

# **Universidad de los Andes**

## **Propuesta de metodología de arquitectura** Arquitectura y diseño de Software

### **Grupo LANN**

Laura Castro

Nicolás Gómez

Andrés Sandoval

Nicolás Vásquez Murcia

Agosto 2016

# Tabla de contenidos

## **Entrega parcial**

### Pre-experimentación

Problemática

Objetivo del experimento

Descripción del experimento

Artefactos a construir

Recursos de la experimentación

Resultados esperados

Duración y etapas

### Post-experimentación

Resultados obtenidos

Duración real

Artefactos contruidos

Análisis

Conclusiones

## **Entrega definitiva**

Ejecutar pruebas de carga

Analizar y comparar los resultados

Programa que envíe datos desde el controlador

Lógica de negocio asociada al microcontrolador

Conclusiones

# Pre-experimentación

## Problemática

En este caso resulta bastante sencillo identificar la problemática a partir del enunciado que fue entregado. En pocas palabras, se tiene una empresa extractora de petróleo llamada OilCol S.A, la cual actualmente no tiene un buen manejo de la información que tiene a su disposición, por lo cual desean implementar un sistema que los ayude con esto. Se desarrollará una aplicación web con el fin de ayudar a la empresa a tener un registro confiable de sus campos, pozos, sensores, y poder manipular como sea necesario los mensajes que estos últimos envían para generar reportes, cambiar el estado de los pozos y demás funcionalidades que OilCol S.A considere necesarias. Para el desarrollo de esta aplicación, se propone implementar una arquitectura Play.

## Objetivo del experimento

Play es un framework que implementa la arquitectura modelo-vista-controlador, y es una arquitectura reactiva, sin estados, y asíncrona. Resulta atractiva dado que busca disminuir problemas relacionados al desarrollo, la escalabilidad y la tolerancia a fallos. Dado que todos los anteriores son atributos importantes para la aplicación a desarrollar, y la facilidad que presenta Play para los desarrolladores, se optó por esta opción. Para probarla, se hará una implementación básica, creando modelos para cada entidad que interactúa en el software (zonas, campos, pozos, sensores, usuarios, y mensajes), y los controladores que sean necesarios, para poder probar todas las funciones mediante peticiones HTTP. Se espera que la aplicación soporte 4800 peticiones por segundo, en un peor escenario.

## Descripción del experimento

Se sabe que OilCol S.A posee 90 campos, 1200 pozos y 4800 sensores, de 4 tipos. Estos 4 tipos de sensores envían diferente información sobre el pozo al que corresponden, y ésta es la información más importante que debe manejar la aplicación. A través de los registros de los sensores, se deben generar reportes y cambiar los estados de los pozos. Por lo anterior, es necesario diseñar una lógica que sea capaz de manipular todos los posibles sensores y almacenar la información en una base de datos para su uso posterior. Adicionalmente, aunque no es tan imprescindible, también es necesario almacenar datos de zonas, campos, pozos, y usuarios en la base de datos.

## Artefactos a construir

Se deben crear los siguientes elementos dentro del proyecto play-java:

- Entidades (Models):
  - Usuario
  - Zona
  - Campo
  - Pozo
  - Sensor
  - Mensaje
- Controladores:
  - UsuarioController
  - ZonaController
  - CampoController
  - PozoController
  - SensorController
  - MensajeController
- Dispatcher de Akka

## Recursos de la experimentación

Se utilizan dos máquinas para el desarrollo del proyecto:

- Máquina principal: Dell Inspiron 5547, sistema operativo Windows 10 Pro de 64 bits, 8gb de RAM, procesador Intel Core i7-4510U@2.6GHz.
- Máquinas de apoyo: Máquinas virtuales previstas por la universidad, sistema operativo Windows 7, con 8gb de RAM, procesador Intel Xeon CPU E5-2640@2.6GHz

Adicionalmente, se utilizan los siguientes softwares:

- Como IDE, se utiliza IntelliJ IDEA 2016 Ultimate Edition.
- Se utiliza pgAdmin III para manejar las bases de datos de PostgreSQL.
- Se utiliza Apache JMeter para realizar las pruebas necesarias.

