VAGRANT-ANSIBLE MQTT + STACK TIG

ENTREGA - 3

12 Abril 2022

Escuela Superior de Informática

Mariola Zarco de Gracia

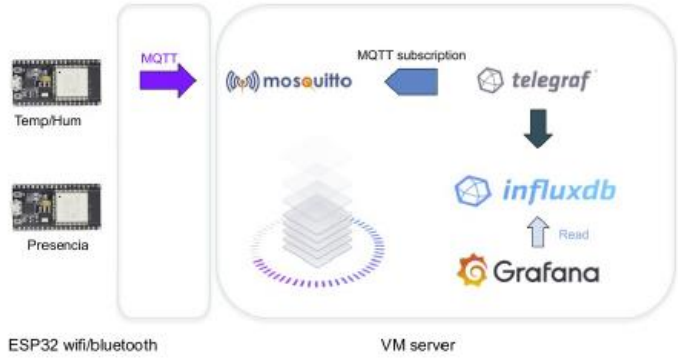
Índice de contenidos

1. Enunciado

Automatice el despliegue y configuración de un sistema que disponga de al menos tres actores / servicios que interactúen entre sí, mediante el uso de Vagrant y Ansible. El ejercicio queda abierto a cualquier servicio, por lo tanto es parte del / la estudiante elegir qué servicios serán configurados y desplegados en las máquinas descritas en Vagrant.

Se propone como ejemplo el stack TIG (Telegraf + InfluxDB + Grafana) visto en la asignatura de sistemas ciberfísicos, donde cada uno de los servicios puede ser desplegado y configurado en diferentes máquinas.

Incluso el bróker MQTT puede ser también virtualizado mediante Vagrant y Ansible.



Se deberá entregar una pequeña memoria con las decisiones tomadas y razonadas (recursos hardware asignados a cada nodo, configuraciones de red, …), una imagen de la arquitectura TI implementada y el funcionamiento del mismo. Se deberá incluir un fichero adicional con los pasos a realizar para desplegar toda la arquitectura.

https://github.com/jkogut/tigck-playbooks

2. Tecnologías utilizadas

2.1. VirtualBox

Es un software de virtualización desarrollado por Oracle Corporation, soporta múltiples sistemas operativos como GNU/Linux, Mac OS X, Windows, además de ser una herramienta que tiene una versión de código abierto con licencia GPL versión 2.

VitualBox permite virtualizar los sistemas operativos FreeBSD, GNU/Linux, OpenBSD, Windows, Solaris y muchos otros. En lo que respecta a los discos duros, utiliza emulación hardware en el que los discos de los sistemas invitados se guardan en el anfitrión como archivos individuales llamados discos duros virtuales.

2.2. Vagrant

Vagrant es una herramienta para la creación y configuración de entornos virtualizados en un solo flujo de trabajo. Es bastante simple, creada por HashiCorp puede ejecutarse tanto en Mac Os, Linux como en Windows. Además, es una herramienta de código abierto, con licencia MIT.

Esta desarrollada en Ruby y su configuración se basa en ficheros de configuración llamados **Vagrantfiles.** En estos ficheros se puede definir todo un entorno virtual que necesites desplegar, desde el sistema operativo a las interfaces de red.

También nos permite gestionar el aprovisionamiento de las instancias, compatible con orquestadores como Ansible, Docker y scripts de bash. La integración con estas tecnologías permite un control total sobre la configuración de las instancias.

Otra funcionalidad muy interesante es a lo que Vagrant denomina **Boxes**[1], son imágenes preinstaladas de diferentes sistemas operativos versionados. Hasicorp proporciona estas Boxes a través de un repositorio, donde se puede encontrar todos los sistemas en sus versiones oficiales. Y también se pueden encontrar Boxes subidas por usuarios con modificaciones y preinstalaciones de aplicaciones listas para ser usadas.

2.3. Ansible

Es una plataforma de software libre para aprovisionar y administrar infraestructura y aplicaciones. Está desarrollado principalmente por RedHat, ha sido categorizado como herramienta de orquestación. Gestiona los nodos a través de SSH y no necesita de ningún software adicional (excepto Python).

Con Ansible podemos actuar contra plataformas en la nube, como host virtualizados, hipervisores e incluso dispositivos de red o servidores físicos. Se distribuye más comúnmente en arquitecturas GPU/Linux a través de los paquetes EPEL o PIP.

