento como para la visualización de la información. Lo que implica utilizar Telegraf para la recopilación de los datos antes de ser almacenados.

Dejamos como parte opcional de este laboratorio, la visualización de datos utilizando Grafana, un software que sigue siendo un poco más versátil para ello.

Objetivos generales

- Establecer una comunicación bidireccional entre el servidor MQTT y los nodos clientes conectados a la red mediante LoRaWAN y NB-IoT.
- Generar un panel de visualización de datos enviados por los nodos.

Objetivos particulares

- Programar un nodo con tecnología de comunicación LoRaWAN que sea capaz de enviar y recibir datos utilizando el estándar CayenneLPP.
- Programar un nodo con tecnología de comunicación NB-IoT que sea capaz de enviar y recibir datos utiliando el protocolo MQTT.
- Desplegar un servidor que permita procesar, almacenar y mostrar los datos reportados por los nodos, en forma práctica y adaptable a distintos desarrollos.

Fundamentos (para tomar con pinzas)

InfluxDB

InfluxDB es una base de datos (DB) de series temporales; a diferencia de las

DB relacionales más comunes como MySQL, esta herramienta se especializa en guardar datos temporales y georreferenciados.

Podemos pensarlo como una cajonera (*bucket*) con cajones asociados a un proyecto, sector o emprendimiento (*meassurement*), y cada cajón tiene listas o planillas de mediciones de una determinada variable (*field*), con dos columnas: tiempo y valor.

En este laboratorio vamos a enviar cuatro valores: latitud, longitud, altitud y temperatura. InfluxDB va a generar cuatro planillas, una para cada una de esas magnitudes, y para cada valor recibido le va a asociar la marca de tiempo del momento en que la recibió.

Nota: Si bien vamos a enviar el timestamp y la altitud, en realidad esos datos no los vamos a utilizar; el timestamp puede ser útil para humanos, porque en realidad no es un timestamp de verdad, es la secuencia hhmmss en la que el BG96 recibió la ubicación GNSS (InfluxDB no utilizará este valor, lo considerará una magnitud más a guardar en una lista). La altitud está por mera curiosidad.

Si bien InfluxDB puede utilizarse en la web (InfluxDB Cloud), lo que vamos a hacer es instalar una versión local que es más completa.

Telegraf

Telegraf es un producto de la misma empresa/comunidad que desarrolla InfluxDB. Se encarga de recopilar datos de distintas fuentes, procesarlos, y guardarlos en InfluxDB.

No existe una versión en la nube de este programa, por lo que naturalmente lo instalaremos junto con InfluxDB.

Docker

Para instalar InfluxDB y Telegraf vamos a usar Docker.

Docker es un software libre y de código abierto que permite el manejo de contenedores; se trata de un nivel intermedio entre la instalación *nativa* de programas, y la instalación de una máquina virtual que se abstraiga completamente del sistema operativo anfitrión.

Tiene amplias ventajas para el manejo de servidores, siendo una metodología más robusta y segura (respecto a la instalación *nativa*) a la vez de facilitar el mantenimiento y actualización tanto del sistema global como de cada uno de los contenedores en particular.

El gran defecto es que como todo lo nuevo, al principio cuesta familiarizarse con la tecnología.

Docker en cinco minutos Con Docker vamos a crear una red con acceso a internet. A esta red se van a conectar los contenedores, y por ende cada uno de

los contenedores va a poder acceder a internet y también conectarse a cada uno de los otros contenedores utilizando simplemente el nombre del contenedor.

En general los contendores no guardan internamente datos en forma persistente, por lo tanto vamos a tener que enlazar un directorio interno de cada contenedor con uno del sistema operativo anfitrión.

En la guía vamos a crear los contenedores que quedarán *andando* (si todo sale bien) y mostrando mensajes de log. Con Ctrl-C paramos los contenedores. Si queremos crear nuevamente el contenedor con el mismo nombre nos va a dar error. Tenemos dos opciones para resolver eso:

1. Borrar el contenedor ya creado, usando:

```
docker rm <nombre_que_le_pusimos_al_contenedor>
```

1. Si no queremos cambiarle parámetros al contenedor, simplemente iniciamos el contenedor que ya habíamos creado:

```
docker start <nombre_que_le_pusimos_al_contenedor>
```

Cuando queremos volver a ejecutar un contendor que ya habíamos creado (por ejemplo luego de reiniciar la computadora), lo iniciamos con

```
docker start <nombre_que_le_pusimos_al_contenedor>
```

si queremos ver los contenedores en ejecución podemos usar

```
docker ps
```

y si queremos ver los mensajes de log de un contenedor (algo no funciona y hay que ver por qué), usamos

```
docker logs <nombre_que_le_pusimos_al_contenedor>
```

Tareas a desarrollar

Este laboratorio consta de tres tareas, las cuales pueden desarrollarse en forma prácticamente independiente. El orden propuesto es sólo una sugerencia.

