Integrantes
Hugo Hernández - 201614900
William Duarte - 201620791
Gabriel Pinto - 201515275
Juan Trujillo - 201618006
Marlo Forero - 201614328

Elay - Entrega 1

Arduino y Node-Red

Para la ejecución y detección correcta de alarmas, se hace uso de un Arduino, el cual tiene conectado, por medio de un protoboard, un sensor de presencia, un buzzer, un potenciómetro, diferentes LEDs, una fuente de alimentación compuesta por 2 baterías y el módulo Wifi ESP8266. El arduino se encarga de procesar toda la información suministrada por los diferentes componentes y de transmitirle los mensajes por serial al módulo Wifi, el cual se encarga de publicar las alarmas en un tópico definido para que luego puedan empezar a ser procesadas por NodeRed.

Con el fin de implementar correctamente las alarmas, se implementó la siguiente estructura para los mensajes:
“A:idAlarma:idDispositivo:unidadResidencial:torre:apartamento”

Donde todos los valores están definidos en el código y son constantes, menos “idAlarma”, el cual depende del tipo de alarma generada. En nuestro caso, manejamos 5 tipos de alarma:

- Alarma si la puerta lleva abierta más de 30 segundos. En este caso se envía el código “1”.
- Alarma si se ha introducido una contraseña incorrecta en el *keypad* más del número de veces permitido, que es 3. En este caso se envía el código “2”.
- Alarma si una persona se acercó a la cerradura en un horario no permitido y duró más de un minuto cerca de está. En este caso se envía el código “3”.
- Alarma si una persona entró a la casa en un horario no permitido. En este caso se envía el código “4”.
- Alarma si el nivel de batería de la cerradura llega a un nivel crítico. En este caso se envía el código “5”.

Siempre que se genera una de las 4 primeras alarmas, el buzzer continúa sonando hasta que esta sea controlada de forma adecuada. En el caso de la alarma 5, el buzzer suena durante 2 segundos y lo hace cada 30 segundos.

Las alarmas son enviadas por medio del puerto serial, para luego ser recibidas por el módulo Wifi y publicarlas por medio de MQTT en el tópico “alarmasCerradura.” Luego de estos son procesadas por Node-Red.

En Node-Red definimos casos para tratar los diferentes mensajes que transmite el Arduino, por medio del módulo Wifi, diferenciando los casos de mensajes para propietarios, administrador, seguridad y YALE, según el código que haya enviado la placa de Arduino

- Para el código “1” de alarma, el mensaje asignado es “La puerta lleva abierta un largo periodo de tiempo.”.
- Para el código “2” de alarma, el mensaje asignado es “Intento de apertura sospechoso. Alguien ha ingresado la clave más de 3 veces de forma errónea”.
- Para el código “3” de alarma, el mensaje asignado es “Se detectó la presencia de una persona cerca de la cerradura, en un horario no permitido.”.
- Para el código “4” de alarma, el mensaje asignado es “Ingreso sospechoso. Alguien entró al inmueble en un horario no permitido.”.
- Para el código “5” de alarma, el mensaje asignado es “Nivel de batería crítico. Recargue la cerradura.”

A las alarmas no contempladas por los códigos anteriores, se les asigna el código 0.

A los mensajes se les añade su *timestamp* y se formatean en formato JSON y según el rol asignado se añaden al tópico de MQTT correspondiente, para esto debe configurarse correctamente el nodo MQTT con la IP y el tópico requerido. Esto se hace por medio de una función que analiza el id de la alarma y a partir de ello define los roles y tópicos en los cuales es necesario publicar dicha alarma por medio de MQTT.

En nuestro caso, definimos que las alarmas 1, 2, 3, 4 y 5 deben ser informadas tanto al propietario como a la seguridad. La alarma 0 a Yale y a partir de los alcances de esta entrega, aún no es necesario notificar al administrador en tiempo real.

Notificación y persistencia

Para asegurarnos que el propietario sea notificado lo más pronto posible de cualquier alarma, implementamos una arquitectura lambda. Esta nos permite subdividir los procesos de notificar y persistir la información asegurando que la parte que notifica sea lo más pronta y cercana al tiempo real. Adicionalmente se

implementó un nuevo nodo, el *disparador*, el cual invoca los servicios REST de los nodos *Speed* y *Batch* con el fin de atomizar las tareas realizadas.
El nodo *disparador* está pendiente de cualquier cosa que sea publicada en el tópico al cual está suscrito. Inmediatamente es publicada una alarma en el tópico el *disparador* invoca el servicio POST del nodo *Speed* (que se encuentra otro nodo) para que este se encargue de enviar el correo al rol (propietario, administrador, seguridad o Yale) que le pasaron por Path. Seguido a esto, el *disparador* invoca el servicio REST del nodo *Batch* el cual se va a encargar de persistir los datos en una base de datos no relacional (Cassandra). Lo anterior bajo los resultados vistos en los laboratorios de clase.

