Universidad Politécnica de Madrid
Escuela Técnica Superior de Ingenieros de Telecomunicación

DISEÑO, PROGRAMACIÓN Y DESPLIEGUE DE UN PROCESADOR PARA PERSISTENCIA DE DATOS DE CONTEXTO EN UNA BASE DE DATOS DISTRIBUIDA UTILIZANDO FIWARE-DRACO Y CASSANDRADB.

TRABAJO FIN DE MÁSTER

Giuseppe André Brando Ortega

2019

Universidad Politécnica de Madrid
Escuela Técnica Superior de Ingenieros de Telecomunicación

Máster Universitario en
Ingeniería de Redes y Servicios Telemáticos

TRABAJO FIN DE MÁSTER

DISEÑO, PROGRAMACIÓN Y DESPLIEGUE DE UN PROCESADOR PARA PERSISTENCIA DE DATOS DE CONTEXTO EN UNA BASE DE DATOS DISTRIBUIDA UTILIZANDO FIWARE-DRACO Y CASSANDRADB.

Autor
Giuseppe André Brando Ortega

Tutor
Andrés Muñoz Arcentales
Ponente
Álvaro Alonso González

Departamento de Ingeniería de Sistemas Telemáticos

2019

Resumen

El presente proyecto consiste en el diseño, programación e implementación de un procesador para la persistencia de datos aplicados para resolver las dificultades vinculadas a la gestión de grandes cantidades de datos que se centran en la recolección, almacenamiento, y análisis para las plataformas de ciudades inteligentes. Para ello todos los datos que provienen de los dispositivos IoT son recolectados por un componente de la plataforma FIWARE la cual se encarga de recibir los datos y entregarlos con formato NGSIv2, para luego ser clasificados y procesados haciendo uso de la herramienta FIWARE-DRACO mediante la implementación de un procesador que permite almacenar los datos en CassandraDB enmarcándose en los lineamientos establecidos en el estándar para persistencia de datos estructurados.

El desarrollo del presente trabajo se divide en varias etapas, donde se definen cada uno de los elementos que conforman la realización del procesador planteado. La primera etapa es el diseño y definición de las características del procesador que se desarrolla bajo el nombre NGSItoCassandra y se implementa haciendo uso de la herramienta de manipulación de datos Apache NiFi. Posteriormente a la creación del procesador, se analiza los datos de entrada y se los procesa para poder establecer la conexión a la base de datos CassandraDB con el objetivo de poder crear, actualizar, borrar o leer los datos definidos en la misma. Finalmente, se despliega un escenario de prueba con todos los componentes necesarios para determinar el correcto funcionamiento del software desarrollado.

Abstract

This project is about the design, programming and implementation of a processor for the persistence of applied data to solve the difficulties related to the management of large amounts of data that focus on the collection, storage and analysis for the smart cities' platforms. For achieving the above mentioned, the data that come from the IoT devices are collected by a component that belongs to the FIWARE platform, which is responsible of receiving the data and deliver it with a NGSIv2 format, for later to be classified and processed making use of the FIWARE-DRACO tool through the implementation of a processor that allows us to store the data in CassandraDB adjusting to the guidelines established in the persistent data structure standard.

The development of this work is divided in many stages, where are defined each of the elements that make up the setup of the proposed processor. The first stage is the design and definition of the characteristics of the processor that is developed under the NGSItoCassandra name and it's implemented using the Apache NiFi tool that will help us with the data manipulation. Later to the processor creation stage, the input data is analyzed and processed to establish the connection with the CassandraDB database in order to be able to create, update, delete or read the data defined in it. Finally, a test environment will be deployed with all the needed components to determine the right operation of the software developed.

Índice general

Índice de figuras

Siglas

NGSI	Next Generation Services Interface.
IoT	Internet of Things.
SOA	Service Oriented Architecture.
JVM	Java Virtual Machine.
CQL	Cassandra Query Language.
MQTT	Message Queue Telemetry Transport.
COAP	Constrained Application Protocol.
HTTP	Hypertext Transfer Protocol.
API	Application Programming Interface.

1 Introducción

En la actualidad gran porcentaje de las ciudades europeas priorizan la investigación sobre los diferentes procesos de transformación hacia nuevos modelos de gestión de los servicios, infraestructuras y sobre todo de la interacción de los ciudadanos con las nuevas tecnologías aplicadas para las ciudades inteligentes. Estos procesos, que durante los últimos años se ha desarrollado principalmente en el ámbito de la Investigación y Desarrollo están siendo ampliamente implantados en estas ciudades debido al gran impacto que generan estos nuevos servicios.[1]

Existen muchos factores tecnológicos que debido a sus prestaciones están dando lugar a esta transformación. Tecnologías como Cloud Computing permiten a entidades gubernamentales y locales, como ministerios y ayuntamientos, externalizar los servicios de infraestructuras y gestión. Otro de los factores es la capacidad de analizar y gestionar grandes cantidades de información (Big Data). Finalmente, el aumento de los teléfonos móviles inteligentes; pero, sobre todo, la conexión a internet y la posibilidad de comunicarse con todo tipo de dispositivos IoT.[2]

Para los primeros despliegues de estos servicios e infraestructuras en ciudades inteligentes se ha optado por soluciones propias o soluciones específicas sobre servicios como: gestión de movilidad, medio ambiente, tráfico, etc. Aunque estas soluciones permiten obtener resultados eficaces a estos problemas, existen factores que dificultan su despliegue en escenarios diferentes. Es decir, las soluciones son independientes de cada ciudad debido a que los recursos son diferentes y cada una enfoca sus servicios de acuerdo con sus propias necesidades. Como consecuencia, se genera el problema de la inexistencia de ecosistemas globales. Debido a ello, se hace imperativa la necesidad de realizar un estándar o un modelo para la obtención y compartimiento de datos para que las ciudades puedan hacer uso de él y aprovechar las ventajas que este modelo les provee y de esta forma poder potenciar el cambio tecnológico que se está dando en la actualidad. De esta forma, se puede llegar a generar múltiples entornos colaborativos en las que los emprendedores pueden ver los proyectos de ciudades inteligentes como una plataforma para el desarrollo de aplicaciones y servicios que se puedan implantar en múltiples ciudades.

Una de las plataformas que han conseguido posicionarse como soluciones innovadoras para el mundo del Internet de las cosas es FIWARE. Esta plataforma de software abierto nace de la colaboración público-privada entre la Comisión Europea y la industria europea en 2011. Es una plataforma de software que se ha creado en un ecosistema en donde los emprendedores o empresas pequeñas pueden materializar sus ideas y construir aplicaciones/servicios en la nube de manera eficaz y sencilla. [3][4]

FIWARE es una plataforma de propósito general, pero resulta de especial interés en el ámbito de las ciudades inteligentes. Una de sus aplicaciones es FIWARE-DRACO que es un mecanismo alternativo de persistencia para administrar datos de contexto. Está implementada en Apache NiFi, la cual es una herramienta fundamentada en los conceptos de programación de flujos de datos y que admite diagramas de gran alcance para automatizar dicho flujo entre sistemas. Aunque el término 'flujo de datos' se usa en una variedad de contextos, en este trabajo se lo usa para indicar el flujo de información automatizado y administrado entre sistemas.

El objetivo principal de este trabajo es usar la plataforma FIWARE-DRACO para poder diseñar, programar y desplegar un procesador para persistencia de datos. Para ello se utilizan las prestaciones de otro componente de la plataforma FIWARE el cual obtiene eventos representados como datos de contexto acordes con el estándar NGSIv2. Luego, por medio de las librerías que provee Apache NiFi, se implementa el procesador de persistencia de datos, cuya función no es únicamente insertar datos, sino también crear los Keyspaces y tablas de forma dinámica en el gestor de base de datos CaassandraDB. Todo este proceso se realiza manteniendo como norma general los lineamientos que el estándar ha definido para este tipo de procesos.

1.1 Motivación

En los últimos años, es cada vez más común encontrar nuevas tecnologías que incluyen el análisis de grandes cantidades de datos para la toma de decisiones sobre un servicio en específico. Por ejemplo, el análisis que conlleva a la evolución a ciudades inteligentes da paso a la investigación de nuevas tecnologías que pudieran implementar sistemas que cumplan los requisitos de estas y brinda la oportunidad que entidades públicas o privadas inviertan en infraestructura y desarrollo de sistemas.

Las dificultades más habituales vinculadas a la gestión de estas cantidades de datos se centran en la recolección, almacenamiento, búsqueda, compartición, análisis, y visualización. La tendencia a manipular enormes cantidades de datos se debe a la necesidad, en muchos casos, de incluir dicha información para la creación de informes estadísticos y modelos predictivos utilizados en diversas materias, como los análisis de negocio, los datos de enfermedades infecciosas y seguimiento a la población.[5]

En la actualidad los análisis de Big Data son recursos que están teniendo mayor demanda para nuevos proyectos como ciudades inteligentes. Estos proyectos requieren que posteriormente a la captura de datos, estos puedan ser procesados, analizados y almacenados para la toma de decisiones que influirá en la sociedad de las ciudades que los implementan.

Teniendo en cuenta esta problemática, que surge con la forma en que se realiza la recolección, almacenamiento y análisis de datos para la implementación de servicios en ciudades inteligentes, este proyecto, pretende diseñar e implementar un procesador de persistencia de datos basado en la plataforma de FIWARE-DRACO que servirá para el proceso de análisis y toma de decisiones de los servicios.

1.2 Objetivo General

Implementar un procesador para persistencia de datos de contexto utilizando FIWARE-DRACO que se encargará de recibir datos con formato NGSIv2 para luego procesarlos y almacenarlos de forma dinámica en CassandraDB enmarcándose en los lineamientos establecidos en el estándar para persistencia de datos estructurados.

1.3 Objetivo Específicos.

- Definir la arquitectura para el funcionamiento del procesador.
- Desarrollar un programa que establezca la conexión y permita la escritura de los datos procesados CassandraDB.
- Implementar un escenario de prueba y validar el funcionamiento del procesador de persistencia de datos.
- Contribuir al proyecto de código abierto FIWARE-DRACO permitiendo proveer de herramientas adicionales para el análisis de datos de contexto en Ciudades Inteligentes.

2 Estado del Arte

En el contexto de este capítulo se describe el estado de arte de cada una de las técnicas y tecnologías utilizadas para la implementación de este trabajo. En la primera sección se hace énfasis en el concepto, características y componente de FIWARE-DRACO, que es un sistema fácil de usar y confiable para el proceso y la distribución de datos. Posteriormente, se tiene un apartado describiendo la aplicación Apache NiFi en la cual se desarrolla el procesador NGSItoCassandra. Finalmente, se tiene un resumen sobre la arquitectura y funcionamiento del gestor de base de datos CassandraDB.