## Resultados esperados

A partir de los recursos disponibles, el diseño del proyecto y la arquitectura implementada se espera que, además del buen funcionamiento de la aplicación (el registro de los pozos de la empresa), el sistema sea capaz de soportar la recepción de la información de todos los sensores con un margen de error de menos de un segundo al hacer pruebas de carga. A partir de estas pruebas se evaluará el nivel de rendimiento de la aplicación y se podrá constatar si la arquitectura Play es la adecuada.

Por otro lado se espera que el tiempo de respuesta y el error medido incremente en la medida en que el número de muestras también aumente y supere las 4800, las cuales son el máximo previsto o peor caso posible en el cual todos los sensores están transmitiendo información a la vez. Para muestras más pequeñas el tiempo de respuesta y error debe ser mínimo y debe cumplir el requerimiento no funcional pedido de manejar la información en una ventana de un segundo.

## Duración y etapas

Se espera que esta primera etapa tenga una duración aproximada de 12 horas de trabajo. Como primera etapa se tiene desarrollar los modelos de las entidades. La segunda etapa es implementar los servicios correspondientes a estas entidades. Por último, como tercera etapa, se realizan las pruebas sobre el sistema desarrollado.

## Post-experimentación

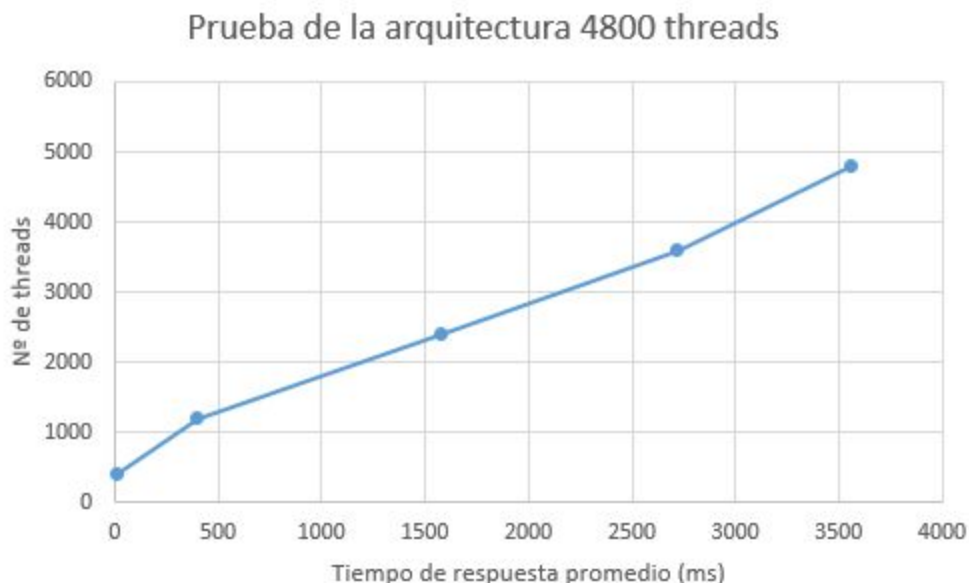
### Resultados obtenidos

Se realizaron varios tipos de pruebas. En primer lugar, se tuvo en cuenta el caso de que los sensores mandaran al mismo tiempo el mensaje desde todos los pozos existentes, generando el peor escenario posible. Los resultados para esto fueron:

Tabla N° 1:

Etiqueta	#Muestras	Media	DesvEstandar	Error
Peticion HTTP	4800	3554	1374	18,44%
Peticion HTTP	3600	2721	931,35	16,03%
Peticion HTTP	2400	1573	662,31	0,00%
Peticion HTTP	1200	401	197,82	0,00%
Peticion HTTP	400	10	20,99	0,00%