Escenarios de prueba
Para los escenarios de prueba se reunieron *Collections* de peticiones en Postman, esto con el fin de poblar las tablas para poder trabajar sobre estos valores. Se cargaron las tablas de las alarmas (las cuales tienen los datos básicos de donde se generó la alerta, el tipo, y un ID autogenerado), unidad residencial, inmueble, hub y dispositivo. A continuación, con los datos ya generados se procedió a realizar las pruebas de carga con la ayuda de JMeter.

Comparación de los datos obtenidos vs. esperados
Tanto para Speed como para Batch, no se obtuvieron los valores esperados en términos de porcentaje de error ni de latencia.
Para el caso de Speed, en la prueba de JMeter con 200 hilos de ejecución, el porcentaje de error fue del 0% y la latencia no excedió 1s. Similarmente, para Batch no hubo errores y tampoco excedió 1s. Sin embargo, para la prueba de 300.000 hilos, se produjeron errores de memoria tanto para Speed como para Batch, ya que en ningún caso se pudo terminar la prueba. Lo anterior puede ser debido a que no se implementó un balanceador de carga, por lo tanto se debe implementar para satisfacer exitosamente los escenarios de calidad propuestos. A continuación se muestran los resultados obtenidos en estos escenarios mediante la consola de JMeter:
Speed 200 hilos

```
C:\ISIS2503\JMeter\bin>jmeter -n -t "C:\Users\ma.forero11\Desktop\Test for Polyg
lot Persistence - Read and Write Test(RW)_v2.jmx"
Writing log file to: C:\ISIS2503\JMeter\bin\jmeter.log
Creating summariser <summary>
Created the tree successfully using C:\Users\ma.forero11\Desktop\Test for Polygl
ot Persistence - Read and Write Test(RW)_v2.jmx
Starting the test @ Sun Mar 25 20:48:16 COT 2018 (1522028896181)
Waiting for possible Shutdown/StopTestNow/Heapdump message on port 4445
summary +     47 in 00:00:14 =    3,4/s Avg:    36 Min:     2 Max:   818 Err:
  0 (0,00%) Active: 1 Started: 47 Finished: 46
summary +    100 in 00:00:30 =    3,3/s Avg:     2 Min:     2 Max:     7 Err:
  0 (0,00%) Active: 1 Started: 147 Finished: 146
summary =    147 in 00:00:44 =    3,3/s Avg:    13 Min:     2 Max:   818 Err:
  0 (0,00%)
summary +     53 in 00:00:16 =    3,3/s Avg:     2 Min:     2 Max:     5 Err:
  0 (0,00%) Active: 0 Started: 200 Finished: 200
summary =    200 in 00:01:00 =    3,3/s Avg:    10 Min:     2 Max:   818 Err:
  0 (0,00%)
Tidying up ...    @ Sun Mar 25 20:49:16 COT 2018 (1522028956046)
... end of run
```

