Monitorar l'estat del sistema de refrigeració i energia d'un CPD mitjançant IoT

Xavier Ramon Vilamú

Màster en Enginyeria de Telecomunicacions (Pla UOC-URL)

Enginyeria Telemàtica

Jose Lopez Vicario

Xavi Vilajosana Guillen

09/06/2019

FITXA DEL TREBALL FINAL

Títol del treball:	*Monitorar l'estat del sistema de refrigeració i energia d'un CPD mitjançant IoT*
Nom de l'autor:	*Xavier Ramon Vilamú*
Nom del consultor/a:	*Jose Lopez Vicario*
Nom del PRA:	*Xavi Vilajosana Guillen*
Data de lliurament:	*06/2019*
Titulació o programa:	*Màster en Enginyeria de Telecomunicacions (Pla UOC-URL)*
Àrea del Treball Final:	*TFM – Enginyeria Telemàtica*
Idioma del treball:	*Català*
Paraules clau	*Monitoratge* *ELK*

Resum del Treball (màxim 250 paraules): *Amb la finalitat, context d'aplicació, metodologia, resultats i conclusions del treball*

El projecte se centrarà en crear un sistema de monitoratge de l'alimentació elèctrica, temperatura i humitat. El projecte es basarà en configurar un equip basat en Raspberry Pi. Cada dispositiu s'encarregarà de rebre un tipus de dades: d'alimentació elèctrica i de temperatura i humitat. Aquesta informació que recolliran els diferents sensors seran enviats a logstash per parseja els fitxers de log i els transformar les dades en el format adient per a enviar-les a un servidor d'elasticsearch i finalment per mostrar les dades recollides per Kibana que realitzar consultes a elasticsearch i retorna gràfics. Finalment, aquest projecte desenvoluparà la gestió i emmagatzematge de les dades en el cloud.

Índex

Llista de taules

Llista de figures

1. Introducció

1.1. Antecedents del Treball

El passat 28 de gener un problema de subministrament elèctric en una empresa va produir que el CPD és quedes sense subministrament elèctric. Per anàlisis posteriors, es va poder determinar que es van produir continus talls en el servei elèctric que van impedir que el grup electrogen és possés en funcionament alhora que les bateries dels SAIs es recaiguessin a temps.

Per aquest motiu es proposa crear un sistema de monitoratge dels sistemes bàsics de funcionament d'un CPD com són l'electricitat, la temperatura, la humitat i l'estat dels SAIs i del grup electrogen.

Per fer-ho utilitzarem un equip estàndard de IoT que s'encarregarà de rebre les diferents lectures dels sensors o les notificacions dels mateixos equips per poder-los enviar a un servidor en el Cloud que ens garantirà el continu funcionament de les notificacions cap als interessats independentment del estat del nostre CPD.

1.2. Objectius del Treball

- Construir un dispositiu que permeti monitorar l'alimentació elèctrica, temperatura, humitat i estat dels SAIs.
- Enviar les dades recollides pels sensors al cloud.
- Emmagatzemar les dades en un sistema amb escalable i redundant.
- Tractar les dades rebudes i enviar senyals d'avis segons les alertes preconfigurades.
- Mostrar les dades recollides per poder conèixer l'històric d'informació.

1.3. Enfocament i mètode seguit

Per implementar la recollida de les dades que volem analitzar pel nostre sistema de monitoratge haurem de capturar les dades de temperatura, humitat i estat de funcionament dels SAIs, grup electrogen i escomesa general.

Per capturar les dades de temperatura utilitzarem un sensor DHT11 que ens permetrà captura la temperatura i la humitat de l'espai on es trobi. Per capturar el funcionament del SAI serà senzill perquè podem realitzar-li

consultes SNMP per saber l'estat en què es troba. El problema principal el tindrem per detectar l'estat del grup electrogen i de l'escomesa general perquè no tindrem cap manera de connectar-nos a ells per consultar el seu estat. Per això construirem un petit circuit que ens permetrà saber si el circuit al què esta connecta està subministrant corrent o no. Aquest sensor l'haurem de connectar a l'entrada de l'escomesa general i a la sortida del generador elèctric per distingir d'on s'estan alimentant els SAIs.

La sortida del SAI no veiem necessari el seu monitoratge perquè els sensors estaran alimentats per la mateixa alimentació del SAI i serà el propi cloud que notificarà que no esta rebent informació dels diferents sensors.

Pel que farà la infraestructura del cloud la basarem amb alguna de les tres solucions líders en el mercat com són Microsoft, Amazon o Google.

Dintre d'aquesta plataforma ubicarem el software de recollida d'informació que la basarem en una solució basada en Elasticsearch, Kibana, Logstash i Filebeat. Filebeat ens permetrà recollir la informació que generen els sensor des dels seus logs per enviar-los a Elasticsearch o Logstash. Logstash ens permetrà separar i seleccionar les dades que rebem i que ens interessin i formatar-les en el format adequat per poder-les treballar correctament. Elasticsearch ens permetre recollir les dades dels diferents sensors amb una infraestructura propia que permet un facil emmagatzematge, creixement i distribució de les dades entre diferents nodes de forma automàtica. Finalment, utilitzarem Kibana per presentar les dades recollides pels sensors.

Per poder implementar aquest projecte amb aquetes caracteristiques haurem de contemplar una part d'electrònica dels sensors, de programació dels sensors i recollida de dades i finalment la part d'infraestructura al cloud i de sistemes d'Elasticsearch.

1.4. Planificació del Treball

Activitat	18-feb	25-feb	04-mar	11-mar	18-mar	25-mar	01-abr	08-abr	15-abr	22-abr	29-abr	06-mai	13-mai	20-mai	27-mai	Juny
Planificació del projecte			06-mar													
Disseny del dispositiu: Alimentació elèctrica																
Disseny del dispositiu: Sensor climatològic																
Disseny de la infraestructura Cloud																
Generació scripts Cloud																
Implementació Logstash i Elasticsearch																
Disseny del grafisme																
Implementació Kibana																
Implementació de les alertes																
Redacció de la memòria: Part 1																
Entrega 1										24-abr						
Redacció de la memòria: Part 2																
Revisió del document																
Entrega 2														22-mai		
Realitzar correccions																
Entrega memòria TFM																09-jun
Realitzar presentació																
Entrega presentació																16-jun
Presentació del TFM																23-jun

Taula 1: Planificació del Projecte

1.5. Breu sumari de productes obtinguts

Els producte que obtindrem serà un circuit electrònic connectat a una Raspberry per poder capturar les dades dels sensors i enviar-les al cloud i una infraestructura al cloud que ens permetrà processa la informació i generar alarmes i un històric dels sensors per poder-ho reportar als interessats.

1.6. Breu descripció dels altres capítols de la memòria

2. Estat de l'art: es realitzarà una explicació de la solució que es vol implementar i es fa un breu resum de les diferents eines que hi ha al mercat per implementar la nostra solució.

3. Disseny i construcció del dispositiu: En aquest capítol mostrarem el circuit amb el qual es recolliran les dades i la programació que hi ha darrere perquè es generin els logs que s'enviaran al servidor finalment. A més, inclourà la programació del servidor SNMP per recollir les dades del servidor.

4. Disseny i implementació de la infraestructura al cloud: presentarem la solució del cloud per emmagatzemar el nostre sistema de recollida de dades en algunes de les infraestructures cloud existents en el món actualment.

5. Desplegament i implementació dels equips del cloud: es configuraran els sistemes que formen la infraestructura cloud per presentar la màxima resistència i autoconfiguració possible en cas de canvis o desastres.

6. Disseny de la implementació gràfica: el sistema tindrà un servei d'històric amb el qual podrem consultar els diferents esdeveniments que s'han produït en el sistema en els darrers mesos per si es volguessin cercar patrons o conèixer la disponibilitat del sistema.

7. Implementació de les alertes: el sistema enviarà alertes per correu electrònic quan es produeixi algun esdeveniment programar amb anterioritat per avisar-nos que algun dels paràmetres està fora del que seria correcte.

2. Estat de l'art

2.1. Disseny funcional

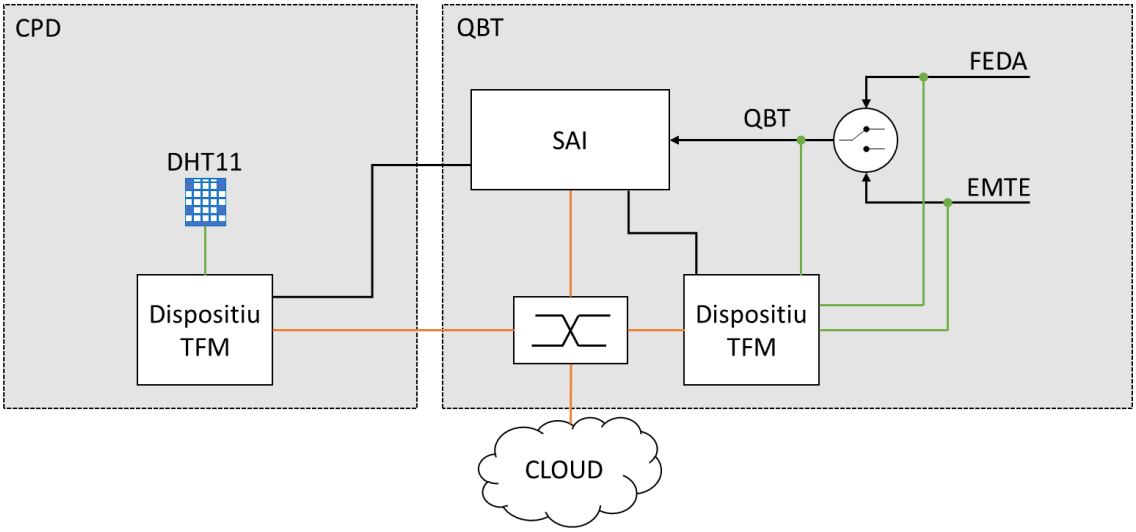

Figura 1: Disseny abstracte de la solució.

L'objectiu del treball és aconseguir tenir un dispositiu en cada una de les sales que volem monitorar per conèixer algun dels estats dels sensors que disposa el nostre dispositiu. En el cas d'aquesta empresa, haurem d'ubicar un dispositiu en la sala de baixa tensió (QBT) i en el CPD. En cada un dels llocs, podrem conèixer diferents estats dels equips. En el cas dels sensors de temperatura i de tensió, estarem limitats pel nombre de pins del dispositiu.

En la sala de baixa tensió de l'empresa disposem del quadre de commutació entre l'escomesa de la companyia elèctrica (FEDA) i el grup electrogen (EMTE). El problema que teníem en aquest punt era que aquest quadre de commutació no ens permetia saber per quina de les dues fonts de corrent elèctric ens arriba el corrent que consumim. Per aquest motiu, necessitem alguna forma de saber d'on està arribant el corrent.

Per solucionar el problema de detectar d'on prové el corrent, inicialment es va optar per unes bobines que detectessin la intensitat però per la potència contractada no hi havia cap dispositiu amb un cost raonable que és pogués connectar a algun dels dispositius que volíem utilitzar.

L'alternativa que vam trobar per detectar la font de corrent era mesura la tensió de cada una de les fonts. El quadre de commutació de baixa tensió no permet que les dues fonts corrent estiguin connectades alhora. D'aquesta manera, quan el grup electrogen té tensió, voldrà dir que estem

funcionant amb el grup electrogen i quan no en tingui, voldrà dir que estem treballant amb el corrent de la companyia. Tot i això, per si hi hagués algun problema en el quadre de commutació, també mesurem el voltatge de sortida del quadre i el d'entrada de la companyia elèctrica.

L'altre element que volem controlar en la sala de baixa tensió seràn els SAIs. Dels SAIs volem saber la carrega de les seves bateries, el consum que tenim i el mode de funcionament en què es troben. La sort que tenim en aquests dispositius es que ells mateixos disposen d'un mòdul de monitoratge SNMP amb connectivitat Ethernet. D'aquesta manera podrem conèixer l'estat del SAI realitzant consultes a aquest modul mitjançant el protocol de comunicació SNMP.