Gráfica N°1: Prueba con 4800 hilos de ejecución



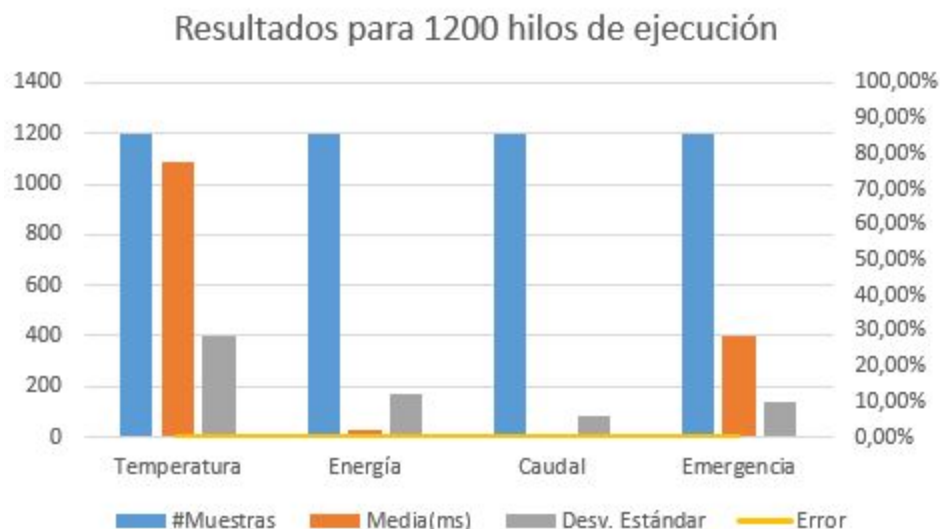
Se incrementó el número de hilos de ejecución hasta llegar a los 4800 puesto que éstos representaban el número total de sensores que pertenecen a OilCol S.A. El tiempo de subida de los hilos fue de 1 segundo, siguiendo el escenario de calidad esperado.

Luego, se realizaron 1200 hilos por cada tipo de sensor que se encontraban, todos en un intervalo de tiempo de 1 segundo. Esto se realizó dado que, en el momento que lleguen todos los mensajes al mismo tiempo, serán 1200 peticiones de cada tipo de sensor. Para cada petición, los datos fueron:

Tabla N° 2:

Etiqueta	#Muestras	Media(ms)	Desv. Estándar	Error
Temperatura	1200	1088	402,71	0,00%
Energía	1200	30	167,52	0,00%
Caudal	1200	14	82,83	0,00%
Emergencia	1200	401	141,6	0,00%
<b>Total</b>	<b>4800</b>	<b>383,25</b>	<b>198,665</b>	<b>0,00%</b>

Gráfica N° 2:



Por último, se hicieron pruebas para ver cómo se desempeñaba la aplicación en un ambiente de ejecución normal, es decir, cómo se va a comportar la aplicación la gran mayoría del tiempo. Esto, se hizo tomando en cuenta el caudal y la energía con una creación de threads en tiempos de 5 y 15 minutos con 1200 hilos. Esto se hizo para tener en cuenta que en 5 y 15 minutos, respectivamente, se recibirá al menos la información de todos los sensores de ese tipo de todos los pozos. A continuación se encuentran los resultados obtenidos:

Tabla N° 3:

Petición	#Muestras	Tiempo (min)	Media respuesta(ms)	Error
<b>Energía</b>	1200	5	13	0%
	1200	15	12	0%
<b>Caudal</b>	1200	5	13	0%
	1200	15	12	0%