Parte 1: Integración con InfluxDB

Para instalar InfluxDB y Telegraf vamos a usar Docker. Las instrucciones las daremos para Linux pero son análogas para Windows (es la ventaja de utilizar Docker frente a la opción de realizar la instalación *nativa* en el sistema operativo). La instalación de Docker está por fuera del alcance de esta guía, en caso que no lo tengan instalado pueden empezar por los primeros pasos.

Habiendo instalado Docker e iniciado el *backend*, lo primero es crear una red de contenedores con acceso a internet (en este caso llamada `red-contenedores-tiot`):

```
docker network create --driver bridge red-contenedores-tiot
```

Luego creamos algunos directorios donde se guardarán los datos luego de que el contenedor se *apague*:

```
cd <camino_a_mi_espacio_de_trabajo>
mkdir datos_persistentes
mkdir datos_persistentes/telegraf
mkdir datos_persistentes/influxdb
```

Instalación de InfluxDB Comenzamos creando el contenedor de InfluxDB (la primera vez demora un poco porque se baja la imagen, luego queda guardada aunque borremos el contenedor):

```
docker run --name=influxdb -p 8086:8086 \
    -v $PWD/influxdb:/var/lib/influxdb2 \
    --network red-contenedores-tiot     \
    influxdb:2.6.1
```

Si lo anterior no dio errores, podemos abrir el sitio `http://localhost:8086` en el navegador donde seguiremos los pasos de inicialización, ingresando un nombre de usuario y una contraseña. El nombre de la organización es como el despacho de cajoneras, se sugiere algo sencillo como `InfluxDBenCasa`. El nombre del *bucket* podría ser `tiot`. Luego hacemos click en *Quick Start* y ya estamos en el espacio de trabajo.

En *Load Data->Sources* buscamos el plugin *MQTT Consumer*, lo seleccionamos, hacemos click en *Use this plugin->create a new configuration*. Le ponemos un nombre como por ejemplo Mosquitto, elegimos el *bucket* y al continuar nos abrirá un editor de texto, continuaremos como si lo hubiésemos leído, y en la siguiente ventana copiaremos el valor de `INFLUX_TOKEN` en el portapaples para luego finalizar el proceso.

Para editar la configuración de esa fuente de datos, hacemos click en el nombre de la configuración, y se abrirá un editor de texto (casi igual que el anterior pero más completo) donde estableceremos las siguientes configuraciones:

- Dentro de `[[outputs.influxdb_v2]]`, la url del servidor debe ser `http://influxdb:8086` porque influxdb es el nombre que le puso docker a este contenedor.
- También en la sección `[[outputs.influxdb_v2]]`, cambiaremos el token por el que acabamos de copiar (el token va entre comillas).
- Dentro de `[[inputs.mqtt_consumer]]`, el servidor debe ser el de Mosquitto: `tcp://test.mosquitto.org:1883`
- En esa misma sección los tópicos deben ser acordes a los configurados en nuestro dispositivo, perfectamente podemos poner `TIoT/#` y aprovechar a registrar todos los datos que envían nuestros compañeros.
- Continuando en la sección `[[inputs.mqtt_consumer]]`, el `data_format` debe ser "json" y hay que agregar dos parámetros más, lo que quedaría así:

```
data_format = "json"
json_strict = false
name_override = "tiot-lab"
```

donde `tiot-lab` es el nombre del *cajón* donde se guardarán las medidas.

Finalmente descargaremos ese archivo de configuración como `moquitto.conf` y guardamos la configuración.

Instalación de Telegraf Crearemos primero la carpeta con archivos de configuración de Telegraf:

```
cd <camino_a_mi_espacio_de_trabajo>
mkdir datos_persistentes/telegraf/telegraf.conf.d
```

y en ese directorio colocaremos el archivo `mosquitto.conf` que acabamos de generar. Luego iniciaremos el contenedor de telegraf:

```
docker run --name=telegraf           \
    -v $PWD/telegraf:/etc/telegraf   \
    --network red-contenedores-tiot  \
    telegraf:alpine
```

y si no hubo errores el comando debe decir al final

```
Connected [tcp://test.mosquitto.org:1883]
```

Creación de un gráfico Ahora ya podemos probar con el cliente MQTTX a enviar datos en formato JSON del estilo

```
{"temperature":33.5,"otro_dato":24.7,"un_texto":"blah, blah"}
```

y Telegraf no debe mostrar errores. Tener cuidado que los valores númericos sean exclusivamente numéricos y sin comillas, mientras que los valores tipo *string* deben estar entre comillas.