Arquitectura de ansible:

Ansible tiene dos tipos de servidores:

Automatización de la creación de escenarios para protocolos de red.

- Controlador: Servidor desde el que se lanza la orquestación y tiene el código fuente a ejecutar. Tiene Ansible instalado. En nuestro caso será la máquina anfitriona.
- Nodo: Al que se conecta el controlador mediante SSH y es aprovisionado. No necesita de ningún software aparte de conectividad SSH como ya hemos comentado.

3. Distribución de Arquitectura

Hemos decidido proceder con los datos recogidos de nuestro anterior ejercicio de la asignatura de IoT, que contenía información de sensores de temperatura, sonido y nivel de líquido, que reemplazarían los de presencia en este dibujo, y quedaría la distribución por nodos como sigue:

[1] Vagrant boxes: https://app.vagrantup.com/boxes/search

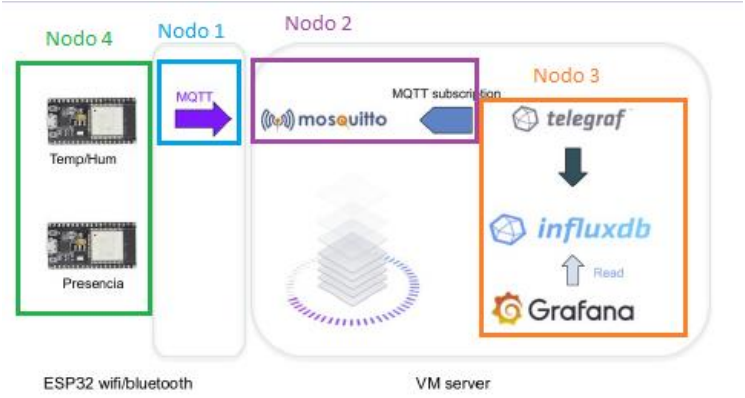


Figura 3.1 – Arquitectura y distribución de Nodos.

4. Configuraciones y documentación

4.1. Vagrant File

Para la configuración del Vagrant file se han tenido en cuenta las siguientes cuestiones:

COMANDO: *$ vagrant init*

- En primer lugar, hemos revisado el listado de Boxes de Vagrant, filtrando por los conceptos de mosquitto, mqtt, grafana y ansible y aunque hemos encontrado algunas boxes ya desarrolladas para algunas de las aplicaciones, no se correspondían 100% con nuestro ejercicio, por lo que hemos decidido dejar la box genérica de Ubuntu.[2]

- Hemos decidido trabajar con la distribución de Ubuntu. Se podría haber elegido la distribución CentOs por ofrecer mayor estabilidad y haber aplicado el comando yum para la descarga e instalación de paquetes, pero hemos preferido mantener Ubuntu por disponer de una comunidad de desarrollo e informaciones más amplia en red para esta etapa inicial ya que los datos en la aplicación tampoco son abundantes.

```
Vagrant.configure("2") do |config|
  config.vm.box = "ubuntu/xenial64"
```

- Ya que son distintas las aplicaciones que van a necesitar de acceso a la red y la red de las máquinas debe ser la misma que las del PC anfitrión, hemos optado por configurar

[2] https://app.vagrantup.com/vadoc2020/boxes/mosquitto

https://app.vagrantup.com/codeyourinfra/boxes/monitor

https://app.vagrantup.com/superbogiuseppe/boxes/grafana_server

una red pública, creando un puente entre redes para que cada máquina aparezca como otro dispositivo físico y tener una mejor identificación. Esto lo definimos dentro de cada nodo.

```
N1.vm.network "public_network", bridge: "enp_03"
```

- Se debe redirigir el puerto 80 de las máquinas invitadas o SO invitado a un puerto de la máquina que las alberga (anfitrión).