## Duración real

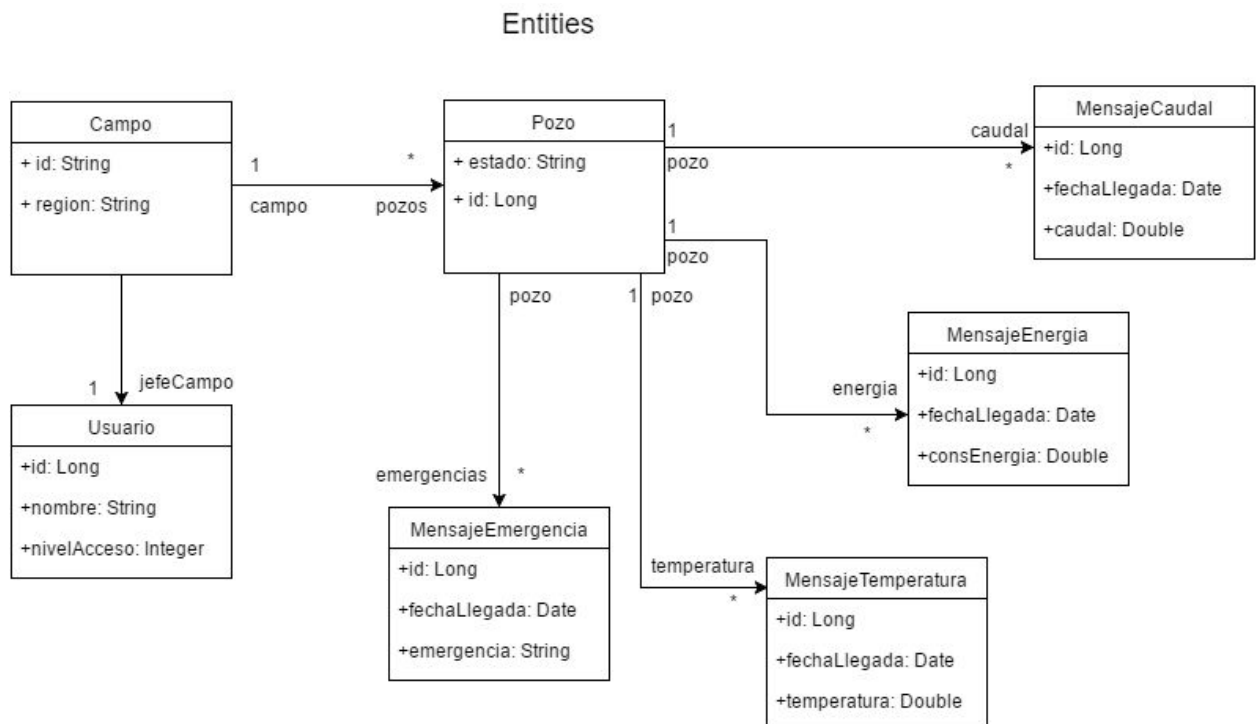
La duración real del experimento fue de 24 horas totales incluyendo las horas de todo el equipo. Esto resulta un trabajo de 24h/persona, para un equipo de 4 personas y representa unas horas totales de 96 horas. Este valor, es más alto al esperado en la Pre-experimentación, puesto que se proponían tiempos totales de 12 horas por persona. Esto sucedió gracias a varios factores que impidieron un óptimo desarrollo de la aplicación. En primer lugar, hubo

cambios en los artefactos generados a lo largo del proyecto, pues se modificó el sensor como una entidad en el ejercicio. En vez, se crearon clases para los mensajes recibidos de los sensores, dado que los sensores existían de manera independiente del proyecto y sólo se obtenían sus registros. Adicionalmente, a medida que se desarrollaba la conexión con una base de datos temporal para realizar pruebas, además de los cambios en la configuración de las rutas, hubo problemas que requirieron más tiempo del estimado e impedían el desarrollo de las demás partes del proyecto. Por último, la estimación del proyecto no tuvo en cuenta que, además de la realización de los diferentes servicios básicos para cada entidad (CRUD) era necesario generar los reportes según el jefe y asignar ciertas capacidades según el tipo de usuario, lo cual también fue un factor determinante en la duración del proyecto.

## Artefactos construidos

Se utilizaron los siguientes artefactos para la implementación de la solución:

- Entidades (Models):
  - Usuario
  - Campo
  - Pozo
  - MensajeCaudal
  - MensajeEnergia
  - MensajeTemperatura
  - MensajeEmergencia





- Controladores:
  - UsuarioController
  - ZonaController
  - CampoController
  - PozoController
  - SensorController
  - MensajeController
- Dispatcher de Akka

## Análisis

Los tiempos de respuesta están acorde a los resultados esperados. Podemos apreciar con la primera gráfica que existe una relación lineal entre el número de threads (o sensores activos) y el tiempo de respuesta. La arquitectura Play nos permite atender un gran número de peticiones y procesar las tareas de forma concurrente, por lo cuál el sistema no se ralentiza de forma exponencial cuando se aumenta el número de muestras.

Para la segunda prueba, la cual mide los tiempos de ejecución para 1200 threads o sensores, vemos una variedad en los resultados para cada tipo de mensaje, sea para temperatura, energía, caudal o emergencia. Vale la pena resaltar que para los cuatro resultados el error es mínimo ( con un 0% para cada uno). Esto se debe a que el número de muestra es mucho menor y es la arquitectura escogida la que nos permite obtener esta precisión en los datos.

Según los resultados de las pruebas reflejados en las gráficas, podemos concluir que no todas las peticiones tienen el mismo tiempo de recepción hacia el sistema. Por esta misma razón es importante tener un Framework que logre mantener un buen rendimiento del sistema sin que este sea dependiente de los datos o de la petición. Play Framework logra esta táctica al ser independiente de cada tipo de mensaje (ver gráfica 1).