En la pestaña de InfluxDB del navegador, vamos a *Dashboard*, creamos uno nuevo, dentro de eso creamos una nueva celda, y tenemos que seleccionar los datos a mostrar. Para eso, en donde se arma el *Query*:

1. En `FROM` seleccionar `tiot`.
2. Filtrar por `_measurement` seleccionando `tiot-lab` que es nuestro *cajón* de medidas. Si no aparece la opción (en general es la última) es porque no se cargaron datos con ese nombre: revisar los logs de Telegraf, enviar datos y volver a recargar la página.
3. Fijarse que en `_field` sólo aparecen los datos numéricos; por defecto los datos que no son numéricos se desechan. Seleccionar la temperatura (o lo que se quiera graficar).

4. Al hacer click en *SUBMIT* se deberían ver los valores en el gráfico (cuando hay uno sólo no es muy visible, enviar más o pasar el mouse por encima del gráfico).

Conclusiones Con lo anterior, ya deberían ser visibles los datos que envía nuestro dispositivo programado según el Laboratorio 3 del curso. Notar que telegraf se suscribe al tópico seleccionado y va a guardar las medidas que lleguen mientras esté suscripto; no vamos a ver medidas anteriores a su ejecución.

Parte 2: Integración con LoRaWAN

En esta parte, tomaremos como referencia el trabajo realizado en el Laboratorio 2 para enviar datos usando FreeRTOS. La idea es usar el sistema de codificación CayenneLPP (Cayenne Low Power Payload) propuesto por My Devices. Se sugiere la siguiente implementación:

1. Utilizar la misma aplicación `tiot-lab2` de TTN compartida por los docentes del curso.
2. Utilizar el ejemplo `CayenneLPP` de la librería `TheThingsNetwork` para adaptar el programa realizado en el Laboratorio 2, enviando tanto la temperatura como el valor aleatorio (este último se sugiere codificarlo como un dato de tipo `Analog Input`).
3. Modificar el *payload formatter* del *uplink* de nuestro dispositivo para que utilice CayenneLPP.
4. Verificar la correcta codificación y decodificación del valor enviado, utilizando como referencia la documentación de CayenneLPP. Para ello codificar manualmente el dato, compararlo con el hexadecimal generado por la librería (ver en Serial Monitor) y luego decodificarlo y compararlo con la decodificación que realiza TTN en su consola. Puede ser útil tener a mano el conversor base64.

Una vez realizado lo anterior, crear una nueva fuente de datos en Telegraf, que se conecte al servidor MQTT de TTN y procese los datos enviados por cualquiera de los dispositivos en la aplicación `tiot-lab2`. Para esto se debe utilizar el procedimiento descripto en la Parte 1, con las siguientes modificaciones:

1. El servidor MQTT de TTN precisa autenticación, esos datos deben especificarse en la configuración (edición del archivo de texto).
2. Se le debe especificar a Telegraf que procese sólamente la parte del JSON que contiene el payload, por lo tanto, dentro de la sección `[[inputs.mqtt_consumer]]` se debe agregar la línea:

```
json_query = "uplink_message.decoded_payload"
```

Más información sobre la configuración para procesar el JSON se puede ver en este README.

Opcional: Downlink de datos Basándose en el ejemplo *Receive* de *TheThingsNetwork*, implementar una funcionalidad análoga a la de la Parte 3; la Arduino deberá encender o apagar el LED conectado al pin 11 dependiendo del mensaje que se reciba. Probar la interacción enviando mensajes al tópico `v3/tiot-lab2@ttn/devices/<mi_dispositivo>/down/push` y viendo el resultado en el LED del dispositivo. Puede ser de utilidad la documentación de la integración MQTT de TTN.

Parte 3: Integración con NB-IoT

Utilizar el código brindado por los docentes para este Laboratorio (se trata de una versión ampliada del denominado `esqueleto_AT` dado en el Laboratorio 3) completándolo para lograr:

- Envío de coordenadas GNSS, armando un texto en formato JSON con los parámetros "timestamp", "altitude", "longitude", "latitude" y "temperature".
- Suscripción a un tópico, recepción de mensajes y actuación en función de ellos: debe encender o apagar el LED que tiene la placa Arduino Zero asociado al pin 11 dependiendo de si recbie el mensaje `{"led":"on"}` o `{"led":"off"}`

> **Nota:** Como se puede apreciar en el código, no es preciso ser robustos en el manejo de espacios y otros parámetros que pueda incluir el JSON.