2.1 FIWARE-DRACO

FIWARE es una iniciativa de código abierto que define un gran conjunto de estándares para la gestión de datos de contexto que permite el desarrollo de soluciones para diferentes dominios como Ciudades Inteligentes, Industria Inteligente y Energía Inteligente. [6]

En toda solución inteligente, existe la necesidad de recopilar y administrar información de contexto, procesar esa información e informar a los actores externos. El componente central de cualquier plataforma "FIWARE" es Context Broker el cual permite al sistema realizar actualizaciones y acceder al estado actual del contexto.

FIWARE-DRACO es un sistema fácil de usar y confiable para procesar y distribuir datos. Internamente se basa en Apache NiFi, el cual es un sistema de flujos de datos que se fundamentan en los conceptos de programación de flujo. Además, admite gráficos de gran alcance y escalables de enrutamiento de datos, transformación y lógica de mediación del sistema. Fue construido para automatizar el flujo de datos entre sistemas. El término 'flujo de datos' es usado en una variedad de contextos y aplicaciones, sin embargo, en el presente trabajo se lo usa para indicar el flujo de información automatizado y administrado entre sistemas.[7]

2.1.1 Interacción de FIWARE DRACO.

En esta subsección se va a detallar la interacción de los componentes de FIWARE-DRACO que requiere de un procesador de persistencia de datos, como Orion y un sistema de almacenamiento distribuido como Cassandra, Mongo o incluso puede interactuar con MySQL que es un sistema de almacenamiento no distribuido.

El conjunto de procesadores que componen FIWARE-DRACO, y que están a cargo de la persistencia, permiten crear una vista histórica de los datos. Para esto utiliza el Context broker Orion, el cual es una implementación en C++ de la vinculación de la API REST NGSIv2 desarrollada como parte de la plataforma FIWARE. Orion permite administrar todo el ciclo de vida de la información de contexto, incluidas las actualizaciones, consultas, registros y suscripciones. Es una implementación del servidor NGSIv2 para administrar la información de contexto y su disponibilidad; para luego dicha información pueda ser almacenada en sistemas de almacenamiento.[8]

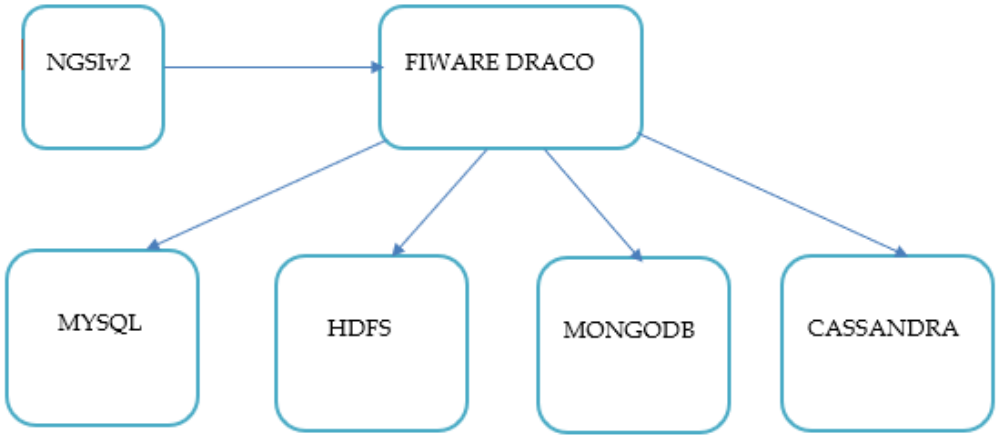

Figura 1. Esquema de Interacción de FIWARE-DRACO

2.1.2 Componentes.

En la subsección anterior se mostraron los elementos que conforman el esquema de interacción de FIWARE-DRACO, dando paso a que en este apartado se mencione brevemente los componentes de la que complementa la arquitectura mostrada y que permiten gestionar a nivel de configuración las diferentes características que provee este tipo de tecnología. [9]

FlowFile: los datos que el usuario trae a NiFi para su procesamiento y distribución se conoce como FlowFile. Este se compone de dos partes: Atributos y Contenido. El contenido es el propio dato del usuario y los atributos son pares (clave-valor) que están asociados con los datos por cada usuario.

Procesador: el procesador es el componente NiFi que se encarga de crear, enviar, recibir, transformar y procesar archivos de flujo de datos.

Relación: estas relaciones se nombran para indicar el resultado de procesar un FlowFile, cada procesador puede tener cero o más relaciones definidas. Después de que un procesador haya terminado de procesar un FlowFile, enrutará (o "enviará") el mismo a una de las relaciones. Luego, un DFM puede conectar cada una de estas relaciones a otros componentes para especificar dónde debe ir el archivo de flujo.

Conexión: un DFM crea un flujo de datos automatizado y luego conecta los componentes a través de Conexiones. Cada conexión consiste en una o más relaciones. Para cada conexión, un DFM puede determinar qué relaciones deben usarse para la conexión. Esto permite que los datos se enruten de diferentes maneras en función de su resultado de procesamiento. Cada conexión alberga una cola de FlowFile. Cuando un FlowFile se transfiere a una Relación particular, se agrega a la cola que pertenece a la Conexión asociada.

Servicio del controlador: los Servicios del controlador son puntos de extensión que, después de que se agregue y configure un DFM en la Interfaz del usuario, se iniciarán cuando se inicie NiFi y brindarán información para el uso de otros componentes (como procesadores u otros servicios del controlador).

2.2 Apache Nifi

NiFi fue construido para automatizar el flujo de datos entre dos o más sistemas debido a la problemática que ha existido desde que las empresas comenzaron a tener más de un sistema, donde estos creaban datos y otros consumían datos. [10]

2.2.1 Características

A lo largo de los años, el flujo de datos ha sido uno de esos males necesarios en una arquitectura. En la actualidad, hay una serie de movimientos activos y en rápida evolución que hacen que el flujo de datos sea mucho más interesante y más vital para el éxito de una empresa determinada. Estos incluyen paradigmas como SOA, IoT y Big Data. Además, el nivel de rigor necesario para el cumplimiento, la privacidad y la seguridad aumenta constantemente. Incluso con todos estos nuevos conceptos, los patrones y las necesidades del flujo de datos siguen siendo en gran medida los mismos. Las diferencias principales son el alcance de la complejidad, la tasa de cambio necesaria para adaptarse, y que a mayor escala el caso extremo se convierte en algo común. NiFi está diseñado para ayudar a enfrentar estos desafíos modernos de flujo de datos.[11]

Algunos de los desafíos de flujo de datos que NiFi ayuda a resolverlos son:

- **Falla de sistemas.**

  Las redes fallan, los discos fallan, el software falla.

- **El acceso a los datos supera la capacidad de consumo.**

  A veces, un determinado origen de datos puede superar alguna parte de la cadena de procesamiento o entrega; solo se necesita un enlace débil para tener un problema.

- **Las condiciones de contorno.**

  Siempre obtendrás datos demasiado grandes, demasiado pequeños, demasiado rápidos, demasiado lentos, corruptos, incorrectos o en el formato incorrecto.

- **Los sistemas evolucionan a diferentes ritmos.**

  Los protocolos y formatos utilizados por un sistema determinado pueden cambiar en cualquier momento y, a menudo, independientemente de los sistemas que los rodean.

- **Cumplimiento y seguridad.**

  Las leyes, regulaciones y políticas cambian. Las interacciones de sistema a sistema y de sistema a usuario deben ser seguras, confiables y responsables.

2.2.2 Arquitectura.

Para lograr identificar las ventajas del uso de Apache NiFi, es necesario hacer una breve descripción de la arquitectura, además de mostrar las componentes principales que hacen posible el funcionamiento y el despliegue de servicios. [12]

NiFi se ejecuta dentro de una JVM en el sistema operativo anfitrión. Los componentes principales de NiFi en la JVM son los siguientes:

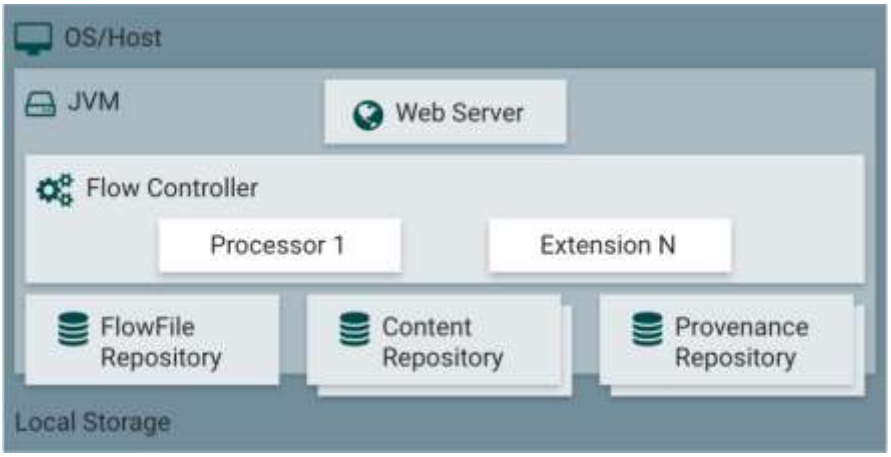

Figura 2. Arquitectura de Apache NiFi

- **Servidor web.**

  El propósito del servidor web es alojar la API de comando y control basada en HTTP de NiFi.

- **Controlador de flujo.**

  El controlador de flujo es el cerebro de la operación. Proporciona subprocesos para que se ejecuten las extensiones y administra la programación de cuándo las extensiones reciben recursos para ejecutar.

- **Repositorio FlowFile.**

  El repositorio de FlowFile es donde NiFi realiza un seguimiento del estado de lo que sabe sobre un FlowFile dado que actualmente está activo en el flujo.

- **Repositorio de contenido.**

  El Repositorio de contenido es donde se encuentran los bytes de contenido real de un FlowFile dado. La implementación del repositorio es conectable. El enfoque predeterminado es un mecanismo bastante simple, que almacena bloques de datos en el sistema de archivos. Se puede especificar más de una ubicación de almacenamiento del sistema de archivos para que se ocupen diferentes particiones físicas a fin de reducir la contención en un solo volumen.

- **Repositorio de procedencias.**

  El repositorio de procedencias es donde se almacenan todos los datos de eventos de procedencias. La construcción del repositorio es conectable con la implementación predeterminada para usar uno o más volúmenes de disco físico. Dentro de cada ubicación, los datos de eventos se pueden indexar y buscar.

2.2.3 Componentes.

NiFi proporciona varios puntos de extensión para brindar a los desarrolladores la capacidad de agregar funcionalidad a la aplicación que satisfagan sus necesidades. La siguiente lista proporciona una descripción de alto nivel de los puntos de extensión más comunes:

- **Procesador:**
  La interfaz del procesador es el mecanismo a través del cual NiFi expone el acceso a FlowFiles, sus atributos y su contenido. El procesador es el bloque de construcción básico utilizado para comprender un flujo de datos de NiFi. Esta interfaz se utiliza para realizar todas las tareas siguientes: [13]
  - Crear archivos de flujo.
  - Leer el contenido de FlowFile.
  - Escribir contenido de FlowFile.
  - Leer los atributos de FlowFile.
  - Actualizar los atributos de FlowFile.
  - Toma de datos.
  - Datos de egreso.
  - Datos de ruta.
  - Extraer datos.
  - Modificar datos.