  COMANDO: $ sudo python3 -m http.server 80

- Hemos configurado los nodos como sigue, teniendo en cuenta que la mayor cantidad de flujo vendrá desde el Broker y el Publisher de Mosquitto, los hemos puesto en nodos independientes 1 y 2, y el resto de aplicaciones TIG, las hemos unificado en un único nodo 3, y por último los sensores en un nodo 4, balanceando así las cargas entre los nodos y manteniendo el acceso a la información lo más cerca posible entre ellos.

```
Vagrant.configure("2") do |config|
  config.vm.box = "ubuntu/xenial64"
  config.vm.define "nodo1" do |n1|
        N1.vm.network "public_network", bridge: "enp_03"
        N1.vm.hostname = "Mosquitto_Broker"
        N1.vm.network "forwarded_port", guest: 80, host:8080, host_ip: "127.0.0.1"
        N1.vm.provider "virtualbox" do |vb|
            vb.cpus = "2"
            vb. Memory = "1024"

  config.vm.define "nodo2" do |n2|
        N2.vm.network "public_network", bridge: "enp_03"
        N2.vm.hostname = "Mosquitto_Publisher"
        N2.vm.network "forwarded_port", guest: 80, host:8081, host_ip: "127.0.0.1"
        N2.vm.provider "virtualbox" do |vb|
            vb.cpus = "2"
            vb. Memory = "512"
```

```
config.vm.define "nodo3" do |n3|
      N3.vm.network "public_network", bridge: "enp_03"
      N3.vm.hostname = "TIG"
      N3.vm.network "forwarded_port", guest: 80, host: 8082, host_ip: "127.0.0.1"
      N3.vm.provider "virtualbox" do |vb|
          vb.cpus = "2"
          vb. Memory = "512"

config.vm.define "nodo4" do |n4|
      N4.vm.network "public_network", bridge: "enp_04"
      N4.vm.hostname = "Sensores"
      N4.vm.network "forwarded_port", guest: 80, host: 8083, host_ip: "127.0.0.1"
      N4.vm.provider "virtualbox" do |vb|
          vb.cpus = "2"
          vb. Memory = "512"
```

Nota: Hemos puesto el hostname con el fin de ilustrar y referenciar visualmente la distribución de los notods, pero se mantendrn sus hostnames como nodo1, nodo2, nodo3 y nodo4.

- Por último hemos definido las rutas para los archivos que deben sincronizarse tanto en el anfitrión como en la máquina invitada como sigue:

```
config.vm.synced_folder "../data/nodo1", "service/ansible/nodo1"
config.vm.synced_folder "../data/nodo2", "service/ansible/nodo2"
config.vm.synced_folder "../data/nodo3", "service/ansible/nodo3"
config.vm.synced_folder "../data/nodo4", "service/ansible/nodo4"
```

- En este caso únicamente tenemos un hypervisor con lo que lo necesitaremos configuración adicional en este sentido.
- Por último, aquí describimos los comandos[3] que se van a utilizar tanto para la configuración, como para el inicio o parada de la máquina virtual, y para la configuración de SSH para tener acceso a la terminal, **especificando el nombre de la máquina virtual**.

```
#Comando para configuración de la máquina virtual
$ vagrant init ubuntu/xenial64

#Comandos para iniciar o detener la máquina virtual
$ vagrant up xenial64
$ vagrant halt

#Comandos para ver el listado y creación
$ vboxmanage list vms | grep xenial64
"xenial64" {a507ba0c-...24bb}

#Configuración de SSH para tener acceso a la terminal
$ vagrant ssh
$ vagrant ssh default
$ vagrant ssh node1

#Comandos para ver el status
$ vagrant status
$ vagrant global-status
```

- Para el tamaño, hemos visitado las páginas con guías sobre el tamaño, como InfluxDB hardware sizing en su versión 1.9.[4] donde se nos aconseja que si tenemos un único nodo, será válido pero no tendremos redundancia de datos, por lo cual si aplicamos el factor de réplica n-1, no podríamos permitirnos perder ningún nodo. Otras consideraciones[i] a tener en cuenta son:
    - Número de escrituras y consultas por segundo
    - Cardinalidad de la serie
    - Complejidad de las consultas
    - Almacenaje de tipo SSD (sino al menos 2000 IOPs para recuperación rápida)

[3] https://www.vagrantup.com/docs/cli/ssh
[4] https://docs.influxdata.com/enterprise_influxdb/v1.9/guides/hardware_sizing/

- Bytes y comprensión - Según sean las medidas*, tags keys, field keys, tag values y consideración de que los field values y timestamps* se guardan en cada punto. Los valores no string, requieren 3 bytes y el resto depende de la compresión.
- Separación de los directorios wal y data en diferentes dispositivos de almacenamiento.