L'alimentació elèctrica del CPD no té cap fase intermèdia entre el SAI i els servidors. D'aquesta manera, no ens és necessari prendre noves mostres de l'electricitat. En el CPD el que ens interessarà detectar les condicions climàtiques que té l'espai. Les més importants són la humitat i la temperatura. En aquest cas, la forma més senzilla de poder-ho controlar és mitjançant una sonda que ubicarem en l'espai. La sonda climàtica més econòmica que trobem al mercat és el DHT11 que la podrem connectar a la majoria de dispositius de programables existents del mercat per recollir les dades. El principal obstacle per implementar aquesta solució serà que haurem d'evitar que el sensor mesuri la calor que dissipa el nostre dispositiu.

3.2. Elecció del dispositiu

La solució que es planteja en aquest treball es basa en Internet of Things, per tant, seguint amb aquesta filosofia, el nostre equip haurà de ser de petites dimensions, amb baix consum d'energia i autogestionable. Sent un treball universitari, les solucions que trobem en el mercat disponible i estandards serien únicament Arduino i Raspberry. Ambdues opcions permeten la inclusió de circuits i la programació dels mateixos per executar diferents codis.

La principal dificultat d'Arduino és que està pensat per realitzar treballs de prototipatge i per la creació de circuits electrònics que desencadenen accions segons les entrades i sortides que té. A més, Arduino realitza una execució seqüencial del codi i utilitza un codi de programació propi. Finalment, el cos de les solucions d'Arduino que hi ha el mercat i que permet la comunicació amb elements externs són més cares que la Raspeberry Pi.

Pel que fa a la Raspberry Pi podrem tenir diferents fils d'execució, permeten que un mateix dispositiu realitzi mesures de diferents sensors alhora. Podrem programar en qualsevol llenguatge de programació i la base de funcionament és un sistema operatiu. Per aquest motiu, en aquest treball, realitzarem el treball amb un Raspberry Pi de tercera generació.

3.3. Elecció de la infraestructura Cloud

En aquests moments els principals líders de la computació en el cloud són Google, Microsoft i Amazon. Tot seguit fem un estudi de les diferents possibilitats que ens donen aquests tres líders. La gran flexibilitat d'aquestes solucions és la possibilitat de disposar d'una infraestructura il·limitada pel teu negoci, sense tenir costos inicials, pagament per ús i facturació per minuts. Per ordre d'aparició tenim AWS, Azure i Google Cloud Platform.

Google disposa de la seva infraestructura anomenada Google Cloud Platform. Està pensada pel treball en contenidors basats en el seu propi estàndard de Kubernetes. La principal contra d'aquesta infraestructura és que té un catàleg de serveis limitats.

Amazon disposa de la seva infraestructura anomenada AWS. Va ser un dels primers a oferir aquests serveis i actualment s'ha convertit en el líder del mercat i el que ofereix una gama de serveis més amplia. AWS té com a referència la capacitat de computació que ofereix mitjançant les seves instàncies EC2. La seva gran debilitat consisteix en la gestió dels costos de funcionament de la infraestructura.

Microsoft disposa de la seva infraestructura anomenada Azure. El gran avantatge que ofereix és la integració que ofereix quan s'implementen aplicacions basades en entorns Windows. Partint dels mateixos programes de Windows que ha passat al Cloud. El principal contra seria la falta de suport i documentació que existeix dels seus serveis.

Per aquest treball escollirem treballar amb el cloud d'AWS. Per totes les possibilitats que ofereixen els seus serveis i perquè és la infraestructura que més coneixem i hem treballat a nivell professional.

3.4. Elecció del sistema de monitoratge

SNMP

SNMP és el protocol de monitoratge de l'estat d'un dispositiu que predomina en les TIC en qualsevol de les versions que té. El principal avantatge que té aquest protocol és que la majoria dels dispositius

suporten alguna de les versions del protocol. El principal desavantatge que tenim és que la majoria d'aquests dispositius només tenen implementades les versions 1 i 2 que no incorporen seguretat. Aquest serà un dels principals desavantatges que trobarem en la nostra implementació perquè la informació ha de viatjar per internet. La segona dificultat que trobarem és que si volem realitzar consultes SNMP, haurem d'obrir els ports per cada un dels dispositius que tinguem o només escoltar missatges Trap.

SYSLOG

Syslog és un sistema que permet la recol·lecció de logs entre diferents equips que formen la xarxa. El principal avantatge que tenim és que centralitza en un únic punt totes les dades que generen els equips i permet recollir fins a 7 nivells de dades. El principal problema que tenim és que s'han d'establir primer de tot els nivells de dades que recol·lectem i segon que haurem de trobar aplicacions que generin aquest logs.

Beats

Beats és un agent que envia informació de diferents tipus de fonts a Elasticsearch directe o passant per un preposat de la base de dades fet per Logstash. Beats té diferents implementacions com són:

- Filebeat: recollida centralitzada de logs
- Metricbeat: recollida de l'estat dels equips.
- Packetbeat: escolta dels paquets de la xarxa.
- Winlogbeat: recollida dels esdeveniments de Windows.
- Auditbeat: recollida dels esdeveniments de Windows.
- Heartbeat: monitoratge de serveis.

Pel nostre treball, utilitzarem Filebeat que ens permetrà recollir els logs de les nostres aplicacions i enviar-les als servidors per tractar-les i emmagatzemar-les. Per tractar-les, el sistema ELK disposa de Logstash, permet tractar les dades de forma que només tinguem les dades que ens interessa guardar i en el format desitjat. Aquestes dades processades seran guardades en Elasticsearch i mitjançant Kibana podrem veure la informació gràficament.

3. Disseny i construcció del dispositiu

3.1. Preparació del dispositiu

Per desenvolupar el projecte treballarem amb una de les plaques més utilitzades en aquest moment en el disseny de solucions IoT i que executa un sistema operatiu basat en UNIX. Aquest dispositiu serà un Raspberry Pi 3 Model B+.

La Raspberry Pi 3 Model B+ és el darrer model disponible en el mercat, compte amb un processador ARM de 64 bits amb 4 nuclis a 1,4 GHz i 1GB de memòria RAM DDR2. A més, de tenir múltiples solucions de connectivitat com són Ethernet PoE 1Gbps, connexió sense fil 802.11ac a 2,4 GHz i 5 GHz i Bluettoth 4.2

Hem seleccionat aquest dispositiu perquè ens permet l'execució d'un sistema operatiu que ens permetrà poder treballar amb diferents llenguatges de programació desacoblats de la placa en qüestió i poder executar de forma concurrent més d'una aplicació. A més, disposa de diferents pins analògics que ens serviran per rebre les dades digitals dels circuits digitals que implementarem i connectarem a ella.

El dispositiu que generarem serà estàndard per les diferents mesures que volem obtenir. Per tant, amb un únic dispositiu haurem de tenir totes les aplicacions i configuracions necessàries per cada un del tipus de dades que volem recollir ja sigui mitjançant algun dels sensors que disposarà el circuit electrònic o recollint la informació de la xarxa mitjançant SNMP o qualsevol altre protocol que treballin els equips als quals escoltarem el seu estat.

Abans de posar-nos a explicar els diferents programes que formaran part de la nostra solució per aquest projecte, carregarem i configurarem el sistema operatiu. El sistema que hem seleccionat per aquest projecte serà la versió més lleugera de Raspbian i a poder ser sense versió gràfica. Des de la propia web de Raspberry podem descarregar la imatge del sistema operatiu. Alhora, podrem consultar les diferents guies per descomprimir el sistema operatiu i carregar-lo en el disc dur. Aquest dispositiu utilitza com a disc dur una targeta microSD.

En el nostre cas, utilitzem un sistema operatiu Windows. Però com ja hem dit, podem trobar en la mateixa web qualsevol de les guies per instal·lar-lo des de qualsevol sistema operatiu domèstic. Un cop tenim la imatge descarregada a l'ordinador, haurem de tenir a l'ordinador el programa

Wind32DiskImage que s'encarregarà d'agafar la imatge .img i descomprimir-la en la targeta SD.

Un cop hàgim finalitzat la descompressió del sistema operatiu podem posar la targeta en la Raspberry i connectar a l'electricitat. La placa es pot alimentar mitjançant un micro-USB o PoE. En el nostre cas, l'entorn de test i preproducció el farem per USB però per l'entorn de producció l'alimentarem per PoE.

En iniciar-lo per primer cop, ens demanarà de realitzar una breu configuració del sistema operatiu com serà l'idioma del teclat, la zona horària, l'idioma del sistema operatiu, canviar la contrasenya i actualitzar-lo. Per agilitzar aquesta primera fase de configuració, no realitzarem l'actualització del sistema operatiu i l'usuari i contrasenya serà l'estàndard per aquest dispositiu (pi / Raspberrry). L'objectiu que tenim és tenir el dispositiu configurat per poder-lo continuar configurant i preparant per connectar-nos per ssh l'abans possible.

Utilitzar l'usuari i contrasenya pi/Raspberry és un error de seguretat molt greu però la idea és substituir aquest usuari i contrasenya per un inici de sessió mitjançant certificats de clau publica i privada que aviat explicarem.

Un cop s'hagi reiniciat el dispositiu per aplicar els canvis en la configuració, haurem d'assegurar-nos que el servei ssh està habilitat. Per revisar-ho, executarem la comanda `service ssh status`. Si el servei indica que està deshabilitat, haurem d'activar-lo amb les següents comandes `systemctl ssh enable` i `systemctl ssh start`, perquè s'habiliti cada vegada que s'inicia el sistema operatiu i per iniciar-lo ara, respectivament. Si per contra, apareix com que el servei no existeix, haurem d'instal·lar-lo amb la comanda `sudo apt-get install ssh` i posteriorment executar les dues comandes anteriors.

Finalment, haurem de canviar la configuració de xarxa per tenir una IP estàtica en el dispositiu. En les darreres actualitzacions dels sistemes UNIX han canviat el mètode de configuració. Per aquest motiu, explicarem els dos sistemes de configuració.

El mètode antic, tenim l'arxiu que haurem de modificar a `/etc/network/interfaces` i haurem de configurar-ho de la següent forma amb el nostre adreçament.

```
auto eth0
iface eth0 inet static
      address 192.168.1.50
      netmask 255.255.255.0
```

```
network 192.168.1.0
broadcast 192.168.1.255
gateway 192.168.1.1
```

Si quan intentem accedir en aquest arxiu ens indica que s'ha traslladat a una nova ubicació, voldrà dir que tenim el nou sistema de configuració de xarxa i que té una estructura yml. Aquest arxiu de configuració, el trobarem a `/etc/netplan/`.

```
network:
  version: 2
  renderer: networkd
  ethernets:
    enp0s3:
     dhcp4: no
     addresses: [192.168.1.222/24]
     gateway4: 192.168.1.1
     nameservers:
       addresses: [8.8.8.8,8.8.4.4]
```

Per aplicar la nova configuració en el mètode antic, haurem de reiniciar l'ordinador i en el mètode nou, podrem executar la comanda `sudo netplan apply` o reiniciar l'equip.

Un cop aplicada aquesta configuració, ja podem accedir mitjançant ssh des de qualsevol equip. Ara ja podem realitzar l'actualització del sistema operatiu i serveis a la darrera versió amb les comandes `sudo apt-get update` i `sudo apt-get upgrade`.

Per la programació del software que utilitzarem per al projecte utilitzarem python 3 amb algunes llibreries complementàries que haurem d'instal·lar però les anirem instal·lant a mesura que les anem utilitzant en els diferents codis que utilitzem.

Finalment, per enviar les dades, haurem d'instal·lar el software filebeat que agafarà els diferents logs que estiguem generant en els diferents programes que haurem programat. El problema que tenim amb aquest programa és que no està disponible en els repositoris de Raspbian i per tant, serà necessari que el compilem pel nostre Kernel per poder-lo utilitzar.

Abans de començar a explicar els diferents programes que hem construït i la compilació de filebeat, assegurarem la comunicació ssh per deixar de treballar amb l'usuari pi. Primer de tot haurem de crear un nou usuari, per exemple, `TFM` i li donarem permisos d'administrador (root), haurem de crear un parell de claus: publica i privada (id_rsa.key i id_rsa.pub) i finalment desactivar l'ús de contrasenyes per l'accés per ssh.