ReportingTask.

La interfaz ReportingTask es un mecanismo que expone NiFi para permitir que las métricas, la información de monitoreo y el estado interno de NiFi se publiquen en puntos finales externos, como archivos de registro, correo electrónico y servicios web remotos.

- **ControllerService.**
  Proporciona un estado y una funcionalidad compartidos entre los procesadores, otros ControllerServices y ReportingTasks dentro de una única JVM. Un ejemplo de caso de uso puede incluir cargar un conjunto de datos muy grande en la memoria. Al realizar este trabajo en un ControllerService, los datos pueden cargarse una vez y exponerse a todos los procesadores a través de este servicio, en lugar de requerir que muchos procesadores diferentes carguen el conjunto de datos por sí mismos.

- **FlowFilePrioritizer.**
  La interfaz FlowFilePrioritizer proporciona un mecanismo por el cual los archivos de flujo en una cola pueden ser priorizados y ordenados para que los archivos de flujo puedan ser procesados en el orden más efectivo para un caso de uso particular.

- **AuthorityProvider.**

  Un AuthorityProvider es responsable de determinar qué privilegios y roles debe otorgarse a un usuario determinado.

2.3 Cassandra DB

Apache Cassandra es una base de datos NoSQL de código abierto masivamente escalable. Cassandra es perfecta para administrar grandes cantidades de datos estructurados, semiestructurados y no estructurados a través de múltiples centros de datos y de la nube. También ofrece disponibilidad continua, escalabilidad lineal y simplicidad operativa en muchos servidores de productos básicos sin un punto único de falla, junto con un poderoso modelo de datos dinámico diseñado para la máxima flexibilidad y tiempos de respuesta rápidos.[14]

2.3.1 Características

La base de datos de Apache Cassandra es la mejor opción cuando se necesita implementar soluciones que requieran de escalabilidad y alta disponibilidad sin comprometer el rendimiento. La escalabilidad lineal y la tolerancia a fallas en el hardware básico o la infraestructura en la nube la convierten en la plataforma perfecta para datos de misión crítica. El soporte de Cassandra para la replicación en múltiples centros de datos es el mejor de su clase, brindando una menor latencia para sus usuarios.[15]

Actualmente está en uso en muchos proyectos y organizaciones como: Constant Contact, CERN, Comcast, eBay, GitHub, GoDaddy, Hulu, Instagram, Intuit, Netflix, Reddit, The Weather Channel y más de 1500 compañías que implementan servicios de big data cumpliendo a cabalidad con las exigencias que requieren estos servicios; debido a esto es una de las bases de datos más utilizadas y tiene las siguientes características:[16]

- **TOLERANTE A FALLOS.**

  Los datos se replican automáticamente en uno o en varios nodos para la tolerancia a fallos y se admite la replicación en diferentes centros de datos. Los nodos que por alguna situación dejan de funcionar se pueden reemplazar sin tener configurado un tiempo de inactividad.

- **DESEMPEÑO.**

  Cassandra supera alternativas populares de NoSQL en los puntos de referencia y las aplicaciones reales, principalmente debido a las elecciones basadas en su arquitectura.

- **DESCENTRALIZADO.**

  Cada nodo en el clúster es idéntico y no existen cuellos de botella en la red.

- **ESCALABLE.**

  Algunas de las implementaciones de producción más grandes como: Apple, con más de 75,000 nodos que almacenan más de 10 PB de datos, Netflix (2,500 nodos, 420 TB, más de 1 billón de solicitudes por día), el motor de búsqueda chino Easou (270 nodos, 300 TB, más de 800 millones solicitudes por día) y eBay (más de 100 nodos, 250 TB) todas estas aplicaciones utilizan cassandra.

- **PERSISTENTE.**

  Cassandra es ideal para aplicaciones que no pueden permitirse perder datos, incluso cuando un centro de datos completo deja de funcionar.

- **CONTROL DE REPLICACION.**

  Tiene el control para elegir entre la replicación síncrona o asíncrona para cada actualización tomando en cuenta que las operaciones asíncronas de alta disponibilidad están optimizadas con funciones como Hinted Handoff y Read Repair.

- **FLEXIBLE.**

  El rendimiento de lectura y escritura aumenta linealmente a medida que lo requiera el servicio y vaya creciendo la cantidad de máquinas, sin interrupciones ni interrupciones en las aplicaciones.

- **Distribuida.**

  Es distribuida, lo quiere decir que la información está repartida a lo largo de los nodos del cluster.

- **APOYO PROFESIONAL.**

  Brinda un buen servicio de soporte de Cassandra los cuales están disponibles a través de terceros.

2.3.2 Arquitectura

Cassandra emplea un sistema distribuido punto a punto donde todos los nodos son los mismos y los datos son distribuidos a través de todos los nodos en el cluster.

Implementa una arquitectura Peer-to-Peer, que elimina todos los puntos de fallo y no declara patrones maestro-esclavo como otros sistemas de almacenamiento. De esta manera cualquier nodo puede tomar el rol de coordinador de una query. Será el sistema que decida y elija el nodo que hará el papel de coordinador. Los datos son repartidos a lo largo del cluster en base a un token único calculado para cada fila por una función hash. [17]

Los nodos se reparten equitativamente el posible rango de tokens que va de -263 a 263, este método selecciona el nodo primario. Internamente Cassandra replicará los datos entre todos los nodos con la política definida, por ejemplo, definiendo el factor de replicación. Además, soporta el concepto de data center para agrupar los nodos lógicamente y tener los datos más cerca del usuario.

El lenguaje de acceso a datos en Cassandra es CQL, es un derivado reducido de SQL. En Cassandra los datos están desnormalizados de manera que el concepto de joins o subqueries no existe.

Cassandra determina una estrategia para poder realizar la replicación de la información la cual está definida de la siguiente forma:

- **Estrategia simple**

  Permite definir un único factor de replicación. Esto determina el número de nodos que deben contener una copia de cada fila. Por ejemplo, si el factor de replicación es 3, entonces tres nodos diferentes deben almacenar una copia de cada fila.

  Esta estrategia trata de que todos los nodos de forma idéntica, ignorando cualquier centro de datos o racks configurados. Para determinar las réplicas de un rango de token, Cassandra itera a través de los tokens en la red, comenzando con el rango de interés del token. Para cada token, comprueba si el nodo propietario se ha agregado al conjunto de réplicas y, si no lo ha hecho, se agrega al conjunto. Este proceso continúa hasta que se agreguen nodos distintos factores de replicación al conjunto de réplicas.

3 Diseño e implementación.

Para llevar a cabo el diseño y la implementación del Procesador NGSItoCassandra se deben cumplir con ciertas especificaciones establecidas previamente, las cuales permiten el buen funcionamiento del proyecto y se detallan a continuación:

- Para el diseño e implementación del procesador NGSItoCassandra se utiliza el software Apache NiFi.
- Lenguaje de Programación "JAVA".
- El desarrollo de software es en "IntelliJ IDEA".
- Para el almacenamiento de datos se necesita CassandraDB.
- El proyecto se desarrolla en un ordenador con sistema operativo Windows, pero sin embargo puede ser implementado en otros sistemas operativos ya sea Linux o en Mac Os.

En este capítulo se presenta la arquitectura utilizada para el desarrollo de este proyecto, presentando así cada uno de los componentes involucrados. Además de las tecnologías descritas anteriormente en el capítulo de estado del arte y se describe como la integración de ellas, permite el despliegue en el escenario para la ejecución y pruebas del sistema.

3.1 Componentes

Para la realización de este proyecto se ha separado en dos componentes importantes como es la implementación del entorno gráfico la cual está realizada en Apache NiFI y por otra parte el desarrollo del software bajo el lenguaje de programación "JAVA" en el software IntelliJ IDEA.

3.1.1 Apache NiFi

Como se mencionó anteriormente en la herramienta Apache NiFi se desarrolla el entorno grafico del proyecto, para la creación de este se realiza los siguientes pasos:

3.1.1.1 Instalación de Apache NiFi

Para la instalación de Apache NiFi en Windows se debe de seguir los siguientes pasos:

- Descargar la herramienta Apache NiFi desde la página oficial (https://nifi.apache.org/download.html).
- Seleccionar la herramienta de acuerdo con el sistema operativo en donde se desea instalar la herramienta.
- Una vez descargado el fichero se debe de instalar esto aproximadamente toma 5 minutos, luego de este tiempo se debe de revisar en los ficheros de configuración que como puerto de acceso a la aplicación web del programa este el puerto:8080.

3.1.1.2 Entorno de Apache NiFi

Una vez instalado Apache NiFi, el proyecto se realiza sobre sobre la instancia predeterminada DRACO que se puede acceder a la interfaz del usuario que normalmente se ejecuta en el puerto 8080 http://localhost: 8080/nifi e identificar sus componentes principales. La figura 3 proporciona la ubicación de muchos componentes de Draco.

Figura 3. Entorno de Apache NiFi

Luego de identificar las componentes de NiFi se procede a describir los dos más importantes que son el procesador y el controlador. El procesador es el componente más utilizado, ya que es responsable del ingreso, enrutamiento y manipulación de datos. Hay muchos tipos diferentes de procesadores. De hecho, este es un Punto de Extensión muy común en NiFi, lo que significa que muchos proveedores pueden implementar sus propios Procesadores para realizar cualquier función que sea necesaria para su caso de uso. Cuando se desea implementar un Procesador, se presenta un cuadro de diálogo para elegir qué tipo de procesador usar. Aquí puede comenzar a encontrar algunos de los procesadores FIWARE creados para administrar eventos NGSI y almacenar los datos en diferentes sumideros. Solo necesita escribir en el campo de filtro la palabra clave FIWARE o NGSI y se mostrará la lista de procesadores de FIWARE como se observa en la figura 4.