En nuestro proyecto teníamos la recepción de 3 sensores diferentes (agua, temperatura, sonido) en diferentes habitaciones de una casa con la siguiente estructura que define el tamaño de los tags utilizados:

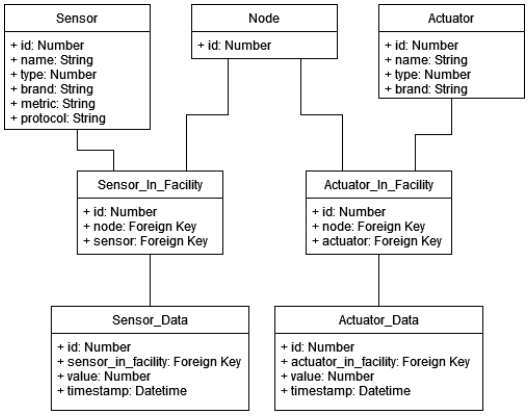


Figura 4.1 – Campos y etiquetas recogidos por cada sensor de la ESP32.

La estimación fue de 82 mensajes a la hora, con lo que tenemos un ratio inferior a 1 mensaje por segundo, con un total de 612 KB estimados al día, con lo que sincronizando bien los tiempos de escritura de los distintos dispositivos y teniendo en cuenta la simplicidad de las escrituras propuestas, tenemos un total de 252,68 KB al año.

➔ Por lo tanto, aun teniendo en cuenta las consideraciones anteriores, estimamos que con dos CPU para evitar la redundancia ya que las mediciones se realizan a la vez y es difícil sincronizarlas y con una capacidad de 512Mb[5], tenemos más que suficiente para el nodo 3 TIG.

El nodo 1 y 2, en un principio tampoco requerirían de mayor capacidad de los 512Mb y con un VirtualBox de 60Gb deberíamos tener suficiente, según los cálculos arriba expuestos ya que no se preveé el aumento de número de sensores o medidas, las consultas a realizar no serían superiores a 2 por hora por cada tipo de dato y siempre podríamos redimensionar. Hemos preferido no limitarla a la baja por si necesitamos realizar la instalación de otras aplicaciones o instalaciones que puedan ocupar espacio en nuestras máquinas.

[5] https://docs.influxdata.com/influxdb/v1.8/guides/hardware_sizing/

4.2. Configuración de los Hosts - Inventario

Configuramos los hosts a través del uso de ssh y los private keyfiles que a través del comando *ssh-keygen -lf /path/to/key* hemos encontrado en[6]:

```
ansible_ssh_private_key_file=/home/mariola/vagrant_xenial64/.vagrant/machines/nodo1/virtualbox/private_key
```

El contenido del Inventario_Hosts es cada nodo con cada private key más su IP correspondiente y nos permite el acceso a través de los comandos ssh, o bien a los puertos de los nodos (forwarded del dispositivo NAT) o a través de la IP.

```
$ ssh -p 2200 \
> -i /home/mariola/vagrant_xenial64/.vagrant/machines/nodo1/virtualbox/private_key \
> vagrant@localhost hostname
xenial64

$ ssh -i /home/mariola/vagrant_xenial64/.vagrant/machines/nodo1/virtualbox/private_key \
> vagrant@192.168.18.50 hostname
xenial64
```

4.3. Configuración genérica – Ping, EPEL e Instalación de servicios

Por último realizamos la instalación de EPEL cada uno de los nodos e instalamos las correspondientes aplicaciones en caso de que no existan, con el become:true me convierto en superusuario y después procedo reiniciando cada uno de los servicios siendo el contenido del archivo *playbook_all.yml* el siguiente (página siguiente):

Nota 1

Las aplicaciones TIG las tenemos ya previamente instaladas a través del comando wget con la última versión de cada una de ellas en nuestro folder /etc/aplicación. Si este no fuera el caso, al no estar utilizando cent0S tendríamos que incluir una opción shell: wget [ruta destino] [link descarga].