Per aplicar aquest canvi en la configuració, haurem d'utilitzar els següents jocs de comandes.

```
sudo adduser TFM
sudo nano /etc/sudoers
```

En aquest arxiu, haurem d'afegir sota `#User privilegi specification`, la següent línia.

```
TFM   ALL=(ALL) ALL
```

Amb això, ja tindrem el nostre usuari creat i amb permisos d'administrador. Tanquem la sessió amb l'usuari pi i entrem de nou per ssh amb l'usuari TFM i ja podem eliminar l'usuari PI. Per eliminar-lo, utilitzarem la comanda:

```
sudo deluser -remove-home pi
```

Per assegurar la comunicació i el login amb el parell de claus, les haurem de generar amb el PC que volem utilitzar per gestionar les Raspberry. Pot ser algú més complicat crear-les amb Windows que no pas amb Linux. Per fer-ho, haurem d'utilitzar el programa Puttygen. Hi haurem de generar una clau SSH-2 RSA de com a mínim 2048 bits. D'aquí generem la nostra clau pública i la nostre clau privada. La clau publica, l'haurem d'exportar i enviar-la a la Raspberry. Dins de la Raspberry, haurem d'ubicar la clau pública dintre de la carpeta personal ~/.ssh/authorized_keys.

Amb això ja podem entrar via ssh tant per usuari i contrasenya com per usuari i clau publica privada. Amb aquesta opció seguim tenint el problema que la contrasenya és més dèbil i l'haurem de desactivar des de l'arxiu de configuració del servei ssh. Per fer-ho, haurem d'editar l'arxiu `/etc/ssh/sshd_config` i descomentar la línia `PasswordAuthetication no` i reiniciar el servei. Amb aquesta acció, aconseguim que per ssh només es pugui accedir amb certificat i la contrasenya ens servirà per entrar com a sudoer o si en algun moment hem d'accedir per teclat i pantalla a la Raspberry.

Amb això ja tindríem el sistema operatiu preparat per funcionar i podem instal·lar la resta de software i aplicacions que utilitzarem.

3.2. Circuit del sensor de temperatura i humitat

Per realitzar aquest sensor, utilitzarem el sensor més comú en electrònica digital per aquests tipus de mesures que és el sensor DHT11. El sensor transforma les dades de temperatura i humitat en una codificació digital que mitjançant una llibreria els rebríem en la nostra aplicació que

programarem a continuació. Aquest és un dels programes més senzills que utilitzarem. Consisteix en un petit codi de python junt amb la llibreria que instal·larem del mateix nom del dispositiu que ens farà automàtica la traducció de les dades binàries a decimals. Un cop tinguem les dades decimals, només les haurem de guardar en un arxiu de logs que per això utilitzarem la llibreria de python syslog.

El circuit que utilitzarem serà el següent:

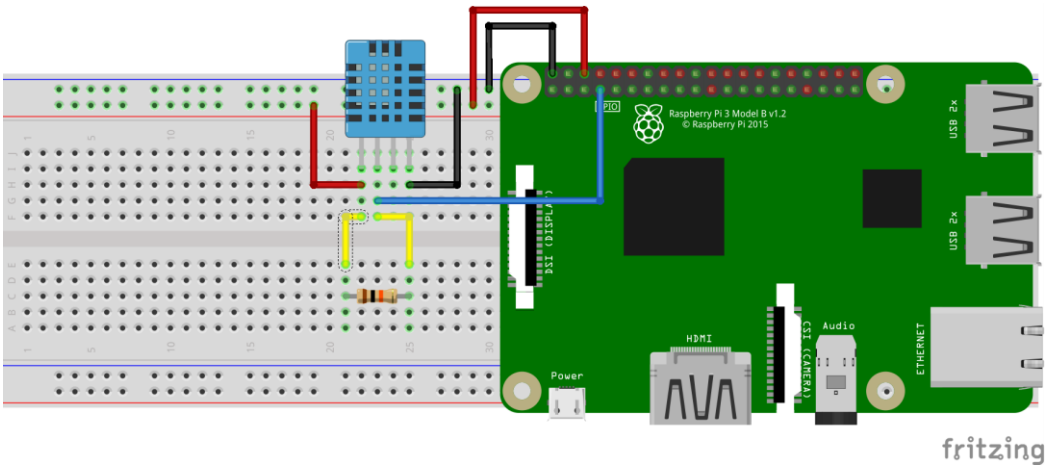

Figura 2: Circuit amb el sensor de temperatura i humitat.

Aquest circuit el connectarem al pin 2 que ens donarà l'alimentació del dispositiu, el pin 6 que serà el neutre i finalment el pin X que serà els que ens donarà les dades. La resistència que utilitzem té una resistència de $10K\Omega$. Aquest darrer pin serà un dels pins que disposa la Raspberry Pi per utilitzar com entrada o sortida de dades. Aquest darrer PIN serà necessari indicar-lo en l'arxiu de configuració de les diferents aplicacions que tindrem. Aquest fitxer de configuració s'estructura amb format json per una millor interpretació i modificació de la configuració dels equips.

3.3. Circuit sensor de presencia de tensió

Per dissenyar aquest circuit, utilitzarem un circuit integrat CPL3700. Aquest circuit ens permetrà detectar si hi ha un voltatge entre dos pols. Aquest dispositiu, es connecta en un extrem del circuit integrat la corren alterna de 230 volts i en l'altre el corrent continu de la Raspberry que ens activarà o desactivarà el PIN on es detecta si hi ha voltatge de 230 v o no.

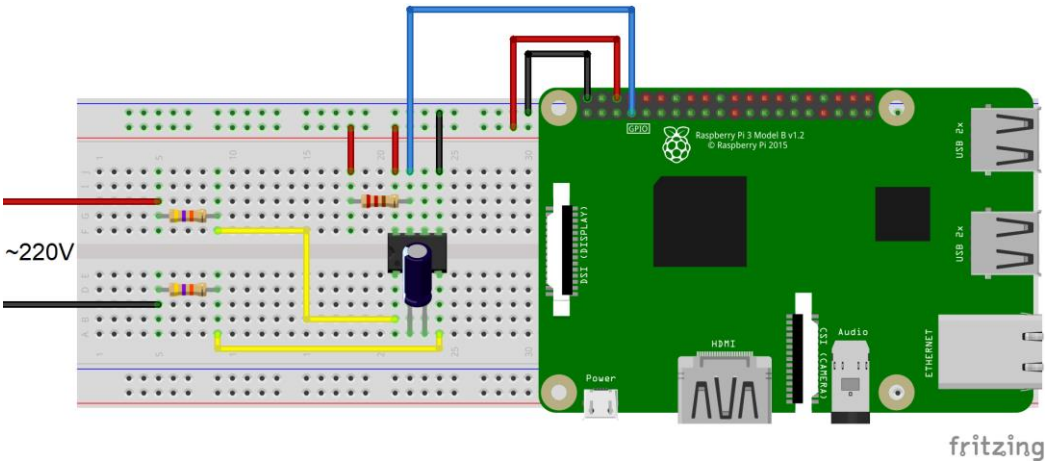

Figura 3: Circuit amb el sensor de tensió.

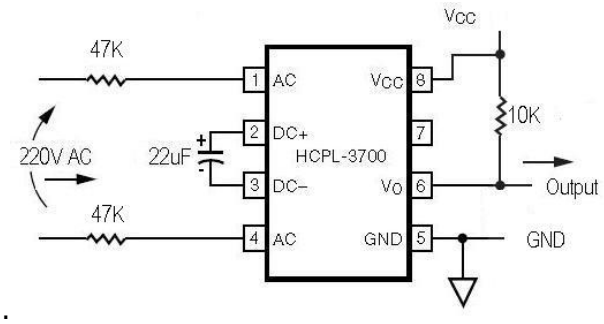

Figura 4: Esquema de conexions del circuit integrat CPL3700.

Internament aquest circuit converteix el corrent alterna a corrent continua per excitar un díode i aquest excita un sensor de llum aïllat de corrent continua que ens permetrà activar o desactivar el flux de corrent entre diferents díodes.

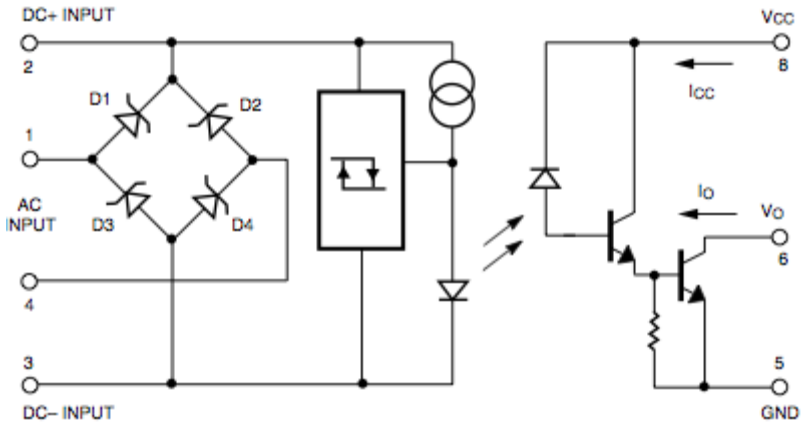

Figura 5: Esquema de funcionament intern del circuit integrat CPL3700.

3.4. Programa sensor de temperatura

DHT11 té una codificació especifica binaria que codifica les dades del sensor en un sol input. Per simplificar aquesta tasca, es pot instal·lar la llibreria dht de python. Per instal·lar-lo en el nostre equip, utilitzarem les següents dues comandes:

```
git clone https://github.com/adafruit/Adafruit_Python_
DHT.git
cd Adafruit_Python_DHT
sudo python3 setup.py install
```

El codi que utilitzarem serà el següent:

```
import Adafruit_DHT
sensor=Adafruit_DHT.DHT11
gpio=7
humidity, temperature = Adafruit_DHT.read_retry(sensor,
gpio)
if humidity is not None and temperature is not None:
print('Temp={0:0.1f}*C
Humidity={1:0.1f}%'.format(temperature, humidity))
else:
print('Failed to get reading. Try again!')
```

Els logs que genera cada una de les aplicacions que estem utilitzant seran emmagatzemats en la mateixa carpeta de logs del sistema, en la capeta /var/log/tfm/.

3.5. Programa sensor de presencia de tensió

Amb el circuit detector de tensió obtindrem un resultat que serà binari, és a dir, tindrem una tensió de 0 V o de 3,3 V (un 0 o un 1 binari). En aquesta primera versió només es considera la possibilitat de conèixer si hi ha tensió o no, no tenim en compte si aquesta tensió és correcta o està per sobre o per sota dels nostres llindars. El valor de tensió concret, el podrem obtenir fent una consulta SNMP al SAI, com veurem en el següent apartat.

Centrant-nos en el codi del sensor de tensió, primer de tot haurem de configurar els PINs que utilitzarem com a PINs INPUT. La configuració dels PINs que utilitzarem per a aquesta funcionalitat, l'obtindrem de l'arxiu de configuració /etc/tfm/config.json. Posteriorment, podrem començar periòdicament l'estat dels PINs per saber si hi ha tensió o no hi ha tensió en el nostre sistema.

El resultat que obtindrem en l'arxiu de logs seran

```
2019-05-12 10:22:06,628 INFO SensorPower1 is up
2019-05-12 10:22:06,630 INFO SensorPower2 is down
```

3.6. Programa receptor d'estat dels SAIs (Server SNMP)

El principal protocol d'informació de l'estat d'un equip a un servidor és SNMP. SNMP és un protocol de comunicació entre agent i servidor que el que fa és informar de l'estat d'un equip. Aquesta informació es pot consultar mitjançant consultes que realitza el servidor a l'agent o mitjançant missatges trap que envia l'agent al servidor quan es produeix un esdeveniment prefixat. L'agent SNMP està suportat per defecte en la majoria de dispositiu i si no es pot activar i configurar amb facilitat.

SNMP disposa de diferents versions que incorporen novetats en l'obtenció de les dades i l'encriptació i autentificació de les dades però la majoria dels dispositius encara suporten la versió 1 del protocol. En el nostre cas, realitzarem consultes molt concretes a la base de dades (MIB) dels SAIs per aquest motiu el nostre codi només implementarà la versió 1 del protocol SNMP. Per programar aquest codi treballarem amb Python 3.

El nostre programa serà l'encàrrec de funcionar com a servidor SNMP en mode consultes, és a dir, periòdicament, haurà d'enviar peticions als agents per saber si hi ha hagut algun canvi en l'estat del dispositiu.