## Conclusiones

La aplicación desarrollada en la arquitectura Play como solución al problema generado en la empresa OilCol S.A. generó resultados positivos teniendo en cuenta un balance entre los resultados obtenidos y los esperados. Si bien algunos se resolvieron parcialmente, la aplicación logra resolver el problema principal de gestionar correctamente los registros de los pozos de la empresa petrolera. De esta manera, se consideró oportuna la decisión de tomar Play Frameworks como la arquitectura que nos ayudaría a resolver los requerimientos no funcionales pedidos por el cliente. Esto no significa que no se obtuvieron fallas en las pruebas, pues al mandar 4800 hilos de ejecución, se obtuvo un porcentaje de error, que se espera que vaya a disminuir con el desarrollo del proyecto, hasta obtener una solución idónea para lo solicitado por la empresa.

# Entrega definitiva experimento 1

## Ejecutar pruebas de carga

Tabla n°1:

Etiqueta	# Muestras	Media	Mín	Máx	Desv. Estándar	% Error	Rendimiento	Kb/sec	Media de Bytes
Temperatura	1200	1235	47	2969	775.65	2.17%	272.4	74.88	281.5
Energía	1200	923	2	2968	561.15	0.25%	207.1	51.38	254.1
Caudal	1200	525	1	2071	416.16	0.00%	420.2	98.89	241
Emergencia	1200	135	1	846	163.07	0.00%	408.4	104.1	261
Total	4800	704	1	2969	671.3	0.60%	595.7	150.9	259.4

La primera prueba se realizó con un número de 1200 muestras por cada tipo de mensaje y se midieron varios factores que contribuyen al desempeño de la aplicación. Con la excepción de la temperatura los datos se mantuvieron relativamente bajos, en particular el porcentaje de error que se mantuvo en 0,00%, lo que indica una experimentación correcta.

Tabla n°2:

Etiqueta	# Muestras	Media	Mín	Máx	Desv. Estándar	% Error	Rendimiento	Kb/sec	Media de Bytes
Emergencia	1200	283	4	814	162.78	0.00%	661.2	168.52	261
Total	1200	283	4	814	162.78	0.00%	661.2	168.52	261

La segunda prueba se enfoca únicamente en los mensajes de emergencia que genera la aplicación. Siendo este un requerimiento de mayor importancia, era necesario diseñar una solución que priorizara la generación de mensajes de emergencia y por ende mejorar el desempeño del mismo. Al igual que en la primera prueba, los tiempos y tasas de procesamiento se mantienen óptimos y el porcentaje de error no sobrepasa el 0,00%. Sin embargo, el rendimiento es significativamente superior, por lo cual se confirma un mejor desempeño para este aspecto del sistema.

## Análisis y comparación de los resultados

Se recuerda que durante la iteración anterior se obtuvieron los siguientes resultados para 1200 muestras:

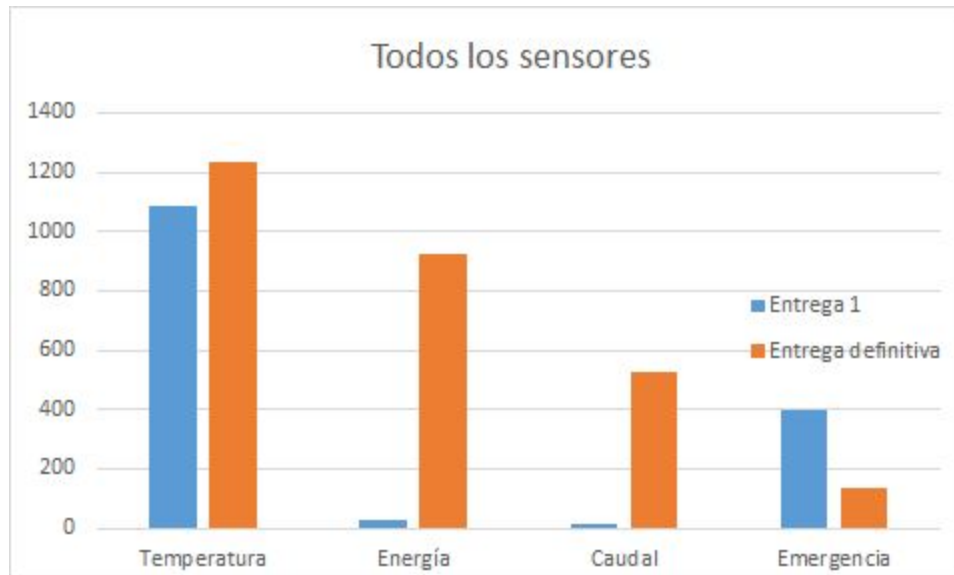
Etiqueta	#Muestras	Media(ms)	Desv. Estándar	Error
Temperatura	1200	1088	402,71	0,00%
Energía	1200	30	167,52	0,00%
Caudal	1200	14	82,83	0,00%
Emergencia	1200	401	141,6	0,00%
<b>Total</b>	<b>4800</b>	<b>383,25</b>	<b>198,665</b>	<b>0,00%</b>