Nota 2

Para ejecutar los playbooks utilizamos el comando:

COMANDO: $ ansible-playbook-i hosts (nameofplaybook.yml)

Nota 3

Para verificar que todos los nodos están activos ejecuto el siguiente comando

COMANDO: $ ansible -i hosts all -m ping

[6] https://martincarstenbach.wordpress.com/2019/12/12/tipsntricks-finding-the-injected-private-key-pair-used-in-vagrant-boxes/

Comentado [MZDG1]:

Comentado [MZDG2R1]:

Comentado [MZDG3]:

C: > Users > Usuario > Desktop > Entrega_3_Vagrant > ! playbook_all.yml

```yaml
---
- hosts: all
  tasks:
    - name: Ping to all nodes
      ping:
    - name: Install EPEL
      become: true
      name: epel-release
      state: present


- hosts: nodo1
  become: true
  tasks:
    - name: Install mosquitto
      become: true
      name: mosquitto
      state: present
    - name: Copy config
      copy:
        src: broker.conf
        dest: /etc/mosquitto/mosquitto.conf
      notify: Restart Broker
  handlers:
    - name: Restart Broker
      service:
        name: mosquitto
        state: started

- hosts: nodo2
  become: true
  tasks:
    - name: Install mosquitto
      become: true
      name: mosquitto
      state: present
    - name: Copy config
      copy:
        src: pub.conf
        dest: /etc/mosquitto/mosquitto.conf
      notify: Restart Publisher
  handlers:
    - name: Restart Publisher
      service:
        name: mosquitto
        state: started
```

C: > Users > Usuario > Desktop > Entrega_3_Vagrant > ! playbook_all.yml

```yaml
- hosts: nodo3
  become: true
  tasks:
   - name: Install Telegraf
     become: true
     name: telegraf
     state: present
    - name: Copy config
     copy:
       src: telegraf.conf
       dest: /etc/telegraf/telegraf.conf
     notify: Restart Telegraf
   - name: Install Influx DB
     become: true
     name: influxDB
     state: present
    - name: Copy config
     copy:
       src: influxDB.conf
       dest: /etc/influxDB/influxDB.conf
     notify: Restart InfluxDB
  - name: Install Grafana
     become: true
     name: grafana
     state: present
    - name: Copy config
     copy:
       src: influxDB.conf
       dest: /etc/grafana/grafana.conf
     notify: Restart Grafana
  handlers:
    - name: Restart Telegraf
     service:
       name: telegraf
       state: started
    - name: Restart InfluxDB
     service:
       name: influxdb
       state: started
    - name: Restart Grafana
     service:
       name: grafana
       state: started
```

4.4. Configuración del Nodo 1 - Broker de Mosquitto

En este caso hemos seguido la configuración del ejercicio anterior practicado en clase, siendo el resultado del contenido del archivo *m_bro.conf:*

```
C: > Users > Usuario > Desktop > Entrega_3_Vagrant > broker.conf
1   listener 2883 192.168.18.53 #IP de sensores
2   allow_anonymous true
```

Por otro lado creamos el playbook del broker *playbook_m_bro.yml* con las tareas correspondientes de comprobación e inicio asignadas, como sigue:

```
1   ---
2   - hosts: broker_mqtt
3   tasks:
4   - name: Copy config
5     become: true
6   copy:
7     src: broker.conf
8     dest: /etc/mosquitto/mosquitto.conf
9   - name: Check Broker
10    service:
11    name: mosquitto
12    state: started   #le digo que inicie mosquito
13
```

4.5. Configuración del Nodo 2 - Publisher de Mosquitto

En este caso hemos seguido con la configuración[7] del archivo *m_pub.conf:*

```
C: > Users > Usuario > Desktop > Entrega_3_Vagrant > pub.conf
1   listener 2883 192.168.18.50 #IP del broker
2   allow_anonymous true
```

Por otro lado creamos el playbook del publisher *playbook_m_pub.yml* con las tareas de publicación asignadas, como sigue:

[7] https://mosquitto.org/man/mosquitto_pub-1.html

https://apimirror.com/ansible~2.9/modules/mqtt_module

```yaml
---
- hosts: pub_mqtt
  tasks:
    - name: Install Paho-MQTT
      become: true
      yum:
        name: python-paho-mqtt
        state: present
    - name: Publish a message on an MQTT topic - Temperature
      mqtt:
        topic: 'service/ansible/nodo4'
        payload: 'Temperature at {{ ansible_date_time.iso8601 }}'
        qos: 0 # calidad el servicio
        retain: False # si queremos que se retenga el mensaje
        server:  192.168.18.51. #IP del nodo 2, que es donde publica.
        port: 2883 #(por defecto es el 1883)
    - name: Publish a message on an MQTT topic - Sound Level
      mqtt:
        topic: 'service/ansible/nodo4'
        payload: 'Sound_level at {{ ansible_date_time.iso8601 }}'
        qos: 0
        retain: False
        server:  192.168.18.51.
        port: 2883
    - name: Publish a message on an MQTT topic - Water Level
      mqtt:
        topic: 'service/ansible/nodo4'
        payload: 'Water_level at {{ ansible_date_time.iso8601 }}'
        qos: 0
        retain: False
        server:  192.168.18.51.
        port: 2883
```