En aquest treball, no s'explica la configuració de l'agent perquè cada sistema disposa del seu propi sistema de configuració segons el proveïdor dels SAI. En qualsevol cas, s'hauria d'activar SNMP amb la versió 1 amb la comunity que correspongui.

Un altre inconvenient que tindrem en treballar amb equips de diferent marca és que haurem de determinar quines són les O.I.D. i el valor que hauran de tenir aquestes per poder saber l'estat del SAI i la seva alimentació. Per això, haurem de conèixer el seu arbre de MIB. Per indicar les O.I.D. que llegirem, haurem de crear un arxiu json que li passarem tota la informació necessària perquè realitzi les diferents consultes SNMP.

En la Raspberry tindrem instal·lada la versió 2 i 3 de python. Això ens obligarà a estar atents a l'hora d'instal·lar possibles paquets en una versió o en una altra. Si instal·lem un paquet amb la funció pip install, aquest paquet s'instal·larà en la versió 2. Per instal·lar un paquet en la versió 3, utilitzarem la comanda python3 -m pip install *Package*.

Per la nostra funció snmp, utilitzarem el paquet pysnmp. Per instal·lar-lo en la versió 3. Utilitzarem la comanda `python3 -m pip install pysnmp`.

L'estructura de la configuració json de l'apartat SNMP serà la següent:

```
"snmp": [
  {
    "name": "snmpSensor1",
    "version": 1,
    "comunity": "PUBLIC",
    "ipAddress": "192.168.1.3",
    "oid": [
      ".1.3.6.1.2.1.1.1.0",
      ".1.3.6.1.2.1.1.3.0",
      ".1.3.6.1.2.1.1.5.0"
    ],
    "names": {
      ".1.3.6.1.2.1.1.1.0" : "SAIBatteryStatus",
      ".1.3.6.1.2.1.1.3.0" : "SAIEstimatedCharge",
      ".1.3.6.1.2.1.1.5.0" : "SAIBatteryCurrent"
    }
  }
]
```

En aquest apartat de l'arxiu de configuració podrem, incloure diferents equips per consultar i les diferents variables que volem consultar de la MIB. Primer de tot, haurem de definir el nom del sensor per quan es guardin els logs, la versió SNMP, la comunitat i l'adreça IP i finalment les diferents OID que volem consultar. Per cada una de les OID es generarà un registre en l'arxiu de logs amb la següent estructura.

```
2019-05-12 10:22:06,628 INFO {'SAIBatteryStatus': 'Normal'}
2019-05-12 10:22:06,630 INFO {'SAIEstimatedCharge': 100 }
```

El codi consisteix a llegir primer la configuració de l'arxiu de configuració, inicialitzar el sistema de log de les dades. De forma iterativa i periòdica, el sistema anirà realitzant consultes SNMP als diferents agents i OID que té configurats i els anirà escrivint en l'arxiu de configuració. En aquesta versió del codi només es donarà suport a la versió 1 de SNMP, tot i que la construcció del codi com l'arxiu de configuració estan pensats per poder incloure la resta versions i paràmetres de configuració.

3.7. Compilació de filebeat

Com ja hem dit a l'inici d'aquest capítol, el programa filebeat no està disponible en els repositoris d'aplicacions de Raspberry. Per aquest motiu, ha sigut necessari cercar quins requisits de software són necessaris per

poder-lo compilar i executar i descarregar el codi. A continuació, es descriu com compilar el codi i establir el programa com un servei de Linux.

Primer de tot, haurem d'instal·lar Golang perquè part del programari de beat està programat amb GO.

```
export golang_ver=$(curl https://golang.org/VERSION?m=text
2> /dev/null)
wget
https://storage.googleapis.com/golang/${golang_ver}.linux-
armv6l.tar.gz
sudo tar -C /usr/local -xzf ${golang_ver}.linux-
armv6l.tar.gz
```

Un cop tinguem Golang en el sistema operatiu, establirem les variables d'entorn necessàries per fer les crides de GO. Perquè les variables d'entorn estiguin disponibles sempre, haurem d'afegir-les al fitxer `/etc/profile`.

```
export GOPATH=$HOME/go
export PATH=$PATH:/usr/local/go/bin
export PATH="$GOPATH/bin:$PATH"
```

I també a l'arxiu ~/.bashrc.

```
export GOPATH=$HOME/go
export PATH="$GOPATH/bin:$PATH"
```

Perquè les variables d'entorn tingui efecte haurem de reiniciar el sistema operatiu. Un cop reiniciat, executarem la comanda `go version` per comprovar que funciona correctament Golang.

Com que haurem de descarregar el codi de git i posteriorment compilar-ho, confirmarem que disposem de totes les eines necessàries per fer-ho amb la comanda:

```
sudo apt-get install build-essential git -y
```

A més, haurem d'instal·lar python, pip, pyyaml i virtualenv. Aquest darrer ens servirà per crear un entorn aïllat d'execució de python.

```
sudo apt-get install python python-pip
sudo pip install virtualenv
sudo /usr/bin/easy_install virtualenv
sudo pip install pyyaml --upgrade
```

Amb aquestes darreres comandes, ja tenim tot el programari necessari per compilar filebeat. Primer de tot, crearem la carpeta de Go.

```
mkdir -p ${GOPATH}
```

Descarregarem magefile mitjançant el gestor de go.

```
go get github.com/magefile/mage
cd $GOPATH/src/github.com/magefile/mage
go run bootstrap.go
```

A continuació, creem un directori per descarregar el codi i descarreguem elàstics de git.

```
mkdir -p ${GOPATH}/src/github.com/elastic
cd ${GOPATH}/src/github.com/elastic
git clone https://github.com/elastic/beats.git
cd beats/
```

Un dels beneficis de treballar amb git és que tenim accés a infinitat de versions del mateix programari i podem seleccionar amb quina volem treballar. Per conèixer les versions disponibles, utilitzarem la comanda `git tag`. En el nostre cas, treballarem en la darrera versió que no es troba en fase de desenvolupament. Per treballar amb aquesta versió, haurem d'obtenir primer el hash de la commit a la qual pertany amb la comanda `git rev-list -n 1 versió`. El hash que obtindrem l'haurem d'incloure a la següent comanda:

```
git checkout hash
```

Finalment, ja podem compilar filebeat,

```
cd ${GOPATH}/src/github.com/elastic/beats/filebeat/
GOARCH=arm go build
make
make update
```

Com que les darreres versions de la Raspberry tenen processadors de 64 bits podríem substituir la compilació arm per arm64.

```
cd ${GOPATH}/src/github.com/elastic/beats/filebeat/
GOARCH=arm64 go build
make
make update
```

Un cop finalitza la compilació del codi, podem comprovar el funcionament de filebeat amb la comanda:

```
./filebeat -e -v
```

Amb aquesta acció, tenim el programa compilat i ja el podem executar. El problema que tenim és que encara no esta instal·lat com una aplicació o servei i, per tant, la seva gestió serà més complicada. Per establir filebeat com un servei de la Raspberry haurem de moure alguns arxius a les ubicacions del sistema.

```
sudo mkdir -p /usr/share/filebeat /usr/share/filebeat/bin
/etc/filebeat /var/log/filebeat /var/lib/filebeat

sudo mv filebeat /usr/share/filebeat/bin
sudo mv module /usr/share/filebeat/
sudo mv modules.d/ /etc/filebeat/
sudo cp filebeat.yml /etc/filebeat/
sudo chmod 750 /var/log/filebeat
sudo chmod 750 /etc/filebeat/
sudo chown -R root:root /usr/share/filebeat/*
```

Per crear el servei, haurem de crear l'arxiu del servei a `/lib/systemd/system/filebeat.service` amb les següents dades:

```
[Unit]
Description=filebeat
Documentation=https://www.elastic.co/guide/en/beats/filebea
t/current/index.html
Wants=userwork-online.target
After=network-online.target

[Service]
ExecStart=/usr/share/filebeat/bin/filebeat              -c
/etc/filebeat/filebeat.yml -path.home /usr/share/filebeat -
path.config /etc/filebeat -path.data /var/lib/filebeat -
path.logs /var/log/filebeat
Restart=always

[Install]
WantedBy=multi-user.target'
```

Amb aquest darrer arxiu, ja tenim filebeat configurat com a servei. Per iniciar-lo podríem utilitzar la comanda `service filebeat start` i si volem que s'inicia amb el sistema operatiu la comanda `systemctl enable filebeat.service`, ambdues amb permisos de superusuari.

3.8. Configuració de Filebeat

Per configurar Filebeat, haurem d'accedir a la carpeta filebeat.yml on haurem d'activar la recol·lecció de logs, la ruta dels logs i l'adreça d'Elasticsearch o Logstash que utilitzarem. Les línies concretes que haurem d'incloure seran.

```
- type: log
  # Change to true to enable this input configuration.
  enabled: true
  # Paths that should be crawled and fetched. Glob based
paths.
  paths:
    - /var/log/tfm/*.log
```

Segons si volem enviar les dades a Logstash o directament a Elasticsearch haurem d'indicar-ho en les següents línies de configuració. Filebeat només permet enviar les dades a un dels dos destins, si no, el servei no iniciarà.

```
output.elasticsearch:
  # Array of hosts to connect to.
  hosts: ["localhost:9200"]

output.logstash:
  # The Logstash hosts
  hosts: ["localhost:5044"]
```

3.9. KeepAlive i purgat de logs

Un dels riscos que podem tenir a l'hora de rebre les dades al servidor del cloud és que l'aplicació local hagi deixat de funcionar per algun motiu. Per això, es disposa d'un procés que anomenem keepAlive que s'executa cada minut i ens serveix per comprovar que les aplicacions tfmsnmp.py, tfmpower.py i tfmtemp.py segueixin funcionant. Si l'aplicació detecta que alguna de les tres aplicacions ha deixat de reportar la informació a Filebeat, aquest procés s'assegura de finalitzar el procés anterior si encara estava en execució i inicia un nou procés d'execució.

El mecanisme d'aquesta aplicació és molt simple. Les tres aplicacions que presentem en aquest treball escriuen amb certa periodicitat en els fitxers de logs. Per això, aquesta aplicació es dedicà a controlar que aquests arxius de logs s'estan modificant amb la periodicitat que els hi pertoca dins de la carpeta /var/log/tfm.

Aquest procés està implementat dintre de les taules de cron del mateix sistema operatiu de la Raspeberry Pi i programat sota llenguatge bash de UNIX. El programa estarà ubicat junt amb la resta del codi en la carpeta /etc/tfm/keepAlive.sh.

Els programes anteriors estaven dins d'un bucle d'un True, en execució continua, però amb un temps de pausa determinat per reduir el nombre de consultes i logs a les nostres necessitats. Aquesta iteració continua, és

un risc en cas de quedar-se alguna part del codi que inclòs esperant alguna resposta. Un dels motius per existir la funció KeepAlive és aquest.

El sistema operatiu de forma periòdica s'encarrega d'executar el codi de KeepAlive mitjançant la taula cron. La taula cron és un servei del mateix sistema operatiu que executa de forma periòdica les aplicacions que té programades. Per defecte, aquest procés està configurat per executar-se cada hora, dia, setmana o mes però si volem establir una periodicitat més concreta, podrem incloure-la a la taula crontab que trobarem a /etc/crontab. La taula cron d'aquest fitxer, permet configurar accions amb un mínim d'execució 1 minut.

Per configurar aquest procés haurem d'incloure la següent línia de codi l'arxiu /etc/crontab.

```
echo "* * * * * /bin/bash /opt/tfm/keepAlive.sh " >> /etc/crontab
```

Aquest mateix procés serà l'encarregat d'iniciar automàticament la resta d'aplicacions quan s'iniciï el dispositiu. Aquests programes s'executaran en screens diferents per poder revisar en tot moment l'estat de les aplicacions i poder-les revisar si fos necessari.

Una screen és una connexió independent al ordinador des del mateix ordinador amb la qual podem executar programes en primer pla sense afectar a la resta d'execució en primer pla que poden haver-hi en altres pantalles.