Al realizar las pruebas con la implementación del microcontrolador, la capa de persistencia y el manejo de la información de los sensores, se observa un incremento en los resultados.

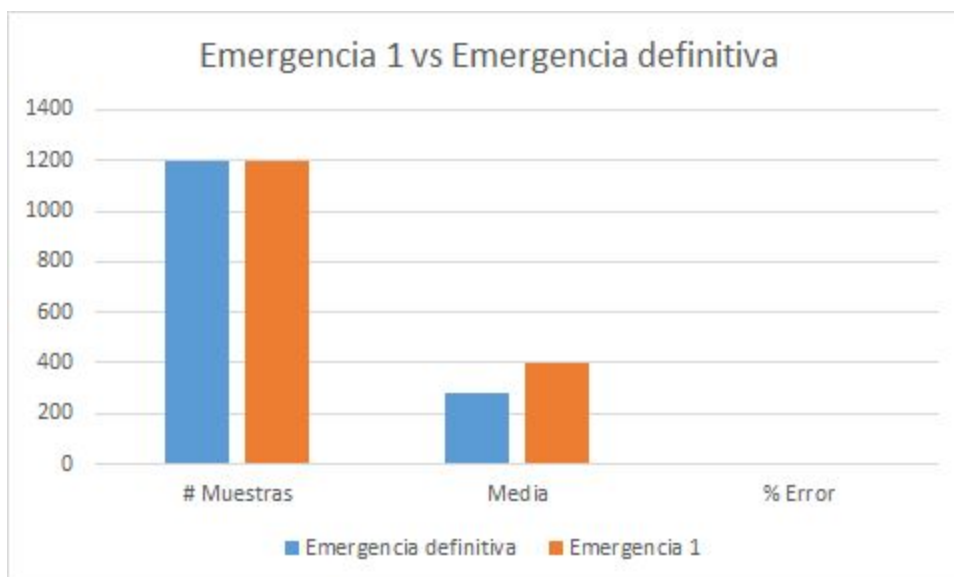
Etiqueta	# Muestras	Media	Mín	Máx	Desv. Estándar	% Error	Rendimiento	Kb/sec	Media de Bytes
Temperatura	1200	1235	47	2969	775.65	2.17%	272.4	74.88	281.5
Energia	1200	923	2	2968	561.15	0.25%	207.1	51.38	254.1
Caudal	1200	525	1	2071	416.16	0.00%	420.2	98.89	241
Emergencia	1200	135	1	846	163.07	0.00%	408.4	104.1	261
Total	4800	704	1	2969	671.3	0.60%	595.7	150.9	259.4

Gráficas:

Gráfica N° 1 Ejecución de todos los sensores:



Gráfica N° 2 Ejecución del servicio de mensaje de emergencia:



Existe una diferencia entre algunas de las variables (media, desviación estándar). Sin embargo el porcentaje de error se mantiene en 0,00% lo que indica una implementación correcta de la persistencia y envío de datos desde los sensores. Observando los resultados al añadir la capa de persistencia, se puede notar que el tiempo medio de respuesta, o latencia, es tiene datos parecidos. Sin embargo, para esta entrega esta medida aumentó en casi todos los servicios. Esto se da, gracias a que, al añadir la capa de persistencia y las relaciones con las entidades hay un mayor uso de recursos para cada petición. Aún así, es notable que se cumple con los escenarios de calidad estipulados con anterioridad y que el programa es capaz de responder