4.6. Configuración del Nodo 3 - TIG

Configuración de los archivos de configuración de cada una de las aplicaciones *playbook_tel_conf, playbook_inf_conf y playbook_graf_conf:*

C: > Users > Usuario > Desktop > Entrega_3_Vagrant > ! playbook_tel_conf.yml

```yaml
---
- hosts: nodo3 #suscriptor y publicador telegraf
tasks:
- name: Copy config
  become: true
copy:
  src: telegraf.conf
  dest: /etc/telegraf/telegraf.conf
- name: Comprobar si suscriptor y publicador de telegraf activo
  service:
  name: telegraf
  state: started   #le digo que inicie telegraf
```

C: > Users > Usuario > Desktop > Entrega_3_Vagrant > ! playbook_inf_conf.yml

```yaml
---
- hosts: nodo3 #suscriptor InfluxDB
tasks:
- name: Copy config
  become: true
copy:
  src: influxDB.conf
  dest: /etc/influxdb/influxDB.conf
- name: Check Suscriptor
  service:
  name: influxDB
  state: started   #le digo que inicie InfluxDB

```

C: > Users > Usuario > Desktop > Entrega_3_Vagrant > ! playbook_graf_conf.yml

```yaml
---
- hosts: nodo3 #suscriptor y publicador grafana
tasks:
- name: Copy config
  become: true
copy:
  src: grafana.conf
  dest: /etc/grafana/grafana.conf
- name: Comprobar si suscriptor de grafana esta activo
  service:
  name: grafana
  state: started   #le digo que inicie telegraf

```

Nota: No he encontrado el módulo de Telegraf para ver las opciones de su configuración por tanto lo he configurado con la plantilla de mqtt para que al menos figurase el paso en la memoria, pero no es correcto.

Configuración del suscriptor de telegraf con el archivo *telegraf_sus.conf:*

C: > Users > Usuario > Desktop > Entrega_3_Vagrant > telegraf_sus.conf

```
listener 2883 192.168.18.51 #IP del mosquitto_publisher
allow_anonymous true
```

Configuración del publicador de telegraf *playbook_tel_pub.conf:*

C: > Users > Usuario > Desktop > Entrega_3_Vagrant > ! playbook_tel_pub.yml

```yaml
---
- hosts: nodo3 #pub_telegraf
  tasks:
    - name: Transfer the message on an MQTT topic - Temperature from node 2 into InfluxDB
      telegraf:
        topic: 'service/ansible/nodo2'
        payload: 'Temperature at {{ ansible_date_time.iso8601 }}'
        qos: 0 # calidad el servicio
        retain: False # si queremos que se retenga el mensaje
        server:  192.168.18.52. #IP del nodo 3, que es donde publica.
        port: 2883 #(por defecto es el 1883)
```

```
- name: Transfer the message on an MQTT topic - Sound Level from node 2 into InfluxDB
  telegraf:
    topic: 'service/ansible/nodo2'
    payload: 'Sound_level at {{ ansible_date_time.iso8601 }}'
    qos: 0
    retain: False
    server:  192.168.18.52.
    port: 2883
- name: Transfer the message on an MQTT topic - Water Level from node 2 into InfluxDB
  telegraf:
    topic: 'service/ansible/nodo2'
    payload: 'Water_level at {{ ansible_date_time.iso8601 }}'
    qos: 0
    retain: False
    server:  192.168.18.52.
    port: 2883
```

Primero creo una base de datos, cuya lectura la vamos a realizar a nivel interno y en principio sin restricciones de user y password a los usuarios. En este archivo *playbook_inf_DB.yml* también le indicamos los data_points a recolectar.