En el procés bash primer comprovarem que l'arxiu de logs s'hagi modificat en el darrer minut i, en cas contrari, primer tancarà la screen anterior i després iniciarà una nova screen amb el programa en qüestió.

```
if [ ! -f "/var/log/tfm/snmp.log" ]
then
        screen -X -S tfmsnmp quit
        screen -d -S tfmsnmp -m python3 /opt/tfm/snmp.py
else
        FILES=`find "/var/log/tfm/snmp.log" -mtime -0.000695 | wc
-l`
        if [ "$FILES" == "0" ]
        then
                screen -X -S tfmsnmp quit
                screen -d -S tfmsnmp -m python3 /opt/tfm/snmp.py
        fi
fi
```

En aquest codi, primer comprovem si no existeix l'arxiu i s'inicia la screen on tindrem el codi. En cas d'existir l'arxiu de logs, comprovem si aquest

s'ha modificat el darrer minut i en cas contrari tanquem la screen i iniciem la nova screen amb el codi en execució. El programa avalua per separat cada un d'ells, d'aquesta manera, si falla només el programa tfmsnmp.py, els altres dos programes no caldrà iniciar-los.

En el cloud podrem comprovar si alguna d'aquestes aplicacions es reinicien amb certa periodicitat i poden generar un risc en les dades. Les aplicacions, quan s'inicien, escriuen en el log informació conforma s'acaben d'iniciar.

Un de les altres modificacions que s'ha d'incloure és el purgat dels logs que generen les aplicacions per mantenir-les en tot moment dins d'una mida operable pel sistema operatiu i que no acabin omplint l'espai d'emmagatzematge. Per configurar aquesta purga, haurem de crear un arxiu dins de la carpeta logrotate.d on indicarem la configuració de la purga.

```
/var/log/tfm/snmp.log{
        daily
        rotate 10
        copytruncate
        delaycompress
        compress
        notifempty
        missingok
}

/var/log/tfm/temp.log {
        daily
        rotate 10
        copytruncate
        delaycompress
        compress
        notifempty
        missingok
}

/var/log/tfm/power.log {
        daily
        rotate 10
        copytruncate
        delaycompress
        compress
        notifempty
        missingok
}
```

4. Disseny i implementació de la infraestructura al cloud.

4.1. Introducció

Un cop hem determinat en l'estat de l'art que la infraestructura del Cloud que utilitzaríem en el treball seria la infraestructura AWS d'Amazon. En aquest capítol ens disposem a explicar com s'estructurarà el nostre cloud, la connectivitat que hi haurà interna i externa, la seguretat i les instàncies amb les que treballarem per fer funcionar el nostre servei ELK.

La infraestructura que es presenta, només comptarà amb una sol instància per programa ELK que s'executi però la idea amb la qual s'ha dissenyat la infraestructura és que des del principi es pugui desenvolupar per si en un futur es volen implementar nous nodes i noves funcionalitats ELK al nostre sistema i pugui escalar-se de forma fàcil perquè la infraestructura que la suportarà ja estarà preparada.

AWS ens permet treballar amb diferents regions del món on ubicar els nostres serveis. En alguns casos, hi ha certes diferències entre regions del món perquè tenen un procés d'actualització i incorporació de nous serveis que no és homogeni en tot el món però en el nostre cas, aquesta condició és irrellevant per les eines que volem implementar. Per aquest motiu, qualsevol regió serà valida per ubicar els nostres serveis d'ELK.

La infraestructura inicial i la que presentem en aquest treball té com a principal punt d'error que per cada aplicació ELK només disposem d'una única instància que hi treballi però no és abast del projecte perquè la càrrega de dades amb què treballarem serà suficient treballar amb una única instància per aplicació. Ens plantejarem ampliar el nombre d'instàncies ELK quan necessitem garantir que les dades d'Elasticsearch no es perdin si falla un node, si el nombre de dades a operar i consultar per ELK no es pot gestionar amb un únic node, si augmenten les necessitats de processament de dades que realitza Logstasth o si necessitarem que Kibana pugui augmentar el nombre de respostes web.

Quan es produeixin algun d'aquests escenaris serà quan de veritat podrem aprofitar el rendiment i les solucions d'ELK per distribuir la nostra informació i la capacitat de processament de les dades i alhora donar una redundància als nostres serveis.

4.2. Comunicació

Com en la majoria dels sistemes d'avui en dia, ha d'existir una comunicació entre els diferents equips. Dintre del cloud aquesta comunicació és gestionar per una Virtual Private Cloud (VPC) com si fos una xarxa LAN que comunica totes les nostres instàncies. Aquesta VPC es valida per tota la regió en la qual estiguem realitzant el projecte però dintre tenim 3 zones diferents. Aquestes zones corresponen a CPDs independents d'AWS que té en aquella regió, de forma que si una CPD falles les altres dues zones podrien seguir treballant sense problema.

En aquest projecte, necessitem tenir connectivitat en tot moment en les nostres aplicacions ELK, per això serà convenient ubicar totes aquestes aplicacions dintre de la mateixa zona. La zona no se selecciona quan creem la VPC, si no quan creem les subxarxes que en formaran part. Les subxarxes que hem creat són la 192.168.0.0 (Kibana), la 192.168.1.0 (Logstash) i la 192.168.2.0 (Elasticsearch), cada una de classe C.

Les zones les podrem utilitzar quan treballem amb més d'un node, d'aquesta manera si caigués alguna zona, la carrega de treball la podrien assumir les instàncies que s'estarien executant en una altra zona. El tema de les zones el comentarem en més detall quan s'expliqui el balanceig de càrrega.

La comunicació que s'efectuarà dintre del nostre Cloud seguirà el següent gràfic:

Figura 6: Esquema de comunicació entre aplicacions ELK.

La Raspberry Pi mitjançant l'aplicació Filebeat enviarà les dades al nostre cloud a l'aplicació Logstash. Un cop Logstash hagi processat les dades, les emmagatzemarà a Elasticsearch. Per l'altra banda, quan un usuari volguí consultar les dades que hi ha emmagatzemades a Elasticsearch, aquest realitzarà una consulta a Kibana que alhora consultarà les dades a Elasticsearch per poder-les mostrar.

4.3. Grups de seguretat

Un cop coneixem el nostre model de comunicacions entre les diferents instàncies que formaran part del nostre cloud, és a l'hora de definir les

regles de seguretat amb les que treballarem. A AWS aquestes regles de seguretat reben del nom de Security Groups.

Els Security Groups són els encarregats de determinar quina comunicació entrant o sortint es permet entre cada grup d'instàncies, serveis o rangs d'IPs que treballen en el nostre cloud. En el nostre cas, haurem de generar tres Security Groups per cada tipologia de màquina que tindrem en el nostre sistema. Els Security Groups que configurarem seran: SG-Kibana, SG-ElasticSearch i SG-Logstash.

Els security groups que tindrem seran els següents.

SG-Logstash			
Tipus	Protocol	Port	Origen
Custom TCP Rule	TCP	5044	0.0.0.0/0

Taula 2: Security Group de les instàncies de Logstash.

El SG-Logstash haurà d'incloure tots els ports que tingui oberta l'aplicació per rebre els missatges de Filebeat. En aquest cas, hem creat una regla genèrica que permeti tot el transit que provingui de qualsevol adreça IP del món però en producció podem indicar l'adreça IP de la qual s'enviaran les dades si comptem amb una adreça IP estàtica.

Si per contra, l'adreçament IP fos dinàmic, es podria realitzar un petit script que a l'iniciar la comunicació amb el Cloud registres la seva adreça IP dintre d'aquest Security Group per enviar les dades.

SG-ElasticSearch			
Tipus	Protocol	Port	Origen
Custom TCP Rule	TCP	9200	SG-Logstash
Custom TCP Rule	TCP	9200	SG-Kibana

Taula 3: Security Group de les instàncies de Elasticsearch.

El SG-ElasticSearch haurà de permetre totes les entrades que se li efectuïn des de Logstash i les consultes que rebrà de Kibana.

SG-Kibana			
Tipus	Protocol	Port	Origen
HTTP	TCP	80	0.0.0.0/0

Taula 4: Security Group de les instàncies de Kibana.

El SG-Kibana haurà de permetre rebre totes les consultes HTTP o HTTPS, segons la configuració de Kibana, que es realitzin des de Internet.

A més, es crearà un Security Group que ens servirà per controlar l'accés a la gestió dels equips

SG-SSH			
Tipus	Protocol	Port	Origen
SSH	TCP	22	(IP de gestió)

Taula 5: Secuirty Group de les instàncies.

En aquest Security Group podrem incloure totes les adreces IP que considerem necessàries des de les quals ens haurem de connectar per poder-los gestionar però haurem de ser molt curosos alhora de no tenir regles molt permissives o eliminar les IPs un cop ja no s'estiguin utilitzant.

4.3. Instancies

Per les tres aplicacions ELK que ubicarem disposarem d'una instància EC2 per cada una. D'aquesta manera, seran independents i es poden dimensionar i destinar els recursos concrets per cada aplicació.

El sistema operatiu que utilitzarem com a base serà Ubuntu perquè hem considerat que serà el sistema operatiu Ubuntu que més implementacions s'estan realitzant i perquè trobarem més compatibilitat i repositoris per les nostres necessitats. L'alternativa que hi havia era la distribució Amazon Linux2 del mateix AWS però té uns repositoris més limitats i no està tan contrastat el funcionament d'ELK com amb Ubuntu, d'aquesta manera mitigàvem un possible risc. Per tant, la imatge amb la qual treballarem serà l'AMI Ubuntu Server 18.04 LTS (HVM) SSD Volume Type.

El tipus d'instàncies i la capacitat que tindran l'haurem d'adaptar a cada necessitat de cada aplicació ELK.

Logstash és una petita aplicació que només processa les dades i les transmet. A més, el nostre processat en aquests moments serà molt simple. De manera que les necessitats d'aquestes instàncies seran molt petites, és a dir, amb t2.micro i el mínim d'emmagatzematge serà suficient.

Les instàncies que executaran Elasticsearch han de tenir una major memoria RAM per poder carregar la majoria de les dades en memòria i reduir el temps de resposta de les peticions. Així i tot, pel nostre escenari seleccionarem instàncies amb aquestes característiques podria disparar l'import del servei. Per aquest motiu, treballarem amb una màquina amb 2 processadors i 4 GB de memòria RAM. Si és veies que es queda curt podríem migrar el servei a una instància amb 2 processadors i 8 GB de

memòria RAM. L'emmagatzematge que haurà de tenir és de 40 GB per poder encabir totes les dades.

Finalment, segons els requisits de Kibana podrem instal·lar l'aplicació en un servidor de 2 processadors i 4 GB de memòria RAM i un espai de 20 GB de disc.

Podem veure un resum de les característiques de les EC2 en la següent taula:

Servei	Tipus	vCPUs	Memoria	Tipus SSD	Espai
Logstash	t2.micro	1	1	GP2	8 GB
Elasticsearch	t2.medium t2.large	2 2	4 8	GP2	40 GB
Kibana	t2.medium	2	4	GP2	20 GB

Taula 6: Caracteristiques de les instàncies EC2.

Per crear les instàncies, haurem de seleccionar primer de tot l'AMI que tindran com a base (Ubuntu Server 18.04 LTS (HVM) SSD Volume Type), posteriorment el tipus d'instància, la subxarxa en què s'ubicaran i finalment els Security Groups que tindran relacionats. En aquest darrer cas, haurem de tenir en compte afegir el Security Group SG-SSH i SG-[Servei].

Finalment, un cop tinguem les instàncies inicialitzades podrem accedir per SSH i el primer que hauríem de realitzar és fer una actualització de les aplicacions, serveis i sistema operatiu.

Nota: per accedir per SSH a aquestes màquines és important recordar que l'usuari AWS d'aquestes instàncies és Ubuntu.

4.5. DNS

Les instàncies amb les quals treballarem tindran una adreça IP dinàmica i una adreça semblant aquesta: ec2-35-180-111-179.eu-west-3.compute.amazonaws.com. Per tant, utilitzar qualsevol d'aquestes dues opcions és inviable per garantir el funcionament del sistema ELK. Per aquest motiu, el més apropiat serà incloure un domini o subdomini dintre de Route53 i a partir d'aquí relacionar cada instància amb una adreça concreta. Haurem de definir dues adreces externes per accedir a Logstash i Kibana i una adreça interna per accedir a Elasticsearch.