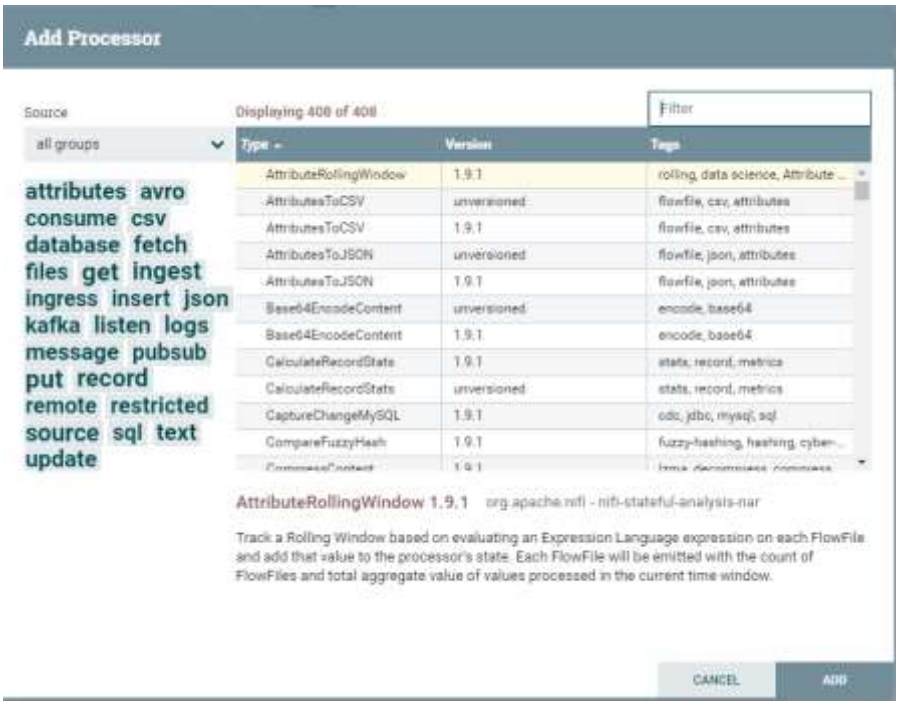

Figura 4. Procesadores predeterminados en Apache NiFi

Los controladores juegan un papel importante en Draco. Este componente se encarga de administrar todas las funciones necesarias para usar diferentes fuentes de almacenamiento de datos externos. En este caso, se usan controladores para establecer y configurar la conexión entre Draco y el almacenamiento de terceros.

Para agregar un Servicio de Controlador para un flujo de datos, se debe de hacer clic con el botón derecho en cualquier lugar del espacio de usuario de Draco y seleccionar Configurar, o hacer clic en Configurar en la Panel de operación como se muestra en la figura 5.

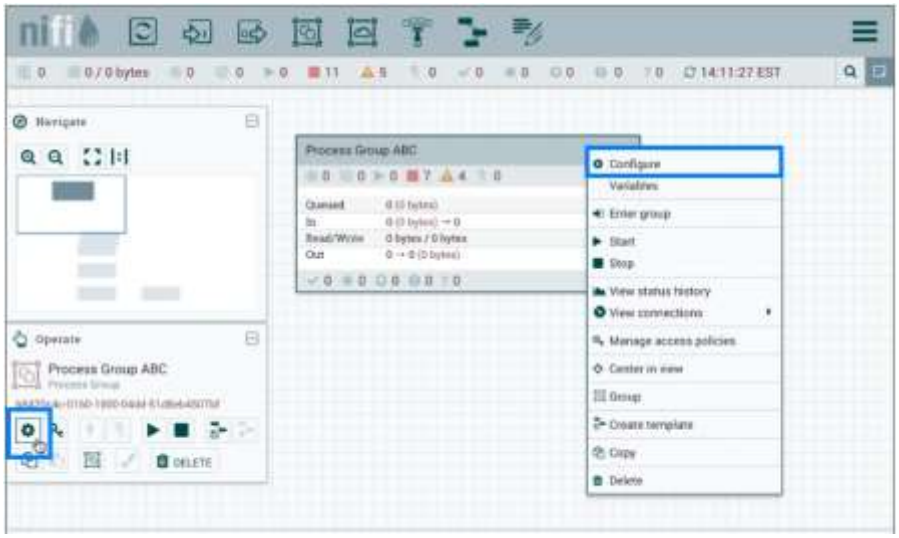

Figura 5. Controladores en Apache NiFi

Para poder agregar un servicio de controlador se debe de seguir lo siguientes pasos:

Hacer clic en Configurar, ya sea desde el panel de control, o desde cualquier lugar del espacio de usuario de Draco. Esto muestra la ventana de configuración del grupo de procesos. La ventana tiene dos pestañas: General y Servicios de Controlador como se muestra en la figura 6.

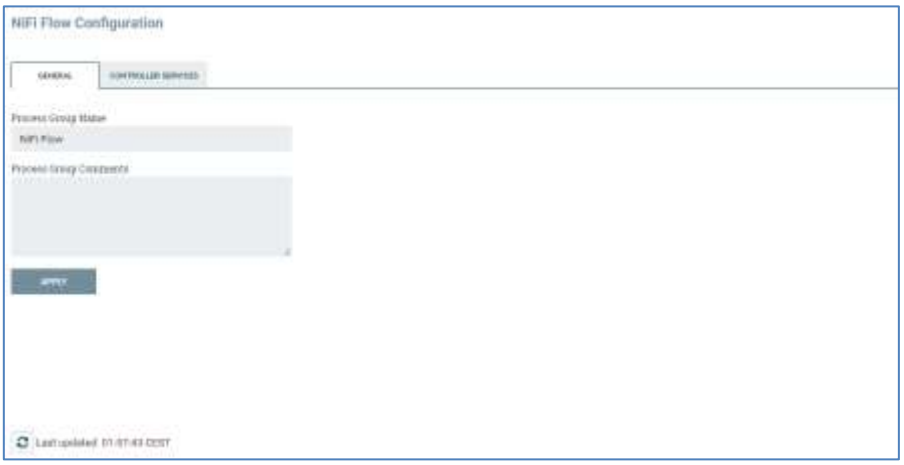

Figura 6. Configuración General de Controladores

La pestaña de servicios de controladores es donde se puede seleccionar los diferentes tipos de controladores; para el presente proyecto se elige el controlador de CassandraSessionProvider como se observa en la figura 7.

Figura 7. Lista de Controladores de Servicios

Los controladores tienen dos opciones muy importantes que se detallan a continuación: Configuración y Habilitar.

- Configuración.

  Para poder mostrar la configuración del controlador se debe accionar el icono ⚙ el cual presenta una ventana donde se puede observar las características, propiedades e incluso comentarios del controlador como se ilustra en la figura 8.

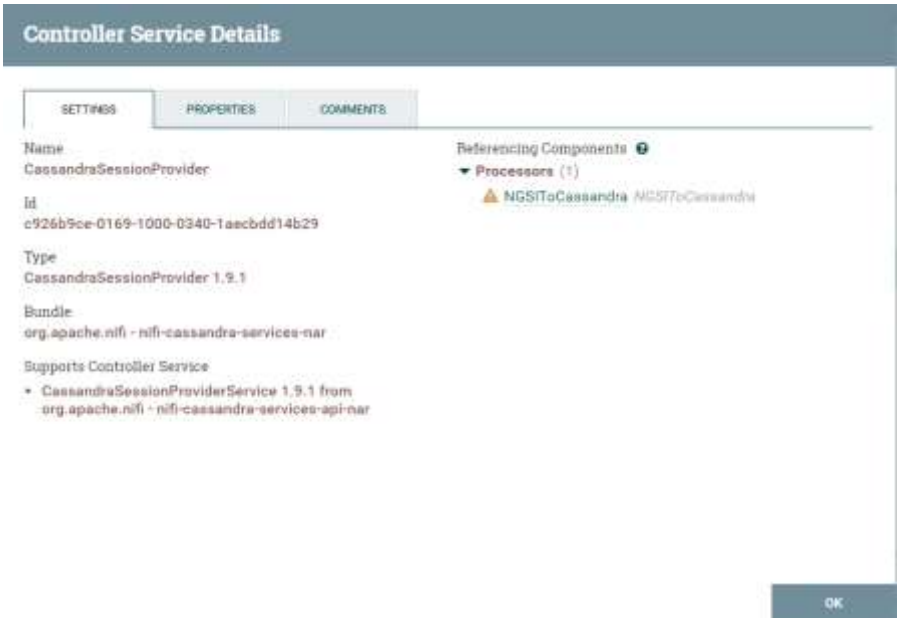

Figura 8. Detalles de Controladores de Servicios

- Habilitar:

  Después de configurar un Servicio de Controlador, debe habilitarse para poder ejecutarse. Esto se lo puede hacer con el botón ⚡ que se encuentra en la columna del extremo derecho de la pestaña Servicios del controlador como se muestra en la figura 9.

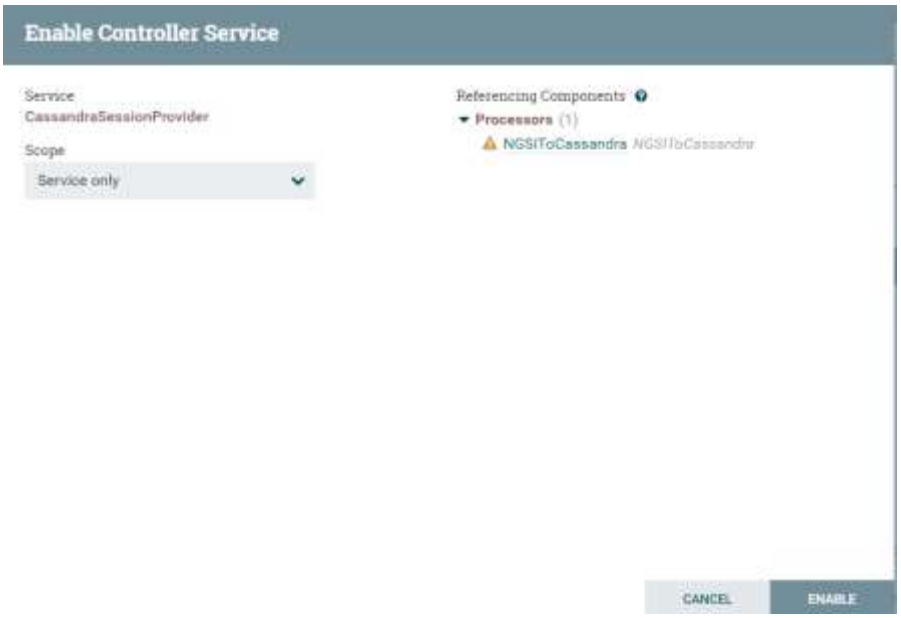

Figura 9. Habilitar un Controlador de Servicios

- Deshabilitar:

  Para modificar un servicio de controlador existente o en ejecución, el DFM necesita detenerlo o deshabilitarlo (así como todas las tareas de informes de referencia y los servicios del controlador). Esto se puede hacer usando el botón Desactivar  .

Figura 10. Deshabilitar un Controlador de Servicios

Por último, una vez que los procesadores y otros componentes se hayan agregado al espacio de usuario y se hayan configurado, el siguiente paso es conectarlos entre sí para que NiFi sepa qué hacer con cada FlowFile después de que se haya procesado. Esto se logra creando una conexión entre cada componente el icono para poder crear dicha conexión es .