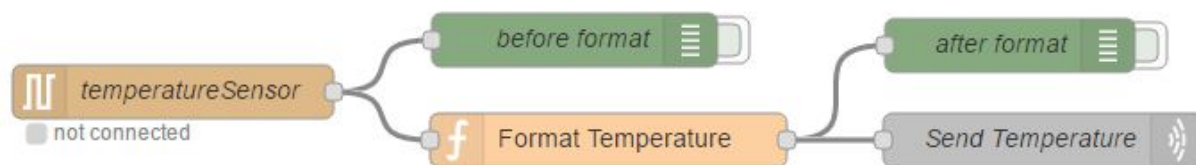
con las solicitudes para el caso de mayor concurrencia posible (uno de los picos) en un ambiente de ejecución sobrecargado. Este último escenario de calidad está asociado con el atributo de calidad de escalabilidad, puesto que se demuestra cómo responde el programa ante cambios abruptos en su ejecución.

Por otro lado, la ejecución para el servicio de reportar una emergencia de un pozo, fue tratado en esta entrega con mayor prioridad, por lo que se decidieron ajustar algunas tácticas de desempeño para mejorar su desempeño. En primer lugar, se aumentó la concurrencia para acceder la base de datos en el pool de AKKA. Segundo, se mejoró el acceso a la base de datos con el uso de índices y así mismo con mayor concurrencia. Gracias a esto, se alcanzó a reducir el tiempo promedio de respuesta, o latencia. Sin embargo, el cambio no fue tan significativo, gracias a que la capa de persistencia representa un mayor número de instrucciones por petición.

## Implementación del microcontrolador

Para consumir la lógica del negocio desarrollada hasta el momento desde un microcontrolador, se desarrollaron los siguientes flujos en node-red:

*Flujo de telemetría:*

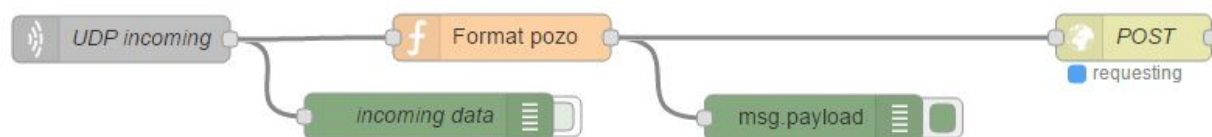


Se recibe la señal desde el microcontrolador, a la cual se le da el siguiente formato:

```
9/23/2016, 12:17:35 AM  after format  
currentTemp : msg.payload : string [27]  
{ "temperatura":20,"pozo":1 }
```

Dentro de esta misma función que da el formato, se tienen en cuenta las unidades en las que viene la temperatura para hacer la conversión y siempre retornar un número en grados celsius. Finalmente se simula un envío UDP.

*Flujo de procesamiento:*



Para procesar los datos que se “envían” desde el microprocesador, se tiene una función que da al mensaje el siguiente formato:

9/23/2016, 12:53:03 AM 8b0eb0f2.c155f

POST : msg.payload : Object

```
{ "temperatura": "20", "pozo": { "id": "1" } }
```

Este formato se da con el fin de modelar el cuerpo de la petición POST de HTTP que se enviará para guardar la información en la base de datos. A continuación se muestra la configuración que se le da al nodo HTTP que hace la petición:

**Edit http request node**

Cancel

Done

Method

POST

URL

http://localhost:9000/temperatura

☐ Enable secure (SSL/TLS) connection

☐ Use basic authentication

Return

a UTF-8 string

Name

POST

## Conclusiones

Como conclusión, se puede afirmar que para la solución deseada por OilCol se debe lograr un equilibrio entre el manejo de la base de datos y la forma en que se desarrolla la lógica. Los sensores de los pozos pueden llegar a generar un gran volumen de datos. Esta información no se puede perder, es decir, se debe guardar. Debido a esto se decidió escoger una base de datos SQL en Postgresql, que nos permitiera archivar esta gran cantidad de información y realizar consultas con esta. Pero se debe tener en cuenta que las diferentes consultas y la manera de desarrollarlas va a afectar el tiempo de respuesta del sistema. Por esto, al momento de diseñarlas e implementarlas, se decidió realizar consultas con el menor número de búsquedas posibles y se realizaron índices sobre las tablas. Es decir, al desarrollar una arquitectura de software, no es simplemente realizando lo que el cliente necesita, sino también lo que solicita.