4.5. Balanceig de la carrega

Tot i que en la implementació que estem efectuant del nostra sistema ELK només disposarem d'una instància per cada aplicació, si necessitéssim dispossar de més instàncies per una aplicació, seria relativament senzill. El motiu és que totes les aplicacions tenen com a protocol de comunicació HTTP o HTTPs i la majoria d'infraestructures del Cloud incorporen eines per gestionar la distribució de les peticions entre diferents instàncies, com Load Balancers.

Ubicarem el balancejador de càrrega davant de cada grup d'instàncies per agafar les peticions HTTP i distribuir-les entre aquestes instàncies. Alhora, aquest sistema ens permetrà centralitzar totes les peticions en una única adreça IP pública o privada o adreça web i no haver de publicar en cada servei totes les IPs que tindran les instàncies.

El nou esquema de comunicació de les aplicacions ELK amb els balancejadors de càrrega seria el següent:

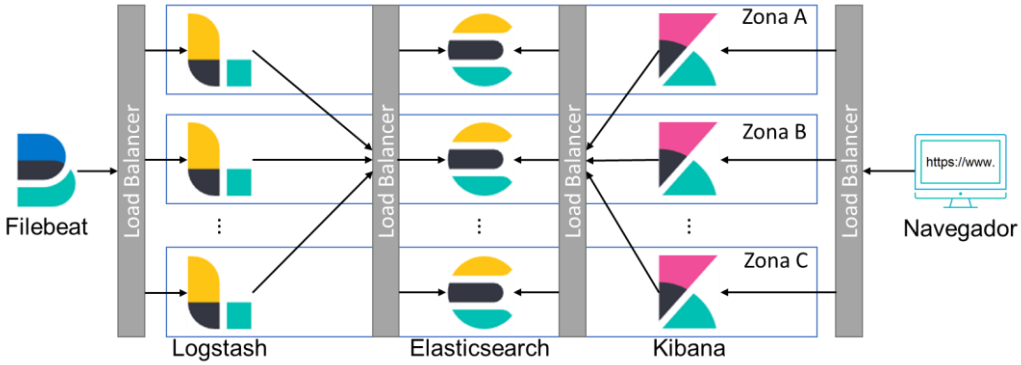

Figura 7: Esquema de comunicació entre aplicacions ELK amb Balancejadors de Carrega.

Cal comentar que aquesta implementació no s'hauria de realitzar per totes les aplicacions alhora, es podria anar realitzant a mesura que la càrrega de treball anés augmentat en cada una de les aplicacions. Fins i tot, si hi ha diferents inputs a processar per part de Logstash es podria assignar a cada una de les instàncies a un tipus de filtre concret a aplicar.

Un altre detall que es pot veure en la figura anterior és que fem referència a les zones d'AWS. Quan treballem en instàncies iguals és convenient distribuir aquestes instàncies en zones diferents per si hi hagués algun problema en una zona, l'altra zona pogués assumir la càrrega. En el cas dels balancejadors de carrega d'AWS per defecte t'obliguen a treball en dues zones diferents per si hi hagués algun problema el balancejador pogués seguir distribuint la càrrega de treball.

Fins ara el model de comunicacions l'hem estat basant en HTTP però és un model de comunicació que envia les dades en text clar. Per aquest motiu, l'ús de HTTPS seria primordial en la implementació d'ELK. El problema principal de treballar amb HTTPS és que les instàncies han de destinar una part dels seus processadors a encriptar i desencriptar les comunicacions. Aquesta tasca es pot veure simplificada, si treballar amb balancejadors de càrrega.

Els balancejadors a part de distribuir la càrrega de treball també poden encriptar i desencriptar les dades HTTPS a HTTP. D'aquesta manera, el processat utilitzat per les instàncies EC2 serà realitzat per la mateixa electrònica de xarxa d'AWS. Llavors per la nostra VPC només tindrem trànsit HTTP.

4.6. Lambda

La informàtica no està absent dels errors i com tot sistema pot fallar. Per aquest motiu, també serà necessari fiscalitzar el funcionament de la nostra infraestructura Cloud. L'eina que ens facilita AWS per aquest control és Lambda.

Lambda és un servei d'AWS que permet el processament de codi sense necessitat d'aprovisionar ni administrar un equip. D'aquesta manera podem desenvolupar un codi que periòdicament consulti les nostres màquines el seu estat i segons la resposta o la no resposta que rebi, el sistema pugui executar les accions pertinents que té programat.

Les tres aplicacions que utilitzem treballen amb HTTP. Per tant, efectuant una petició GET en cada una de les instàncies ens pot ser suficient per saber si les instàncies estan funcionant o no correctament segons la resposta esperem. Consegüentment, segons la resposta rebuda podem notificar-ho per correu electrònic, apagar la instància i tornar-la a aixecar, reiniciar el servei, etc.

5. Desplegament i implementació dels equips del cloud

5.1. Preparació del sistema operatiu

Com hem comentat en la part d'infraestructura en el cloud, treballarem amb un Ubuntu Server 18.04 LTS. Aquesta imatge del sistema operatiu ja inclou alguns dels paquets que necessitarem pero, tot i això, necessitarem instal·lar algunes aplicacions que utilitzarem en la instal·lació de Kibana i ElasticSearch.

Primer de tot, haurem d'actualitzar els repositoris d'Ubuntu i el sistema operatiu.

```
apt-get update
apt-get dist-upgrade -y
reboot
apt-get upgrade -y
reboot
```

En aquest punt és interessant realitzar una imatge del sistema operatiu per si posteriorment haguéssim de partir de nou en la instal·lació del sistema operatiu. Aquesta imatge que generaríem la podríem eliminar un cop generéssim la imatge del sistema operatiu amb tots els programes instal·lats.

En aquesta imatge no ens és necessari assegurar la connexió SSH perquè per defecte el sistema AWS ja obliga a realitzar les connexions entre client servidor mitjançant un sistema de clau pública i privada.

Ambdues aplicacions treballen amb java. Per tant, primer de tot, haurem d'instal·lar java en el sistema operatiu. Com que java no està dins dels repositoris per defecte d'Ubuntu, primer haurem d'agregar el seu repositori al sistema operatiu.

```
add-apt-repository ppa:linuxuprising/java
apt-get update
apt install oracle-java11-installer
apt install oracle-java11-set-default
```

Definim una variable d'entorn que serà la ubicació de l'entorn d'execució de Java afegint la següent línia a l'arxiu /etc/environament.

```
JAVA_HOME="/usr/lib/jvm/java-11-oracle/"
```

5.2. Instal·lar Elasticsearch i configuració

Primer de tot, haurem d'importar el repositoris Elastic:

```
wget -qO - https://artifacts.elastic.co/GPG-KEY-
elasticsearch | sudo apt-key add –
apt-get install apt-transport-https
echo "deb https://artifacts.elastic.co/packages/7.x/apt
stable main" | sudo tee -a /etc/apt/sources.list.d/elastic-
7.x.list
apt-get update
apt-get install elasticsearch
```

Un cop instal·lat Elasticsearch només ens quedarà configura el servei perquè s'iniciï automàtica quan s'inicialitzi el sistema operatiu.

```
/bin/systemctl daemon-reload
/bin/systemctl enable elasticsearch.service
systemctl start elasticsearch.service
```

Un cop inicialitzat el servei, passat un minut, podem comprovar si el servei d'Elasticsearch preguntar-li si funciona amb una petició http.

```
curl -XGET http://localhost:9200
```

Per optimitzar el funcionament d'ElasticSearch serà interessant configurar l'ús de la memòria RAM i la memòria swap. Per treure les limitacions de la memòria RAM, haurem de modificar els següents arxius:

- /etc/elasticsearch/elasticsearch.yml

  `bootstrap.memory_lock: true` (descomentar la linia)

- /etc/default/elasticsearch

  `MAX_LOCKED_MEMORY=unlimited` (descomentar la linia)

- /etc/security/limits.conf (afegir les línies)

  ```
  elasticsearch soft memlock unlimited
  elasticsearch hard memlock unlimited
  ```

- /usr/lib/systemd/system/elasticsearch.service ()

  `LimitMEMLOCK=infinity`

D'aquesta manera, ja hem eliminat les limitacions que el mateix Elasticsearch té per defecte. Tot i això, hem de recordar que Elasticsearch

treballa sobre una màquina virtual de Java. Per tant, també haurem de dir-li a Java l'ús de memòria RAM que pot utilitzar l'aplicació. Aquest valor haurà de deixar la RAM necessària per permetre treballar la resta de serveis que té la màquina i el sistema operatiu. Per configurar-ho, serà necessari modificar l'arxiu /etc/elasticsearch/jvm.options les dues següents línies de codi amb els gigabytes que volem permetre que utilitzi.

```
-Xms1g
-Xmx1g
```

En la nostra implementació, utilitzarem la mateixa màquina virtual tant per Kibana com per Elasticsearch com per Logstash. Quan s'han de processar moltes dades o distribuïm en diferents servidors les dades, és convenient tenir ambdos aplicacions en equips separats i dedicar el màxim de ram a cada aplicació.

Perquè aquests canvis, tinguin afecta haurem de recarregar la configuració del servei i reiniciar-lo.

```
/bin/systemctl daemon-reload
systemctl stop elasticsearch.service
systemctl start elasticsearch.service
```

L'altre element que podem optimitzar és desactivar la memòria swap del sistema operatiu per efectuar totes les consultes en memòria RAM i millorar el temps de resposta de les consultes. Per desactivar-ho, haurem d'anar a l'arxiu /etc/fstab i comentar la línia que fa referència a l'espai de la memòria swap.

```
/swap.img       none    swap    sw      0       0
```

Quan hàgem guardat aquest canvi, haurem de reiniciar el sistema operatiu perquè tingui efecte.

Per poder comprovar que aquests canvis han tingut efecte, podem executar les següents comandes per veure el comportament del sistema operatiu i de l'aplicació abans i després dels canvis.

Per comprovar la reserva de memòria RAM de l'aplicació, haurem de mirar les primeres línies dels logs d'ElasticSearch a /var/log/ /elasticsearch/elasticsearch.log on cercarem la següent línia i els megabytes que indica seran els que l'aplicació hi té accés per treballar.

```
[2019-05-14T18:38:09,829][INFO ][o.e.e.NodeEnvironment    ]
[RmVVAZg] heap size [1015.6mb], compressed ordinary object
pointers [true]
```

Per la memòria RAM executarem la comanda `free`.

Abans del canvi:

```
              total        used        free      shared  buff/cache
Mem:        4015748     3272884      340416         296      402448
Swap:       4015100       42252     3972848
```

Després del canvi:

```
              total        used        free      shared  buff/cache
Mem:        4015748     3272884      340416         296      402448
Swap:             0           0           0
```

En aquest punt, podíem configurar alguns dels paràmetres que utilitzarem d'Elasticsearch com el nom del node, el nom del clúster, l'adreça IP i el port sobre el qual treballarem. Per defecte, l'adreça IP és l'adreça de loopback. Per tant, si les consultes es realitzen des del mateix servidor es pot deixar configurat així però si ha de rebre consultes des de diferents servidors, haurem d'indicar que escolti la interfície. Per exemple, haurem d'indicar la interfície eth0 dintre de l'arxiu de configuració /etc/elasticsearch/elasticsearch.yml.

```
network.host: _local_,_eth0:ipv4_
```

Si volguéssim treballar amb més d'un node, podríem partir de la configuració que hem realitzat fins ara i a partir d'aquí modificar de nou la configuració de /etc/elasticsearch/elasticsearch.yml perquè els nodes dels clúster de Elasticsearch es comuniquin entre ells. Per això haurem de descomentar les línies següents i indicar els rangs IP de la resta de hosts que formen part del clúster.

```
discovery.zen.ping.multicast.enabled: false
discovery.zen.ping.unicast.hosts: ["IP_host1", "IP_host2",
"IP_host3"]
discovery.zen.minimum_master_nodes: 2
```

Aquest sistema funciona quan tenim un nombre de nodes petit i podem conèixer les IPs de cada un dels nodes. Si el nombre de nodes fos molt gran, s'hauria d'habilitar la funcionalitat de multicast. Per defecte, la IP de multicast és la 224.2.2.4 i el port 54328. El mateix sistema s'encarregarà d'enviar periòdicament missatges a l'adreça multicast per descobrir la resta de nodes i agregar-los de forma periòdica al clúster.