Para eso se sugiere analizar el código brindado por los docentes y completarlo, en particular en las partes que explícitamente se indica. Se puede tomar como insumo el código desarrollado para el Laboratorio 3 (en particular la inicialización del módulo).

Esta extensión de la librería anterior permite guardar una parte de la respuesta obtenida del módulo BG96, utilizando un comodín (por defecto es el caracter `^`). Por ejemplo, la línea

```
sendCommandAndWaitFor("AT+QGPSLOC=2\r\n", "+QGPSLOC: ^,", rcv_data_buffer);
```

guarda en el buffer `rcv_data_buffer` todo lo que está después de `+QGPSLOC:` pero antes de la primer coma. Notar que se utiliza la instrucción `AT+QGPSLOC=2` para lograr que el módulo responda las coordenadas en formato decimal y de este modo se puedan enviar fácilmente a la base de datos de InfluxDB.

También se implementó una espera por mensajes nuevos con un timeout. Esto permite consultar si se recibieron nuevos mensajes en el tópico suscripto. Notar que el mensaje se guarda incluyendo las comillas.

Parte 4 (opcional): Integración con Grafana

La integración con Grafana nos permitirá generar un mapa interactivo con los dispositivos NB-IoT georeferenciados.

Al igual que con los contenedores anteriores, primero creamos el directorio de datos:

```
cd <camino_a_mi_espacio_de_trabajo>
mkdir datos_persistentes/grafana
```

y luego iniciaremos el contenedor de grafana:

```
docker run --name=grafana -p 3000:3000  \
    -v $PWD/grafana:/var/lib/grafana    \
    --network red-contenedores-tiot     \
    -u $(id -u) grafana:grafana-oss
```

En caso que no haya errores, abrir en el navegador el sitio `http://localhost:3000`, ingresar con nombre de usuario `admin`, contraseña `admin`, y luego configurar una nueva contraseña.

Lo primero es ir a *DATA SOURCES* y seleccionar InfluxDB. Esto nos permitirá tomar datos de nuestra base de datos. En *Query Language* seleccionamos *Flux*, el servidor será `http://influxdb:8086`, la autenticación la dejaremos en blanco, en *Organization* ponemos `InfluxDBenCasa`, el *bucket* por defecto será `tiot` y el *token* hay que generarlo en el sitio de InfluxDB. Finalmente con *Save & test* verificamos que se haya conectado correctamente.

Luego creamos un nuevo *dashboard*, agregamos un nuevo panel y arriba a la derecha, donde dice *Time series* lo cambiamos seleccionando la opción *Geomap*.

En el *Query* seleccionamos los datos que queremos mostrar[1]:

```
from(bucket: "tiot")
  |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
  |> filter(fn: (r) => r["_measurement"] == "tiot-lab")
  |> filter(fn: (r) => r["_field"] == "latitude" or r["_field"] == "longitude"
  or r["_field"] == "temperature")
  |> group(columns: ["_time", "_measurement", "host", "topic"])
  |> drop(columns: ["_measurement", "_start", "_stop", "host", "topic"])
  |> pivot(rowKey: ["_time"], columnKey: ["_field"], valueColumn: "_value")
  |> filter(fn: (r) => exists r["latitude"]
  and exists r["longitude"] and exists r["temperature"])
  |> group(columns: ["latitude", "longitude"])
  |> rename(columns: {temperature: "_value"})
  |> last()
```

[1] Lo que hace la sentencia es recuperar datos, y para cada nodo, y cada juego de coordenadas latitud/longitud se queda con el último valor de temperatura reportado. La sintaxis de Flux puede consultarse en https://docs.influxdata.com/influxdb/cloud/query-data/get-started/

```
|> group()
```

En el panel de la derecha, en la sección *Map layers*, seleccionamos *Markers* como tipo de capa, en *Data* seleccionamos el *query* que acabamos de hacer, y en *Location Mode* dejamos *Auto*.

Más configuraciones pueden hacerse a gusto, como colorear los puntos según la temperatura. Debe tenerse en cuenta que muchas veces suceden cosas no esperadas debido a que el rango de tiempo seleccionado no contiene datos válidos; se sugiere trabajar con un rango amplio donde podamos confirmar que allí hay datos para luego configurar el panel.

Entregables y Defensa

Cada estudiante debe asistir a la defensa con el laboratorio realizado en su completitud, y habiendo contestado de manera individual -y en tiempo y forma- la encuesta correspondiente publicada en la plataoforma EVA.

El día de la defensa deberán poder explicar cada una de las partes de este laboratorio.