Este icono se encuentra sobre cada procesador como se puede observar a continuación:

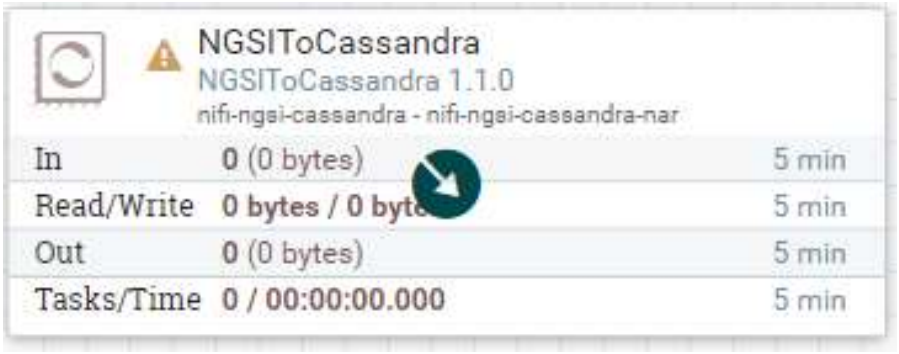

Figura 11. Procesador NGSItoCassandra

Para poder establecer una conexión se tiene que arrastrar el icono hacia el otro procesador, luego se abre una ventana indicando que se desea crear la conexión y al dar clic en el boto añadir se crea la conexión.

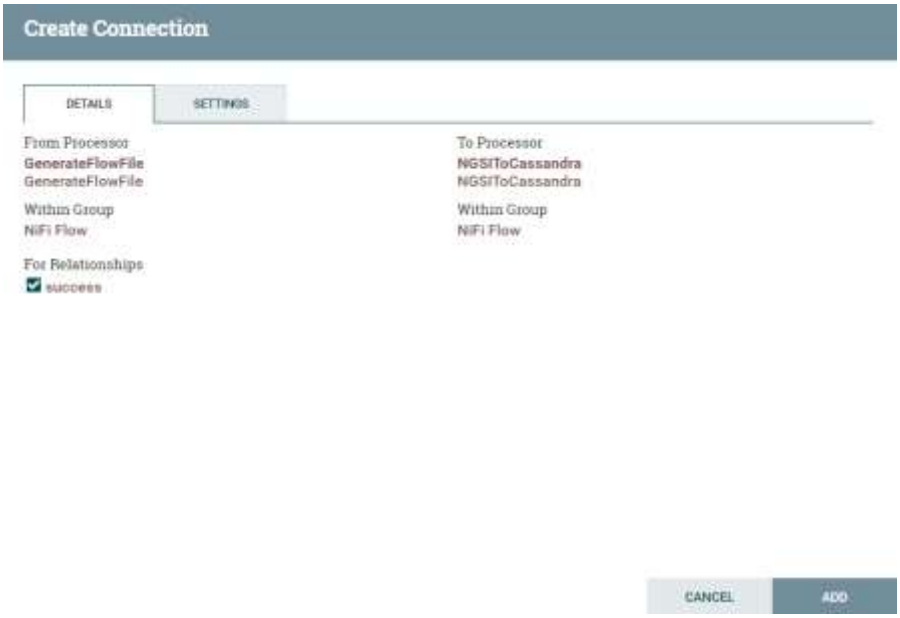

Figura 12. Crear Conexión entre dos procesadores

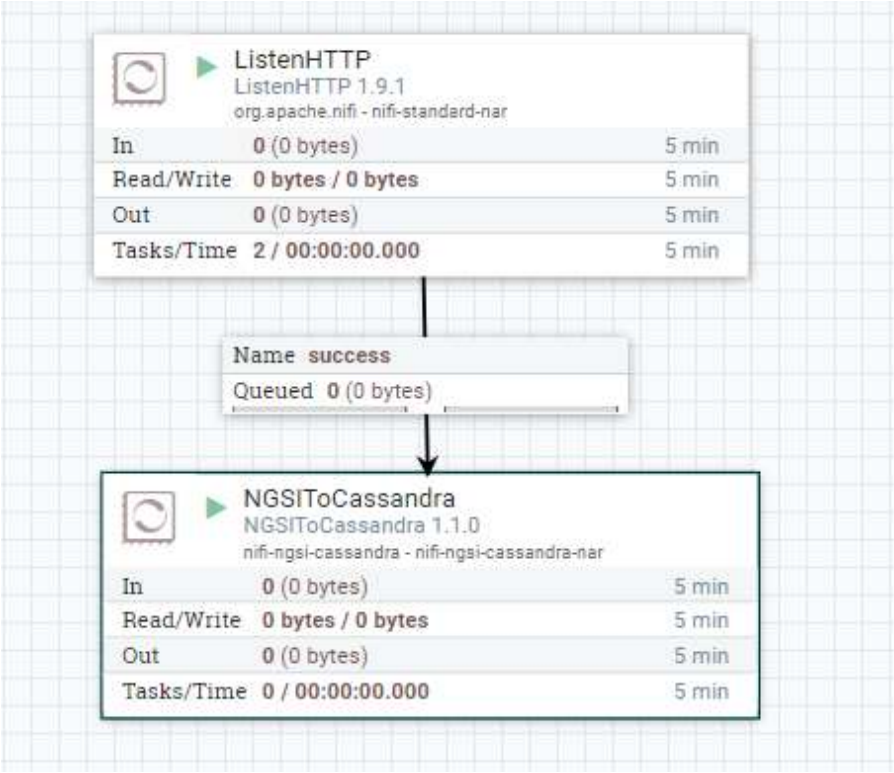

Figura 13. Conexión Creada entre dos procesadores

3.1.1.3 Configuración del procesador NGSItoCassandra

En esta subsección se detalla la configuración del procesador NGSItoCassandra el cual toma el archivo FLowfile generado por el procesador ListenHTTP y lo transforma en un objeto NGSI. Después, este procesador se comunicará con el servicio del controlador Cassandra Connection Provider (establecido en las propiedades del procesador). Finalmente, el procesador, utilizando las características del controlador, crea la base de datos, las tablas e inserta los datos en la base de datos Cassandra utilizando la estructura definida por el estándar NGSI.

A continuación, se detallan las propiedades establecidas por el procesador las cuales están basadas de las características de CassandraDB.

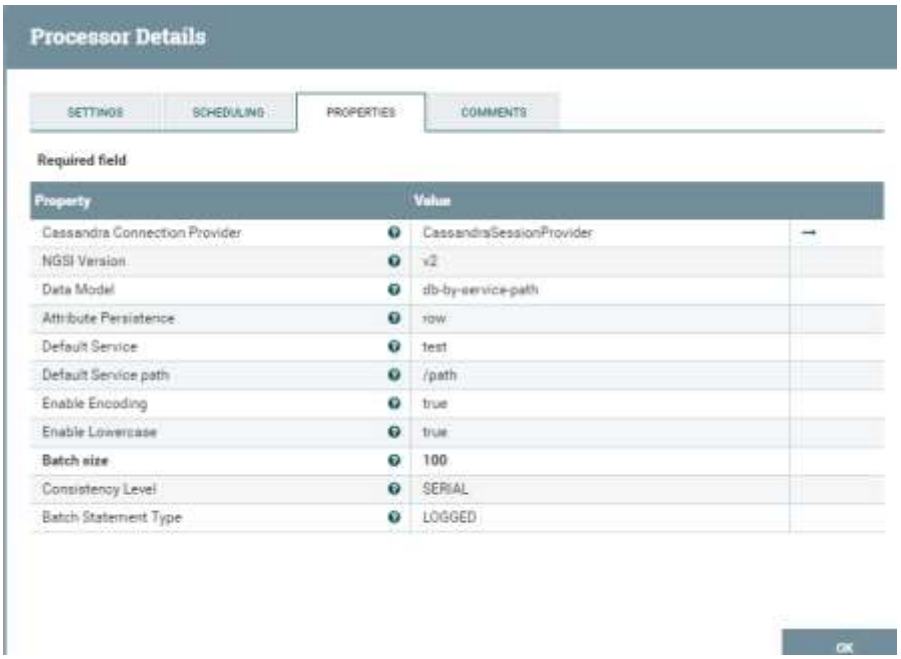

Figura 14. Propiedades del procesador NGSItoCassandra

Name	Default Value	Allowable Values	Description
Cassandra Connection Provider			Especifica la conexión de Cassandra que proporciona el servicio de controlador que se utilizará para conectarse al clúster de Cassandra.
NGSI Version	**v2**	**v2**	La versión de NGSI de sus eventos de entrada.
Data Model	db-by-service-path	db-by-service-path/ db-by-entity	El modelo de datos para crear las tablas cuando se ha recibido un evento puede elegir entre: db-by-service-path o db-by-entity, el valor predeterminado es db-by-service-path
Attribute Persistence	row	row	El modo de almacenar los datos dentro de la tabla.

Default Service	test		Servicio Fiware predeterminado para construir el nombre de la base de datos.
Default Service path	/path		Fiware ServicePath predeterminado para construir el nombre de la tabla.
Enable Encoding	true	True false	verdadero aplica la nueva codificación, falso aplica la antigua codificación.
Enable Lowercase	true	True false	verdadero o falso, verdadero para crear el nombre del esquema y las tablas en minúsculas.
Batch size	100		Especifica el número de 'Insertar declaraciones' que se agruparán para ejecutarse como un lote (BatchStatement).
Consistency Level	ONE		La estrategia de cuántas réplicas debe responder antes de que se devuelvan los resultados.
Batch Statement Type			Especifica el tipo de 'Declaración de lote' que se utilizará.

3.2 Implementación del procesador NGSItoCassandra

En esta sección se describe cada uno de los componentes necesarios para poder realizar la implementación del procesador NGSItoCassandra descrito en los capítulos anteriores. Se toma como punto de partida la descripción del escenario de prueba describiendo cada uno de los parámetros.