5.3. Instal·lar Kibana

Kibana està inclòs dintre dels repositoris que hem inclòs en els apartats anteriors. Per aquest motiu, la seva instal·lació serà tan senzilla com executar la comanda d'instal·lació i habilitar el servei.

```
apt-get install kibana
/bin/systemctl daemon-reload
/bin/systemctl enable kibana.service
systemctl start kibana.service
```

En aquest cas, per comprovar el funcionament d'aquesta aplicació, haurem d'accedir als logs de la mateixa per comprovar que funciona correctament perquè la resposta que ens donaria la comanda curl seria en llenguatge http. Per comprovar-ho podem fer un `cat` al següent arxiu:

```
cat /var/log/kibana/kibana.log
```

Com que haurem d'accedir des de fora a kibana, haurem de modificar la configuració perquè pugui escoltar les peticions que li vinguin de qualsevol IP. La seguretat del servei com en el cas d'Elasticsearch serà garantit per les regles de seguretat del Cloud. Per permetre qualsevol consulta, haurem de modificar l'arxiu de configuració /etc/kibana/kibana.yml

```
server.host: "0.0.0.0"
```

Si tenim multiples nodes elasticsearch, haurem de configurar també Kibana per accedir a la informació des de qualsevol dels nodes d'elasticsearch que estigui funcionant.

5.3. Instal·lar Logstash

Primer de tot, haurem de tenir en compte tenir instal·lat i configurat java i tenir important el repositori d'elasticsearch al sistema. Si ho tenim fet, podem instal·lar directament l'aplicació. En cas contrari, hauríem de repetir la instal·lació que hem realitzat en els apartats anteriors. La comanda per instal·lar-lo serà.

```
apt-get install Logstash
```

Per comprovar que funciona correctament, podem realitzar una petita execució per línia de codi. Amb la següent comanda:

```
/usr/share/logstash/bin/logstash -e 'input { stdin { } }
output { stdout {} }'
```

D’aquesta manera, quan hagi arrencat, tot el que escriguem ens ho mostrarà per pantalla d’aquesta manera.

```
{
      "@version" => "1",
       "message" => "prova",
    "@timestamp" => 2019-05-18T15:20:04.885Z,
          "host" => "administrator"
}
```

Per configurar el servei, haurem d’accedir a /etc/logstash/conf.d i crearem un arxiu tfm.conf. Un cop tinguem l’arxiu correcte, podrem configurar logstash com un servei i que s’executi automàticament.

```
/bin/systemctl daemon-reload
/bin/systemctl enable logstash.service
systemctl start logstash.service
```

6. Disseny de la implementació gràfica.

Un cop tenim les diferents aplicacions comunicant-se i les dades correctament introduïdes en el nostre sistema, es moment de donar-li sentit. Amb les diferents visualitzacions gràfiques que ens facilita Kibana.

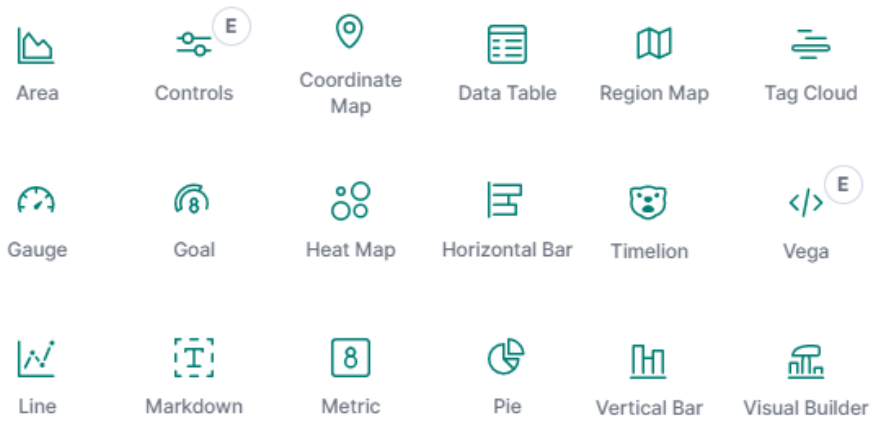

Figura 8: Tipus de gràfics de Kibana.

Entre les diferents representacions que hi ha tenim gràfics, controls, representacions sobre mapes i mapes de calor, paraules repetides, taules, etc. Amb tots aquest dissenys podem aconseguir representar gran varietat o almenys les dades que recollim amb els nostres sensors.

Per les dades de temperatura i humitat les representares de forma gràfica en línies. Per les dades de voltatge i carrega utilitzarem la galga que ens indicarà com es troba la bateria de carrega o el voltatge de sortida del SAI o companyia. Tot i que aquest darrer serà sempre 230 V. Pel consum que estem tenint en el CPD utilitzarem una mètrica i finalment per la font de corrent, també utilitzarem un gràfic que ens indicarà en quin estat es troba.

La principal dificultat que ens hem trobat alhora de crear els gràfics ha estat la coherència entre les dades recollides i les dades d'entrada del gràfic. Per això, vam haver de modificar alguna de les dades que tractàvem amb Logstach i emmagatzemar-les en el format que pertocava. En la majoria dels casos, estàvem guardant valors numèrics en strings.

Un cop s'han creat els diferents elements que volem mostrar en el nostre sistema de monitoratge, podrem crear-nos el nostre propi dashboard perquè amb un cop d'ull puguem saber l'estat del nostre CPD.

Per la temperatura, s'ha seleccionat un gràfic on es representa tant la temperatura com la humitat de la sala.

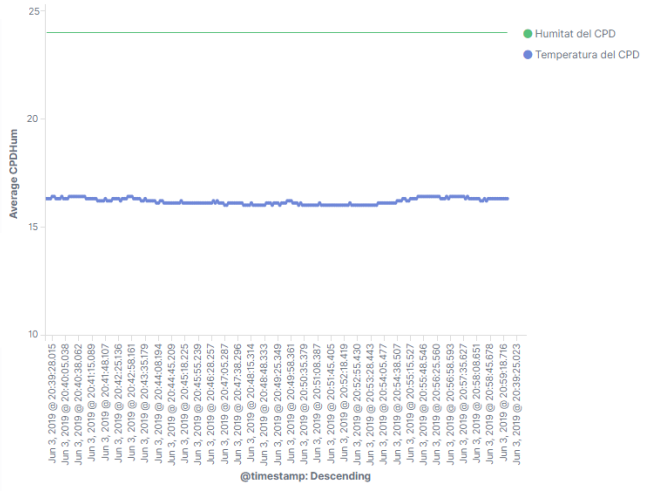

Figura 9: Visualització de les dades d'humitat i temperatura a Kibana.

Per indicar el voltatge de sortida i la càrrega de les bateries, s'ha mostrar mitjanánt una galga per poder veure quins son els valors correctes de sortida.

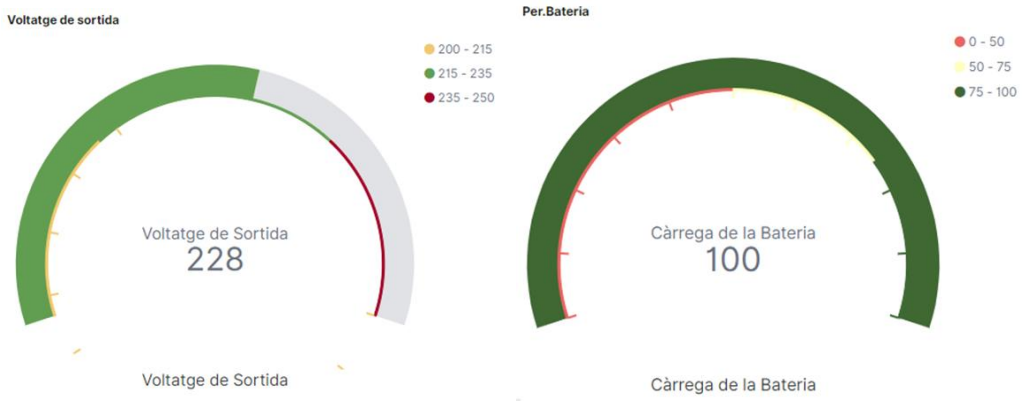

Figura 10: Visualització del voltatge de sortida i de la carrega de les bateries a Kibana.

Finalment, pel consum d'electricitat és mostra el valor actual dels ampers que s'estan consumint..

Figura 11: Visualització de la intensitat a Kibana.

7. Implementació de les alertes

El sistema d'alertes de ELK no es una de les funcionalitats gratuïtes, està inclòs dintre del X-PACK. X-PACK és la versió de pagament de diferents de les funcionalitat que inclou serveis de seguretat, alertes i notificacions, grafs, aprenentatge automàtic i eines de generació d'informes.

Es veritat que de les funcionalitats més cridanera de totes les que ofereix seria la de Machine Learning però pel volum de dades amb que treballarem no es necessari aquest funcionament. Machine Learning ens permetria detectar comportaments anormals en les dades que recollim de forma automàtica. A més, si es detecta alguna anomalia, ens pot ajudar a analitzar informació passada per poder detectar esdeveniments relacionats o semblants als detectats.

Un cop descartat l'ús del Machine Learning en el nostre sistema, ens plantegem la configuració d'alertes automàtiques quan en el sistema és produeixi un esdeveniment preconfigurat perquè ens avisi. En el nostre projecte, consistirà en avisar-nos d'alguna incidència en els sistemes d'alimentació i refrigeració.

El sistema que utilitzarem serà dins de Kibana. Tot i que també, es podria configurar dintre de Logstach mentre es preparen les dades. La idea de fer-ho un cop ja tenim les dades dintre de ELK es per poder tenir un històric de les dades i veure si l'esdeveniment es puntual o continuat. Per exemple, en el sistema de detecció d'on prové la corrent al nostre CPD pot detectar una caiguda de pocs milisegons en el subministrament, llavors, si fem l'alerta des del Logstach, només que hi hagi un missatge amb el problema, ja ens haurà de comunicar que hi hagut algun problema. A favor, d'Elasticsearch podem esperar uns segons no crítics per determinar si és un problema necessari d'informar o a estat només un instant de temps.

Per configurar aquests mecanisme d'alertes, haurem de determinar amb quina periodicitat es revisa, la consulta que es realitzarà a Elàstic Search, la condició que s'ha de donar i les accions que realitzarà.

Per fer-ho, primer de tot haurem d'instal·lar mailutils i configura les dades dintre dels arxius de configuració de Elasticsearch i postfix.

La alerta la haurem de configurar a partir d'un json. En aquest json, s'haurà d'indicar les dades mencionades més amunt de la següent manera.

1. Indicar la periodicitat amb que s'ha d'executar la consulta.

```
"trigger": {
  "schedule": {
    "interval": "1m"
} }
```

2. Escriure la consulta que volem realitzar a elàsticsearch en llenguatge DSL (Domain Specific Language). Per més facil, la generació d'aquest text, es pot fer primer un gràfic amb la clau que volem recuperar per analitzar i un cop la tenim, podem obtenir el codi DSL.

```
"input": {
  "search": {
    "request": {
      "indices": "file*",
      "types": "doc",
      "body": {
        "query": {
          "bool": {
            "should": [ {
              SensorName : "powerFeda"
       } ] } },
      "sort": [ {
        "@timestamp": { "order": "desc" } } ], "size": 1
} } } }
```

3. La condició que activarà l'acció a partir de la consulta anterior.

```
"condition": {
  "compare": {
    "ctx.payload.hits.hits.0._source.monitor.status": {
      "eq": "0"
} }
```

4. L'acció que ha de realitzar

```
"actions" : {
  "send_email" : {
    "throttle_period": "10s",
    "email" : {
      "from" : "alertes@tfm.cat",
      "to" : "tfm@tfm.cat",
      "subject" : "",
      "body" : "Hi hagut algun problema en el subministrament
                elèctric"
} } }
```

Amb aquesta configuració el servei, ja analitzarà periòdicament el codi json que l'haurem configurat.

8. Conclusions

En el transcurs del treball s'han pogut assolir els objectius que estaven previstos en el projecte dintre del temps total del projecte però no s'ha pogut seguir la planificació prevista a l'inici del projecte i s'han produït petites desviacions. Sobretot s'han produït demores en la redacció dels documents. Per desgracia no es va poder desplegar el projecte en un entorn real, tot i que, tampoc es va contemplar des d'un inici del projecte i només es va poder realitzar en entorn de test.