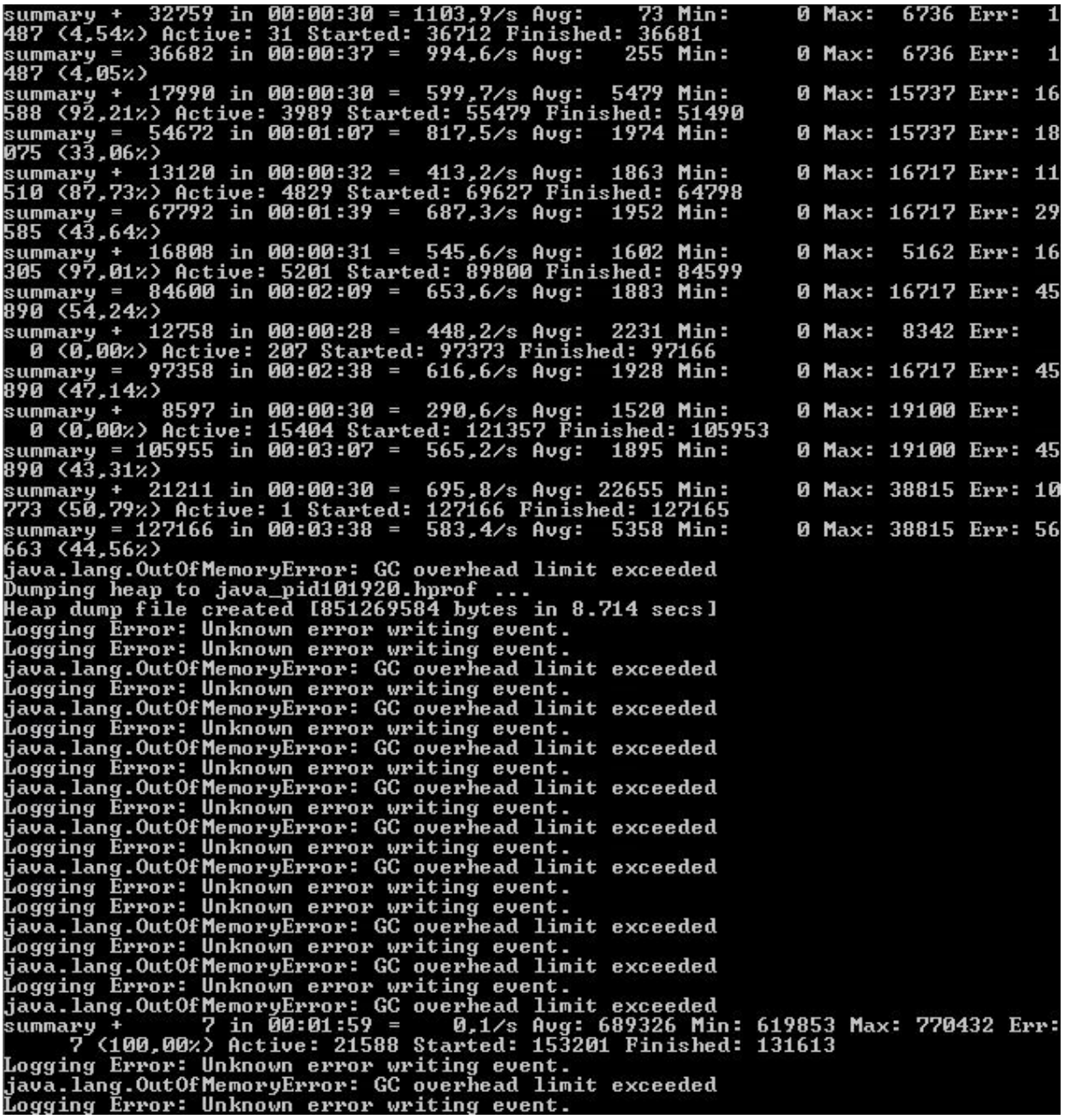
Batch 200 hilos

```
C:\ISIS2503\JMeter\bin>jmeter -n -t "C:\ISIS2503\JMeter\bin\Grupo de Hilos.jmx"
Writing log file to: C:\ISIS2503\JMeter\bin\jmeter.log
Creating summariser <summary>
Created the tree successfully using C:\ISIS2503\JMeter\bin\Grupo de Hilos.jmx
Starting the test @ Sun Mar 25 20:43:18 COT 2018 (1522028598666)
Waiting for possible Shutdown/StopTestNow/Heapdump message on port 4445
summary +     39 in 00:00:11 =    3,4/s Avg:    28 Min:     5 Max:   842 Err:
  0 (0,00%) Active: 1 Started: 39 Finished: 38
summary +    100 in 00:00:30 =    3,3/s Avg:     6 Min:     5 Max:    13 Err:
  0 (0,00%) Active: 1 Started: 139 Finished: 138
summary =    139 in 00:00:41 =    3,3/s Avg:    12 Min:     5 Max:   842 Err:
  0 (0,00%)
summary +     61 in 00:00:18 =    3,3/s Avg:     5 Min:     5 Max:    12 Err:
  0 (0,00%) Active: 0 Started: 200 Finished: 200
summary =    200 in 00:01:00 =    3,3/s Avg:    10 Min:     5 Max:   842 Err:
  0 (0,00%)
Tidying up ...    @ Sun Mar 25 20:44:18 COT 2018 (1522028658530)
... end of run
```

Speed 300.000 hilos

```
C:\ISIS2503\JMeter\bin>jmeter -n -t "C:\Users\ma.forero11\Desktop\Test for Polyg
lot Persistence - Read and Write Test(RW)_v2.jmx"
Writing log file to: C:\ISIS2503\JMeter\bin\jmeter.log
Creating summariser <summary>
Created the tree successfully using C:\Users\ma.forero11\Desktop\Test for Polygl
ot Persistence - Read and Write Test(RW)_v2.jmx
Starting the test @ Sun Mar 25 20:56:12 COT 2018 (1522029372769)
Waiting for possible Shutdown/StopTestNow/Heapdump message on port 4445
java.lang.OutOfMemoryError: GC overhead limit exceeded
Dumping heap to java_pid80252.hprof ...
Heap dump file created [763663457 bytes in 2.767 secs]
Logging Error: Unknown error writing event.
java.lang.OutOfMemoryError: GC overhead limit exceeded
        at java.util.ResourceBundle$CacheKey.<init>(Unknown Source)
        at java.util.ResourceBundle.getBundleImpl(Unknown Source)
        at java.util.ResourceBundle.getBundle(Unknown Source)
        at sun.util.resources.LocaleData$1.run(Unknown Source)
        at sun.util.resources.LocaleData$1.run(Unknown Source)
```

Batch 300.000 hilos



Reflexión arquitectura actual

La arquitectura actual no satisface los escenarios de calidad propuestos, ya que ocurren errores de memoria al sobrecargarla con la llegada masiva de peticiones. Para poder cumplir el objetivo es necesario mantener divididos los programas Speed y Batch en máquinas separadas, como está ahora, pero hacer escalamiento horizontal. Además, es necesario implementar un balanceador de carga que nos permita probar la arquitectura con los escenarios de prueba mencionados y que esta pueda soportar el volumen de peticiones tope (300.000 en un minuto) sin que se produzcan errores y logrando que la latencia no supere 1s.