El entorno de simulación que se utiliza en este desarrollo tiene como base la siguiente estructura:

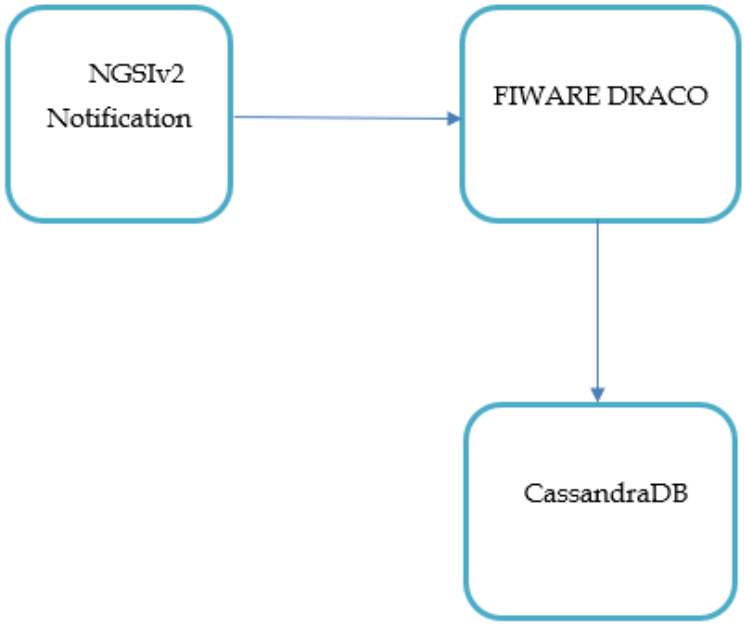

Figura 15. Escenario presentado para el proyecto

El escenario presentado en este proyecto está compuesto de tres partes importantes

- Notificaciones NGSIv2:
  Normalmente estos datos de entradas vienen de la arquitectura IoT que trabaja con protocolos como MQTT, COAP, HTTP, entre otros, estos datos son generados por la lectura de los sensores instalados para la implementación de proyectos de ciudades inteligentes y son procesados en un Context Broker el cual almacena los datos cuando se presenta una variación, pero no tiene historial de estos, es decir que siempre tiene el último cambio. Para la implementación de este proyecto la parte de notificaciones se genera a través de un script manualmente. [18]
  La estructura del script lleva los parámetros del formato NGSIv2 y se envía bajo el protocolo HTTP.

```
curl "http://127.0.0.1:5060/v2/notify" -v -s -S --header "Content-Type:
application/json" --header "Accept: application/json" --header "Fiware-Service:
qsg" --header "Fiware-ServicePath: test" -d "{"subscriptionId":
"51c0ac9ed714fb3b37d7d5a8","data": [{"temperature": {"type": "Float", "value":
30.73,"metadata": {}},"type": "Room","id": "Room1" }]}"
```

Figura 16. Script de notificación NGSIv2

- Fiware Draco:

  En esta parte se implementa el procesador NGSIToCassandra para poder hacerlo se debe de realizar lo siguiente:

  Primero, vaya a su navegador y abra Draco usando esta URL http://localhost:8080/nifi. La siguiente imagen muestra el escenario de Apache NiFi donde se va a ejecutar el proyecto.

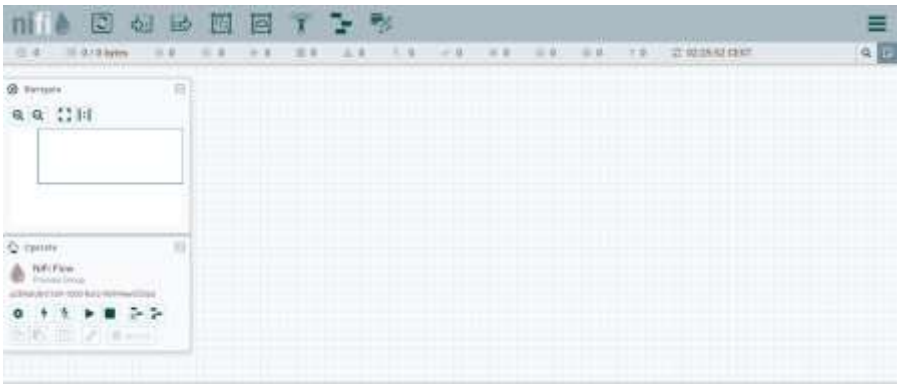

Figura 17. Espacio del usuario de Draco

Ahora vaya a la barra de herramientas de Componente que se encuentra en la sección superior de la interfaz gráfica de usuario de NiFi, encuentre el ícono del procesador arrástrelo y suéltelo dentro del espacio de usuario de Draco. En este punto, se mostrará una ventana emergente con una lista de todos los procesadores disponibles. Por favor, seleccione el procesador NGSIToCassandra.

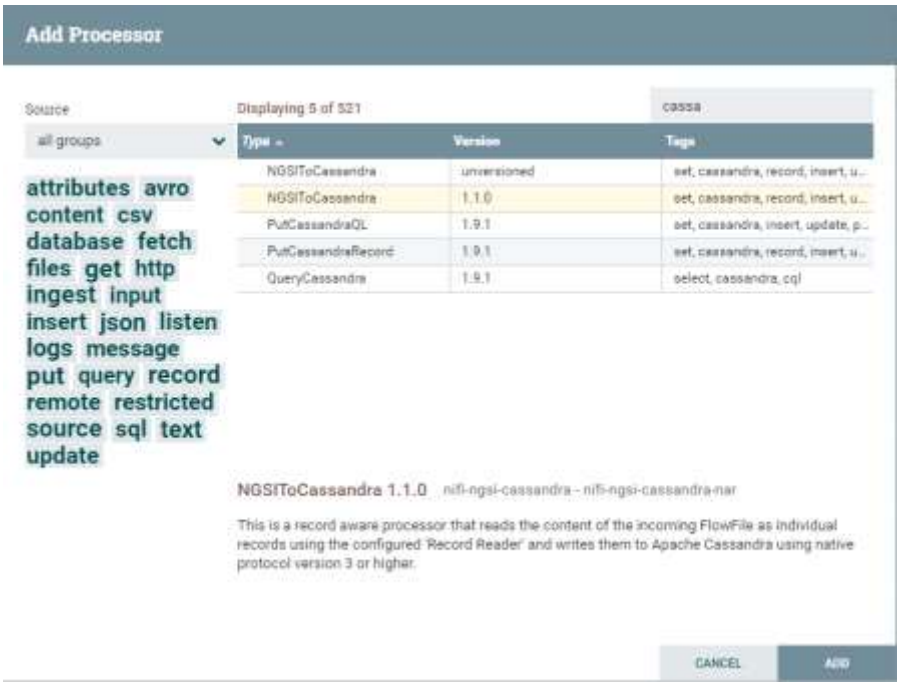

Figura 18. Lista de procesadores en Draco

Luego de seleccionar el procesador indicado en el paso anterior se puede observar que se tienes algunos avisos advertencias por parte del Apache Nifi como muestra en la figura.

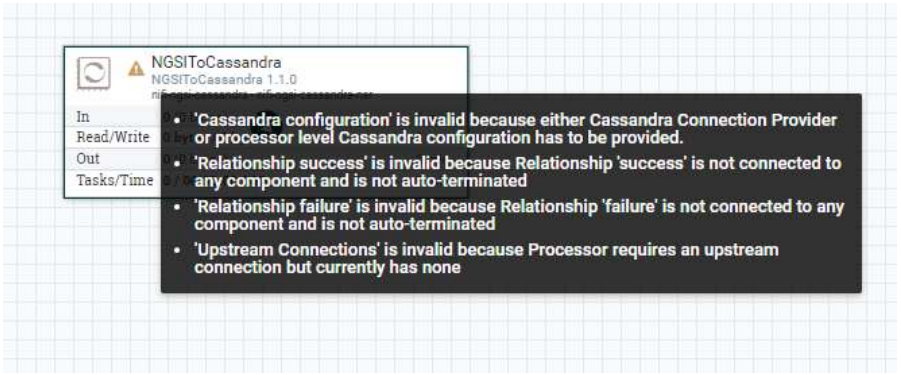

Figura 19. Procesador NGSIToCassandra

Para eliminar estas advertencias se debe de realizar las siguientes configuraciones:

- El procesador debe de estar conectado hacia el controlador Cassandra Conenection Provider el cual se encuentra en la lista por defecto de Apache Nifi para poder habilitarlo se debe ingresar en configuraciones y dar clic en el icono de habilitar.

Figura 20. Conexión con el controlador

- Debe de tener habilitadas las opciones de relationship tanto como la de success y failure activadas.

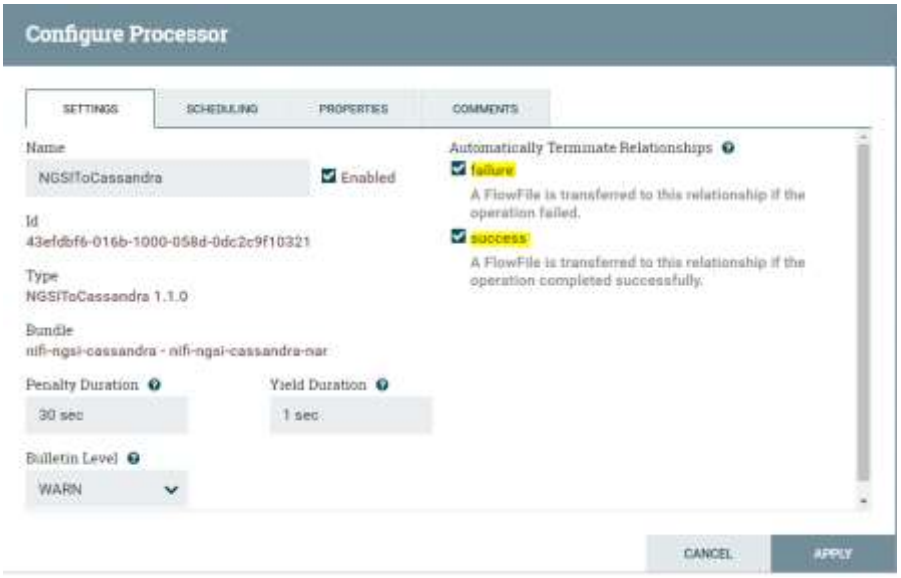

Figura 21. Configuración del procesador NGSIToCassandra

- Por último, el procesador debe de estar sin advertencia como lo indica la siguiente figura.

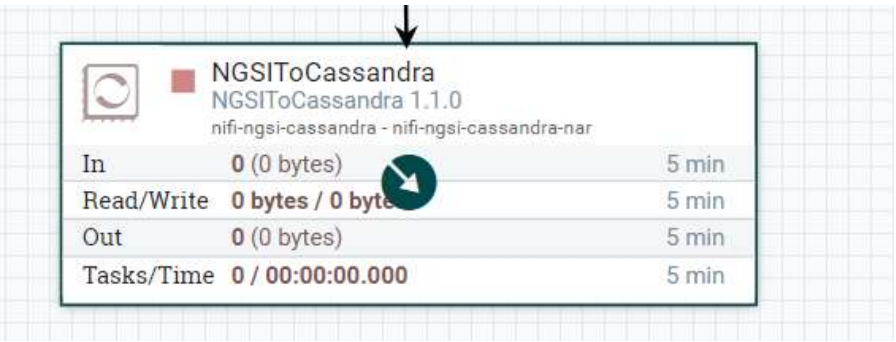

Figura 22. Procesador NGSIToCassandra sin advertencias de posibles fallos

El siguiente paso es configurar el procesador ListenHTTP que genera las notificaciones NGSIv2 el cual se encuentra en el panel de componentes luego de tenerlo en el escenario del usuario se debe realizar las siguientes configuraciones.

- En la propiedad de Base Path se debe colocar v2/notify indica la ruta donde se va a ejecutar el script que genera las notificaciones.