C: > Users > Usuario > Desktop > Entrega_3_Vagrant > ! playbook_inf_DB.yml

```
---
- hosts: nodo3 #suscriptor InfluxDB
tasks:
- name: Create database
  influxdb_database:
      hostname: "192.168.18.53"
      database_name: "DB_Vivienda_1"
- name: Write points into database
  influxdb_write:
      hostname: "192.168.18.53"
      database_name: "DB_Vivienda_1"
      data_points:
        - measurement: Temperature
          tags:
            host: nodo4
            sensor: 1
          time: "{{ ansible_date_time.iso8601 }}"
          fields:
            value: 2000 #desconoczco el rango
        - measurement: Sound_Level
          tags:
            host: nodo4
            sensor: 2
          time: "{{ ansible_date_time.iso8601 }}"
          fields:
            value: 2000 #desconoczco el rango
        - measurement: Water_Level
          tags:
            host: nodo4
            sensor: 3
          time: "{{ ansible_date_time.iso8601 }}"
          fields:
            value: 2000 #desconoczco el rango
```

Por último la configuración del contenido del archivo *graf_datasource.yml* para poder escuchar y representar los registros de influxDB sería el siguiente:

C: > Users > Usuario > Desktop > Entrega_3_Vagrant > ! playbook_graf_datasource.yml

```yaml
---
- name: Create influxdb datasource
  grafana_datasource:
    name: "datasource-influxdb"
    grafana_url: "https://grafana.company.com"
    grafana_user: "admin"
    grafana_password: "xxxxxx"
    org_id: "1"
    ds_type: "influxdb"
    ds_url: "https://influx.company.com:8086"
    database: "telegraf"
    time_interval: ">10s"
    tls_ca_cert: "/etc/ssl/certs/ca.pem"
```

Si quisiéramos visualizar la información a través de un dashboard podríamos utilizar la siguiente configuración:

```yaml
- hosts: localhost
  connection: local
  tasks:
    - name: Import Grafana dashboard foo
      grafana_dashboard:
        grafana_url: http://grafana.company.com
        grafana_api_key: "{{ grafana_api_key }}"
        state: present
        message: Updated by ansible
        overwrite: yes
        path: /path/to/dashboards/foo.json

    - name: Export dashboard
      grafana_dashboard:
        grafana_url: http://grafana.company.com
        grafana_user: "admin"
        grafana_password: "{{ grafana_password }}"
        org_id: 1
        state: export
        uid: "000000653"
        path: "/path/to/dashboards/000000653.json"
```

4.7. Configuración del Nodo 4 - Sensores

Aunque lo hemos puesto en último lugar, realmente deberíamos haber empezado por esta configuración para poder ir comprobando paso a paso que el resto de las configuraciones anteriores eran correctas conforme construíamos la arquitectura. La configuración del nodo 4 que contiene la publicación de información por parte de los sensores sería la siguiente *playbook_sens_pub.yml*:

C: > Users > Usuario > Desktop > Entrega_3_Vagrant > ! playbook_sen_pub.yml

```yaml
---
- hosts: nodo4 #sensores
  tasks:
    - name: Install Paho-MQTT
      become: true
      yum:
        name: python-paho-mqtt
        state: present
    - name: Publish a message on an MQTT topic - Temperature
      mqtt:
        topic: 'service/ansible/nodo4'
        payload: 'Temperature at {{ ansible_date_time.iso8601 }}'
        qos: 0 # calidad el servicio
        retain: False # si queremos que se retenga el mensaje
        server:  192.168.18.50. #IP del nodo 1, que es donde publica.
        port: 2883 #(por defecto es el 1883)
    - name: Publish a message on an MQTT topic - Sound Level
      mqtt:
        topic: 'service/ansible/nodo4'
        payload: 'Sound_level at {{ ansible_date_time.iso8601 }}'
        qos: 0
        retain: False
        server:  192.168.18.50.
        port: 2883
    - name: Publish a message on an MQTT topic - Water Level
      mqtt:
        topic: 'service/ansible/nodo4'
        payload: 'Water_level at {{ ansible_date_time.iso8601 }}'
        qos: 0
        retain: False
        server:  192.168.18.50.
        port: 2883
```

4.8. Desinstalación [8]

Por último existen unos comandos útiles para la desinstalación del programa:

```
rm -rf /opt/vagrant
rm -f /usr/bin/vagrant
```

Y de los datos asociados:

En todas las plataformas, este directorio se encuentra en la raíz de su directorio personal, y se llama vagrant.d. Eliminamos el directorio ~/.vagrant.d para borrar los datos del usuario.

[8] https://www.vagrantup.com/docs/installation/uninstallation