Durant la redacció del projecte, es va haver de substituir alguns dels programes que s'havien utilitzats durant el desenvolupament del projecte perquè havien quedat obsolets i fora dels repositoris d'Ubuntu. Per poder presentar la documentació el més actualitzada possible, es va haver de tornar a realitzar proves de funcionament.

Pel treball de desenvolupament la majoria de la informació va ser bastant facil de trobar, tot i que alhora, de realitzar l'script de Logstach i el de configuració del servei de notificació la informació va ser bastant més difícil de trobar i de crear l'estructura de les dades.

Algunes de les millores que es podien aportat al nostre dispositiu seria substituir el sensor de tensió binaria que em creat per un sensor digital on podem saber el valor concret de tensió entrant i es podria substituir la Raspberry Pi per un dispositiu propi i personalitzat per les nostres funcions. A més, podríem utilitzar puppet o chef per actualitzar els programes i configuracions dels programes. Pel que fa a la infraestructura clau seria interessant provar i implementar alguna solució d'Elasticsearch que ofereix els mateixos proveïdors i implementar les funcions Lambda per garantir el funcionament de totes les aplicacions.

9. Bibliografia

https://www.datamation.com/cloud-computing/aws-vs-azure-vs-google-cloud-comparison.html

https://stackoverflow.com/questions/20898384/ssh-disable-password-authentication 14/04/2019

https://kb.iu.edu/d/aews 14/04/2019

https://raspi.tv/2012/how-to-create-a-new-user-on-raspberry-pi 14/04/2019

http://jacoburibeais.com/myhomeprojects/2019/02/19/elastic-search-project.html

https://www.ictshore.com/sdn/python-snmp-tutorial/

https://www.raspberrypi.org/forums/viewtopic.php?t=43069

https://tecadmin.net/install-oracle-java-11-ubuntu-18-04-bionic/

http://snmplabs.com/pysnmp/index.html

https://www.ictshore.com/sdn/python-snmp-tutorial/

http://www.circuitbasics.com/how-to-set-up-the-dht11-humidity-sensor-on-the-raspberry-pi/

https://www.raspberrypi-spy.co.uk/2012/06/simple-guide-to-the-rpi-gpio-header-and-pins/

https://forum.arduino.cc/index.php?topic=413083.0

https://github.com/elastic/elasticsearch/issues/19868

10. Annexos

10.1. Taula consultes SNMPv1 SAI Generex

En aquest punt es presenta un exemple de les consultes SNMP que podríem realitzar al SAI per saber el seu estat i enviar les dades al cloud.

OID	Nom	Descripció
.1.3.6.1.2.1.33.1.2.1.0	upsBatteryStatus	Estat de la bateria
.1.3.6.1.2.1.33.1.2.4.0	upsEstimatedChargeRemaing	% de bateria
.1.3.6.1.2.1.33.1.4.1.0	upsOutputSource	Font de sortida
.1.3.6.1.2.1.33.1.2.5.0	upsBatteryVoltage	Voltatge de sortida
.1.3.6.1.2.1.33.1.2.6.0	upsBatteryCurrent	Intensitat de sortida

Taula 7: Taula consultes SNMPv1 SAI Generex.

10.2. Exemple d'arxiu de configuració (/etc/tfm/config.json)

```
{
  "temperture": [
    {
      "name": "tempCPD",
      "pin": 22
    }
  ],
  "power": [
    {
      "name": "powerQBT",
     "pin": 4
    },
    {
      "name": "powerFEDA",
      "pin": 17
    },
    {
      "name": "powerEMTE",
      "pin": 27
    }
  ],
  "snmp": [
    {
      "name": "SAI",
      "version": 1,
      "comunity": "PUBLIC",
      "ipAddress": "192.168.1.3",
      "oid": [
        ".1.3.6.1.2.1.33.1.2.1.0",
        ".1.3.6.1.2.1.33.1.2.4.0",
        ".1.3.6.1.2.1.33.1.4.1.0",
        ".1.3.6.1.2.1.33.1.2.5.0"
```

```
        ".1.3.6.1.2.1.33.1.4.1.0",
      ],
      "names": {
        ".1.3.6.1.2.1.33.1.2.1.0" : "upsBatteryStatus",
        ".1.3.6.1.2.1.33.1.2.4.0" : "upsEstimatedChargeRemaing",
        ".1.3.6.1.2.1.33.1.4.1.0" : "upsOutputSource",
        ".1.3.6.1.2.1.33.1.2.5.0" : "upsBatteryVoltage",
        ".1.3.6.1.2.1.33.1.2.6.0" : "upsBatteryCurrent"
      }
    }
  ]
}
```

10.3. Codi programes dispositiu

/opt/tfm/tfmpower.py

```
import json
import time
import logging
import RPi.GPIO as GPIO

logfile='/var/log/tfm/power.log'

logging.basicConfig(filename=logfile,         format='%(asctime)s
%(levelname)s %(message)s', level=logging.INFO)
console = logging.StreamHandler()
logging.getLogger('').addHandler(console)

with open('/etc/tfm/config.json') as json_file:
      data = json.load(json_file)['power']


GPIO.setmode(GPIO.BCM)
for sensor in data:
      GPIO.setup(sensor['pin'], GPIO.IN)

while True:
      for sensor in data:
            if (GPIO.input(sensor['pin']) == True):
                  logging.info("%s is up" % sensor['name']);
            else:
                  logging.info("%s is down" % sensor['name']);
        time.sleep(60)
```

/opt/tfm/tfmtemp.py

```
import Adafruit_DHT
import time
import logging
import json
```

```
logfile='/var/log/tfm/temp.log'

logging.basicConfig(filename=logfile,         format='%(asctime)s
%(levelname)s %(message)s', level=logging.INFO)
console = logging.StreamHandler()
logging.getLogger('').addHandler(console)

sensorDHT = Adafruit_DHT.DHT11

with open('/etc/tfm/config.json') as json_file:
  data = json.load(json_file)['temperture']

while True:
      for sensor in data:
            humidity,                temperature                =
Adafruit_DHT.read_retry(sensorDHT, sensor['pin'])
            if humidity is not None and temperature is not None:
                  text             =             'Temp={0:0.1f}*C
Humidity={1:0.1f}%'.format(temperature, humidity)
                  logging.info("%s - %s" % (sensor['name'],text))
            else:
                  logging.error('Failed to get reading sensor: %s.
Try again!' % sensor['name'])

            time.sleep(1)
```

/opt/tfm/tfmsnmp.py

```
import json
import time
import logging
from pysnmp import hlapi
from snmpv1 import get

logfile='/var/log/tfm/snmp.log'

logging.basicConfig(filename=logfile,         format='%(asctime)s
%(levelname)s %(message)s', level=logging.INFO)
console = logging.StreamHandler()
logging.getLogger('').addHandler(console)

with open('/etc/tfm/config.json') as json_file:
  data = json.load(json_file)['snmp']

while True:
  for sensor in data:
    for oid in sensor['oid']:
      if sensor['version'] == 1:
        try:
          resposta     =     get(sensor['ipAddress'],     [oid],
hlapi.CommunityData(sensor['comunity']))
          for key, value in resposta.items():
```

```
            text = sensor['name']+sensor['names'][oid] + "-" +
str(value)
            logging.info(text)
        except:
          logging.error('Sensor %s not respon %s in time' %
(sensor['ipAddress'], oid))
      elif sensor['version'] == 2:
        logging.error('NOT IMPLEMENTED')
      elif sensor['version'] == 3:
        logging.error('NOT IMPLEMENTED')
      else:
        logging.error('ERROR JSON')
  time.sleep(5)
```

/opt/tfm/snmpv1.py

(Codi obtingut de la pàgina web: https://www.ictshore.com/sdn/python-snmp-tutorial/)

```
from pysnmp import hlapi

def cast(value):
   try:
      return int(value)
   except (ValueError, TypeError):
      try:
         return float(value)
      except (ValueError, TypeError):
         try:
            return str(value)
         except (ValueError, TypeError):
            pass
   return value

def fetch(handler, count):
   result = []
   for i in range(count):
      try:
         error_indication, error_status, error_index, var_binds =
next(handler)
         if not error_indication and not error_status:
            items = {}
            for var_bind in var_binds:
               items[str(var_bind[0])] = cast(var_bind[1])
               result.append(items)
         else:
            raise RuntimeError('Got SNMP error: {0}'.format(
            error_indication))
      except StopIteration:
         break
   return result

def construct_object_types(list_of_oids):
   object_types = []
```

```
   for oid in list_of_oids:

object_types.append(hlapi.ObjectType(hlapi.ObjectIdentity(oid)))
   return object_types

def      get(target,      oids,      credentials,      port=161,
engine=hlapi.SnmpEngine(), context=hlapi.ContextData()):
   handler = hlapi.getCmd(
      engine,
      credentials,
      hlapi.UdpTransportTarget((target, port)),
      context,
      *construct_object_types(oids)
   )
    return fetch(handler, 1)[0]
```

/opt/tfm/ keepAlive.sh

```
#!/bin/bash

if [ ! -f "/var/log/tfm/snmp.log" ]
then
   screen -X -S tfmsnmp quit
   screen -d -S tfmsnmp -m python3 /opt/tfm/snmp.py
else
   FILES=`find "/var/log/tfm/snmp.log" -mtime -0.000695 | wc -l`
   if [ "$FILES" == "0" ]
   then
      screen -X -S tfmsnmp quit
      screen -d -S tfmsnmp -m python3 /opt/tfm/snmp.py
   fi
fi

if [ ! -f "/var/log/tfm/temp.log" ]
then
   screen -X -S tfmtemp quit
   screen -d -S tfmtemp -m python3 /opt/tfm/tfmtemp.py
else
   FILES=`find "/var/log/tfm/temp.log" -mtime -0.000695 | wc -l`
   if [ "$FILES" == "0" ]
   then
      screen -X -S tfmtemp quit
      screen -d -S tfmtemp -m python3 /opt/tfm/tfmtemp.py
   fi
fi

if [ ! -f "/var/log/tfm/power.log" ]
then
   screen -X -S tfmpower quit
   screen -d -S tfmpower -m python3 /opt/tfm/tfmpower.py
else
   FILES=`find "/var/log/tfm/power.log" -mtime -0.000695 | wc -l`
   if [ "$FILES" == "0" ]
```

```
   then
      screen -X -S tfmpower quit
      screen -d -S tfmpower -m python3 /opt/tfm/tfmpower.py
   fi
fi
```

10.4. Configuració de l'arxiu de logstash

```
input {
  beats {
    port => 5044
  }
}

filter {
  mutate {
    remove_field => [ "@version" ]
    remove_field => [ "agent" ]
    remove_field => [ "host" ]
    remove_field => [ "log" ]
    remove_field => [ "file" ]
    remove_field => [ "offset" ]
    remove_field => [ "input" ]
    remove_field => [ "ecs" ]
    remove_field => [ "tags" ]
  }

  date {
    match => ["message", "YYYY-MM-dd HH:mm:ss"]
    target => "customlog"
  }

  if [message] =~ /^*power*/ {
    grok {
      match  =>  {  "message"  =>  "%{TIMESTAMP_ISO8601:logdate}
%{LOGLEVEL:log-level}         %{GREEDYDATA:SensorName}         is
%{GREEDYDATA:state}" }
    }
    if [message] =~ /^*up*/ {
      mutate {
        add_field => { "voltageoutput" => 230 }
      }
    } else {
      mutate {
        add_field => { "voltageoutput" => 0 }
      }
    }
    mutate { convert => { "voltageoutput" => "integer" } }
  }

  if [message] =~ /^*SAI*/ {
    grok {
```

```
      match  =>  {  "message"  =>  "%{TIMESTAMP_ISO8601:logdate}
%{LOGLEVEL:log-level} %{GREEDYDATA:textjson}" }
    }
    json {
      source => "textjson"
    }
    mutate {
      remove_field => [ "textjson" ]
    }
  }

  if [message] =~ /^*CPD*/ {
    grok {
      match  =>  {  "message"  =>  "%{TIMESTAMP_ISO8601:logdate}
%{LOGLEVEL:log-level} %{GREEDYDATA:textjson}" }
    }
    json {
      source => "textjson"
    }
    mutate {
      remove_field => [ "textjson" ]
    }
  }


}

output {
  elasticsearch {
    hosts => ["127.0.0.1:9200"]
  }
  stdout { codec => rubydebug }
}
```