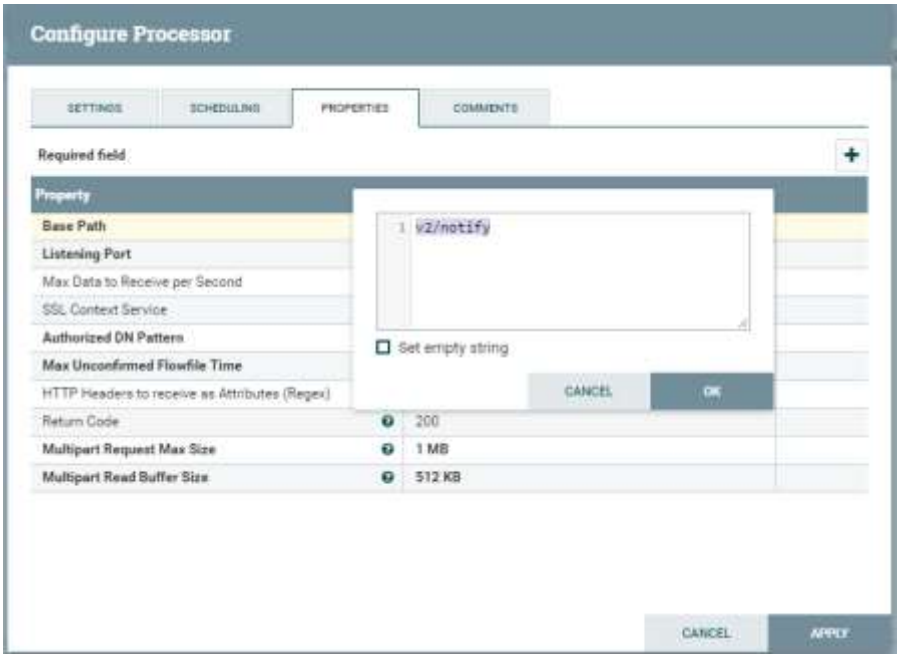

Figura 23. Configuración de ruta del procesador ListenHTTP

- Se debe colocar el puerto que va a escuchar el procesador se escogió el puerto 5060.

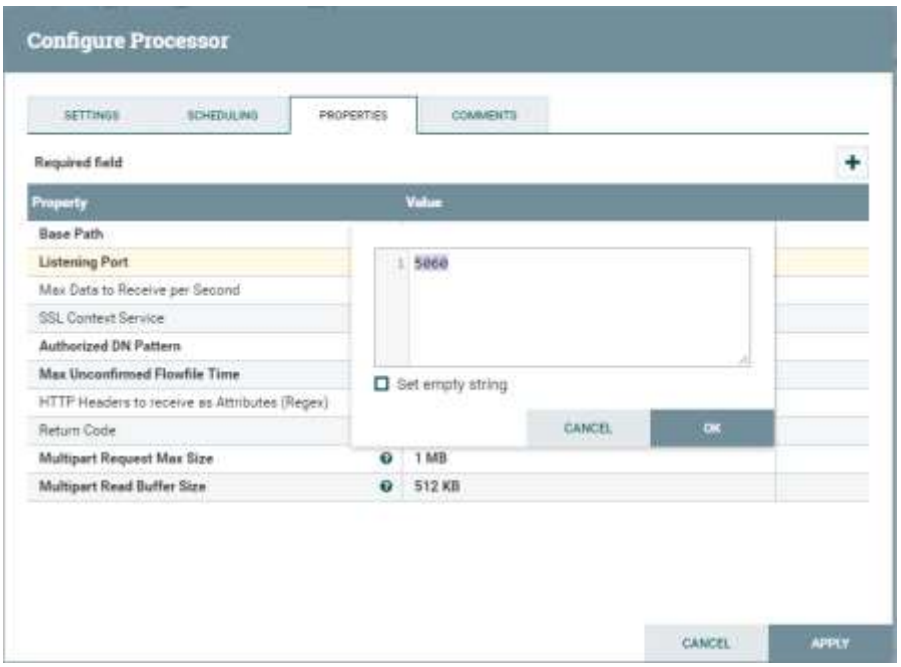

Figura 24. Configuración de puerto del procesador ListenHTTP

- Por último, se realiza la conexión entre el procesador que genera las notificaciones con el procesador NGSItoCassandra los resultados obtenidos serán presentados en ultimo capitulo.

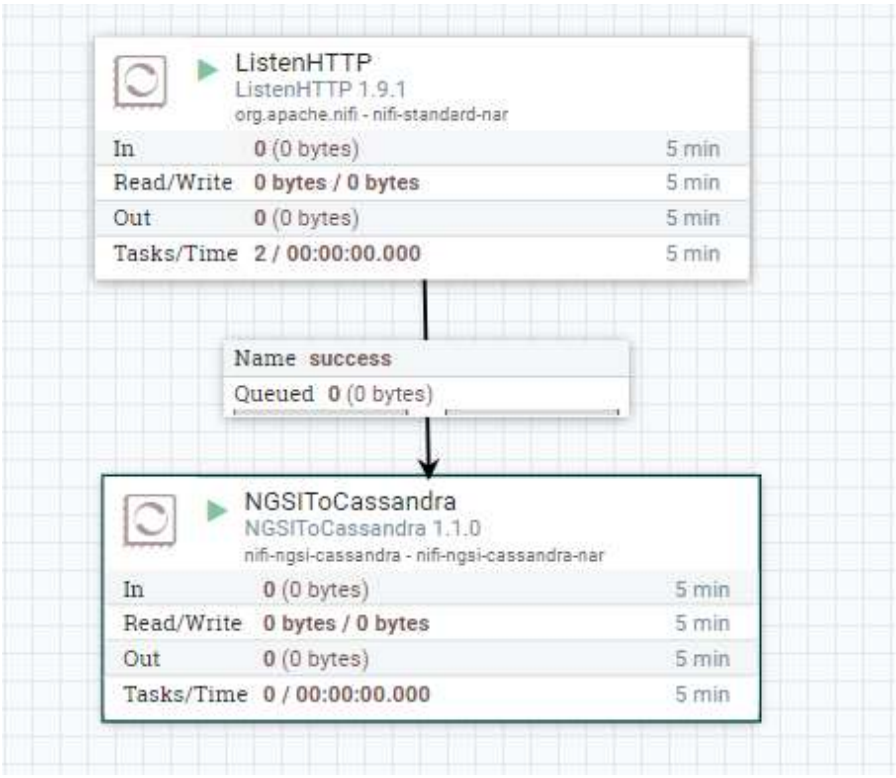

Figura 25. Conexión entre los Procesadores NGSIToCassandra y ListenHTTP

- CassandraDB:

  En esta parte del proyecto se establece la conexión y escritura con la base de datos. Para poder realizar esto se tiene que iniciar el proceso de CassandraDB el cual establece las conexiones con los diferentes cluster y keyspaces. Luego que comprueba que su entorno no tiene problemas puede realizar la escritura y almacenar los datos.

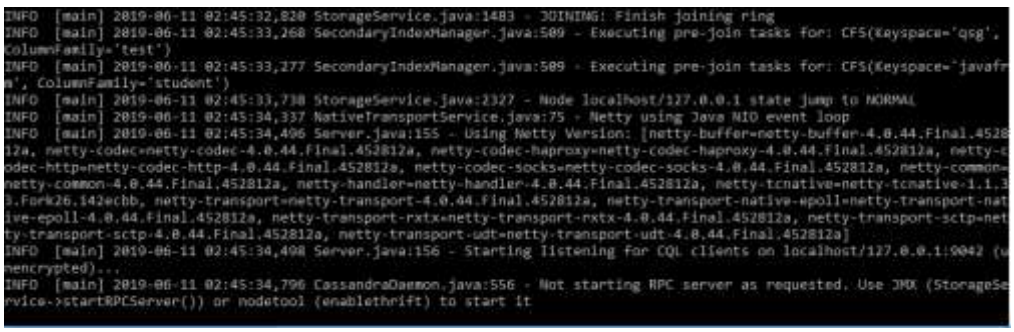

Figura 26. Inicio del proceso de Cassandra DB en el ordenador

Para llevar a cabo la conexión y escritura con la base de datos CassandraDB a la implementación del proyecto, se programa en el lenguaje JAVA sobre el software IntelliJ. El resultado de este programa es un archivo con extensión NAR el cual se lo incluye en las librerías de Apache NiFi para que se pueda visualizar en la lista de procesadores y poder utilizarlo en el entorno gráfico. La programación está dividida en dos clases de java importantes la cuales se van a detallar a continuación:

- AbstractCassandraProcessor.
  En esta clase se define la conexión de la base de datos, la creación del cluster y se define las propiedades del procesador NGSIToCassandra.

  Con respecto a las propiedades del procesador se define los nombres de los campos, el valor por defecto, la descripción que indica lo que realiza y la acción o el valor que debe de tener para su funcionamiento.

```
public static final int DEFAULT_CASSANDRA_PORT = [redacted];

// Common descriptors
static final PropertyDescriptor CONNECTION_PROVIDER_SERVICE = new PropertyDescriptor.Builder()
        .name("cassandra-connection-provider")
        .displayName("Cassandra Connection Provider")
        .description("Specifies the Cassandra connection providing controller service to be used to
        connect to Cassandra cluster.")
        .required(false)
        .identifiesControllerService(CassandraSessionProviderService.class)
        .build();

static final PropertyDescriptor NGSI_VERSION = new PropertyDescriptor.Builder()
        .name("ngsi-version")
        .displayName("NGSI Version")
        .description("The version of NGSI of your incoming events. You can choose between v2 for
        NGSIv2 and ld for NGSI-LD. NGSI-LD is not supported yet ")
        .required(false)
        .allowableValues("v2")
        .defaultValue("v2")
        .addValidator(StandardValidators.NON_EMPTY_VALIDATOR)
        .build();
```

Figura 27. Código de las propiedades del procesador NGSIToCassandra

Luego se configura la conexión hacia la base de datos y la creación del cluster.

```
@OnScheduled
public void onScheduled(ProcessContext context) {
    final boolean connectionProviderIsSet = context.getProperty(CONNECTION_PROVIDER_SERVICE).isSet();

    if (connectionProviderIsSet) {
        CassandraSessionProviderService sessionProvider = context.getProperty(
        CONNECTION_PROVIDER_SERVICE).asControllerService(CassandraSessionProviderService.class);
        cluster.set(sessionProvider.getCluster());
        cassandraSession.set(sessionProvider.getCassandraSession());
        return;
    }

    try {
        connectToCassandra(context);
    } catch (NoHostAvailableException nhae) {
        getLogger().error("No host in the Cassandra cluster can be contacted successfully to
        execute this statement", nhae);
        getLogger().error(nhae.getCustomMessage(10, true, false));
        throw new ProcessException(nhae);
    } catch (AuthenticationException ae) {
        getLogger().error("Invalid username/password combination", ae);
        throw new ProcessException(ae);
    }
}
```

Figura 28. Código para la conexión de la base de datos y la creación del cluster

- CassandraBackend
  Esta clase define los métodos CRUD que se aplica en la base de datos CassandraDB; es decir se establece la creación, lectura, actualización y la eliminación de una tabla. También se define la construcción del keyspace.

```
public String createTable(String keyspace, String tableName, String
attrPersistence){

    String query = "CREATE TABLE IF NOT EXISTS "+ keyspace+"."+tableName
    +getFieldsForCreate(attrPersistence);
    System.out.println(query);
    return query;
}
```

Figura 29. Código de Creación de una Tabla

```
public String createKeyspace(String keyspace)
{
    //Query
    String query = "CREATE KEYSPACE IF NOT EXISTS "+keyspace+" WITH
    replication " + "= {'class':'SimpleStrategy',
    'replication_factor':1};";
    return query;
} // createDatabase
```

Figura 30. Código de Creación de un Keyspace

```
public String buildKeyspace(String service,boolean enableEncoding,boolean enableLowercase) throws Exception {
    String name;

    if (enableEncoding) {
        name = NGSICharsets.encodeCassandra((enableLowercase)?service.toLowerCase():service);
    } else {
        name = NGSICharsets.encode((enableLowercase)?service.toLowerCase():service, false, true);
    } // if else

    if (name.length() > NGSIConstants.CASSANDRA_MAX_KEYSPACE_NAME_LEN) {
        throw new Exception("Building KEYSPACE name '" + name
                + "' and its length is greater than " + NGSIConstants.CASSANDRA_MAX_KEYSPACE_NAME_LEN);
    } // if

    return name;
} // buildKeyspace
```

Figura 31. Código para la construcción de un keyspace

Siguiendo los pasos y especificaciones mostradas en esta sección queda finalizada la implementación respetando los lineamientos del diseño propuesto las pruebas y los resultados obtenidos se detalla en el siguiente capítulo.

4 Resultados Obtenidos

Para realizar la prueba de ejecución del escenario propuesto es necesario, hacer uso de varios componentes, en primer lugar, se debe de iniciar los servicios de Apahce NiFi y de la base de datos CassandraDB como se hizo referencia en capítulos anteriores. Luego una vez arrancado los procesos antes mencionados se debe de revisar las conexiones entre los procesadores tanto como el de NGSIToCassandra como el de ListenHTTP para poder empezar el inicio del escenario planteado. Finalmente se genera el Script y se revisa si se escribe en la base de datos.

A continuación, se indica los pasos a seguir y la obtención de los resultados.

- Inicio del proceso de Cassandra DB.

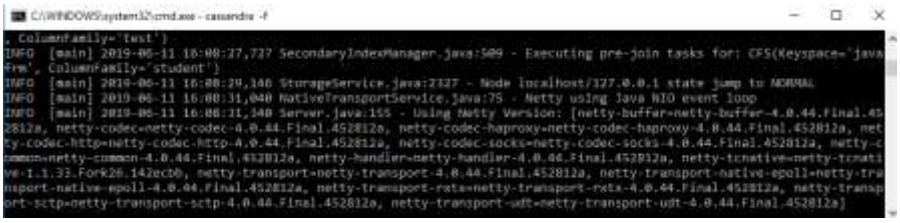

Figura 32. Inicio de Cassnadra DB

- Inicio del proceso Apache Nifi.

Figura 33. Inicio del proceso de Apache Nifi

- Comprobar que los dos procesadores estén sin errores y sin advertencias

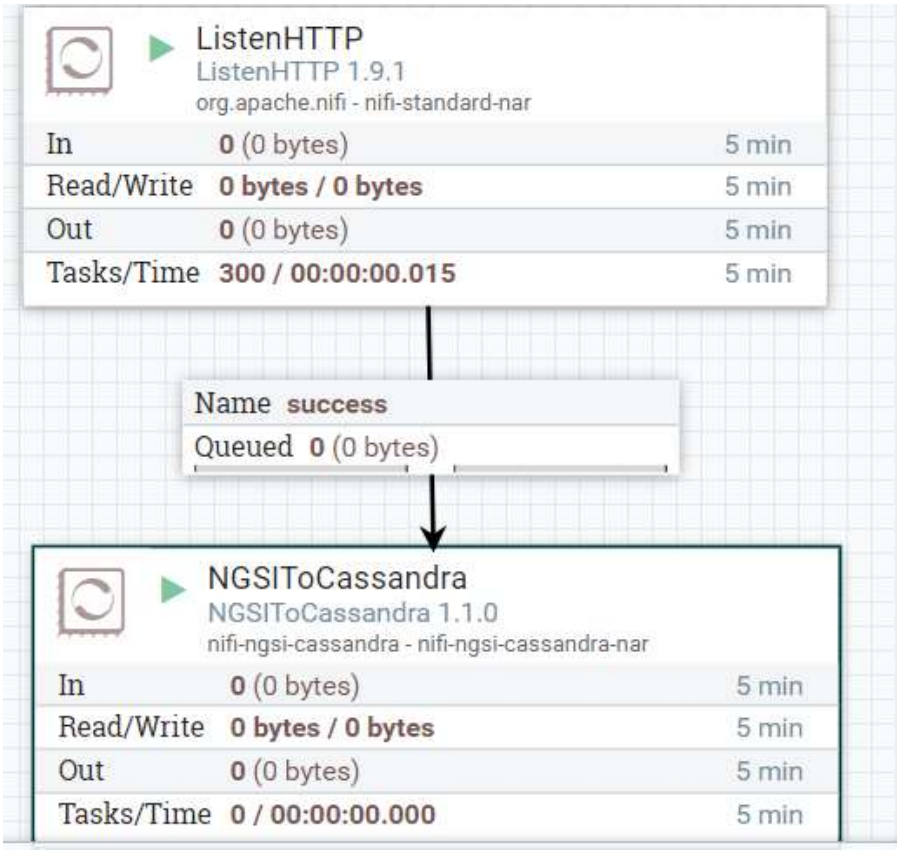

Figura 34. Procesadores sin errores y sin advertencias

- Generar el Script de notificaciones NGDIv2.

Figura 35. Generar Script de notificaciones

- Revisar si el script se escribió en la base de datos.

Figura 36. Revisión de escritura en la base de datos

5 Conclusiones

Por medio de la realización del proyecto presentado en este documento, se ha permitido realizar el diseño y la implementación de un procesador NGSIToCassandra que permite gestionar y procesar notificaciones con formato NGSIv2. Estas son enviadas por los sensores o equipos de capturas de datos de una arquitectura IoT para la implementación en los proyectos de Ciudades Inteligentes. La arquitectura y tecnologías utilizadas permiten que esta implementación se utilice en entornos donde se manejan grandes volúmenes de datos.

La arquitectura presentada en este escenario en combinación con las tecnologías utilizadas, permiten que se pueda implementar y desarrollar el proyecto de forma fácil, rápida y confiable gracias a la utilización de FIWARE-DRACO para la implementación, despliegue, integración, y comunicación de cada uno de los servicios.

Las pruebas realizadas en este proyecto permitieron comprobar que las prestaciones de FIWARE-DRACO se adaptan y dan facilidades para poder trabajar en conjunto con diversas tecnologías de sistemas de almacenamiento, sin importar la base de datos escogida. El trabajo con bases de datos como CassandraDB, requiere en la mayoría de los casos, conocer bien el formato de los datos para definir adecuadamente la estructura en la que se van a almacenar y poder optimizar los resultados de las consultas desde la etapa de diseño.

El desarrollo de este proyecto permite que se pueda realizar una adaptación a otros entornos, debido a la flexibilidad del software desarrollado. Por esa razón se puede usar este proyecto para la colaboración en el desarrollo de trabajos a futuros en trabajos basados en la arquitectura IoT. Una posible mejora sería la de conseguir que los datos de entrada no solo tengan el formato NGSIv2, por el contrario, los datos pueden ser de varios formatos siempre y cuando sean estandarizados y que cumpla con los lineamientos de persistencia de datos definidos por la comunidad FIWARE.

Finalmente, este proyecto forma parte del repositorio Github “FIWARE-DRACO” para la colaboración de la comunidad FIWARE la cual se encarga en desarrollar tecnologías y soluciones sobre los nuevos sistemas de Ciudades inteligentes.

Bibliografía

[1] T. Marcos Paramio, AENOR (2015, Octubre.) Las Normas para las Ciudades Inteligentes [Online] Available: https://www.paradigmadigital.com /dev/cassandra-la-dama-de-las-bases-de-datos-nosql/

[2] S. García Gómez, Smart City Solutions, Telefónica I+D (2015, Agosto.) Fiware: Una plataforma abierta y estándar para Ciudades Inteligentes [Blog] Available: https://www.esmartcity.es/comunicaciones/i-congreso-ciudades-inteligentes-fiware

[3] T. Marcos Paramio, AENOR (2015, Agosto.) Normalización en ciudades inteligentes – España [Online] Available: https://portal.aenormas.aenor.com /descargasweb/normas/aenor-Spanish-standardization-on-Smart-Cities-CTN-78.pdf

[4] H.Villarejo Galende, REE (2015, Junio.) Smart cities una apuesta de la Unión Europea para mejorar los servicios públicos urbanos [Online] Available: https://dialnet.unirioja.es/servlet/articulo?codigo=5488698

[5] Kumar. R., (2014, Mayo.) Open Source Solution for Cloud Computing Platform Using OpenStack [Online] Available: https://edisciplinas.usp.br/ pluginfile.php/318402/course/section/93669/V3I5201427_3.pdf

[6] FIWARE Foundation. (2018, Enero.) Developers [Online] Available: https://www.fiware.org/developers/

[7] FIWARE-DRACO. (2018, Mayo.) Docs/Home [Online] Available: https://fiware-draco.readthedocs.io/en/latest/index.html

[8] FIWARE-ORION. (2018, Mayo.) Docs/Home [Online] Available: https://fiware-orion.readthedocs.io/en/master/

[9] FIWARE-DRACO, (2018, Mayo.). docs installation and administration guide [Online] Available: https://fiware-draco.readthedocs.io

[10] D. Calvo. (2018, Junio.) Apache Nifi. [ONLINE] Available: http://www.diegocalvo.es/nifi/

[11] Apache NiFi. (2019, Abril.) NiFi Documentation/overview [Online] Available: https://nifi.apache.org/docs.html

[12] Apache NiFi. (2018, Julio.) Overview [Online] Available: https://hortonworks.com/apache/nifi/NIFI APACHE

[13] Apache NiFi. (2015, Enero.) Simple tasks in nifi find the processor [Online] Available: https://kisstechdocs.wordpress.com/2015/01/15/simple-tasks-in-nifi-find-the-processor-you-want/

[14] The Apache Software Foundation. (2019, Abril.) Home [Online] Available: http://cassandra.apache.org/

[15] Datastax. (2019, Junio.) Cassandra Documentation [Online] Available: https://docs.datastax.com/en/landing_page/doc/landing_page/cassandra.html

[16] The Apache Software Foundation. (2019, Abril.) Docuementation [Online] Available: http://cassandra.apache.org/doc/latest/getting_started/index.html

[17] M. Zaforas. (2016, Marzo.) Cassandra, la dama de las bases de datos NoSQL. [Online] Available: https://www.paradigmadigital.com/dev/cassandra-la-dama-de-las-bases-de-datos-nosql/

[18] R. Ferreiroa. (2018, Agosto.) Internet of Things se comunica a través de MQTT. [Online] Available: https://www.ciberfabrica.com/internet-of-things-mqtt/ s